1

INTRODUCTION

Unit Structure

Introduction: The World Wide Web, WWW Architecture, Web Search Engine, Web Crawling, Web Indexing, Web Searching, Search Engine Optimization and Limitations, Introduction to the Semantic Web

1. Introduction:

1.1 THE WORLD WIDE WEB

World Wide Web

The Web's historic logo designed by Robert Cailliau

Inventor	Sir Tim Berners Lee[1]
Launch year	1990
Company	CERN
Available?	Worldwide

The **World Wide Web**, abbreviated as **WWW** and commonly known as **the Web**, is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them by using hyperlinks. Using concepts from earlier hypertext systems, British engineer and computer scientist Sir Tim Berners-Lee, now the Director of the World Wide Web Consortium, wrote a proposal in March 1989 for what would eventually become the World Wide Web. He was later joined by Belgian computer scientist Robert Cailliau while both were working at CERN in Geneva, Switzerland.

"The World-Wide Web (W3) was developed to be a pool of human knowledge, which would allow collaborators in remote sites to share their ideas and all aspects of a common project."

➢ History of the World Wide Web

Arthur C. Clarke was quoted in Popular Science in May 1970, in which he predicted that satellites would one day "bring the accumulated knowledge of the world to our fingertips" using an office console that would combine the functionality of the xerox, telephone, TV and a small computer so as to allow both data transfer and video conferencing around the globe.

In March 1989, Tim Berners-Lee wrote a proposal that referenced ENQUIRE, a database and software project he had built in 1980, and described a more elaborate information management system.

With help from Robert Cailliau, he published a more formal proposal (on November 12, 1990) to build a "Hypertext project" called "WorldWideWeb" (one word, also "W3") as a "web" of "hypertext documents" to be viewed by "browsers" using a client–server

A NeXT Computer was used by Berners-Lee as the world's first web server and also to write the first web browser, WorldWideWeb, in 1990. By Christmas 1990, Berners-Lee had built all the tools necessary for a working Web: the first web browser (which was a web editor as well); the first web server; and the first web pages, which described the project itself.

➢ **WWW prefix**

Many web addresses begin with *www*, because of the long-standing practice of naming Internet hosts (servers) according to the services they provide. The hostname for a web server is often *www*, as it is *ftp* for an FTP server, and *news* or *nntp* for a USENET news server. These host names appear as Domain Name System (DNS) subdomain names, as in www.example.com.

When a single word is typed into the address bar and the return key is pressed, some web browsers automatically try adding "www." to the beginning of it and possibly ".com", ".org" and ".net" at the end. For example, typing 'microsoft<enter>' may resolve to *http://www.microsoft.com/* and 'openoffice<enter>' to *http://www.openoffice.org*. This feature was beginning to be included in early versions of Mozilla Firefox.

The 'http://' or 'https://' part of web addresses *does* have meaning: These refer to Hypertext Transfer Protocol and to HTTP Secure and so define the communication protocol that will be used to request and receive the page, image or other resource. The HTTP network protocol is fundamental to the way the World Wide Web works, and the encryption involved in HTTPS adds an essential layer if confidential information such as passwords or bank details are to be exchanged over the public internet.

➢ **Standards**

Many formal standards and other technical specifications and software define the operation of different aspects of the World Wide Web, the Internet, and computer information exchange.

Usually, when web standards are discussed, the following publications are seen as foundational:

- Recommendations for markup languages, especially HTML and XHTML, from the W3C. These define the structure and interpretation of hypertext documents.
- Recommendations for stylesheets, especially CSS, from the W3C.
- Standards for ECMAScript (usually in the form of JavaScript), from Ecma International.

- Recommendations for the Document Object Model, from W3C.
- Additional publications provide definitions of other essential technologies for the World Wide Web, including, but not limited to, **Uniform Resource Identifier** (URI), **HyperText Transfer Protocol** (HTTP)

➢ **Speed issues**

Frustration over congestion issues in the Internet infrastructure and the high latency that results in slow browsing has led to an alternative, pejorative name for the World Wide Web: the *World Wide Wait*.[69] Speeding up the Internet is an ongoing discussion over the use of peering and QoS technologies. Other solutions to reduce the World Wide Wait can be found at W3C.[70] Standard guidelines for ideal Web response times are:[71]

- 0.1 second (one tenth of a second). Ideal response time. The user doesn't sense any interruption.
- 1 second. Highest acceptable response time. Download times above 1 second interrupt the user experience.
- 10 seconds. Unacceptable response time. The user experience is interrupted and the user is likely to leave the site or system.

➢ **Caching**

If a user revisits a Web page after only a short interval, the page data may not need to be re-obtained from the source Web server. Almost all web browsers cache recently obtained data, usually on the local hard drive. HTTP requests sent by a browser will usually only ask for data that has changed since the last download. If the locally cached data are still current, it will be reused. Caching helps reduce the amount of Web traffic on the Internet. The decision about expiration is made independently for each downloaded file, whether image, stylesheet, JavaScript, HTML, or whatever other content the site may provide. Thus even on sites with highly dynamic content, many of the basic resources only need to be refreshed occasionally. Web site designers find it worthwhile to collate resources such as CSS data and JavaScript into a few site-wide files so that they can be cached efficiently. This helps reduce page download times and lowers demands on the Web server.

Questions based on WWW:

1. Explain the invention of WWW?
2. What are the Advantage of WWW?
3. What were the speed issues caused by WWW?

1.2 WORLD WIDE WEB ARCHITECTURE

The *World Wide Web* (*WWW*, or simply *Web*) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (*URI*).

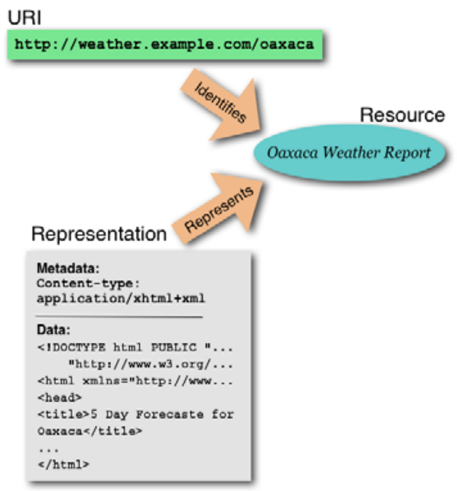
All TAG participants, past and present, have had a hand in many parts of the design of the Web. In the Architecture document, they emphasize what characteristics of the Web must be preserved when inventing new technology. They notice where the current systems don't work well, and as a result show weakness. This document is a pithy summary of the wisdom of the community.

This scenario illustrates the three architectural bases of the Web :

- **Identification (§2)**. URIs are used to identify resources. In this travel scenario, the resource is a periodically updated report on the weather in Oaxaca, and the URI is “http://weather.example.com/oaxaca”.
- **Interaction (§3)**. Web agents communicate using standardized protocols that enable interaction through the exchange of messages which adhere to a defined syntax and semantics. By entering a URI into a retrieval dialog or selecting a hypertext link, Nadia tells her browser to perform a retrieval action for the resource identified by the URI. In this example, the browser sends an HTTP GET request (part of the HTTP protocol) to the server at "weather.example.com", via TCP/IP port 80, and the server sends back a message containing what it determines to be a representation of the resource as of the time that representation was generated. Note that this example is specific to hypertext browsing of information—other kinds of interaction are possible, both within browsers and through the use of other types of Web agent; our example is intended to illustrate one common interaction, not define the range of possible interactions or limit the ways in which agents might use the Web.
- **Formats (§4).** Most protocols used for representation retrieval and/or submission make use of a sequence of one or more messages, which taken together contain a payload of representation data and metadata, to transfer the representation between agents. The choice of interaction protocol places limits on the formats of representation data and metadata that can be transmitted. HTTP, for example, typically transmits a single octet stream plus metadata, and uses the "Content-Type" and

"Content-Encoding" header fields to further identify the format of the representation. In this scenario, the representation transferred is in XHTML, as identified by the "Content-type" HTTP header field containing the registered Internet media type name, "application/xhtml+xml". That Internet media type name indicates that the representation data can be processed according to the XHTML specification.

The diagram shows the relationship between identifier, resource, and representation.



```
Metadata:
Content-type:
application/xhtml+xml

Data:
<!DOCTYPE html PUBLIC "...
    "http://www.w3.org/...
<html xmlns="http://www...
<head>
<title>5 Day Forecaste for
Oaxaca</title>
...
</html>
```

- **Global Identifiers**

Global naming leads to global network effects.

- **Identify with URIs**

To benefit from and increase the value of the World Wide Web, agents should provide URIs as identifiers for resources.

- **URIs Identify a Single Resource**

Assign distinct URIs to distinct resources.

- **Avoiding URI aliases**

A URI owner SHOULD NOT associate arbitrarily different URIs with the same resource.

- **Consistent URI usage**

An agent that receives a URI SHOULD refer to the associated resource using the same URI, character-by-character.

- **Reuse URI schemes**

A specification SHOULD reuse an existing URI scheme (rather than create a new one) when it provides the desired properties of identifiers and their relation to resources.

- **URI opacity**

Agents making use of URIs SHOULD NOT attempt to infer properties of the referenced resource.

- **Reuse representation formats**

New protocols created for the Web SHOULD transmit representations as octet streams typed by Internet media types.

- **Data-metadata inconsistency**

Agents MUST NOT ignore message metadata without the consent of the user.

- **Metadata association**

Server managers SHOULD allow representation creators to control the metadata associated with their representations.

- **Safe retrieval**

Agents do not incur obligations by retrieving a representation.

- **Available representation**

A URI owner SHOULD provide representations of the resource it identifies

- **Reference does not imply dereference**

An application developer or specification author SHOULD NOT require networked retrieval of representations each time they are referenced.

- **Consistent representation**

A URI owner SHOULD provide representations of the identified resource consistently and predictably.

- **Version information**

A data format specification SHOULD provide for version information.

- **Namespace policy**

An XML format specification SHOULD include information about change policies for XML namespaces.

- **Extensibility mechanisms**

A specification SHOULD provide mechanisms that allow any party to create extensions.

- **Extensibility conformance**

Extensibility MUST NOT interfere with conformance to the original specification.

- **Unknown extensions**

A specification SHOULD specify agent behavior in the face of unrecognized extensions.

- **Separation of content, presentation, interaction**

A specification SHOULD allow authors to separate content from both presentation and interaction concerns.

- **Link identification**

A specification SHOULD provide ways to identify links to other resources, including to secondary resources (via fragment identifiers).

- **Web linking**

A specification SHOULD allow Web-wide linking, not just internal document linking.

- **Generic URIs**

A specification SHOULD allow content authors to use URIs without constraining them to a limited set of URI schemes.

- **Hypertext links**

A data format SHOULD incorporate hypertext links if hypertext is the expected user interface paradigm.

- **Namespace adoption**

A specification that establishes an XML vocabulary SHOULD place all element names and global attribute names in a namespace.

- **Namespace documents**

The owner of an XML namespace name SHOULD make available material intended for people to read and material optimized for software agents in order to meet the needs of those who will use the namespace vocabulary.

- **QNames Indistinguishable from URIs**

Do not allow both QNames and URIs in attribute values or element content where they are indistinguishable.

- **QName Mapping**

A specification in which QNames serve as resource identifiers MUST provide a mapping to URIs.

- **XML and "text/*"**

In general, a representation provider SHOULD NOT assign Internet media types beginning with "text/" to XML representations.

- **XML and character encodings**

In general, a representation provider SHOULD NOT specify the character encoding for XML data in protocol headers since the data is self-describing.

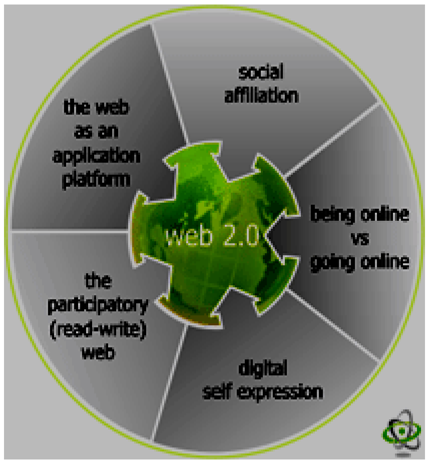
- **Orthogonality**

Orthogonal abstractions benefit from orthogonal specifications.

- **Error recovery**

Agents that recover from error by making a choice without the user's consent are not acting on the user's behalf.

➢ <u>Web 2.0</u>


social affiliation
being online vs going online
digital self expression
the participatory (read-write) web
the web as an application platform
web 2.0

Within a very short stint of 17 years since Tim Berners Lee came up with the concept of World Wide Web, the growth of Internet has become unimaginable. Initially the web pages on the Internet were static html pages and the hosting servers found it very easy to support innumerous web pages on a single server since the demand on the server due to the use of static web pages was very low.

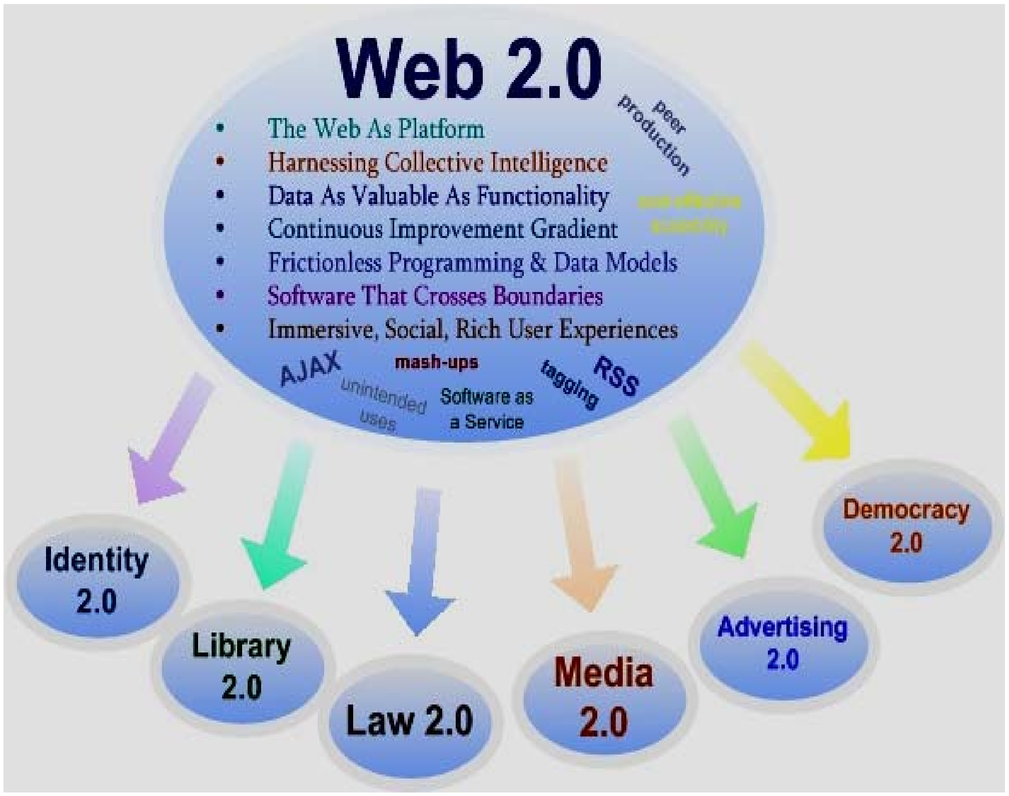
But, of Late, websites have started using dynamic contents and the demand on the servers hosting those pages has increased enormously. Web 2.0 concept penetrates into the Internet right here.

Web 2.0 is providing the required support to host the collection of second-generation web applications/web pages that utilize the dynamic technologies like AJAX enabling the user to make dynamic updates in their web page and providing a bunch of value added services for the customer. Google continues to be the vanguard of this innovation of using web2.0 applications! Google Suggest, A9 search of Amazon, Gmail, Google Maps are a few web URLs that have initiated the growth of Web 2.0 technology over the past few years! Ad-on to this list are YouTube and MySpace. The list of websites that have adopted this technology as on date is much more.

In the year and a half since, the term "Web 2.0" has clearly taken hold, with more than 9.5 million citations in Google. But there's still a huge amount of disagreement about just what Web 2.0 means, with some people decrying it as a meaningless marketing buzzword, and others accepting it as the new conventional wisdom.

Questions based on WWW Architecture?

1. Explain the Architecture of WWW?
2. Explain the relationship of the three architectural bases of the Web?
3. Explain the next version of Web1.0?


Web 2.0
The Web As Platform
Harnessing Collective Intelligence
Data As Valuable As Functionality
Continuous Improvement Gradient
Frictionless Programming & Data Models
Software That Crosses Boundaries
Immersive, Social, Rich User Experiences
peer production
AJAX
mash-ups
unintended uses
Software as a Service
tagging
RSS
Identity 2.0
Library 2.0
Law 2.0
Media 2.0
Advertising 2.0
Democracy 2.0

1.3 WEB SEARCH ENGINE

A **web search engine** is designed to search for information on the World Wide Web. The search results are usually presented in a list of results and are commonly called *hits*. The information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

➢ **How Search Engines Work**

The term "search engine" is often used generically to describe both crawler-based search engines and human-powered directories. These two types of search engines gather their listings in radically different ways.

- **Crawler-Based Search Engines**

Crawler-based search engines, such as Google, create their listings automatically. They "crawl" or "spider" the web, then people search through what they have found.

If you change your web pages, crawler-based search engines eventually find these changes, and that can affect how you are listed. Page titles, body copy and other elements all play a role.

- **Human-Powered Directories**

A human-powered directory, such as the Open Directory, depends on humans for its listings. You submit a short description to the directory for your entire site, or editors write one for sites they review. A search looks for matches only in the descriptions submitted.

Changing your web pages has no effect on your listing. Things that are useful for improving a listing with a search engine have nothing to do with improving a listing in a directory. The only exception is that a good site, with good content, might be more likely to get reviewed for free than a poor site.

- **"Hybrid Search Engines" Or Mixed Results**

In the web's early days, it used to be that a search engine either presented crawler-based results or human-powered listings. Today, it extremely common for both types of results to be presented. Usually, a hybrid search engine will favor one type of listings over another. For example, MSN Search is more likely to present human-powered listings from LookSmart. However, it does also present crawler-based results (as provided by Inktomi), especially for more obscure queries.

➢ <u>A List of All-Purpose Search Engines</u>

1. <u>Google</u>

In the last few years, Google has attained the ranking of the #1 search engine on the Net, and consistently stayed there.

2. Yahoo

Yahoo is a search engine, subject directory, and web portal. Yahoo provides good search results powered by their own search engine database, along with many other Yahoo search options.

3. MSN Search

MSN Search is Microsoft's offering to the search world. Learn about MSN Search: its ease of use, cool search features, and simple advanced search accessibility.

4. AOL Search

Learn why so many people have chosen AOL Search to be their jumping off point when searching the Web. With its ease of use, simple accessibility, and nifty search features, AOL Search has carved itself a unique niche in the search world.

5. Ask

Ask.com is a very popular crawler-based search engine. Some of the reasons that it has stayed so popular with so many people are its ease of use, cool search features (including Smart Answers), and powerful search interface.

6. AlltheWeb

AlltheWeb is a search engine whose results are powered by Yahoo. AlltheWeb has some very advanced search features that make it a good search destination for those looking for pure search.

7. <u>AltaVista</u>

AltaVista has been around in various forms since 1995, and continues to be a viable presence on the Web.

8. <u>Lycos</u>

Lycos has been around for over ten years now (started in September of 1995), and has some interesting search features to offer. Learn more about Lycos Search, Lycos Top 50, Lycos Entertainment, and more.

9. <u>Gigablast</u>

Gigablast is a search engine with some interesting features, good advanced search power, and an excellent user experience.

10. <u>Cuil</u>

Cuil is a slick, minimalist search engine with a magazine look and feel. Cuil claims to have indexed over 121 billion Web pages, so it is quite a large search engine, plus, the search interface returns quite a few related categories and search terms that can potentially launch your search net quite a bit wider.

<u>Questions based on Web Search Engine:</u>

1. How Web Search Engines are useful for Web search?
2. How Web Search Engine works? List all the Search Engines.

1.4 WEB CRAWLING

A web crawler is a relatively simple automated program, or script, that methodically scans or "crawls" through Internet pages to create an index of the data it's looking for. Alternative names for a web crawler include web spider, web robot, bot, crawler, and automatic indexer.

When a search engine's web crawler visits a web page, it "reads" the visible text, the hyperlinks, and the content of the various tags used in the site, such as keyword rich meta tags. Using the information gathered from the crawler, a search engine will then determine what the site is about and index the information. The website is then included in the search engine's database and its page ranking process.

Search engines, however, are not the only users of web crawlers. Linguists may use a web crawler to perform a textual analysis; that is, they may comb the Internet to determine what words are commonly used today. Market researchers may use a web crawler to determine and assess trends in a given market. There are numerous nefarious uses of web crawlers as well. In the end, a web crawler may be used by anyone seeking to collect information out on the Internet.

Web crawlers may operate one time only, say for a particular one-time project. If its purpose is for something long term, as is the case with search engines, they may be programed to comb through the Internet periodically to determine whether there has been any significant changes. If a site is experiencing heavy traffic or technical difficulties, the spider may be programmed to note that and revisit the site again, hopefully after the technical issues have subsided.

Web crawling is an important method for collecting data on, and keeping up with, the rapidly expanding Internet. A vast number of web pages are continually being added every day, and information is constantly changing. A web crawler is a way for the search engines and other users to regularly ensure that their databases are up to date.

- **Crawler Overview**

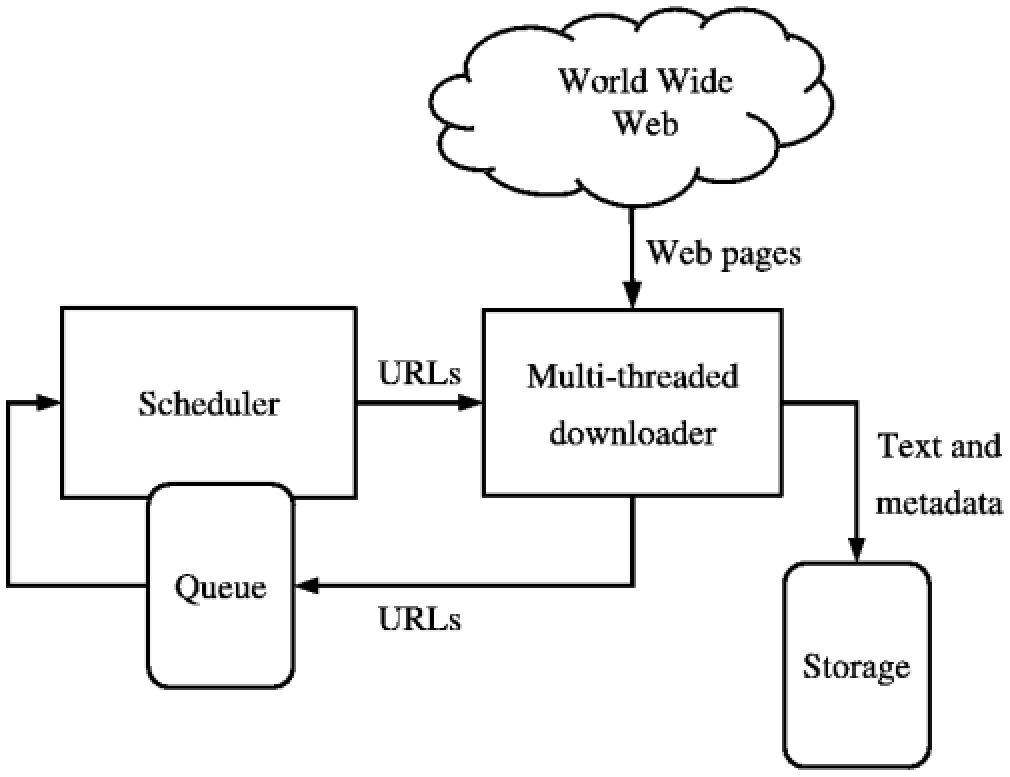
In this article, it will introduce a simple Web crawler with a simple interface, to describe the crawling story in a simple C# program. My crawler takes the input interface of any Internet navigator to simplify the process. The user just has to input the URL to be crawled in the navigation bar, and click "Go".

The crawler has a URL queue that is equivalent to the URL server in any large scale search engine. The crawler works with multiple threads to fetch URLs from the crawler queue. Then the retrieved pages are saved in a storage area as shown in the figure.

The fetched URLs are requested from the Web using a C# Sockets library to avoid locking in any other C# libraries. The retrieved pages are parsed to extract new URL references to be put in the crawler queue, again to a certain depth

Questions based on web crawling:

1. What is Web Crawling ? How is it useful?
2. Explain Web Crawling Overviews.



1.5 WEB INDEXING

Web indexing (or "Internet indexing") includes back-of-book-style indexes to individual websites or an intranet, and the creation of keyword metadata to provide a more useful vocabulary for Internet or onsite search engines. With the increase in the number of periodicals that have articles online, web indexing is also becoming important for periodical websites.

Back-of-the-book-style web indexes may be called "web site A-Z indexes." The implication with "A-Z" is that there is an alphabetical browse view or interface. This interface differs from that of a browse through layers of hierarchical categories (also known as a taxonomy) which are not necessarily alphabetical, but are also found on some web sites.

Web site A-Z indexes have several advantages over Search Engines - Language is full of homographs and synonyms and not all the references found will be relevant.

A human-produced index has someone check each and every part of the text to find everything relevant to the search term, while a Search Engine leaves the responsibility for finding the information with the enquirer.

Although an A-Z index could be used to index multiple sites, rather than the multiple pages of a single site, this is unusual.

Metadata web indexing involves assigning keywords or phrases to web pages or web sites within a meta-tag field, so that the web page or web site can be retrieved with a search engine that is customized to search the keywords field. This may or may not involve using keywords restricted to a controlled vocabulary list

Questions based on web indexing:

1. Explain Web Indexing.

1.6 WEB SEARCHING

Web Searching defines searching of information on World Wide Web
The search technology uses semantic and extraction capabilities to recognize the best answer from within a sea of relevant pages.

Web Searching is done through an engine called Web Search Engine
The search results are generally presented in a list of results and are often called *hits*. The information may consist of web pages, images, information and other types of files.

Some search engines also mine data available in databases or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

➢ **Web Search Tools**

- **Choose the Right Tool:** There are three distinct types of Web search tools: Web directories, Web indexes, and specialized databases.

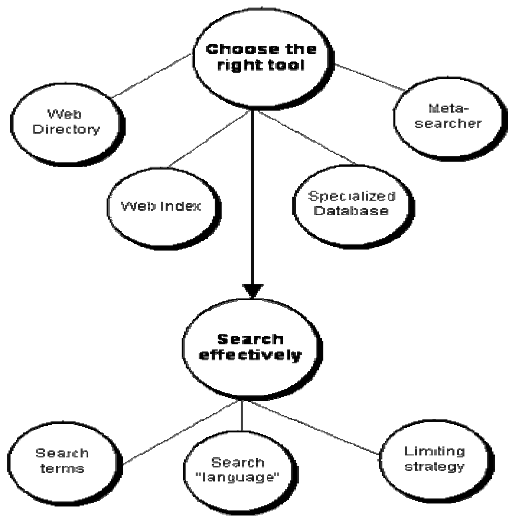
- **Browse the Best Sites:** Web directories are selective. They provide short descriptions of Web sites and are a good place to start a general search or to survey what's available on a broad topic.

- **Search for Specific Information:** Web indexes ("search engines") are huge databases containing the full text of millions of Web pages. Start here when your search is specific or well-defined. Specialized factual databases (the "invisible Web") are also good sources for answering specific questions.

- **Meta-Search to Save Time:** A meta-searcher allows you to send one search to many different Web tools (key directories and indexes) simultaneously.

- **Smart Search Techniques:** Use effective search techniques in all of these sources. Choose good search terms, speak the "language" of the search tool (symbols, boolean operators) and use limiting to focus search results.

-



Questions based on Web Searching:

1. Explain Web searching? What are the web searching tool?

1.7 SEARCH ENGINE OPTIMIZATION (SEO) AND LIMITATIONS

➢ **Search Engine Optimization (SEO)**

SEO is an acronym for "search engine optimization" or "search engine optimizer." Deciding to hire an SEO is a big decision that can potentially improve your site and save time, but you can also risk damage to your site and reputation. Make sure to research the potential advantages as well as the damage that an irresponsible SEO can do to your site. Many SEOs and other agencies and consultants provide useful services for website owners, including:

- Review of your site content or structure
- Technical advice on website development: for example, hosting, redirects, error pages, use of JavaScript
- Content development
- Management of online business development campaigns
- Keyword research
- SEO training
- Expertise in specific markets and geographies.
- SEO is a key part of any web site to drive and promote traffic, and not just any traffic, the most relevant traffic possible.

➢ **Limitations**

- **Great Expectations**

Search engine optimisation features, such as those mentioned on our SEO page, will help to get your website noticed, but they won’t work miracles. People with a website to advertise tend to expect too much of search engines, either through underestimating the sheer number of websites that touch on a particular topic, or through overestimating the abilities of the search engines.

They also overestimate the ability of internet users to make the most of what the search engines offer. Few users delve beyond the first couple of pages of search results, and fewer still read the search engines’ guidelines to efficient searching

You should be aware that merely submitting a website to a search engine does not guarantee that the search engine will include that website in its search results. Different search engines work in different ways, with varying levels of efficiency. They also work at different speeds: some become aware of new websites almost instantly, while others may take weeks.

- **Ratings**

Search engines, imperfect though they are, attempt to rank websites mainly according to two factors:

✓ relevance, which can be increased by skilled search engine optimisation,
✓ and popularity, which is largely out of the hands of the website's owner and its designer.

Most search engines place great emphasis on the number of significant links to particular websites, and are able to detect the approximate number and quality of these links. The greater the number of relevant links, the more significant the website will appear to be.

Obviously, the number of links to your website will be largely out of your control, but there are legitimate ways to increase the number. Co–operation between websites that deal with a particular topic, in which each website includes links to the others, is one way of increasing your profile with the search engines..

The sad truth is that most new websites start near the bottom of most search engines' rankings and work their way up over time. You should be very wary of organisations claiming to guarantee that your website will instantly appear near the top of the rankings. There are many underhand ways of achieving this, and the search engines are wise to most of them. It is quite possible that your website will indeed appear near the top of the rankings, but it won't stay there for long if the wrong methods are used. Once the search engines identify fraud, they will penalise your website, and perhaps even blacklist it.

Questions based on SEO:

1. What is SEO? How is SEO useful in day-to-day life?
2. Explain the limitations of SEO.

1.8 INTRODUCTION TO THE SEMANTIC WEB

The Semantic Web is a web that is able to describe things in a way that computers can understand.

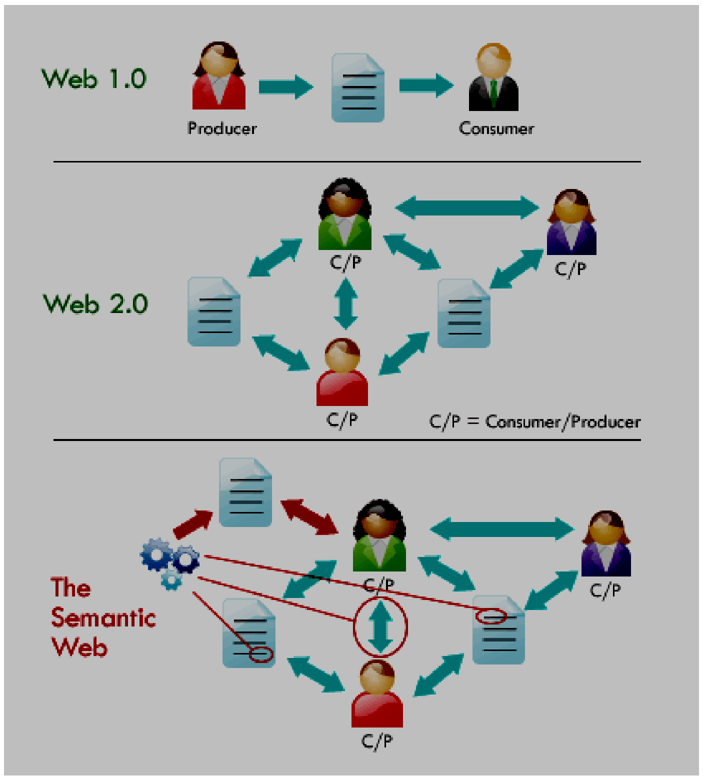
- The Beatles was a popular band from Liverpool.

- John Lennon was a member of the Beatles.
- "Hey Jude" was recorded by the Beatles.

Sentences like the ones above can be understood by people. But how can they be understood by computers?

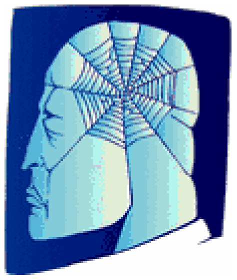
Statements are built with syntax rules. The syntax of a language defines the rules for building the language statements. But how can syntax become semantic?

This is what the Semantic Web is all about. Describing things in a way that computers applications can understand it.



The Semantic Web is **not about links** between web pages.

The Semantic Web describes the **relationships between things** (like A is a part of B and Y is a member of Z) and the **properties of things** (like size, weight, age, and price)



"If HTML and the Web made all the online documents look like one huge **book**, RDF, schema, and inference languages will make all the data in the world look like one huge **database**"

Tim Berners-Lee, Weaving the Web, 1999

An Introduction To Social Networks

Wikipedia defines a social network service as a service which "*focuses on the building and verifying of online social networks for communities of people who share interests and activities, or who are interested in exploring the interests and activities of others, and which necessitates the use of software.*".

What Can Social Networks Be Used For?

Social networks can provide a range of benefits to members of an organisation:

Support for learning: Social networks can enhance informal learning and support social connections within groups of learners and with those involved in the support of learning.

Support for members of an organisation: Social networks can potentially be used my all members of an organisation, and not just those involved in working with students. Social networks can help the development of communities of practice.

Engaging with others: Passive use of social networks can provide valuable business intelligence and feedback on institutional services (although this may give rise to ethical concerns).

Ease of access to information and applications: The ease of use of many social networking services can provide benefits to users by simplifying access to other tools and applications. The Facebook Platform provides an example of how a social networking service can be used as an environment for other tools.

Common interface: A possible benefit of social networks may be the common interface which spans work / social boundaries. Since such services are often used in a personal capacity the interface and the way the service works may be familiar, thus minimising training and support needed to exploit the services in a professional context. This can, however, also be a barrier to those who wish to have strict boundaries between work and social activities

A report published by OCLC provides the following definition of social networking sites: "*Web sites primarily designed to facilitate interaction between users who share interests, attitudes and activities, such as Facebook, Mixi and MySpace.*"

Examples of popular social networking services include:

Facebook: Facebook is a social networking Web site that allows people to communicate with their friends and exchange information. In May 2007 Facebook launched the Facebook Platform which provides a framework for developers to create applications that interact with core Facebook features [3].

MySpace: MySpace [4] is a social networking Web site offering an interactive, user-submitted network of friends, personal profiles, blogs and groups, commonly used for sharing photos, music and videos.

Ning: An online platform for creating social Web sites and social networks aimed at users who want to create networks around specific interests or have limited technical skills [5].

Twitter: Twitter [6] is an example of a micro-blogging service [7]. Twitter can be used in a variety of ways including sharing brief information with users and providing support for one's peers.

Opportunities And Challenges

The popularity and ease of use of social networking services have excited institutions with their potential in a variety of areas. However effective use of social networking services poses a number of challenges for institutions including long-term sustainability of the services; user concerns over use of social tools in a work or study context; a variety of technical issues and legal issues such as copyright, privacy, accessibility

Exercise:

1. Explain Semantic Web? How does it differ from Web1.0 and Web2.0?
2. What is search engine? Explain its working.
3. What is web crawler? Explain how it works.
4. Explain the architecture of web describing various components.
5. Explain the difference between website and web portal.
6. What is search engine optimization? State its importance.
7. Give the overview of different search engines.
8. Write a note on caching.

❖❖❖❖

2

SERVLETS

Unit Structure

Introduction to servlets, Servlet Life Cycle, Servlet Classes, Servlet, ServletRequest, ServletResponse, ServletContext, Threading Models, HttpSessions

2.1 INTRODUCTION TO SERVLETS

SERVLET: A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.

This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server.

❖ What are JAVA Servlets?

A **Servlet** is a Java class which conforms to the **Java Servlet API**, a protocol by which a Java class may respond to HTTP requests. Thus, a software developer may use a servlet to add dynamic content to a Web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets are the Java counterpart to non-Java dynamic Web content technologies such as CGI and ASP.NET. Servlets can maintain state in session variables across many server transactions by using HTTP cookies, or URL rewriting.

Servlets are snippets of Java programs which run inside a **Servlet Container**. A Servlet Container is much like a Web Server which handles user requests and generates responses. Servlet Container is different from a Web Server because it can not only serve requests for static content like HTML page, GIF images, etc., it can also contain Java Servlets and JSP pages to generate dynamic response. Servlet Container is responsible for loading and maintaining the lifecycle of the a Java Servlet. Servlet Container can be used standalone or more often used in conjunction with a Web server. Example of a Servlet Container is Tomcat and that of Web Server is Apache.

2.1.1 Servlets vs CGI

The traditional way of adding functionality to a Web Server is the Common Gateway Interface (CGI), a language-independent interface that allows a server to start an external process which gets information about a request through environment variables, the command line and its standard input stream and writes response data to its standard output stream. Each request is answered in a separate process by a separate instance of the CGI program, or CGI script (as it is often called because CGI programs are usually written in interpreted languages like Perl).

Servlets have several advantages over CGI:

- A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
- A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.

2.2 SERVER LIFE CYCLE:

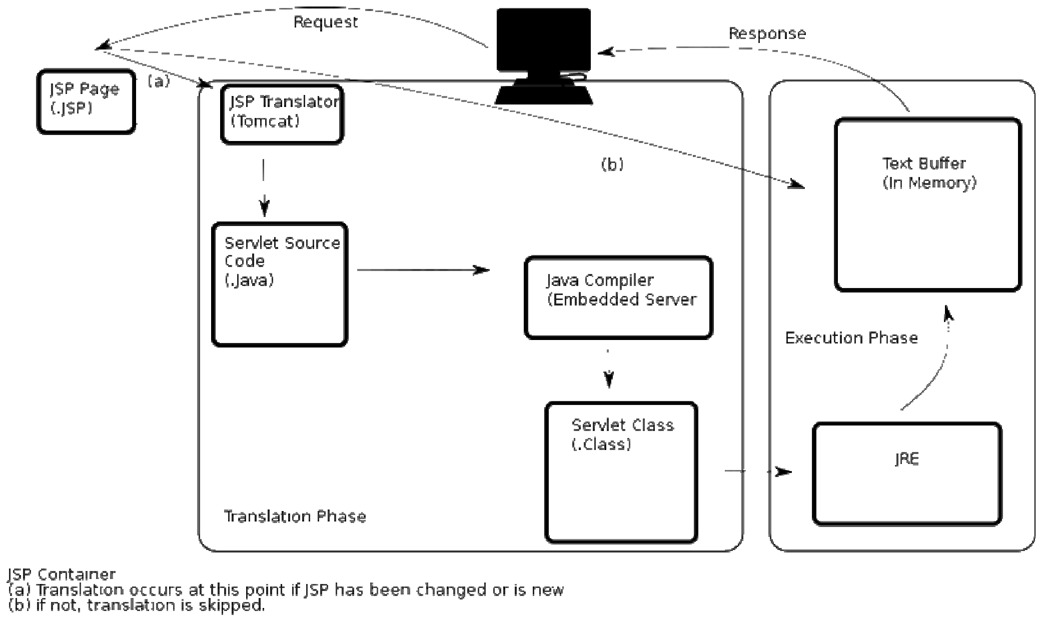
The servlet lifecycle consists of the following steps:

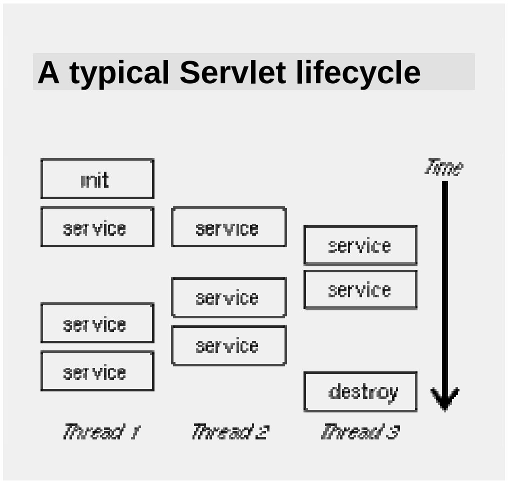
1. The servlet class is loaded by the Web container during start-up.
2. The Web container calls the init() method. This method initializes the servlet and must be called before the servlet can service any requests. In the entire life of a servlet, the init() method is called only once.

3. After initialization, the servlet can service client requests. Each <u>request</u> is serviced in its own separate thread. The Web container calls the service() method of the servlet for every request. The service() method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the servlet must provide an implementation for these methods. If a request for a method that is not implemented by the servlet is made, the method of the parent class is called, typically resulting in an error being returned to the requester.
4. Finally, the Web container calls the destroy() method that takes the servlet out of service. The destroy() method, like init(), is called only once in the lifecycle of a servlet.

Here is a simple servlet that just generates HTML. Note that HttpServlet is a subclass of GenericServlet, an implementation of the Servlet interface. The service() method dispatches requests to methods doGet(), doPost(), doPut(), doDelete(), etc., according to the HTTP request.

<u>LIFECYCLE:</u>


Request
Response
JSP Page (.JSP)
(a)
JSP Translator (Tomcat)
(b)
Text Buffer (In Memory)
Servlet Source Code (.Java)
Java Compiler (Embedded Server
Execution Phase
Servlet Class (.Class)
JRE
Translation Phase
JSP Container
(a) Translation occurs at this point if JSP has been changed or is new
(b) if not, translation is skipped.



2.2.1 The Basic Servlet Architecture

1. A Servlet, in its most general form, is an instance of a class which implements the javax.servlet.Servlet interface. Most Servlets, however, extend one of the standard implementations of that interface, namely javax.servlet.GenericServlet and javax.servlet.http.HttpServlet. In this tutorial we'll be discussing only HTTP Servlets which extend the javax.servlet.http.HttpServlet class.
2. In order to initialize a Servlet, a server application loads the Servlet class (and probably other classes which are referenced by the Servlet) and creates an instance by calling the no-args constructor. Then it calls the Servlet's init(ServletConfig config) method. The Servlet should performe one-time setup procedures in this method and store the ServletConfig object so that it can be retrieved later by calling the Servlet's getServletConfig() method. This is handled by GenericServlet. Servlets which extend GenericServlet (or its subclass HttpServlet) should call super.init(config) at the beginning of the init method to make use of this feature. The ServletConfig object contains Servlet parameters and a reference to the Servlet's ServletContext. The init method is guaranteed to be called only once during the Servlet's lifecycle. It does not need to be thread-safe because the service method will not be called until the call to init returns.
3. When the Servlet is initialized, its service(ServletRequest req, ServletResponse res) method is called for every request to the Servlet. The method is called concurrently (i.e. multiple threads may call this method at the same time) so it should be implemented in a thread-safe manner. Techniques for ensuring that the service method is not called concurrently, for the cases where this is not possible.
4. When the Servlet needs to be unloaded (e.g. because a new version should be loaded or the server is shutting down) the

destroy() method is called. There may still be threads that execute the service method when destroy is called, so destroy has to be thread-safe. All resources which were allocated in init should be released in destroy. This method is guaranteed to be called only once during the Servlet's lifecycle.

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML
4.0 " +
            "Transitional//EN\">\n" +
            "<html>\n" +
            "<head><title>Hello WWW</title></head>\n" +
            "<body>\n" +
            "<h1>Hello WWW</h1>\n" +
            "</body></html>");
  }
}
```

2.3 SERVLET CLASSES:

Servlets are actually simple Java classes which must implement the javax.servlet.Servlet interface. This interface contains a total of five methods. **javax.servlet** package already provides two classes which implement this interface i.e. *GenericServlet* and *HttpServlet*. So all we need to do is to extend one of these classes and override the method(s) you need for your Servlet.

- *GenericServlet* is a very simple class which only implements the *javax.servlet.Servlet* interface and provides only basic functionality.
- On the other hand, HttpServlet is a more useful class which provides methods to work with the HTTP protocol. So if your Servlet works with HTTP protocol (in most cases this will be the case) then you *should* extend javax.servlet.http.HttpServlet class to build Servlets and this is what we are going to do in this article.

Servlets once initialized are kept in memory. So every request which the Servlet Container receives, is delegated to the in-memory Java Servlet which then generates the response. This 'kept in memory' feature makes Java Servlets, a fast and efficient method of building web applications.

2.3.1 SERVLET:

A Servlet is an object that receives a request and generates a response based on that request. The basic servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package javax.servlet.http defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the Web server and a client. Servlets may be packaged in a WAR file as a Web application.

Servlets can be generated automatically by JavaServer Pages (JSP) compiler, or alternately use template engines such as WebMacro or Apache Velocity to generate HTML. Often servlets are used in conjunction with JSPs in a pattern called "Model 2", which is a flavor of the model-view-controller pattern.

2.3.2 SERVLET REQUEST

This interface is for getting data from the client to the servlet for a service request. Network service developers implement the ServletRequest interface. The methods are then used by servlets when the service method is executed; the ServletRequest object is passed as an argument to the service method.

Some of the data provided by the ServletRequest object includes parameter names and values, attributes, and an input stream. Subclasses of ServletRequest can provide additional protocol-specific data. For example, HTTP data is provided by the interface HttpServletRequest, which extends ServletRequest. This framework provides the servlet's only access to this data.

- **getAttribute**

public abstract Object getAttribute(String name)

Returns the value of the named attribute of the request, or null if the attribute does not exist. This method allows access to request information not already provided by the other methods in this interface. Attribute names should follow the same convention as package names. The following predefined attributes are provided.

Attribute Name	Attribute Type	Description
javax.net.ssl. cipher_suite	string	The string name of the SSL cipher suite in use, if the request was made using SSL
javax.net.ssl. peer_certificates	array of javax.security. cert.X509 Certificate	The chain of X.509 certificates which authenticates the client. This is only available when SSL is used with client authentication is used.
javax.net. ssl.session	javax.net.ssl. SSLSession	An SSL session object, if the request was made using SSL.

The package (and hence attribute) names beginning with java.*, and javax.* are reserved for use by Javasoft. Similarly, com.sun.* is reserved for use by Sun Microsystems.

Parameters:

name - the name of the attribute whose value is required

- **getAttributeNames**

public abstract Enumeration getAttributeNames()
Returns an enumeration of attribute names contained in this request.

- **getCharacterEncoding**

public abstract String getCharacterEncoding()
Returns the character set encoding for the input of this request.

- **getContentLength**

public abstract int getContentLength()
Returns the size of the request entity data, or -1 if not known. Same as the CGI variable CONTENT_LENGTH.

- **getContentType**

public abstract String getContentType()
Returns the Internet Media Type of the request entity data, or null if not known. Same as the CGI variable CONTENT_TYPE.

- **getInputStream**

public abstract ServletInputStream getInputStream() throws IOException Returns an input stream for reading binary data in the request body.

> **Throws:** IllegalStateException
>
> if getReader has been called on this same request.
>
> **Throws:** IOException
>
> on other I/O related errors.
>
> **See Also:**
>
> getReader

- **getParameter**

public abstract String getParameter(String name)
Returns a string containing the lone value of the specified parameter, or null if the parameter does not exist. For example, in an HTTP servlet this method would return the value of the specified query string parameter. Servlet writers should use this method only when they are sure that there is only one value for the parameter. If the parameter has (or could have) multiple values, servlet writers should use getParameterValues. If a multiple valued parameter name is passed as an argument, the return value is implementation dependent.

Parameters:

name - the name of the parameter whose value is required.

- **getParameterNames**

public abstract Enumeration getParameterNames()
Returns the parameter names for this request as an enumeration of strings, or an empty enumeration if there are no parameters or the input stream is empty. The input stream would be empty if all the data had been read from the stream returned by the method getInputStream.

- **getParameterValues**

public abstract String[] getParameterValues(String name)
Returns the values of the specified parameter for the request as an array of strings, or null if the named parameter does not exist.

For example, in an HTTP servlet this method would return the values of the specified query string or posted form as an array of strings.

Parameters:

name - the name of the parameter whose value is required.

- **getProtocol**

public abstract String getProtocol()
Returns the protocol and version of the request as a string of the form <protocol>/<major version>.<minor version>. Same as the CGI variable SERVER_PROTOCOL.

- **getScheme**

public abstract String getScheme()
Returns the scheme of the URL used in this request, for example "http", "https", or "ftp". Different schemes have different rules for constructing URLs, as noted in RFC 1738. The URL used to create a request may be reconstructed using this scheme, the server name and port, and additional information such as URIs.

- **getServerName**

public abstract String getServerName()
Returns the host name of the server that received the request. Same as the CGI variable SERVER_NAME.

- **getServerPort**

public abstract int getServerPort()
Returns the port number on which this request was received. Same as the CGI variable SERVER_PORT.

- **getReader**

public abstract BufferedReader getReader() throws IOException
Returns a buffered reader for reading text in the request body. This translates character set encodings as appropriate.

Throws: UnsupportedEncodingException

if the character set encoding is unsupported, so the text can't be correctly decoded.

Throws: IllegalStateException if getInputStream has been called on this same request.

Throws: IOException on other I/O related errors.

- **getRemoteAddr**

public abstract String getRemoteAddr()
Returns the IP address of the agent that sent the request. Same as the CGI variable REMOTE_ADDR.

- **getRemoteHost**

public abstract String getRemoteHost()
Returns the fully qualified host name of the agent that sent the request. Same as the CGI variable REMOTE_HOST.

- **setAttribute**

public abstract void setAttribute (String key, Object o)
This method stores an attribute in the request context; these attributes will be reset between requests. Attribute names should follow the same convention as package names.

The package (and hence attribute) names beginning with java.*, and javax.* are reserved for use by Javasoft. Similarly, com.sun.* is reserved for use by Sun Microsystems.

Parameters:

key - a String specifying the name of the attribute

o - a context object stored with the key.

Throws: IllegalStateException if the named attribute already has a value.

- **getRealPath**

public abstract String getRealPath(String path)
getRealPath() is deprecated. *This method has been deprecated in preference to the same method found in the ServletContext interface.*

Applies alias rules to the specified virtual path and returns the corresponding real path, or null if the translation can not be performed for any reason. For example, an HTTP servlet would resolve the path using the virtual docroot, if virtual hosting is enabled, and with the default docroot otherwise. Calling this method with the string "/" as an argument returns the document root.

Parameters:

path - the virtual path to be translated to a real path

2.3.2 SERVLET RESPONSE:

Defines an object to assist a servlet in sending a response to the client. The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.

To send binary data in a MIME body response, use the ServletOutputStream returned by getOutputStream(). To send character data, use the PrintWriter object returned by getWriter(). To mix binary and text data, for example, to create a multipart response, use a ServletOutputStream and manage the character sections manually.

The charset for the MIME body response can be specified with setContentType(java.lang.String). For example, "text/html; charset=Shift_JIS". The charset can alternately be set using setLocale(java.util.Locale). If no charset is specified, ISO-8859-1 will be used. The setContentType or setLocale method must be called before getWriter for the charset to affect the construction of the writer.

Various methods used and in detail:

getCharacterEncoding

public java.lang.String **getCharacterEncoding**()

Returns the name of the charset used for the MIME body sent in this response.

If no charset has been assigned, it is implicitly set to ISO-8859-1 (Latin-1).

See RFC 2047 (http://ds.internic.net/rfc/rfc2045.txt) for more information about character encoding and MIME.

Returns:

a String specifying the name of the charset, for example, ISO-8859-1

getOutputStream

public ServletOutputStream **getOutputStream**()
throws java.io.IOException

Returns a ServletOutputStream suitable for writing binary data in the response. The servlet container does not encode the binary data.

Calling flush() on the ServletOutputStream commits the response. Either this method or getWriter() may be called to write the body, not both.

Returns:

a ServletOutputStream for writing binary data

Throws:

IllegalStateException - if the getWriter method has been called on this response java.io.IOException - if an input or output exception occurred

getWriter

public java.io.PrintWriter **getWriter**()
throws java.io.IOException

Returns a PrintWriter object that can send character text to the client. The character encoding used is the one specified in the charset= property of the setContentType(java.lang.String) method, which must be called *before* calling this method for the charset to take effect.

If necessary, the MIME type of the response is modified to reflect the character encoding used.

Calling flush() on the PrintWriter commits the response.

Either this method or getOutputStream() may be called to write the body, not both.

Returns:

a PrintWriter object that can return character data to the client

Throws:

java.io.UnsupportedEncodingException - if the charset specified in setContentType cannot be used

IllegalStateException - if the getOutputStream method has already been called for this response object

java.io.IOException - if an input or output exception occurred

setContentLength

public void **setContentLength**(int len)
Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.

Parameters:

len - an integer specifying the length of the content being returned to the client; sets the Content-Length header

setContentType

public void **setContentType**(java.lang.String type)
Sets the content type of the response being sent to the client. The content type may include the type of character encoding used, for example, text/html; charset=ISO-8859-4.

If obtaining a PrintWriter, this method should be called first.

Parameters:

type - a String specifying the MIME type of the content

setBufferSize

public void **setBufferSize**(int size)
Sets the preferred buffer size for the body of the response. The servlet container will use a buffer at least as large as the size requested. The actual buffer size used can be found using getBufferSize.

A larger buffer allows more content to be written before anything is actually sent, thus providing the servlet with more time to set appropriate status codes and headers. A smaller buffer decreases server memory load and allows the client to start receiving data more quickly.

This method must be called before any response body content is written; if content has been written, this method throws an IllegalStateException.

Parameters:

size - the preferred buffer size

Throws:

IllegalStateException - if this method is called after content has been written

getBufferSize

public int **getBufferSize**()
Returns the actual buffer size used for the response. If no buffering is used, this method returns 0.

Returns:

the actual buffer size used

flushBuffer

public void **flushBuffer**()
throws java.io.IOException
Forces any content in the buffer to be written to the client. A call to this method automatically commits the response, meaning the status code and headers will be written.

resetBuffer

public void **resetBuffer**()
Clears the content of the underlying buffer in the response without clearing headers or status code. If the response has been committed, this method throws an IllegalStateException.

Since: 2.3

isCommitted

public boolean **isCommitted**()
Returns a boolean indicating if the response has been committed. A commited response has already had its status code and headers written.

Returns: a boolean indicating if the response has been committed

reset

public void **reset**()

Clears any data that exists in the buffer as well as the status code and headers. If the response has been committed, this method throws an IllegalStateException.

Throws:

IllegalStateException - if the response has already been committed

setLocale

public void **setLocale**(java.util.Locale loc)

Sets the locale of the response, setting the headers (including the Content-Type's charset) as appropriate. This method should be called before a call to getWriter(). By default, the response locale is the default locale for the server.

Parameters:

loc - the locale of the response

getLocale

public java.util.Locale **getLocale**()

Returns the locale assigned to the response.

2.3.4 SERVLETCONTEXT:

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per "web application" per Java Virtual Machine. (A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace such as /catalog and possibly installed via a .war file.)

In the case of a web application marked "distributed" in its deployment descriptor, there will be one context instance for each virtual machine. In this situation, the context cannot be used as a location to share global information (because the information won't be truly global). Use an external resource like a database instead.

The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.

Methods and details used in details:

getContext

public ServletContext **getContext**(java.lang.String uripath)
Returns a ServletContext object that corresponds to a specified URL on the server.

This method allows servlets to gain access to the context for various parts of the server, and as needed obtain RequestDispatcher objects from the context. The given path must be begin with "/", is interpreted relative to the server's document root and is matched against the context roots of other web applications hosted on this container.

In a security conscious environment, the servlet container may return null for a given URL.

Parameters:

uripath - a String specifying the context path of another web application in the container.

Returns:

the ServletContext object that corresponds to the named URL, or null if either none exists or the container wishes to restrict this access.

getMajorVersion

public int **getMajorVersion**()
Returns the major version of the Java Servlet API that this servlet container supports. All implementations that comply with Version 2.3 must have this method return the integer 2.

Returns: 2

getMinorVersion

public int **getMinorVersion**()
Returns the minor version of the Servlet API that this servlet container supports. All implementations that comply with Version 2.3 must have this method return the integer 3.

Returns: 3

getMimeType

public java.lang.String **getMimeType**(java.lang.String file)

Returns the MIME type of the specified file, or null if the MIME type is not known. The MIME type is determined by the configuration of the servlet container, and may be specified in a web application deployment descriptor. Common MIME types are "text/html" and "image/gif".

Parameters:

file - a String specifying the name of a file

Returns:

a String specifying the file's MIME type

getResourcePaths

public java.util.Set **getResourcePaths**(java.lang.String path)

Returns a directory-like listing of all the paths to resources within the web application whose longest sub-path matches the supplied path argument. Paths indicating subdirectory paths end with a '/'. The returned paths are all relative to the root of the web application and have a leading '/'. For example, for a web application containing

/welcome.html
/catalog/index.html
/catalog/products.html
/catalog/offers/books.html
/catalog/offers/music.html
/customer/login.jsp
/WEB-INF/web.xml
/WEB-INF/classes/com.acme.OrderServlet.class,

getResourcePaths("/") returns {"/welcome.html", "/catalog/", "/customer/", "/WEB-INF/"}

getResourcePaths("/catalog/") returns {"/catalog/index.html", "/catalog/products.html", "/catalog/offers/"}.

Parameters:

the - partial path used to match the resources, which must start with a /

Returns:

a Set containing the directory listing, or null if there are no resources in the web application whose path begins with the supplied path.

Since:

Servlet 2.3

getResource

public java.net.URL **getResource**(java.lang.String path)
throws java.net.MalformedURLException

Returns a URL to the resource that is mapped to a specified path. The path must begin with a "/" and is interpreted as relative to the current context root.

This method allows the servlet container to make a resource available to servlets from any source. Resources can be located on a local or remote file system, in a database, or in a .war file.

The servlet container must implement the URL handlers and URLConnection objects that are necessary to access the resource.

This method returns null if no resource is mapped to the pathname.

Some containers may allow writing to the URL returned by this method using the methods of the URL class.

The resource content is returned directly, so be aware that requesting a .jsp page returns the JSP source code. Use a RequestDispatcher instead to include results of an execution.

This method has a different purpose than java.lang.Class.getResource, which looks up resources based on a class loader. This method does not use class loaders.

Parameters:

path - a String specifying the path to the resource

Returns:

the resource located at the named path, or null if there is no resource at that path

Throws:

java.net.MalformedURLException - if the pathname is not given in the correct form

getResourceAsStream

publicjava.io.InputStream

getResourceAsStream(java.lang.String path)
Returns the resource located at the named path as an InputStream object.

The data in the InputStream can be of any type or length. The path must be specified according to the rules given in getResource. This method returns null if no resource exists at the specified path.

Meta-information such as content length and content type that is available via getResource method is lost when using this method.

The servlet container must implement the URL handlers and URLConnection objects necessary to access the resource.

This method is different from java.lang. Class. Get Resource As Stream, which uses a class loader. This method allows servlet containers to make a resource available to a servlet from any location, without using a class loader.

Parameters:

name - a String specifying the path to the resource

Returns:

the InputStream returned to the servlet, or null if no resource exists at the specified path

getRequestDispatcher

publicRequestDispatcher
getRequestDispatcher(java.lang.String path)
Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname must begin with a "/" and is interpreted as relative to the current context root. Use getContext to obtain a RequestDispatcher for resources in foreign contexts. This method returns null if the ServletContext cannot return a RequestDispatcher.

Parameters:

path - a String specifying the pathname to the resource

Returns:

a RequestDispatcher object that acts as a wrapper for the resource at the specified path

See Also:

RequestDispatcher, getContext(java.lang.String)

getNamedDispatcher

publicRequestDispatcher
getNamedDispatcher(java.lang.String name)
Returns a RequestDispatcher object that acts as a wrapper for the named servlet.

Servlets (and JSP pages also) may be given names via server administration or via a web application deployment descriptor. A servlet instance can determine its name using ServletConfig.getServletName().

This method returns null if the ServletContext cannot return a Request Dispatcher for any reason.

Parameters:

name - a String specifying the name of a servlet to wrap

Returns:

a RequestDispatcher object that acts as a wrapper for the named servlet

2.4 THREADING MODELS:

In computer science, the term **threaded code** refers to a compiler implementation technique where the generated code has a form that essentially consists entirely of calls to subroutines. The code may be processed by an interpreter, or may simply be a sequence of machine code call instructions.

One of the main advantages of threaded code is that it is very compact, compared to code generated by alternative code generation techniques and alternative calling conventions. This advantage usually comes at the expense of slightly slower execution speed (usually just a single machine instruction). However, sometimes there is a synergistic effect—sometimes more compact code is smaller *and* significantly faster than non-threaded code.[1] A program small enough to fit entirely in a computer processor's cache may run faster than a less-compact program that suffers constant cache misses.

To save space, programmers squeezed the lists of subroutine calls into simple lists of subroutine addresses, and used a small loop to call each subroutine in turn. For example:

```
start:          thread:       pushA:  *sp++ = A
  ip = &thread    &pushA              jump top
top:              &pushB      pushB:  *sp++ = B
  jump *ip++      &add                jump top
                ...         add:   *sp++ = *--sp + *--sp
                                   jump top
```

In this case, decoding the bytecodes is performed once, during program compilation or program load, so it is not repeated each time an instruction is executed. This can save much time and space when decode and dispatch overhead is large compared to the execution cost.

Note, however, addresses in thread for &pushA, &pushB, etc., are two or more bytes, compared to one byte, typically, for the decode and dispatch interpreter described above. In general, instructions for a decode and dispatch interpreter may be any size. For example, a decode and dispatch interpreter to simulate an Intel Pentium decodes instructions that range from 1 to 16 bytes. However, bytecoded systems typically choose 1-byte codes for the most-common operations. Thus, the thread often has a higher space cost than bytecodes. In most uses, the reduction in decode cost outweighs the increase in space cost.

Note also that while bytecodes are nominally machine-independent, the format and value of the pointers in threads generally depend on the target machine which is executing the interpreter. Thus, an interpreter might load a portable bytecode program, decode the bytecodes to generate platform-dependent threaded code, then execute threaded code without further reference to the bytecodes.

The loop is simple, so is duplicated in each handler, removing jump top from the list of machine instructions needed to execute each interpreter instruction. For example:

```
start:          thread:       pushA:  *sp++ = A
  ip = thread      &pushA             jump *ip++
  jump *ip++        &pushB       pushB:  *sp++ = B
                &add                jump *ip++
                ...          add:    *sp++ = *--sp + *--sp
                                   jump *ip++
```

This is called **direct threaded code** (DTC). Although the technique is older, the first widely circulated use of the term "threaded code" is probably Bell's article "Threaded Code" from 1973.

Threading models

Practically all executable threaded code uses one or another of these methods for invoking subroutines (each method is called a "threading model").

- **Direct threading**

Addresses in the thread are the addresses of machine language. This form is simple, but may have overheads because the thread consists only of machine addresses, so all further parameters must be loaded indirectly from memory. Some Forth systems produce direct-threaded code. On many machines direct-threading is faster than subroutine threading (see reference below).

As example, a stack machine might execute the sequence "push A, push B, add". That might be translated to the following thread and routines, where ip is initialized to the address &thread.

```
thread:      pushA: *sp++ = A          pushB: *sp++ = B          add:
*sp++ = *--sp + *--sp
&pushA          jump *ip++              jump *ip++              jump *ip++
&pushB
&add
...
```

Alternatively, operands may be included in the thread. This can remove some indirection needed above, but makes the thread larger:

```
thread:     push: *sp++ = *ip++      add: *sp++ = *--sp + *--sp
  &push          jump *ip++              jump *ip++
  &A
  &push
  &B
  &add
```

- **Indirect threading**

Indirect threading uses pointers to locations that in turn point to machine code. The indirect pointer may be followed by operands which are stored in the indirect "block" rather than storing them repeatedly in the thread. Thus, indirect code is often more compact than direct-threaded code, but the indirection also typically makes it slower, though still usually faster than bytecode interpreters. Where the handler operands include both values and types, the space savings over direct-threaded code may be significant. Older FORTH systems typically produce indirect-threaded code.

As example, if the goal is to execute "push A, push B, add", the following might be used. Here, ip is initialized to address &thread, each code fragment (push, add) is found by double-indirecting through ip; and operands to each code fragment are found in the first-level indirection following the address of the fragment.

```
thread:     i_pushA:   push:                  add:
  &i_pushA    &push      *sp++ = *(*ip + 1)      *sp++ = *--sp + *--sp
  &i_pushB    &A       jump *(*ip++)           jump *(*ip++)
  &i_add     i_pushB:
              &push
              &B
            i_add:
              &add
```

- **Subroutine threading**

So-called "subroutine-threaded code" (also "call-threaded code") consists of a series of machine-language "call" instructions (or addresses of functions to "call", as opposed to direct threading's use of "jump"). Early compilers for ALGOL, Fortran, Cobol and some Forth systems often produced subroutine-threaded code. The code in many of these systems operated on a last-in-first-out (LIFO) stack of operands, which had well-developed compiler theory. Most modern processors have special hardware support for subroutine "call" and "return" instructions, so the overhead of one extra machine instruction per dispatch is somewhat diminished. Anton Ertl has stated "that, in contrast to popular myths, subroutine threading is usually slower than direct threading."[3] However, Ertl's most recent tests[4] show that subroutine threading is faster than direct threading in 15 out of 25 test cases. Ertl's most recent tests show that direct threading is the fastest threading model on Xeon, Opteron, and Athlon processors; indirect threading is the fastest threading model on Pentium M processors; and subroutine threading is the fastest threading model on Pentium 4, Pentium III, and PPC processors.

As an example of call threading "push A, push B, add":

```
thread:         pushA:          pushB:       add:
  call pushA      *sp++ = A       *sp++ = B    *sp++ = *--sp + *--sp
  call pushB      ret             ret          ret
  call add
```

- **Token threading**

Token threaded code uses lists of 8 or 12-bit indexes to a table of pointers. Token threaded code is notably compact, without much special effort by a programmer. It is usually half to three-fourths the size of other threaded-codes, which are themselves a quarter to an eighth the size of compiled code. The table's pointers can either be indirect or direct. Some Forth compilers produce token threaded code. Some programmers consider the "p-code" generated by some Pascal compilers, as well as the byte codes used by .NET, Java, Basic and some C compilers to be token-threading.

A common approach historically is bytecode, which uses 8-bit opcodes and, often, a stack-based virtual machine. A typical interpreter is known as a "decode and dispatch interpreter", and follows the form

```
bytecode:       top:                pushA:       pushB:       add:
  0 /*pushA*/     i = decode(vpc++)     *sp++ = A     *sp++ = B
*sp++ = *--sp + *--sp
  1 /*pushB*/     addr = table[i]       jump top      jump top      jump
top
  2 /*add*/       jump *addr
```

If the virtual machine uses only byte-size instructions, decode() is simply a fetch from bytecode, but often there are commonly-used 1-byte instructions plus some less-common multibyte instructions, in which case decode() is more complex. The decoding of single byte opcodes can be very simply and efficiently handled by a branch table using the opcode directly as an index.

For instructions where the individual operations are simple, such as "push" and "add", the overhead involved in deciding what to execute is larger than the cost of actually executing it, such interpreters are often much slower than machine code. However for more complex ("compound") instructions, the overhead percentage is proportionally less significant.

- **Huffman threading**

Huffman threaded code consists of lists of Huffman codes. A Huffman code is a variable length bit string used to identify a unique item. A Huffman-threaded interpreter locates subroutines using an index table or tree of pointers that can be navigated by the Huffman code. Huffman threaded code is one of the most compact representations known for a computer program. Basically the index and codes are organized by measuring the frequency that each subroutine occurs in the code. Frequent calls are given the shortest codes. Operations with approximately equal frequencies are given codes with nearly equal bit-lengths. Most Huffman-threaded systems have been implemented as direct-threaded Forth systems, and used to pack large amounts of slow-running code into small, cheap microcontrollers. Most published uses have been in toys, calculators or watches.

- **Lesser used threading**

String threading, where operations are identified by strings, usually looked-up by a hash table. This was used in Charles H. Moore's earliest Forth implementations and in the University of Illinois's experimental hardware-interpreted computer language. It is also used in Bashforth.

2.5 HTTPSESSIONS:

public interface **HttpSession**

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time

- Bind objects to sessions, allowing user information to persist across multiple user connections

When an application stores an object in or removes an object from a session, the session checks whether the object implements HttpSessionBindingListener. If it does, the servlet notifies the object that it has been bound to or unbound from the session.

A servlet should be able to handle cases in which the client does not choose to join a session, such as when cookies are intentionally turned off. Until the client joins the session, isNew returns true. If the client chooses not to join the session, getSession will return a different session on each request, and isNew will always return true.

Session information is scoped only to the current web application (ServletContext), so information stored in one context will not be directly visible in another.

Methods used in detail:

getCreationTime

public long **getCreationTime**()

Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

Returns:

a long specifying when this session was created, expressed in milliseconds since 1/1/1970 GMT

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

getId

public java.lang.String **getId**()
Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.

Returns:

a string specifying the identifier assigned to this session

getLastAccessedTime

public long **getLastAccessedTime**()

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

Actions that your application takes, such as getting or setting a value associated with the session, do not affect the access time.

Returns:

a long representing the last time the client sent a request associated with this session, expressed in milliseconds since 1/1/1970 GMT

setMaxInactiveInterval

public void **setMaxInactiveInterval**(int interval)

Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout.

Parameters:

interval - An integer specifying the number of seconds

getMaxInactiveInterval

public int **getMaxInactiveInterval**()

Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses. After this interval, the servlet container will invalidate the session. The maximum time interval can be set with the setMaxInactiveInterval method. A negative time indicates the session should never timeout.

Returns:

an integer specifying the number of seconds this session remains open between client requests

getSessionContext

public HttpSessionContext **getSessionContext**()

Deprecated. *As of Version 2.1, this method is deprecated and has no replacement. It will be removed in a future version of the Java Servlet API.*

getAttribute

public java.lang.Object **getAttribute**(java.lang.String name)
Returns the object bound with the specified name in this session, or null if no object is bound under the name.

Parameters:

name - a string specifying the name of the object

Returns:

the object with the specified name

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

getValue

public java.lang.Object **getValue**(java.lang.String name)
Deprecated. *As of Version 2.2, this method is replaced by getAttribute(java.lang.String).*

Parameters:

name - a string specifying the name of the object

Returns:

the object with the specified name

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

getAttributeNames

public java.util.Enumeration **getAttributeNames**()
Returns an Enumeration of String objects containing the names of all the objects bound to this session.

Returns:

an Enumeration of String objects specifying the names of all the objects bound to this session

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

getValueNames

public java.lang.String[] **getValueNames**()
Deprecated. *As of Version 2.2, this method is replaced by getAttributeNames()*

Returns:

an array of String objects specifying the names of all the objects bound to this session

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

setAttribute

public void **setAttribute**(java.lang.String name,
java.lang.Object value)
Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.

After this method executes, and if the object implements Http Session Binding Listener, the container calls Http Session Binding Listener.value Bound.

Parameters:

name - the name to which the object is bound; cannot be null

value - the object to be bound; cannot be null

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

putValue

public void **putValue**(java.lang.String name,
                java.lang.Object value)
Deprecated. *As of Version 2.2, this method is replaced by* *setAttribute(java.lang.String, java.lang.Object)*

Parameters:

name - the name to which the object is bound; cannot be null

value - the object to be bound; cannot be null

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

removeAttribute

public void **removeAttribute**(java.lang.String name)
Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.

After this method executes, and if the object implements Http Session Binding Listener, the container calls Http Session Binding Listener. value Unbound.

Parameters:

name - the name of the object to remove from this session

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

removeValue

public void **removeValue**(java.lang.String name)
Deprecated. *As of Version 2.2, this method is replaced by* *setAttribute(java.lang.String, java.lang.Object)*

Parameters:

name - the name of the object to remove from this session

Throws:

java.lang.IllegalStateException - if this method is called on an invalidated session

invalidate

public void **invalidate**()
Invalidates this session and unbinds any objects bound to it.

Throws:

java.lang.IllegalStateException - if this method is called on an already invalidated session

isNew

public boolean **isNew**()
Returns true if the client does not yet know about the session or if the client chooses not to join the session. For example, if the server used only cookie-based sessions, and the client had disabled the use of cookies, then a session would be new on each request.

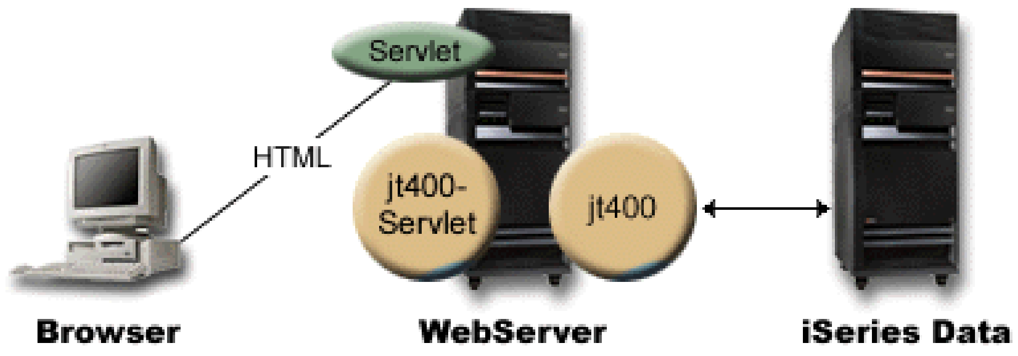
Returns:

true if the server has created a session, but the client has not yet joined

Throws:

java.lang.IllegalStateException - if this method is called on an already invalidated session

❖ **How servlets work?**



Exercise:

1. What is servlet? Give comparision of CGI and Servlet.
2. Explain the life cycle of servlet.
3. Explain the architecture of servlet.
4. Demostrate with an example the typical execution of servlet.
5. Explain the HTTPServletResponse and Request.
6. List and explain the methods of HTTP Session Interface.

❖❖❖❖

3

JAVA SERVER PAGES

Unit Structure

JSP: JSP Development Model, Components of JSP page, Request dispatching, Session and Thread Management Java Server Pages (JSP) are an afterbirth of Java Servlets. When Java Servlets were introduced it opened many avenues to a Java programmer. Java became a full fledged application server programming language. Though Java Servlets were great, it posed one great problem.

What is the need for JSP?

If you are a programmer or a web designer you will agree with me that not every programmer is a good designer and not every good designer is a good programmer. This is the exact problem posed by Java Servlets. Which means Java Servlets required the Java programmer to know the designing skills because the Java Servlets did not separate the Programming logic from the presentation layer.

Therefore there was a need to separate the design aspects from the Core Java programmers. This was the reason why, JSP was introduced.

How does JSP solve this problem?

Java Server Pages or JSP solved just this issue. It separated the designing issues from the programming logic. Simply put, if a company were to design a JSP based website, it would first design the layout using a professional web designer. This design can then be passed onto the JSP programmer who can then insert Java code (JSP code) inside these HTML pages. Once inserted, this pure HTML pages becomes a JSP page. It is as simple as that.

To give more re-usability and to further separate the programming logic Java Beans can be used. The 'usebean' property of a JSP page can just use these Java beans which is nothing but a Java class and then use the bean's methods from inside the JSP page making the JSP page very powerful. The Java bean on the other hand handle issues like connecting to the database, or making another HTTP connection etc.

Having understood the basics of a JSP page, it is then necessary to understand how to get started with JSP.

3.1 JSP DEVELOPMENT MODEL

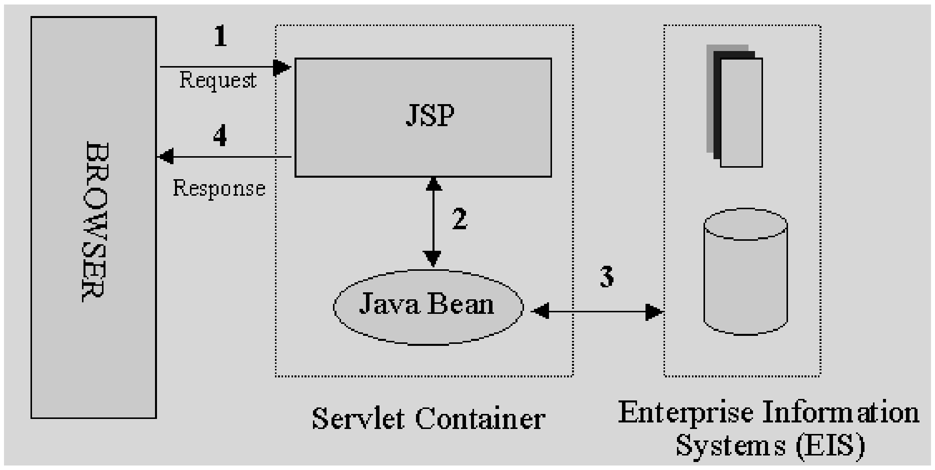
JSP provides a declarative, presentation-centric method of developing servlets. JSP specification itself is defined as a standard extension on top the Servlet API.

The early JSP specifications advocated two philosophical development models:

- Model 1 architecture
- Model 2 architecture

The 2 approaches differ essentially in the location at which the bulk of the request processing was performed.

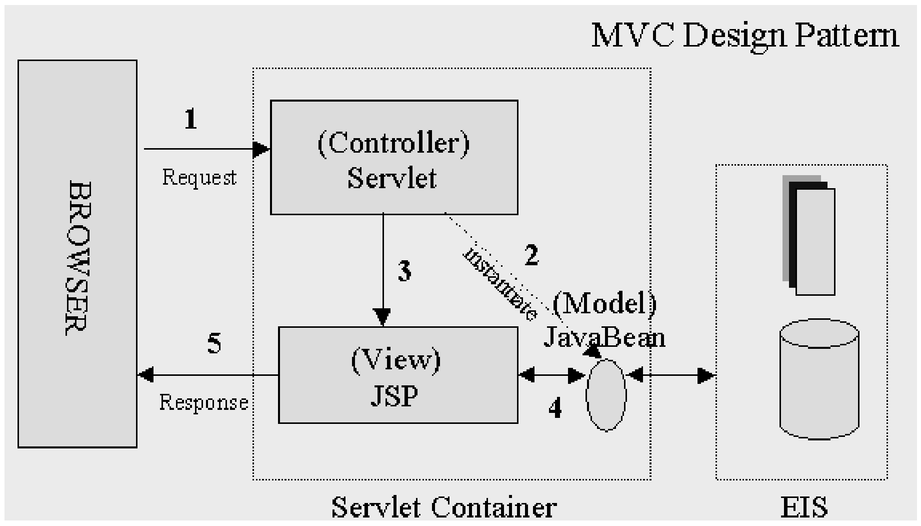
3.1.1 Model 1 architecture



- In the Model 1 architecture, the incoming request from a web browser is sent directly to the JSP page, which is responsible for processing it and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans.

- Model 1 architecture is suitable for simple applications. It may not be desirable for complex implementations. Indiscriminate usage of this architecture usually leads to a significant amount of scriptlets i.e. Java code embedded within the JSP page.
- Another downside of this architecture is that each of the JSP pages must be individually responsible for managing application state and verifying authentication and security.

3.1.2 Model 2 architecture



- The Model 2 architecture is a server-side implementation of the popular Model/View/Controller design pattern.
- Here, the processing is divided between:

  - **Presentation components**: They are JSP pages that generate the HTML/XML response that determines the user interface when rendered by the browser.
  - **Front components**: They are the controllers. They do not handle any presentation issues, but rather, process all the HTTP requests. Here, they are responsible for creating any beans or objects used by the presentation components, as well as deciding, depending on the user's actions, which presentation component to forward the request to. Front components can be implemented as either a servlet or JSP page.

— There is no processing logic within the presentation component itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the controller, and extracting the dynamic content within for insertion within its static templates.
— It cleanly separates the roles and responsibilities of the developers and page designers on the programming team.

— The front components present a single point of entry into the application, thus making the management of application state, security, and presentation uniform and easier to maintain.

3.2 COMPONENTS OF JSP PAGE :

A JSP page typically contains the following components:

1. Directives
2. Declarations
3. Expressions
4. Scriptlets
5. Comments

3.2.1 Directives

- JSP directives are messages for the JSP engine. They do not directly produce any visible output, but tell the engine what to do with the rest of the JSP page.
- JSP directives are always enclosed within the <%@ ... %> tag.
- The two primary directives are page and include. (Note that JSP 1.1 also provides the taglib directive, which can be used for working with custom tag libraries)

3.2.1.1 Page Directive

- Typically, the page directive is found at the top of almost all of your JSP pages.
- There can be any number of page directives within a JSP page, although the attribute/value pair must be unique. Unrecognized attributes or values result in a translation error.
- For example,

```
<%@ page import="java.util.*, com.foo.*" buffer="16k" %>
```

3.2.1.2 Include Directive

- The include directive lets you separate your content into more manageable elements, such as those for including a common page header or footer.
- The page included can be a static HTML page or more JSP content.
- For example, the directive:

```
<%@ include file="copyright.html" %>
```

- It can be used to include the contents of the indicated file at any location within the JSP page.

3.2.2 Declarations

- JSP declarations let you define page-level variables to save information or define supporting methods that the rest of a JSP page may need.
- Note that too much of declarations would turn out to be a maintenance nightmare. For that reason, and to improve reusability, it is best that logic-intensive processing is encapsulated as JavaBean components.
- Declarations are found within the <%! ... %> tag.
- Always end variable declarations with a semicolon, as any content must be valid Java statement.

```
<%! int i=0; %>
```

- You can also declare methods. For example, you can override the initialization event in the JSP life cycle by declaring:

```
<%! public void jspInit()
          {
          //some initialization code
            }
%>
```

3.2.3 Expressions

- The results of evaluating the expression are converted to a string and directly included within the output page.
- Typically expressions are used to display simple values of variables or return values by invoking a bean's getter methods.
- JSP expressions begin within <%= ... %> tags and do not include semicolons:

```
<%= fooVariable %>
<%= fooBean.getName() %>
```

3.2.4 Scriptlets

- Scriptlets are embedded within <% ... %> tags. This code is run when the request is serviced by the JSP page. You can have just about any valid Java code within a scriptlet, and is not limited to one line of source code.

- Following example combines both expressions and scriptlets:

```
<% for (int i=1; i<=4; i++) { %>
   <H<%=i%>>Hello</H<%=i%>>
<% } %>
```

3.2.5 Comments

- You can include HTML comments in JSP pages. But users can view these if they view the page's source. If you don't want users to see your comments, embed them within the <%-- ... --%> tag:

```
<%-- comment for server side only --%>
```

- A most useful feature of JSP comments is that they can be used to selectively block out scriptlets or tags from compilation. Thus, they can play a significant role during the debugging and testing process.

Note that there are some objects are implicitly available within a JSP page. They can be used within scriptlets and expressions, without the page author first having to create them. These objects act as wrappers around underlying Java classes or interfaces typically defined within the Servlet API. The nine implicit objects:

- request: represents the HttpServletRequest triggering the service invocation. Request scope.
- response: represents HttpServletResponse to the request. Not used often by page authors. Page scope.
- pageContext: encapsulates implementation-dependent features in PageContext. Page scope.
- application: represents the ServletContext obtained from servlet configuration object. Application scope.
- out: a JspWriter object that writes into the output stream. Page scope.
- config: represents the ServletConfig for the JSP. Page scope.
- page: synonym for the "this" operator, as an HttpJspPage. Not used often by page authors. Page scope.
- session: An HttpSession. Session scope. More on sessions shortly.
- exception: the uncaught Throwable object that resulted in the error page being invoked. Page scope.

Note that these implicit objects are only visible within the system generated _jspService() method. They are not visible within methods you define yourself in declarations.

3.3 REQUEST DISPATCHING

A RequestDispatcher object can forward a client's request to a resource or include the resource itself in the response back to the client. A resource can be another servlet, or an HTML file, or a JSP file, etc.

RequestDispatcher acts as an object as a wrapper for the resource located at a given path that is supplied as an argument to the getRequestDispatcher method.

For constructing a RequestDispatcher object, you can use either the ServletRequest.getRequestDispatcher() method or the ServletContext.getRequestDispatcher() method. They both do the same thing, but impose slightly different constraints on the argument path. For the former, it looks for the resource in the same webapp to which the invoking servlet belongs and the pathname specified can be relative to invoking servlet. For the latter, the pathname must begin with '/' and is interpreted relative to the root of the webapp.

To illustrate, suppose you want Servlet_A to invoke Servlet_B. If they are both in the same directory, you could accomplish this by incorporating the following code fragment in either the service method or the doGet method of Servlet_A:

```
RequestDispatcher dispatcher = getRequest Dispatcher
("Servlet_B");
dispatcher.forward ( request, response );
```

where request, of type HttpServletRequest, is the first parameter of the enclosing service method (or the doGet method) and response, of type HttpServletResponse, the second. You could accomplish the same by

```
Request Dispatcher dispatcher = get Servlet Context().get Request
Dispatcher("/servlet/Servlet_B");
dispatcher.forward( request, response );
```

The request dispatching functionality allows a servlet to delegate request handling to other components on the server. A servlet can either forward an entire request to another servlet or include bits of content from other components in its own output. In either case, this is done with a RequestDispatcher object that is obtained from the ServletContext with its new getRequestDispatcher() method. When you call this method, you specify the path to the servlet to which you are dispatching the request.

When you dispatch a request, you can set request attributes using the setAttribute() method of ServletRequest and read them using the getAttribute() method. A list of available attributes is returned by getAttributeNames().

3.3.1 RequestDispatcher provides two methods for dispatching requests:

- **forward**

void **forward** (Servlet Request request, Servlet Response response) throws ServletException,IOException

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. This method allows one servlet to do preliminary processing of a request and another resource to generate the response.

For a RequestDispatcher obtained via getRequestDispatcher(), the ServletRequest object has its path elements and parameters adjusted to match the path of the target resource.

forward should be called before the response has been committed to the client. If the response already has been committed, this method throws an IllegalStateException. Uncommitted output in the response buffer is automatically cleared before the forward.

Parameters:

request - a ServletRequest object that represents the request the client makes of the servlet

response - a ServletResponse object that represents the response the servlet returns to the client

Throws:

ServletException - if the target resource throws this exception

IOException - if the target resource throws this exception

IllegalStateException - if the response was already committed

- **include**

void **include**(ServletRequest request,ServletResponse response)
throws ServletException,IOException

It includes the content of a resource (servlet, JSP page, HTML file) in the response. In essence, this method enables programmatic server-side includes.

The ServletResponse object has its path elements and parameters remain unchanged from the caller's. The included servlet cannot change the response status code or set headers; any attempt to make a change is ignored.

Parameters:

request - a ServletRequest object that contains the client's request

response - a ServletResponse object that contains the servlet's response

Throws:

ServletException - if the included resource throws this exception

IOException - if the included resource throws this exception

3.3.2 getRequestDispatcher() method:

RequestDispatcher **getRequestDispatcher**(String path)
returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname specified may be relative, although it cannot extend outside the current servlet context. If the path begins with a "/" it is interpreted as relative to the current context root. This method returns null if the servlet container cannot return a RequestDispatcher.

The difference between this method and Servlet Context.get Request Dispatcher(java.lang.String) is that this method can take a relative path.

3.4 SESSION AND THREAD MANAGEMENT

3.4.1 Session Management

3.4.1.1 Using session object:

- The HttpSession API provides a simple mechanism for storing information about individual users on the application server. The API provides access to a session object that can be used to store other objects. The ability to tie objects to a particular user is important when working in an object-oriented environment.
- It allows you to quickly and efficiently save and retrieve JavaBeans that you may be using to identify your site's visitors, to hold product information for display on your online store, or to track products that potential customers have placed in their shopping carts.
- A session object is created on the application server, usually in a Java servlet or a JavaServerPage. The object is stored on the application server and a unique identifier called a session ID is assigned to it.
- The session object and session ID are handled by a session manager on the application server. Each session ID assigned by the application server has zero or more key/value pairs tied to it. The values are objects that you place in the session. Assign each of those objects a name, and each name must have an object with it because a null is not allowed.

3.4.1.2 Using cookie:

- A cookie is used to store the session ID on the Web site visitor's computer. This is automatically handled by the application server. Simply create the session object and begin using it.
- The application server will, by default, create the session ID and store it in a cookie. The browser will send the cookie back to the server every time a page is requested. The application server, via the server's session manager, will match the session ID from the cookie to a session object.
- The session object is then placed in the HttpServletRequest object and you retrieve it with the getSession() method.

3.4.1.3 Using URL rewritting:

- The procedure for URL rewriting is quite simple and requires only the use of two methods found in the HttpServletResponse interface.

- These two methods, encodeURL() and encodeRedirectURL(), are used to append the session ID to the URL. This allows the server to track users as they move through your Web pages, but it requires that every URL be rewritten.
- The string returned by the methods will have the session ID appended to it only if the server determines that it's required. If the user's browser supports cookies, the returned URL will not be altered.
- The following line of HTML code from a JSP creates a link to another JSP:

```
<A HREF="/products/product.jsp">Product Listing</A>
```

- Clicking on this link would send the user to the product.jsp page. Using URL rewriting, the same code would be written as follows:

```
<A HREF="

<%= response.encodeURL("/product/product.jsp")%>

">Product listing</A>
```

- The returned string from the encodeURL() method would contain the session ID. On a Tomcat 3.2 application server, the result of this line of code would be:

```
<A HREF="http://www.yourservername.com/products/
product.jsp;$sessionid$xxxx">Product Listing</A>
```

The xxxx would actually be a unique session ID generated by the server.

You should now have a good understanding of how the session ID is tracked and matched to a session object on the server.

The first step in using the session object is creating it. The method getSession() is used to create a new session object and to retrieve an already existing one. The getSession() method is passed a Boolean flag of true or false.

A false parameter indicates that you want to retrieve a session object that already exists. A true parameter lets the session manager know that a session object needs to be created if one does not already exist.

Following are some of the methods defined in the Java Servlet specification that can be used for session management:

- ***setAttribute(String name, Object value):*** Binds an object to this session using the name specified. Returns nothing (void).
- ***getAttribute(String name):*** Returns the object bound with the specified name in this session, or null if no object is bound under this name.
- ***removeAttribute(String name):*** Removes the object bound with the specified name from this session. Returns nothing (void).

 invalidate(): Invalidates this session and unbinds any objects bound to it. Returns nothing (void).

 isNew(): Returns a Boolean with a value of true if the client does not yet know about the session or if the client chooses not to join the session.

EXAMPLE : you can save shopping cart as a session attribute. This allows the shopping cart to be saved between requests and also allows cooperating servlets to access the cart. Some servlet adds items to the cart; another servlet displays, deletes items from, and clears the cart; and next servlet retrieves the total cost of the items in the cart.

```
public class CashierServlet extends HttpServlet
{
          public void doGet (HttpServletRequest req,
HttpServletResponse res)throws ServletException, IOException
          {

                    // Get the user's session and shopping cart
                    HttpSession session = request.getSession();
                    ShoppingCart cart = (ShoppingCart) session.get
          Attribute("cart");
                    ...
                    // Determine the total price of the user's books
                    double total = cart.getTotal();
                    ...
          }
}

package javax.servlet.http;

public interface HttpSession
{

          public java.lang.Object getAttribute(java.lang.String name);
          public java.util.Enumeration getAttributeNames();
          public void removeAttribute(java.lang.String name);
```

```
        public void setAttribute(java.lang.String name,
java.lang.Object value);

}
```

3.4.2 Thread Management

There are two major issues with Java Threads:

1. Concurrency
2. Control

- Failure to address both these issues means the endeavor will fail sooner or later.
- Java threads are most difficult to control. What if a thread gets stuck in a blocking method? What if something is wrong and the thread doesn't get CPU time? What if there is a bug? There are lots of 'what if" situations.
- Threads are not your traditional pool threads. Every Queue Thread has its own <u>management structure</u>. Each event in the life of a Queue Thread <u>is timed</u>.
- Thread "interrupt()" is a disaster. The original developers probably had a vision that programmers would want to interrupt an executing thread. But they never perfected that vision. What we have now are threads interrupting themselves as well as other threads sometimes with erroneous results.

  Let's say you create thread "A" and you expect that thread to complete some work within a time limit.

  - You execute a timed wait for thread "A".
  - Thread "A" does not complete within the time limit,
    - the time expires and
    - you regain control.
    - Your code continues with other work.
  - Then you have a second timed wait for another thread "B".
  - If thread "A" then issues interrupt(), it interrupts the caller at the second wait.
  - Both NotifyAll() and SignalAll() are shot gun methods. Having multiple threads waiting on a single object is a course grained solution. When the group awakens every thread must do some work to find out if it is needed.
  - Even if each thread is running on a separate CPU it still requires operating system CPU cycles to get the threads

running and put the unnecessary threads back into a blocking state.

- The purpose of the wait(), notify() and notifyAll() methods is to temporarily pause and resume the execution of code in an object.
- Typically the host object is not in a state where it can proceed with a method call it has been given and the thread of execution must literally wait for the object to return to a ready state. A common example would be a limited pool or store of objects where you must wait for a storage slot to be released or an object to be returned to the pool before you can use it.

```
public synchronized Object getNextObject() {

// Waiting loop
while (! objectAvailable())
{

try {

          wait();
}
catch (InterruptedException e)
          {

// Handle exception
}
}

// No longer waiting, get the return object
Object returnObject;

// Assign the returnObject from store

// Notify state change for other waiters

          notify();

          return returnObject;
}
```

- The act of waiting is associated with the Object class because any subclass may need to wait for a ready state to occur. The waiting process acts on a single thread of execution, but the wait mechanism expects that multiple threads may be waiting for the same object. The wait and notify methods are hosted by the Object class so that the Java Virtual

Exercise:

1. What is JSP?
2. Why JSP is required? Also explain when it is required.
3. Explain the architectural model of JSP.
4. Explain the different components of JSP.
5. Explain the expression used in JSP programs.
6. What is scriplet?
7. What is difference between HTML tag and scriplet tag?
8. Write a note on Thread Management.

❖❖❖❖

4

INTRODUCTION TO WEB SERVICES

Unit Structure

Introduction to Web Services: What is a Web Service? Software as a Service, Web Service Architectures, SOA, XML

4.1 WHAT IS A WEB SERVICE?

4.4.1.Definition:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

4.4.2 Agents and Services

A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular Web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language) with the same functionality. Although the agent may have changed, the Web service remains the same.

4.4.3 Requesters and Providers

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization, such as a business or an individual. The *provider entity* is the person or organization that provides an appropriate agent to implement a particular service. A *requester entity* is a person or organization that wishes to make use of a provider entity's Web service. It will use a *requester agent* to exchange messages with the provider entity's *provider agent*.

(In most cases, the requester agent is the one to initiate this message exchange, though not always. Nonetheless, for consistency we still use the term "requester agent" for the agent that interacts with the provider agent, even in cases when the provider agent actually initiates the exchange.)

Note:

A word on terminology: Many documents use the term service provider to refer to the provider entity and/or provider agent. Similarly, they may use the term service requester to refer to the requester entity and/or requester agent. However, since these terms are ambiguous -- sometimes referring to the agent and sometimes to the person or organization that owns the agent -- this document prefers the terms *requester entity*, *provider entity*, *requester agent* and *provider agent*.

In order for this message exchange to be successful, the requester entity and the provider entity must first agree on both the semantics and the mechanics of the message exchange.

4.4.4. Service Description

The mechanics of the message exchange are documented in a Web service description (WSD). The WSD is a machine-processable specification of the Web service's interface, written in WSDL. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. In essence, the service description represents an agreement governing the mechanics of interacting with that service.

4.4.5. Semantics

The semantics of a Web service is the shared expectation about the behavior of the service, in particular in response to messages that are sent to it. In effect, this is the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated. It may be explicit or implicit, oral or written, machine processable or human oriented, and it may be a legal agreement or an informal (non-legal) agreement.

While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction. The dividing line between these two is not necessarily rigid. As more semantically rich languages are used to describe the mechanics of the interaction, more of the essential information may migrate from the informal semantics to the service description. As this migration occurs, more of the work required to achieve successful interaction can be automated.

4.4.6. Overview of Engaging a Web Service

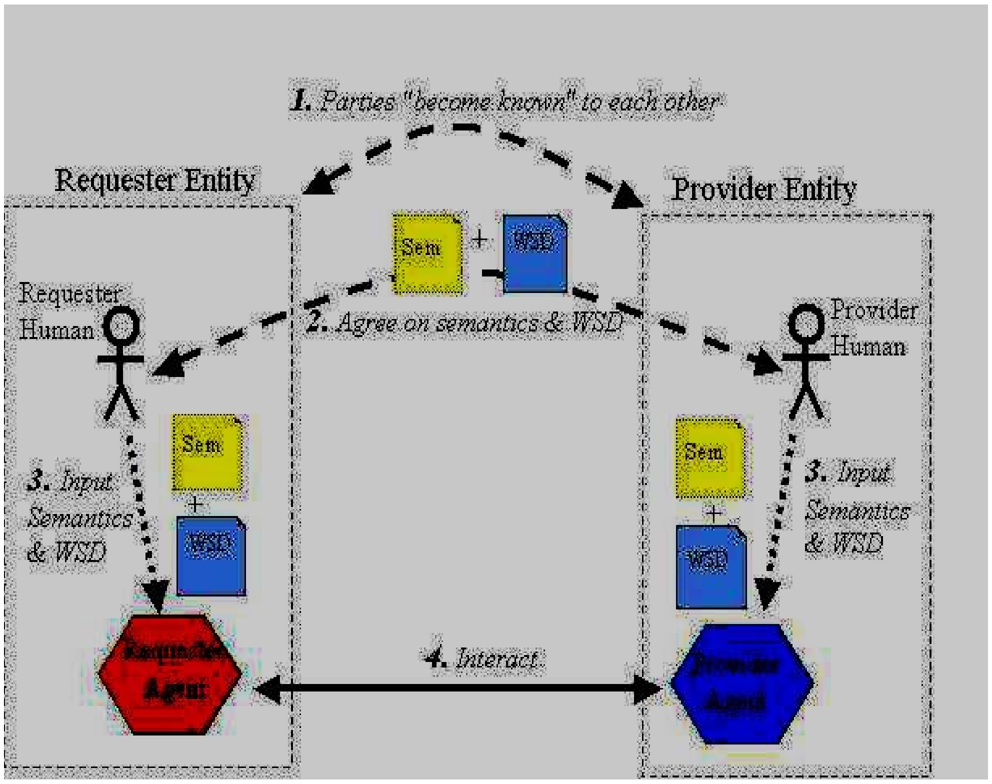
There are many ways that a requester entity might engage and use a Web service. In general, the following broad steps are required:

(1) The requester and provider entities become known to each other (or at least one becomes know to the other);

(2) The requester and provider entities somehow agree on the service description and semantics that will govern the interaction between the requester and provider agents;

(3) The service description and semantics are realized by the requester and provider agents;

(4) The requester and provider agents exchange messages, thus performing some task on behalf of the requester and provider entities. (I.e., the exchange of messages with the provider agent represents the concrete manifestation of interacting with the provider entity’s Web service.)



The General Process of Engaging a Web Service

4.2 SOFTWARE AS A SERVICE

4.2.1. Definition:

Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet.

SaaS is becoming an increasingly prevalent delivery model as underlying technologies that support Web services and service-oriented architecture (SOA) mature and new developmental approaches, such as Ajax, become popular. Meanwhile, broadband service has become increasingly available to support user access from more areas around the world.

SaaS is closely related to the ASP (application service provider) and On Demand Computing software delivery models. IDC identifies two slightly different delivery models for SaaS. The hosted application management (hosted AM) model is similar to ASP: a provider hosts commercially available software for customers and delivers it over the Web. In the software on demand model, the provider gives customers network-based access to a single copy of an application created specifically for SaaS distribution.

4.2.2. Key characteristics

SaaS characteristics include:

- Network-based access to, and management of, commercially available software
- Activities managed from central locations rather than at each customer's site, enabling customers to access applications remotely via the Web
- Application delivery typically closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model, including architecture, pricing, partnering, and management characteristics
- Centralized feature updating, which obviates the need for end-users to download patches and upgrades.
- Frequent integration into a larger network of communicating software—either as part of a mashup or a plugin to a platform as a service

(Service oriented architecture is naturally more complex than traditional models of software deployment.)

SaaS providers generally price applications on a per-user basis and/or per business basis, sometimes with a relatively small minimum number of users and often with additional fees for extra bandwidth and storage. SaaS revenue streams to the vendor are therefore lower initially than traditional software license fees, but are also recurring, and therefore viewed as more predictable, much like maintenance fees for licensed software.

In addition to characteristics mentioned above, SaaS sometimes provides:

- More feature requests from users, since there is frequently no marginal cost for requesting new features
- Faster new feature releases, since the entire community of users benefits

- Embodiment of recognized best practices, since the user community drives the software publisher to support best practice

4.2.3. Benefits

Benefits of the SaaS model include:

- easier administration
- automatic updates and patch management
- compatibility: All users will have the same version of software.
- easier collaboration, for the same reason
- global accessibility.

The traditional model of software distribution, in which software is purchased for and installed on personal computers, is sometimes referred to as *software as a product*.

4.3. WEB SERVICE ARCHITECTURES

4.3.1. Purpose of the Web Service Architecture

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. This document (WSA) is intended to provide a common definition of a Web service, and define its place within a larger Web services framework to guide the community. The WSA provides a conceptual model and a context for understanding Web services and the relationships between the components of this model. The architecture does not attempt to specify how Web services are implemented, and imposes no restriction on how Web services might be combined. The WSA describes both the minimal characteristics that are common to all Web services, and a number of characteristics that are needed by many, but not all, Web services. The Web services architecture is interoperability architecture: it identifies those global elements of the global Web services network that are required in order to ensure interoperability between Web services.

4.3.2. There are two ways to view the web service architecture.

- The first is to examine the individual roles of each web service actor.
- The second is to examine the emerging web service protocol stack.

1. Web Service Roles

There are three major roles within the web service architecture:

- **Service provider:**

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

- **Service requestor**

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

- **Service registry**

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearinghouse for companies and their services.

2. Web Service Protocol Stack

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

- **Service transport**

This layer is responsible for transporting messages between applications. Currently, this layer includes hypertext transfer protocol (HTTP), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP), and newer protocols, such as Blocks Extensible Exchange Protocol (BEEP).

- **XML messaging**

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

- **Service description**

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

- **Service discovery**

This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

As web services evolve, additional layers may be added, and additional technologies may be added to each layer.

4.3.3. Few Words about Service Transport

The bottom of the web service protocol stack is service transport. This layer is responsible for actually transporting XML messages between two computers.

- **Hyper Text Transfer Protocol (HTTP)**

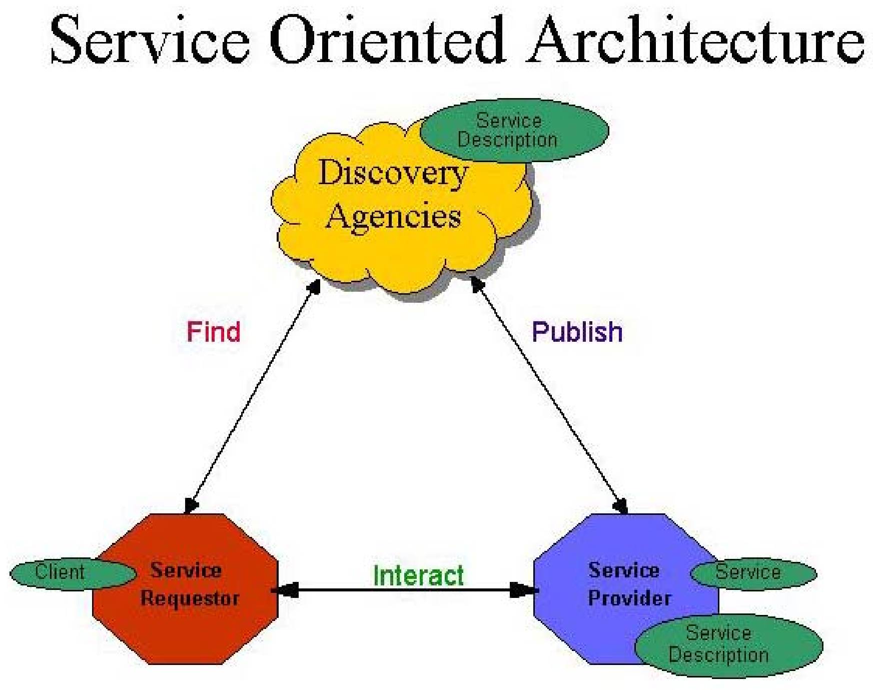
Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to easily integrate remote applications, but it does raise a number of security concerns.

- **Blocks Extensible Exchange Protocol (BEPP)**

One promising alternative to HTTP is the Blocks Extensible Exchange Protocol (BEEP).BEEP is a new IETF framework of best practices for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management

SOAP is not tied to any specific transport protocol. In fact, you can use SOAP via HTTP, SMTP, or FTP. One promising idea is therefore to use SOAP over BEEP.

4.4. SOA (Service Oriented Architecture)



The figure above illustrates the relationships between requesters, providers, services, descriptions, and discovery services in the case where agents take on both requester and provider roles. For example, XML messages compliant with the SOAP specification are exchanged between the requester and provider. The provider publishes a WSDL file that contains a description of the message and endpoint information to allow the requester to generate the SOAP message and send it to the correct destination.

To support the common MEP of request/response, for example, a Web services implementation provides software agents that function as both requesters and providers, as shown in Figure 2. The service requester sends a message in the form of a request for information, or to perform an operation, and receives a message from the service provder that contains the result of the request or operation. The service provider receives the request, processed the message and sends a response. The technologies typically used for this type of Web services interaction include SOAP, WSDL, and HTTP.

Note:

The Web services architecture does not include the concept of automatically correlating requests and responses, as some RPC oriented technologies do. The correletion of request and response messages is typically application-defined.

The following sections provide more formal definitions of the components, roles, and operations in Web services architecture.

4.4.1. Components

- **The Service:** Whereas a web service is an interface described by a service description, its implementation is the service. A service is a software module deployed on network accessible platforms provided by the service provider. It exists to be invoked by or to interact with a service requestor. It may also function as a requestor, using other web services in its implementation.
- **The Service Description:** The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information, and network location. It could also include categorization and other meta data to facilitate discovery and utilization by requestors. The complete description may be realized as a set of XML description documents. The service description may be published to a requestor directly or to a discovery agency.

4.4.2. Roles

- **Service Provider:** From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service. It has also been referred to as a service execution environment or a service container. Its role in the client-server message exchange patterns is that of a server.
- **Service Requestor:** From a business perspective, this is the business that requires certain function to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The requestor role can be played by a browser driven by a person or a program without a user interface, e.g. another web service. Its role in the client-server message exchange patters is that of a client.

- **Discovery Agency:** This is a searchable set of service descriptions where service providers publish their service descriptions. The service discovery agency can be centralized or distributed. A discovery agency can support both the pattern where it has descriptions sent to it and where the agency actively inspects public providers for descriptions. Service requestors may find services and obtain binding information (in the service descriptions) during development for static binding, or during execution for dynamic binding. For statically bound service requestors, the service discovery agent is in fact an optional role in the architecture, as a service provider can send the description directly to service requestors. Likewise, service requestors can obtain a service description from other sources besides a service registry, such as a local filesystem, FTP site, URL, or WSIL document.

4.4.3. Operations

In order for an application to take advantage of Web services, three behaviors must take place: publication of service descriptions, finding and retrieval of service descriptions, and binding or invoking of services based on the service description. These behaviors can occur singly or iteratively, with any cardinality between the roles. In detail these operations are:

- **Publish:** In order to be accessible, a service needs to publish its description such that the requestor can subsequently find it. Where it is published can vary depending upon the requirements of the application (see Service Publication Stck discussion for more details)

- **Find:** In the find operation, the service requestor retrieves a service description directly or queries the registry for the type of service required (see Service Discovery for more details). The find operation may be involved in two different lifecycle phases for the service requestor: at design time in order to retrieve the service's interface description for program development, and at runtime in order to retrieve the service's binding and location description for invocation.

- **Interact:** Eventually, a service needs to be invoked. In the interact operation the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service. Examples of the interaction include: single message one way, broadcast from requester to many services, a multi message conversation, or a business process. Any of these types of interactions can be synchronous or asynchronous.

4.5 XML

4.5.1. What is XML?

- XML (Extensible Markup Language) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification[4] produced by the W3C, and several other related specifications, all gratis open standards.[5]
- XML's design goals emphasize simplicity, generality, and usability over the Internet.[6] It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.
- There are many programming interfaces that software developers may use to access XML data, and several schema systems designed to aid in the definition of XML-based languages.
- As of 2009, hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for most office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org (OpenDocument), and Apple's iWork.

Exercise:

1. What is Web service?
2. Explain the concept of SOA.
3. State and explain the characteristics of Software as service.
4. Explain the role of service providers in SOA model.
5. Explain the XML technology in detail.
6. Explain the architectural view of web services.
7. Demostrate an example for implementing the concept of software as service.

❖❖❖❖

5

INTRODUCTION TO .NET FRAMEWORK

Unit Sturcture

Introduction to .NET Framework : Evolution of .NET, Comparison of Java and .NET, Architecture of .NET Framework, Common Language Runtime, Common Type System, MetaData, Assemblies, Application Domains, CFL, Features of .NET, Advantages and Applications.

The .NET History

Sometime in the July 2000, Microsoft announced a whole new software development framework for Windows called .NET in the Professional Developer Conference (PDC). Microsoft also released PDC version of the software for the developers to test. After initial testing and feedback Beta 1 of .NET was announced. Beta 1 of the .NET itself got lot of attention from the developer community. When Microsoft announced Beta 2, it incorporated many changes suggested by the community and internals into the software. The overall 'Beta' phase lasted for more than 1 ½ years. Finally, in March 2002 Microsoft released final version of the .NET framework.

One thing to be noted here is the change in approach of Microsoft while releasing this new platform. Unlike other software where generally only a handful people are involved in beta testing, .NET was thrown open to community for testing in it's every pre-release version. This is one of the reasons why it created so many waves of excitement within the community and industry as well.

Microsoft has put in great efforts in this new platform. In fact Microsoft says that its future depends on success of .NET. The development of .NET is such an important event that Microsoft considers it equivalent to transition from DOS to Windows. All the future development – including new and version upgrades of existing products – will revolve around .NET.

Flavors of .NET

Contrary to general belief .NET is not a single technology. Rather it is a set of technologies that work together seamlessly to solve your business problems. The following sections will give you insight into various flavors and tools of .NET and what kind of applications you can develop.

What type of applications can I develop?

When you hear the name .NET, it gives a feeling that it is something to do only with internet or networked applications. Even though it is true that .NET provides solid foundation for developing such applications it is possible to create many other types of applications. Following list will give you an idea about various types of application that we can develop on .NET.

1. ASP.NET Web applications: These include dynamic and data driven browser based applications.
2. Windows Form based applications: These refer to traditional rich client applications.
3. Console applications: These refer to traditional DOS kind of applications like batch scripts.
4. Component Libraries: This refers to components that typically encapsulate some business logic.
5. Windows Custom Controls: As with traditional ActiveX controls, you can develop your own windows controls.
6. Web Custom Controls: The concept of custom controls can be extended to web applications allowing code reuse and modularization.
7. Web services: They are "web callable" functionality available via industry standards like HTTP, XML and SOAP.
8. Windows Services: They refer to applications that run as services in the background. They can be configured to start automatically when the system boots up.

As you can clearly see, .NET is not just for creating web application but for almost all kinds of applications that you find under Windows.

.NET Framework SDK

You can develop such varied types of applications. That's fine. But how? As with most of the programming languages, .NET

has a complete Software Development Kit (SDK) – more commonly referred to as **.NET Framework SDK –** that provides classes, interfaces and language compilers necessary to program for .NET. Additionally it contains excellent documentation and Quick Start tutorials that help you learn .NET technologies with ease. Good news is that - .NET Framework SDK is available FREE of cost. You can download it from the MSDN web site. This means that if you have machine with .NET Framework installed and a text editor such as Notepad then you can start developing for .NET right now!
You can download entire .NET Framework SDK (approx 131 Mb) from MSDN web site at
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url
=/msdn-files/027/000/976/msdncompositedoc.xml

Development Tools

If you are developing applications that require speedy delivery to your customers and features like integration with some version control software then simple Notepad may not serve your purpose. In such cases you require some Integrated Development Environment (IDE) that allows for Rapid Action Development (RAD). The new Visual Studio.NET is such an IDE. VS.NET is a powerful and flexible IDE that makes developing .NET applications a breeze. Some of the features of VS.NET that make you more productive are:

- Drag and Drop design
- IntelliSense features
- Syntax highlighting and auto-syntax checking
- Excellent debugging tools
- Integration with version control software such as Visual Source Safe (VSS)
- Easy project management

Note that when you install Visual Studio.NET, .NET Framework is automatically installed on the machine.

Visual Studio.NET Editions

Visual Studio.NET comes in different editions. You can select edition appropriate for the kind of development you are doing. Following editions of VS.NET are available:
- Professional
- Enterprise Developer
- Enterprise Architect

Visual Studio .NET Professional edition offers a development tool for creating various types of applications mentioned previously. Developers can use Professional edition to build Internet and Develop applications quickly and create solutions that span any device and integrate with any platform.

Visual Studio .NET Enterprise Developer (VSED) edition contains all the features of Professional edition plus has additional capabilities for enterprise development. The features include things such as a collaborative team development, Third party tool integration for building XML Web services and built-in project templates with architectural guidelines and spanning comprehensive project life-cycle.

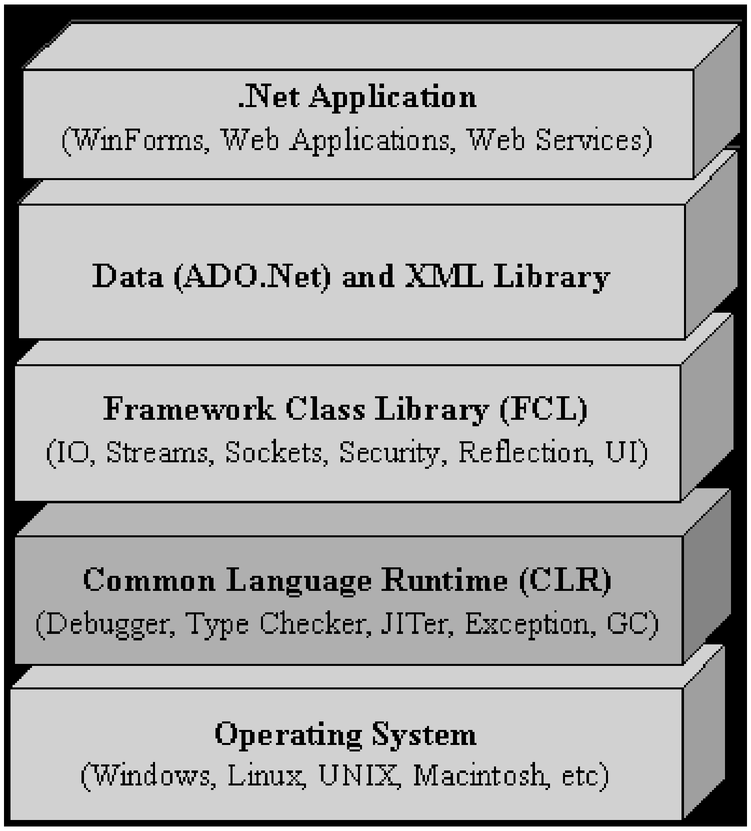
Visual Studio .NET Enterprise Architect (VSEA) edition contains all the features of Visual Studio .NET Enterprise Developer edition and additionally includes capabilities for designing, specifying, and communicating application architecture and functionality. The additional features include Visual designer for XML Web services, Unified Modeling Language (UML) support and enterprise templates for development guidelines and policies.

Special language specific editions are available. They are:
- Visual Basic.NET Standard Edition
- Visual C# Standard Edition
- Visual C++ .NET Standard (soon to be released)

5. Introduction to .NET Framework:

The Microsoft .NET Framework is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework supports multiple programming languages in a manner that allows language interoperability, whereby each language can utilize code written in other languages; in particular, the .NET library is available to all the programming languages that .NET encompasses. The .NET Framework is a Microsoft offering and is intended to be used by most new applications created for the Windows platform. In order to be able to develop and not just run applications for the Microsoft .NET Framework 4.0, it is required to have Visual Studio 2010 installed on your computer.

The framework's Base Class Library provides a large range of features including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.


.Net Application
(WinForms, Web Applications, Web Services)
Data (ADO.Net) and XML Library
Framework Class Library (FCL)
(IO, Streams, Sockets, Security, Reflection, UI)
Common Language Runtime (CLR)
(Debugger, Type Checker, JITer, Exception, GC)
Operating System
(Windows, Linux, UNIX, Macintosh, etc)

Microsoft .NET Framework

5.1 EVOLUTION OF .NET

The Microsoft .NET initiative is all-encompassing, ever-present, and in certain ways, brand-new—but the underlying technologies have been with us for some time. In this article, we'll explore the evolutionary process that made .NET possible, from MS-DOS and the iterations of Windows to ActiveX. It's all come together to culminate in .NET

C:\DOS\Run (MS-DOS)

DOS carved its spot in computing history as the innovation that let regular folks use computers. Prior to that, no *one* user ever operated a computer. It was always operated by a *team* of users.

When DOS came out in 1981, a couple of companies (most notably, Novell) built software to let teams work on bundles of computers—the first PC-based networks. Other companies built products like Telix, PCBoard and Wildcat, enabling the building of the first distributed public networks. The DOS world was great, but everything changed when Microsoft invented Windows.

C:\Windows\Run (Win3.1)

Next came Windows, a GUI even your grandmother could use. The old standby, MS-DOS, was still with us working in the background behind Windows. The first widely accepted iteration of this revolutionary product was Windows 3.1; networking was introduced with 3.11. Users could share files and folders graphically and even send e-mail without having to use the cryptic command-line tool.

With Windows making the PC easier to use, suddenly it became simple to access huge, legacy databases. All this easy access began to tax the resources of mainframes. More efficient use of valuable network resources was needed.

C:\Windows\Crash (Win32)

Microsoft learned early that regular and recurring releases raise revenue. No product can be everything to every user, and bugs needed to be fixed, so constant upgrades and new product releases were necessary. With each release, new features (and bugs) were introduced.

The first release of Win32 was Windows 95. This was a big change, moving from a 16-bit system in Windows 3.x and MS-DOS to a 32-bit operating system in Windows 95. This new version contained robust networking features and tools out of the box. This included standard TCP/IP support and wizards to automate network access/setup.

About the same time, and between Microsoft's planned releases, the World Wide Web blew onto the scene. To effectively support the networking features required to connect users to their ISPs, Microsoft released a couple of service releases and, finally, Windows 98.

Activate the Internet (ActiveX)

When Bill Gates takes one of his famous reading vacations, the result usually affects the course of information technology for years to come. Gates had made a fortune producing operating systems and software for the PC; now he realized another fortune could be made in producing software for the Internet. Upon returning from a mid-1990s reading vacation, he handed down a decree to Microsoft employees: Activate the Internet. Thus, ActiveX technologies were born.

ActiveX is a reworking of Microsoft's component format, the OLE/COM technology found in all Microsoft products. ActiveX, which emerged as the cornerstone of Microsoft's plans (at the time), encompassed all of the current Internet technologies in "objects." Object-oriented programming was all the rage, requiring different components to work well with and within each other. ActiveX would extend that model to include the object's context (such as a desktop application or a Web script) or the environment in which it would run (e.g., over a slow network).

For a time, it looked like Microsoft was going to make the Sun Java technologies a subset of ActiveX. Such a partnership would undoubtedly have benefited both. A few dozen lawsuits and an antitrust case later, however, the split is complete.

Dot Net (.NET)

The advent of .NET brings us to the present. The .NET initiative continues the evolution of the Microsoft technologies including ActiveX and the doomed DNA product. Its extensive support for open standards constitutes an apparent paradigm shift for Microsoft.

Microsoft software has the lion's share of almost every software market out there, but the market is changing fast. The key to remember here is that .NET is a *server* technology initiative. It doesn't matter what client software you're using; it doesn't even matter if you're running a cell phone, PDA, wristwatch, or toaster instead of a PC. The client market has become a commodity system. The important stuff is on the server… or servers.

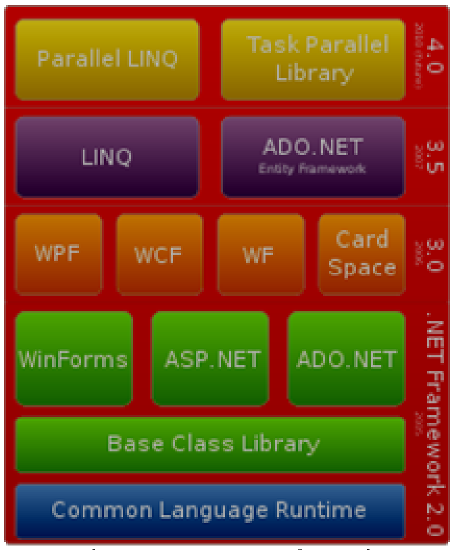
Building an application that tracks an individual or organization's personal, professional, and other information is easy. Any MCSD, MCDBA, Perl, or Java guru can do it in a flash and for a song. Building a suite of applications, with different physical and conceptual architectures, availability requirements, and resources, is a whole other ballgame. The adoption of both existing and emerging open standards like XML promises to ease the burden of integrating disparate systems. This is one of the goals of the .NET framework.

5.2 COMPARISON OF JAVA AND .NET

At root level architecture and components, MS.NET and J2EE platforms are very similar. Both are virtual machine based architecture having CLR and Java Virtual Machine (JVM) as the underlying virtual machine for the management and execution of programs. Both provide memory, security and thread management on behalf of the program and both try to decouple the applications with the execution environment (OS and physical machine). Both, basically, target the Web based applications and especially the XML based web services. Both provide managed access to memory and no direct access to memory is allowed to their managed applications.

However, there are few contrasts in the architecture and design of the two virtual machines. Microsoft .NET framework's architecture is more coupled to the Microsoft Windows Operating System which makes it difficult to implement it on various operating systems and physical machines. Java, on the other hand, is available on almost all major platforms. At the darker side, J2EE architecture and JVM is more coupled to the Java programming language while Microsoft.NET has been designed from the scratch to support language independence and language integration. Microsoft.NET covers the component development and integration in much more detail than Java. The versioning policy of .NET is simply the best implemented versioning solution in the software development history. Java has got the support of industry giants like Sun, IBM, Apache and Oracle while the Microsoft.NET is supported by giants like Microsoft, Intel, and HP.

5.3 ARCHITECTURE OF .NET FRAMEWORK



The .NET Framework Stack

5.3.1 Common Language Runtime

The Common Language Runtime (CLR) is a core component of Microsoft's .NET initiative. It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code. In the CLR, code is expressed in a form of bytecode called the Common Intermediate Language (CIL, previously known as MSIL—Microsoft Intermediate Language).

Developers using the CLR write code in a language such as C# or VB.NET. At compile time, a .NET compiler converts such code into CIL code. At runtime, the CLR's just-in-time compiler converts the CIL code into code native to the operating system. Alternatively, the CIL code can be compiled to native code in a separate step prior to runtime by using the Native Image Generator (NGEN). This speeds up all later runs of the software as the CIL-to-native compilation is no longer necessary.

Although some other implementations of the Common Language Infrastructure run on non-Windows operating systems, Microsoft's implementation runs only on Microsoft Windows operating systems

5.3.2 Common Type System

CTS - Common Type System

The Common Type System, support both Object Oriented Programming like Java as well as Procedural languages like 'C'. It deals with two kinds of entities: Objects and Values. Values are the familiar atomic types like integers and chars. Objects are self defining entities containing both methods and variables.

Objects and Values can be categorized into the following hierarchy:

Types can be of two kinds Value Types and Reference Types. Value Types can further categorized into built-in (for example Integer Types and Float Type) and user defined types like Enum.

Reference Type can be divided into three sub categories: Self Describing Reference Type, Pointers and Interfaces. Pointers can be sub divided into Function pointers, Managed and Unmanaged Types.

Value Types can be converted into Reference Type, and this conversion is called Boxing of Values. De-referencing the Boxed Value Types from the Referenced Type is called Un-Boxing.

Casting rules from one type to another, for example conversion of char to integer types are also defined within the Common Type System.

Common Type System also defines scope and assemblies. An assembly is a configured set of loadable code modules and other resources that together implement a unit of functionality. A scope is a collection of grouped names of different kinds of values or reference types.

5.3.3 Metadata

.NET metadata, in the Microsoft .NET framework, refers to certain data structures embedded within the Common Intermediate Language code that describes the high-level structure of the code. Metadata describes all classes and class members that are defined in the assembly, and the classes and class members that the current assembly will call from another assembly. The metadata for a method contains the complete description of the method, including the class (and the assembly that contains the class), the return type and all of the method parameters.

A .NET language compiler will generate the metadata and store this in the assembly containing the CIL. When the CLR executes CIL it will check to make sure that the metadata of the called method is the same as the metadata that is stored in the calling method. This ensures that a method can only be called with exactly the right number of parameters and exactly the right parameter types

5.3.4 Assemblies

An assembly is the functional unit of sharing and reuse in the Common Language Runtime. It is the equivalent of JAR (Java Archive) files of Java.

Assembly is a collection of physical files package in a .CAB format or newly introduced .MSI file format. The assemblies contained in a .CAB or .MSI files are called static assemblies, they include .NET Framework types (interfaces and classes) as well as resources for the assembly (bitmaps, JPEG files, resource files, etc.). They also include metadata that eliminates the need of IDL file descriptors, which were required for describing COM components.

The Common Language Runtime also provide API's that script engines use to create dynamic assemblies when executing scripts. These assemblies are run directly and are never saved to disk.

Microsoft has greatly diminished the role of Windows Registry system with introduction of assemblies concept, which is an adaptation of Java's JAR deployment technology.

Assemblies is an adaptation, but not a copy of Java's JAR technology. It has been improved upon in some ways, for example it has introduced a versioning system. However, since the .NET framework is skewed towards the Windows architecture some of the Java's JAR portability features may have been sacrificed.

Again, similar to JAR files, the assemblies too contain an entity called manifest. However, manifest in .NET framework plays somewhat wider role. Manifest is a metadata describing the inter-relationship between the entities contained in the assemblies like managed code, images and multimedia resources. Manifest also specifies versioning information.

The manifest is basically a deployment descriptor, having XML syntax. Java programmers can relate it with J2EE (Java 2 Enterprise Edition) deployment descriptors for EjB (Enterprise Java Beans) applications.

The Microsoft documentation stress that assemblies are "logical dlls". This may be a reasonable paradigm for VB or C++ programmers, but Java programmers will find it easier, if we visualize assemblies as an extension of JAR concept. However, unlike JAR, each assembly can have only one entry point defined, which can be either DllMain, WinMain, or Main.

As stated earlier, Assemblies have a manifest metadata. This contains version and digitally signed information. This purports to implement version control and authentication of the software developer. Version and authentication procedure is carried out by the runtime during loading the assembly into the code execution area.

Again, much like Java's trusted lib. concept, .NET Assemblies can be placed in secured area called global assembly cache. This area is equivalent to trusted class path of Java. Only system administrators can install or deinstall Assemblies from the global assembly cache. There is a place for downloaded or transient Assemblies called downloaded assembly cache. The Assemblies loaded from global assembly cache run outside the sandbox and have faster load time as well as enjoy more freedom to access file system resources. The Assemblies loaded from the downloaded cache area are subject to more security checks, therefore are slower to load and since they run inside the sandbox; enjoy much less privileges.

Assemblies manifests also contain information regarding sharing of code by different Applications and Application Domains.

To summarize, the Operating System can have multiple applications running simultaneously, each such application occupies a separate Win32 process and can contain multiple Application Domains. An Application Domain can be constructed from multiple assemblies.

5.3.5 Application Domains

Application domains are light weight process. It can be visualized as an extension of Java's sandbox security and Thread model.

The Common Language Runtime provides a secure, lightweight unit of processing called an application domain. Application domains also enforce security policy.

By light weight it means that multiple application domains run in a single Win32 process, yet they provide a kind of fault isolation, that is fault in one application domain does not corrupt other application domains. This aids in enhancing execution security against viruses as well as helps in debugging faulty codes.

The Common Language Runtime relies on type safety and verifiability features of Common Type System (CTS) to provide fault isolation between application domains. Since type verification can be conducted statically before execution, it is cost efficient and needs less security support from microprocessor hardware.

Each application can have multiple application domains associated with it. And each application domain has a configuration file, containing security permissions. This configuration information is used by the Common Language Runtime to provide sandbox security similar to that of Java sandbox model.

Although multiple application domains can run within a process, no direct calls are allowed between methods of objects in different application domains. Instead, a proxy mechanism is used for code space isolation.

5.3.6 FCL (Framework class library)

.NET Framework provides huge set of Framework (or Base) Class Library (FCL) for common, usual tasks. FCL contains thousands of classes to provide the access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Streams, Threads, Security, Network Programming, Windows Programming, Web Programming, Data Access, etc. It is simply the largest standard library ever shipped with any development environment or programming language. The best part of this library is they follow extremely efficient OO design (design patterns) making their access and use very simple and predictable. You can use the classes in FCL in your program just as you use any other class and can even apply inheritance and polymorphism on these.

5.4 FEATURES OF .NET

Interoperability

Because interaction between new and older applications is commonly required, the .NET Framework provides means to access functionality that is implemented in programs that execute outside the .NET environment. Access to COM components is provided in the System. Runtime.Interop Services and System.EnterpriseServices namespaces of the framework; access to other functionality is provided using the P/Invoke feature.

Common Runtime Engine

The Common Language Runtime (CLR) is the virtual machine component of the .NET Framework. All .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

Language Independence

The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all possible datatypes and programming constructs supported by the CLR and how they may or may not interact with each other conforming to the Common Language Infrastructure (CLI) specification. Because of this feature, the .NET Framework supports the exchange of types and object instances between libraries and applications written using any conforming .NET language.

Base Class Library

The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction, XML document manipulation and so on.

Simplified Deployment

The .NET Framework includes design features and tools that help manage the installation of computer software to ensure that it does not interfere with previously installed software, and that it conforms to security requirements.

Security

The design is meant to address some of the vulnerabilities, such as buffer overflows, that have been exploited by malicious software. Additionally, .NET provides a common security model for all applications.

Portability

The design of the .NET Framework allows it to theoretically be platform agnostic, and thus cross-platform compatible. That is, a program written to use the framework should run without change on any type of system for which the framework is implemented. While Microsoft has never implemented the full framework on any system except Microsoft Windows, the framework is engineered to be platform agnostic, and cross-platform implementations are available for other operating systems (see Silverlight and the Alternative implementations section below). Microsoft submitted the specifications for the Common Language Infrastructure (which includes the core class libraries, Common Type System, and the Common Intermediate Language), the C# language, and the C++/CLI language to both ECMA and the ISO, making them available as open standards. This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

5.5 ADVANTAGES AND APPLICATION

Advantages:

Consistent Programming Model

Different programming languages have different approaches for doing a task. For example, accessing data with a VB 6.0 application and a VC++ application is totally different. When using different programming languages to do a task, a disparity exists among the approach developers use to perform the task. The difference in techniques comes from how different languages interact with the underlying system that applications rely on.

With .NET, for example, accessing data with a VB .NET and a C# .NET looks very similar apart from slight syntactical differences. Both the programs need to import the System.Data namespace, both the programs establish a connection with the database and both the programs run a query and display the data on a data grid. The VB 6.0 and VC++ example mentioned in the first paragraph explains that there is more than one way to do a particular task within the same language. The .NET example explains that there's a unified means of accomplishing the same task by using the .NET Class Library, a key component of the .NET Framework.

The functionality that the .NET Class Library provides is available to all .NET languages resulting in a consistent object model regardless of the programming language the developer uses.

Direct Support for Security

Developing an application that resides on a local machine and uses local resources is easy. In this scenario, security isn't an issue as all the resources are available and accessed locally. Consider an application that accesses data on a remote machine or has to perform a privileged task on behalf of a nonprivileged user. In this scenario security is much more important as the application is accessing data from a remote machine.

With .NET, the Framework enables the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. This makes distributed computing more secure because .NET developers cooperate with network security devices instead of working around their security limitations.

Simplified Development Efforts

Let's take a look at this with Web applications. With classic ASP, when a developer needs to present data from a database in a Web page, he is required to write the application logic (code) and presentation logic (design) in the same file. He was required to mix the ASP code with the HTML code to get the desired result.

ASP.NET and the .NET Framework simplify development by separating the application logic and presentation logic making it easier to maintain the code. You write the design code (presentation logic) and the actual code (application logic) separately eliminating the need to mix HTML code with ASP code. ASP.NET can also handle the details of maintaining the state of the controls, such as contents in a textbox, between calls to the same ASP.NET page.

Another advantage of creating applications is debugging. Visual Studio .NET and other third party providers provide several debugging tools that simplify application development. The .NET Framework simplifies debugging with support for Runtime diagnostics. Runtime diagnostics helps you to track down bugs and also helps you to determine how well an application performs. The .NET Framework provides three types of Runtime diagnostics: Event Logging, Performance Counters and Tracing.

Easy Application Deployment and Maintenance

The .NET Framework makes it easy to deploy applications. In the most common form, to install an application, all you need to do is copy the application along with the components it requires into a directory on the target computer. The .NET Framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET Framework ensures that all the components the application depends on are available on the computer before the application begins to execute.

Real World Application

Microsoft's passport service is an example of a .NET service. Passport is a Web-based service designed to make signing in to Websites fast and easy. Passport enables participating sites to authenticate a user with a single set of sign-in credentials eliminating the need for users to remember numerous passwords and sign-in names. You can use one name and password to sign in to all .NET Passport-participating sites and services. You can store personal information in your .NET Passport profile and, if you choose, automatically share that information when you sign in so that participating sites can provide you with personalized services. If you use Hotmail for your email needs then you should be very much familiar with the passport service.

To find out more about how Businesses are implementing Web Services and the advantages it is providing please visit Microsoft's Website and check out the case studies published.

Exercise:

1. Explain the evolution of .NET framework.
2. Explain the different components of .NET framework.
3. What is the role of CLR in .NET.
4. Write a note on Assemblies and Metadata.
5. Explain the features of .NET.
6. Explain the portability features for .Net applications.
7. Explain the role of .NET in developing the web services.

❖❖❖❖

6

C#

Unit Structure

Basic principles of Object Oriented Programming, Basic Data Types, Building Control, Structures, Operators, Declares Variables, Reference data types, Strings, Arrays, Classes and Objects, Exception Handling, Generics, File Handling, Inheritance and Polymorphism, Database Programming

6.1 BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING.

- **Object-Oriented Programming:-**

At the center of C# is *object-oriented programming* (OOP). The object-oriented methodology is inseparable from C#, and all C# programs are to at least some extent object oriented. Because of its importance to C#, it is useful to understand OOP's basic principles before you write even a simple C# program.

OOP is a powerful way to approach the job of programming. Programming methodologies have changed dramatically since the invention of the computer, primarily to accommodate the increasing complexity of programs. For example, when computers were first invented, programming was done by toggling in the binary machine instructions using the computer's front panel. As long as programs were just a few hundred instructions long, this approach worked. As programs grew, assembly language was invented so that a programmer could deal with larger, increasingly complex programs, using symbolic representations of the machine instructions. As programs continued to grow, high-level languages such as FORTRAN and COBOL were introduced that gave the programmer more tools with which to handle complexity. When these early languages began to reach their breaking point, structured programming languages, such as C, were invented.

At each milestone in the history of programming, techniques and tools were created to allow the programmer to deal with increasingly greater complexity. Each step of the way, the new approach took the best elements of the previous methods and moved forward. The same is true of object-oriented programming. Prior to OOP, many projects were nearing (or exceeding) the point where the structured approach no longer worked. A better way to handle complexity was needed, and object-oriented programming was the solution.

Object-oriented programming took the best ideas of structured programming and combined them with several new concepts. The result was a different and better way of organizing a program. In the most general sense, a program can be organized in one of two ways: around its code (what is happening) or around its data (what is being affected). Using only structured programming techniques, programs are typically organized around code. This approach can be thought of as “code acting on data.”

Object-oriented programs work the other way around. They are organized around data, with the key principle being “data controlling access to code.” In an object-oriented language, you define the data and the code that is permitted to act on that data. Thus, a data type defines precisely the operations that can be applied to that data.

To support the principles of object-oriented programming, all OOP languages, including C#, have three traits in common: encapsulation, polymorphism, and inheritance. Let’s examine each.

- **6.1.0 Encapsulation:-**

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. In an object-oriented language, code and data can be bound together in such a way that a self-contained *black box* is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an *object* is created. In other words, an object is the device that supports encapsulation.

Within an object, code, data, or both may be *private* to that object or *public.* Private code or data is known to and accessible by only another part of the object. That is, private code or data cannot be accessed by a piece of the program that exists outside the object. When code or data is public, other parts of your program can access it even though it is defined within an object. Typically, the public parts of an object are used to provide a controlled interface to the private elements.

C#'s basic unit of encapsulation is the *class.* A class defines the form of an object. It specifies both the data and the code that will operate on that data. C# uses a class specification to construct *objects.* Objects are instances of a class. Thus, a class is essentially a set of plans that specify how to build an object.

Collectively, the code and data that constitute a class are called its *members*. The data defined by the class is referred to as *fields*. The terms *member variables* and *instance variables* also are used. The code that operates on that data is contained within *function members,* of which the most common is the *method.* Method is C#'s term for a subroutine. (Other function members include properties, events, and constructors.) Thus, the methods of a class contain code that acts on the fields defined by that class.

- **6.1.1 Polymorphism:-**

Polymorphism (from the Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions. A simple example of polymorphism is found in the steering wheel of an automobile. The steering wheel (the interface) is the same no matter what type of actual steering mechanism is used. That is, the steering wheel works the same whether your car has manual steering, power steering, or rack-and-pinion steering. Thus, turning the steering wheel left causes the car to go left no matter what type of steering is used. The benefit of the uniform interface is, of course, that once you know how to operate the steering wheel, you can drive any type of car.

The same principle can also apply to programming. For example, consider a *stack* (which is a first-in, last-out list). You might have a program that requires three different types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. In this case, the algorithm that implements each stack is the same, even though the data being stored differs. In a non-object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in C# you can create one general set of stack routines that works for all three specific situations. This way, once you know how to use one stack, you can use them all.

More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods.” This means that it is possible to design a single interface to a group of related activities. Polymorphism helps reduce complexity by allowing the same interface to be used to specify a *general class of action.* It is the compiler’s job to select the *specific action* (that is, method) as it applies to each situation. You, the programmer, don’t need to do this selection manually. You need only remember and utilize the general interface.

- **6.1.2 Inheritance:-**

Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification. If you think about it, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Red Delicious apple is part of the classification *apple,* which in turn is part of the *fruit* class, which is under the larger class *food.* That is, the *food* class possesses certain qualities (edible, nutritious, and so on) that also, logically, apply to its subclass, *fruit.* In addition to these qualities, the *fruit* class has specific characteristics (juicy, sweet, and so on) that distinguish it from other food. The *apple* class defines those qualities specific to an apple (grows on trees, not tropical, and so on). A Red Delicious apple would, in turn, inherit all the qualities of all preceding classes and would define only those qualities that make it unique.

Without the use of hierarchies, each object would have to explicitly define all of its characteristics. Using inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

6.2 BASIC DATA TYPES.

- **Why Data Types Are Important:-**

Data types are especially important in C# because it is a strongly typed language. This means that all operations are type-checked by the compiler for type compatibility. Illegal operations will not be compiled. Thus, strong type-checking helps prevent errors and enhances reliability. To enable strong type-checking, all variables, expressions, and values have a type. There is no concept of a “typeless” variable, for example. Furthermore, the type of a value determines what operations are allowed on it. An operation allowed on one type might not be allowed on another

- **6.2.0 C#’s Value Types:-**

C# contains two general categories of built-in data types: *value types* and *reference types.* C#’s reference types are defined by classes, and a discussion of classes is deferred until later. However, at the core of C# are its 13 value types, which are shown in Table 3-1. These are built-in types that are defined by keywords in the C# language, and they are available for use by any C# program.

Type	**Meaning**
bool	Represents true/false values
byte	8-bit unsigned integer
char	Character
decimal	Numeric type for financial calculations
double	Double-precision floating point
float	Single-precision floating point
int	Integer
long	Long integer
sbyte	8-bit signed integer
short	Short integer
uint	An unsigned integer
ulong	An unsigned long integer
ushort	An unsigned short integer

The term *value type* indicates that variables of these types contain their values directly. (This differs from reference types, in which a variable contains a reference to the actual value.) Thus, the value types act much like the data types found in other programming languages, such as C++. The value types are also known as *simple types.* The term *primitive type* is also occasionally used.

C# strictly specifies a range and behavior for each value type. Because of portability requirements, C# is uncompromising on this account. For example, an **int** is the same in all execution environments. There is no need to rewrite code to fit a specific platform. While strictly specifying the size of the value types may cause a small loss of performance in some environments, it is necessary in order to achieve portability.

Note *C# 2.0 added a new feature called a* nullable type, *which enables a variable to hold an undefined value. A nullable type can be created for any value type, including the built-in types.*

- **6.2.1 Integers:-**

C# defines nine integer types: **char**, **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long**, and **ulong**. However, the **char** type is primarily used for representing characters, and it is discussed later in this chapter. The remaining eight integer types are used for numeric calculations. Their bit-width and ranges are shown here:

Type	**Width in Bits**	**Range**
byte	8	0 to 255
sbyte	8	−128 to 127
short	16	−32,768 to 32,767
ushort	16	0 to 65,535
int	32	−2,147,483,648 to 2,147,483,647
uint	32	0 to 4,294,967,295
long	64	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	64	0 to 18,446,744,073,709,551,615

As the table shows, C# defines both signed and unsigned versions of the various integer types. The difference between signed and unsigned integers is in the way the high-order bit of the integer is interpreted. If a signed integer is specified, then the C# compiler will generate code that assumes that the high-order bit of an integer is to be used as a *sign flag.* If the sign flag is 0, then the number is positive; if it is 1, then the number is negative. Negative numbers are almost always represented using the *two's complement* approach. In this method, all bits in the negative number are reversed, and then 1 is added to this number.

Signed integers are important for a great many algorithms, but they have only half the absolute magnitude of their unsigned relatives. For example, as a **short**, here is 32,767:

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

For a signed value, if the high-order bit were set to 1, the number would then be interpreted as −1 (assuming the two's complement format). However, if you declared this to be a **ushort**, then when the high-order bit was set to 1, the number would become 65,535.

Probably the most commonly used integer type is **int**. Variables of type **int** are often employed to control loops, to index arrays, and for general-purpose integer math. When you need an integer that has a range greater than **int**, you have many options. If the value you want to store is unsigned, you can use **uint**. For large signed values, use **long**. For large unsigned values, use **ulong**. For example, here is a program that computes the distance from the Earth to the sun, in inches. Because this value is so large, the program uses a **long** variable to hold it.

```
// Compute the distance from the Earth to the sun, in inches.

using System;

class Inches {
  public static void Main() {
    long inches;
    long miles;

    miles = 93000000; // 93,000,000 miles to the sun

    // 5,280 feet in a mile, 12 inches in a foot
    inches = miles * 5280 * 12;

    Console.WriteLine("Distance to the sun: " +
                      inches + " inches.");  }
}
```

Here is the output from the program:

```
Distance to the sun: 5892480000000 inches.
```

Clearly, the result could not have been held in an **int** or **uint** variable.

The smallest integer types are **byte** and **sbyte**. The **byte** type is an unsigned value between 0 and 255. Variables of type **byte** are especially useful when working with raw binary data, such as a byte stream of data produced by some device. For small signed integers, use **sbyte**. Here is an example that uses a variable of type **byte** to control a **for** loop that produces the summation of the number 100:

```
// Use byte.

using System;

class Use_byte {
  public static void Main() {
    byte x;
    int sum;

    sum = 0;
    for(x = 1; x <= 100; x++)
      sum = sum + x;

    Console.WriteLine("Summation of 100 is " + sum);
  }
}
```

The output from the program is shown here:

```
Summation of 100 is 5050
```

Since the **for** loop runs only from 0 to 100, which is well within the range of a **byte**, there is no need to use a larger type variable to control it.

When you need an integer that is larger than a **byte** or **sbyte**, but smaller than an **int** or **uint**, use **short** or **ushort**

- **6.2.2 Floating-Point Types:-**

The floating- point types can represent numbers that have fractional components. There are two kinds of floating-point types, **float** and **double**, which represent single- and double-precision numbers, respectively. The type **float** is 32 bits wide and has an approximate range of 1.5E−45 to 3.4E+38. The **double** type is 64 bits wide and has an approximate range of 5E−324 to 1.7E+308.

Of the two, **double** is the most commonly used. One reason for this is that many of the math functions in C#’s class library (which is the .NET Framework library) use **double** values. For example, the **Sqrt( )** method (which is defined by the standard **System**.**Math** class) returns a **double** value that is the square root of its **double** argument. Here, **Sqrt( )** is used to compute the radius of a circle given the circle’s area:

```
// Find the radius of a circle given its area.

using System;
```

```
class FindRadius {
  public static void Main() {
    Double r;
    Double area;

    area = 10.0;

    r = Math.Sqrt(area / 3.1416);



    Console.WriteLine("Radius is " + r);
    }
}
```

The output from the program is shown here:

```
Radius is 1.78412203012729
```

One other point about the preceding example. As mentioned, **Sqrt( )** is a member of the **Math** class. Notice how **Sqrt( )** is called; it is preceded by the name **Math**. This is similar to the way **Console** precedes **WriteLine( )**. Although not all standard methods are called by specifying their class name first, several are, as the next example shows.

The following program demonstrates several of C#'s trigonometric functions, which are also part of C#'s math library. They also operate on **double** data. The program displays the sine, cosine, and tangent for the angles (measured in radians) from 0.1 to 1.0.

```
//  Demonstrate Math.Sin(), Math.Cos(), and Math.Tan().

using System;

class Trigonometry {
  public static void Main() {
    Double theta; // angle in radians

    for(theta = 0.1; theta <= 1.0; theta = theta + 0.1) {
      Console.WriteLine("Sine of " + theta + "  is " +
                  Math.Sin(theta));
      Console.WriteLine("Cosine of " + theta + "  is " +
                  Math.Cos(theta));
      Console.WriteLine("Tangent of " + theta + "  is " +
                  Math.Tan(theta));
      Console.WriteLine();
    } }}
```

Here is a portion of the program's output:

```
Sine of 0.1  is 0.0998334166468282
Cosine of 0.1  is 0.995004165278026
Tangent of 0.1  is 0.100334672085451

Sine of 0.2  is 0.198669330795061
Cosine of 0.2  is 0.980066577841242
Tangent of 0.2  is 0.202710035508673

Sine of 0.3  is 0.29552020666134
Cosine of 0.3  is 0.955336489125606
Tangent of 0.3  is 0.309336249609623
```

To compute the sine, cosine, and tangent, the standard library methods **Math.Sin( )**, **Math.Cos( )**, and **Math.Tan( )** are used. Like **Math.Sqrt( )**, the trigonometric methods are called with a **double** argument, and they return a **double** result. The angles must be specified in radians.

- **6.2.3 The decimal Type:-**

Perhaps the most interesting C# numeric type is **decimal**, which is intended for use in monetary calculations. The **decimal** type utilizes 128 bits to represent values within the range 1E−28 to 7.9E+28. As you may know, normal floating-point arithmetic is subject to a variety of rounding errors when it is applied to decimal values. The **decimal** type eliminates these errors and can accurately represent up to 28 decimal places (or 29 places in some cases). This ability to represent decimal values without rounding errors makes it especially useful for computations that involve money.

Here is a program that uses a **decimal** type in a financial calculation. The program computes the discounted price given the original price and a discount percentage.

```
// Use the decimal type to compute a discount.

using System;

class UseDecimal {
  public static void Main() {
    decimal price;
    decimal discount;
    decimal discounted_price;

    // compute discounted price
    price = 19.95m;
    discount = 0.15m; // discount rate is 15%

    discounted_price = price - ( price * discount);

    Console.WriteLine("Discounted price: $" + discounted_price); }}
```

The output from this program is shown here:

```
Discounted price: $16.9575
```

In the program, notice that the decimal constants are followed by the **m** or **M** suffix. This is necessary because without the suffix, these values would be interpreted as standard floating-point constants, which are not compatible with the **decimal** data type. You can assign an integer value, such as 10, to a **decimal** variable without the use of the **M** suffix, though. (A detailed discussion of numeric constants is found later in this chapter.)

Here is another example that uses the **decimal** type. It computes the future value of an investment that has a fixed rate of return over a period of years.

```
/*
  Use the decimal type to compute the future value of an investment.
*/

using System;

class FutVal {
  public static void Main() {
    decimal amount;
    decimal rate_of_return;
    int years, i;

    amount = 1000.0M;
    rate_of_return = 0.07M;
    years = 10;

    Console.WriteLine("Original investment: $" + amount);
    Console.WriteLine("Rate of return: " + rate_of_return);
    Console.WriteLine("Over " + years + " years");

    for(i = 0; i < years; i++)
      amount = amount + (amount * rate_of_return);

    Console.WriteLine("Future value is $" + amount);
  }
}
```

Here is the output:

```
Original investment: $1000
Rate of return: 0.07
Over 10 years
Future value is $1967.15135728956532249000
```

Notice that the result is accurate to many decimal places—more than you would probably want! Later in this chapter you will see how to format such output in a more appealing fashion.

- **6.2.4 Characters:-**

In C#, characters are not 8-bit quantities like they are in many other computer languages, such as C++. Instead, C# uses a 16-bit character type called *Unicode.* Unicode defines a character set that is large enough to represent all of the characters found in all human languages. Although many languages, such as English, French, or German, use relatively small alphabets, some languages, such as Chinese, use very large character sets that cannot be represented using just 8 bits. To accommodate the character sets of all languages, 16-bit values are required. Thus, in C#, **char** is an unsigned 16-bit type having a range of 0 to 65,535. The standard 8-bit ASCII character set is a subset of Unicode and ranges from 0 to 127. Thus, the ASCII characters are still valid C# characters.

A character variable can be assigned a value by enclosing the character inside single quotes. For example, this assigns X to the variable **ch**:

```
char ch;
ch = 'X';
```

You can output a **char** value using a **WriteLine( )** statement. For example, this line outputs the value in **ch**:

```
Console.WriteLine("This is ch: " + ch);
```

Although **char** is defined by C# as an integer type, it cannot be freely mixed with integers in all cases. This is because there are no automatic type conversions from integer to **char**. For example, the following fragment is invalid:

```
char ch;

ch = 10; // error, won't work
```

The reason the preceding code will not work is that 10 is an integer value, and it won't automatically convert to a **char**. If you attempt to compile this code, you will see an error message. To make the assignment legal, you would need to employ a cast, which is described later in this chapter.

- **6.2.5 The bool Type:-**

The **bool** type represents true/false values. C# defines the values true and false using the reserved words **true** and **false**. Thus, a variable or expression of type **bool** will be one of these two values. Unlike some other computer languages, in C# there is no conversion defined between **bool** and integer values. For example, 1 does not convert to true, and 0 does not convert to false.

Here is a program that demonstrates the **bool** type:

```
// Demonstrate bool values.

using System;

class BoolDemo {
  public static void Main() {
    bool b;

    b = false;
    Console.WriteLine("b is " + b);
    b = true;
    Console.WriteLine("b is " + b);

    // a bool value can control the if statement
    if(b) Console.WriteLine("This is executed.");

    b = false;
    if(b) Console.WriteLine("This is not executed.");

    // outcome of a relational operator is a bool value
    Console.WriteLine("10 > 9 is " + (10 > 9));
  }
}
```

The output generated by this program is shown here:

```
b is False
b is True
This is executed.
10 > 9 is True
```

There are three interesting things to notice about this program. First, as you can see, when a **bool** value is output by **WriteLine( )**, “True” or “False” is displayed. Second, the value of a **bool** variable is sufficient, by itself, to control the **if** statement. There is no need to write an **if** statement like this:

```
if(b == true) ...
```

Third, the outcome of a relational operator, such as **<**, is a **bool** value. This is why the expression **10 > 9** displays the value “True.” Further, the extra set of parentheses around **10 > 9** is necessary because the **+** operator has a higher precedence than the **>**.

6.3 BUILDING CONTROL STRUCTURES, OPERATORS, DECLARES VARIABLES.

- **6.3.0 Control Structures in C#:**

There are three categories of program control statements: *selection* statements, which are the **if** and the **switch**; *iteration* statements, which consist of the **for**, **while**, **do-while**, and **foreach** loops; and *jump* statements, which include **break**, **continue**, **goto**, **return**, and **throw.**

If statement:It is the powerful decision making statement.

The complete form of the **if** statement is

```
if(condition) statement;
else statement;
```

where the targets of the **if** and **else** are single statements. The **else** clause is optional. The targets of both the **if** and **else** can be blocks of statements. The general form of the **if** using blocks of statements is

```
if(condition)
{
 statement sequence
}
else
{
 statement sequence
}
```

If the conditional expression is true, the target of the **if** will be executed; otherwise, if it exists, the target of the **else** will be executed. At no time will both of them be executed. The conditional expression controlling the **if** must produce a **bool** result.

Here is a simple example that uses an **if** and **else** statement to report if a number is positive or negative:

```
// Determine if a value is positive or negative.

using System;

class PosNeg {
  public static void Main() {
    int i;

    for(i=-5; i <= 5; i++) {
      Console.Write("Testing " + i + ": ");

      if(i < 0) Console.WriteLine("negative");
      else Console.WriteLine("positive");
    }
  }
}
```

The output is shown here:

```
Testing -5: negative
Testing -4: negative
Testing -3: negative
Testing -2: negative
Testing -1: negative
Testing 0: positive
Testing 1: positive
Testing 2: positive
Testing 3: positive
Testing 4: positive
Testing 5: positive
```

In this example, if **i** is less than zero, then the target of the **if** is executed. Otherwise, the target of the **else** is executed. In no case are both executed.

- **Nested if:-**

A *nested **if*** is an **if** statement that is the target of another **if** or **else**. Nested **if**s are very common in programming. The main thing to remember about nested **if**s in C# is that an **else** statement always refers to the nearest **if** statement that is within the same block as the **else** and not already associated with an **else**. Here is an example:

```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d;
  else a = c; // this else refers to if(k > 100)
}
else a = d; // this else refers to if(i == 10)
```

As the comments indicate, the final **else** is not associated with **if(j<20)**, because it is not in the same block (even though it is the nearest **if** without an **else**). Rather, the final **else** is associated with **if(i==10)**. The inner **else** refers to **if(k>100)**, because it is the closest **if** within the same block.

The following program demonstrates a nested **if**. In the positive/negative program shown earlier, zero is reported as positive. However, for some applications, zero is considered signless. The following version of the program reports zero as being neither positive nor negative:

```
// Determine if a value is positive, negative, or zero.

using System;

class PosNegZero {
  public static void Main() {
    int i;

    for(i=-5; i <= 5; i++) {

      Console.Write("Testing " + i + ": ");

      if(i < 0) Console.WriteLine("negative");
      else if(i == 0) Console.WriteLine("no sign");
        else Console.WriteLine("positive");
    }
  }
}
```

Here is the output:

```
Testing -5: negative
Testing -4: negative
Testing -3: negative
Testing -2: negative
Testing -1: negative
Testing 0: no sign
Testing 1: positive
Testing 2: positive
Testing 3: positive
Testing 4: positive
Testing 5: positive
```

- **The if-else-if Ladder**

A common programming construct that is based upon the nested **if** is the ***if-else-if** ladder.* It looks like this:

```
if(condition)
  statement;
else if(condition)
  statement;
else if(condition)
  statement;
.
.
.
else
  statement;
```

The conditional expressions are evaluated from the top downward. As soon as a true condition is found, the statement associated with it is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final **else** statement will be executed. The final **else** often acts as a default condition. That is, if all other conditional tests fail, then the last **else** statement is performed. If there is no final **else** and all other conditions are false, then no action will take place.

The following program demonstrates the **if-else-if** ladder. It finds the smallest singledigit factor for a given value.

```
// Determine smallest single-digit factor.

using System;

class Ladder {
  public static void Main() {
    int num;

    for(num = 2; num < 12; num++) {
      if((num % 2) == 0)
        Console.WriteLine("Smallest factor of " + num + " is 2.");
      else if((num % 3) == 0)
        Console.WriteLine("Smallest factor of " + num + " is 3.");
      else if((num % 5) == 0)
        Console.WriteLine("Smallest factor of " + num + " is 5.");
      else if((num % 7) == 0)
        Console.WriteLine("Smallest factor of " + num + " is 7.");
```

```
   else
     Console.WriteLine(num + " is not divisible by 2, 3, 5, or 7.");
   }
  }
}
```

The program produces the following output:

```
Smallest factor of 2 is 2.
Smallest factor of 3 is 3.
Smallest factor of 4 is 2.
Smallest factor of 5 is 5.
Smallest factor of 6 is 2.
Smallest factor of 7 is 7.
Smallest factor of 8 is 2.
Smallest factor of 9 is 3.
Smallest factor of 10 is 2.
11 is not divisible by 2, 3, 5, or 7.
```

As you can see, the last **else** is executed only if none of the preceding **if** statements succeeds.

- **6.3.1 The switch Statement:-**

The second of C#'s selection statements is the **switch**. The **switch** provides for a multiway branch. Thus, it enables a program to select among several alternatives. Although a series of nested **if** statements can perform multiway tests, for many situations the **switch** is a more efficient approach. It works like this: The value of an expression is successively tested against a list of constants. When a match is found, the statement sequence associated with that match is executed. The general form of the **switch** statement is

```
switch(expression) {
  case constant1:
    statement sequence
    break;
   case constant2:
     statement sequence
     break;
   case constant3:
    statement sequence
    break;
 .
 .
 .
```

```
  default:
     *statement sequence*
     break;
}
```

The **switch** expression must be of an integer type, such as **char**, **byte**, **short**, or **int**, or of type **string** (which is described later in this book). Thus, floating-point expressions, for example, are not allowed. Frequently, the expression controlling the **switch** is simply a variable. The **case** constants must be literals of a type compatible with the expression. No two **case** constants in the same **switch** can have identical values.

The **default** statement sequence is executed if no **case** constant matches the expression. The **default** is optional; if it is not present, no action takes place if all matches fail. When a match is found, the statements associated with that **case** are executed until the **break** is encountered.

The following program demonstrates the **switch**:

```
// Demonstrate the switch.

using System;
class SwitchDemo {
  public static void Main() {
    int i;

    for(i=0; i<10; i++)
      switch(i) {
        case 0:
          Console.WriteLine("i is zero");
          break;
        case 1:
          Console.WriteLine("i is one");
          break;
        case 2:
          Console.WriteLine("i is two");
          break;
        case 3:
          Console.WriteLine("i is three");
          break;
        case 4:
          Console.WriteLine("i is four");
          break;
        default:
          Console.WriteLine("i is five or more");
          break;
      }
  }
}
```

The output produced by this program is shown here:

```
i is zero
i is one
i is two
i is three
i is four
i is five or more
i is five or more
i is five or more
i is five or more
i is five or more
```

As you can see, each time through the loop, the statements associated with the **case** constant that matches **i** are executed. All others are bypassed. When **i** is five or greater, no **case** constants match, so the **default** statement is executed.

In the preceding example, the **switch** was controlled by an **int** variable. As explained, you can control a **switch** with any integer type, including **char**. Here is an example that uses a **char** expression and **char** case constants:

```
// Use a char to control the switch.

using System;

class SwitchDemo2 {
  public static void Main() {
    char ch;

    for(ch='A'; ch<= 'E'; ch++)
      switch(ch) {
        case 'A':
          Console.WriteLine("ch is A");
          break;
        case 'B':
          Console.WriteLine("ch is B");
          break;
        case 'C':
          Console.WriteLine("ch is C");
          break;
        case 'D':
          Console.WriteLine("ch is D");
          break;
        case 'E':
```

```
        Console.WriteLine("ch is E");
        break;
    }
  }
}
```

The output from this program is shown here:

```
ch is A
ch is B
ch is C
ch is D
ch is E
```

Notice that this example does not include a **default** statement. Remember, the **default** is optional. When not needed, it can be left out.

In C#, it is an error for the statement sequence associated with one **case** to continue on into the next **case**. This is called the “no fall-through” rule. This is why **case** sequences end with a **break** statement. (You can avoid fall-through in other ways, such as by using the **goto**, discussed later in this chapter, but **break** is by far the most commonly used approach.) When encountered within the statement sequence of a **case**, the **break** statement causes program flow to exit from the entire **switch** statement and resume at the next statement outside the **switch**. The **default** statement must also not “fall through,” and it too usually ends with a **break**.

The no fall-through rule is one point on which C# differs from C, C++, and Java. In those languages, one **case** may continue on (that is, fall through) into the next **case**. There are two reasons that C# instituted the no fall-through rule for **case**s. First, it allows the compiler to freely rearrange the order of the **case** statements, perhaps for purposes of optimization. Such a rearrangement would not be possible if one **case** could flow into the next. Second, requiring each **case** to explicitly end prevents a programmer from accidentally allowing one **case** to flow into the next.

Although you cannot allow one **case** sequence to fall through into another, you can have two or more **case** statements refer to the same code sequence, as shown in this example:

```
// Empty cases can fall through.

using System;
```

```
class EmptyCasesCanFall {
  public static void Main() {
    int i;

    for(i=1; i < 5; i++)
      switch(i) {
        case 1:
        case 2:
        case 3: Console.WriteLine("i is 1, 2 or 3");
          break;
        case 4: Console.WriteLine("i is 4");
          break;
      }

  }
}
```

The output is shown here:

```
i is 1, 2 or 3
i is 1, 2 or 3
i is 1, 2 or 3
i is 4
```

In this example, if **i** has the value 1, 2, or 3, then the first **WriteLine( )** statement executes. If it is 4, then the second **WriteLine( )** statement executes. The stacking of **case**s does not violate the no fall-through rule, because the **case** statements all use the same statement sequence.

Stacking **case** statements is a commonly employed technique when several **case**s share common code. This technique prevents the unnecessary duplication of code sequences.

- **Nested switch Statements**

It is possible to have a **switch** as part of the statement sequence of an outer **switch**. This is called a *nested **switch**.* The **case** constants of the inner and outer **switch** can contain common values and no conflicts will arise. For example, the following code fragment is perfectly acceptable:

```
switch(ch1) {
  case 'A': Console.WriteLine("This A is part of outer switch.");
    switch(ch2) {
      case 'A':
        Console.WriteLine("This A is part of inner switch");
        break;
      case 'B': // ...
    } // end of inner switch
    break;
  case 'B': // ...
```

6.3.2 C# Operators

Results are computed by building expressions. These expressions are built by combining variables and operators together into statements. The following table describes the allowable operators, their precedence, and associativity.

Table 2-4. Operators with their precedence and Associativity

Category (by precedence)	**Operator(s)**	**Associativity**
Primary	x.y f(x) a[x] x++ x-- new typeof default checked unchecked delegate	left
Unary	+ - ! ~ ++x --x (T)x	left
Multiplicative	* / %	left
Additive	+ -	left
Shift	<< >>	left
Relational	< > <= >= is as	left
Equality	== !=	right
Logical AND	&	left
Logical XOR	^	left
Logical OR	|	left
Conditional AND	&&	left
Conditional OR	||	left
Null Coalescing	??	left
Ternary	?:	right
Assignment	= *= /= %= += -= <<= >>= &= ^= |= =>	right

Left associativity means that operations are evaluated from left to right. Right associativity mean all operations occur from right to left, such as assignment operators where everything to the right is evaluated before the result is placed into the variable on the left.

Most operators are either unary or binary. Unary operators form expressions on a single variable, but binary operators form expressions with two variables

- ***Binary Operators: Binary.cs***

```
using System;

class Binary
{
   public static void Main()
   {
      int x, y, result;
      float floatresult;

      x = 7;
      y = 5;

      result = x+y;
      Console.WriteLine("x+y: {0}", result);

      result = x-y;
      Console.WriteLine("x-y: {0}", result);

      result = x*y;
      Console.WriteLine("x*y: {0}", result);

      result = x/y;
      Console.WriteLine("x/y: {0}", result);

      floatresult = (float)x/(float)y;
      Console.WriteLine("x/y: {0}", floatresult);

      result = x%y;
      Console.WriteLine("x%y: {0}", result);

      result += x;
      Console.WriteLine("result+=x: {0}", result);
   }
}
```

And here's the output:

```
   x+y: 12
   x-y: 2
   x*y: 35
```

```
x/y: 1
x/y: 1.4
x%y: 2
result+=x: 9
```

- ***Unary Operators: Unary.cs***

```
using System;

class Unary
{
   public static void Main()
   {
      int unary = 0;
      int preIncrement;
      int preDecrement;
      int postIncrement;
      int postDecrement;
      int positive;
      int negative;
      sbyte bitNot;
      bool logNot;

      preIncrement = ++unary;
      Console.WriteLine("pre-Increment: {0}", preIncrement);

      preDecrement = --unary;
      Console.WriteLine("pre-Decrement: {0}",
preDecrement);

      postDecrement = unary--;
      Console.WriteLine("Post-Decrement: {0}",
postDecrement);

      postIncrement = unary++;
      Console.WriteLine("Post-Increment: {0}",
postIncrement);

      Console.WriteLine("Final Value of Unary: {0}", unary);

      positive = -postIncrement;
      Console.WriteLine("Positive: {0}", positive);

      negative = +postIncrement;
```

```
            Console.WriteLine("Negative: {0}", negative);

            bitNot = 0;
            bitNot = (sbyte)(~bitNot);
            Console.WriteLine("Bitwise Not: {0}", bitNot);

            logNot = false;
            logNot = !logNot;
            Console.WriteLine("Logical Not: {0}", logNot);
        }
```

```
pre-Increment: 1
pre-Decrement 0
Post-Decrement: 0
Post-Increment: -1
Final Value of Unary: 0
Positive: 1
Negative: -1
Bitwise Not: -1
Logical Not: true
```

- **variable declaration:**

```
void F() {
  int x; x = 1;
  int y;
  int z; z = x * 2;
}
```

6.4 REFERENCE DATA TYPES, STRINGS

- **6.4.0 Reference Types**

In contrast to value types, the value of a reference types is allocated on the heap. Another name for a reference type, that you might be more familiar with, is an object. Reference types stores the reference to the data, unlike value types, that stores the value.

Following example gives use of reference type:

```
using System;
public class Cat
{
   private int age;

   public void SetAge(int years)
```

```
    {
       age = years;
    }
    public int GetAge()
    {
      return age;
    }

}
public class RefTest
{
    public static void Main()
    {
      Cat miranda = new Cat();
      miranda.SetAge(6);

      Cat caesar = miranda; //caesar now equals miranda

      Console.WriteLine("Caesar: " + caesar.GetAge());
      Console.WriteLine("Miranda: " + miranda.GetAge());

      miranda.SetAge(10); //change Miranda's age, what happen to
Caesar now?

      Console.WriteLine("Caesar: " + caesar.GetAge());
      Console.WriteLine("Miranda: " + miranda.GetAge());
      Console.WriteLine(caesar == miranda);
    }
}
```

Memory for variables that are reference types are not automatically freed when they go out of scope. Instead the Garbage Collector is responsible for this.

In contrast to value types, a reference type does not necessarily have a value. It can be null. This means that the variable does not reference any data.

- **6.4.1 string:**

string defines and supports character strings. In many other programming languages a string is an array of characters. This is not the case with C#. In C#, strings are objects.

The easiest way to construct a **string** is to use a string literal. For example, here **str** is a **string** reference variable that is assigned a reference to a string literal:

```
string str = "C# strings are powerful.";
```

- **Strings Are Immutable**

Here is something that might surprise you: The contents of a **string** object are immutable. That is, once created, the character sequence comprising that string cannot be altered. This restriction allows C# to implement strings more efficiently. Even though this probably sounds like a serious drawback, it isn't. When you need a string that is a variation on one that already exists, simply create a new string that contains the desired changes. Since unused string objects are automatically garbage-collected, you don't even need to worry about what happens to the discarded strings.

It must be made clear, however, that **string** reference variables may, of course, change the object to which they refer. It is just that the contents of a specific **string** object cannot be changed after it is created.

The string type represents a sequence of zero or more Unicode characters. string is an alias for String in the .NET Framework.

Although string is a reference type, the equality operators (== and !=) are defined to compare the values of string objects, not references. This makes testing for string equality more intuitive. For example

```
string a = "hello";
string b = "h";
// Append to contents of 'b'
b += "ello";
Console.WriteLine(a == b);
Console.WriteLine((object)a == (object)b);
```

6.5 ARRAYS:-

An *array* is a collection of variables of the same type that are referred to by a common name. In C#, arrays can have one or more dimensions, although the one-dimensional array is the most common. Arrays are used for a variety of purposes because they offer a convenient means of grouping together related variables. For example, you might use an array to hold a record of the daily high temperature for a month, a list of stock prices, or your collection of programming books.

One-Dimensional Arrays

A *one-dimensional array* is a list of related variables. Such lists are common in programming. For example, you might use a one-dimensional array to store the account numbers of the active users on a network. Another array might store the current batting averages for a baseball team.

To declare a one-dimensional array, you will use this general form:

type[ ] *array-name* = new *type*[*size*];

sample:

int[] sample = new int[10];

Initializing an Array

In the preceding program, the **nums** array was given values by hand, using ten separate assignment statements. While that is perfectly correct, there is an easier way to accomplish this. Arrays can be initialized when they are created. The general form for initializing a one-dimensional array is shown here:

type[ ] array-name = { *val1*, *val2*, *val3*, ..., *valN* };

Multidimensional Arrays

Although the one-dimensional array is the most commonly used array in programming, multidimensional arrays are certainly not rare. A *multidimensional array* is an array that has two or more dimensions, and an individual element is accessed through the combination of two or more indices.

Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. In a two-dimensional array, the location of any specific element is specified by two indices. If you think of a two-dimensional array as a table of information, one index indicates the row, the other indicates the column.

To declare a two-dimensional integer array **table** of size 10, 20, you would write

int[,] table = new int[10, 20];

C# also allows you to create a special type of two-dimensional array called a *jagged array.* A jagged array is an *array of arrays* in which the length of each array can differ. Thus, a jagged array can be used to create a table in which the lengths of the rows are not the same.

Jagged arrays are declared by using sets of square brackets to indicate each dimension. For example, to declare a two-dimensional jagged array, you will use this general form:

*type[ ] [ ] ar*ray-name = new *type*[*size*][ ];

Here, *size* indicates the number of rows in the array. The rows, themselves, have not been allocated. Instead, the rows are allocated individually. This allows for the length of each row to vary. For example, the following code allocates memory for the first dimension of **jagged** when it is declared. It then allocates the second dimensions manually.

```
int[ ][ ] jagged = new int[3][ ];
jagged[0] = new int[4];
jagged[1] = new int[3];
jagged[2] = new int[5];
```

Assigning Array References

As with other objects, when you assign one array reference variable to another, you are simply changing the object to which the variable refers. You are not causing a copy of the array to be made, nor are you causing the contents of one array to be copied to the other. For example, consider this program:

```
// Assigning array reference variables.

using System;

class AssignARef {
  public static void Main() {
    int i;

    int[ ] nums1 = new int[10];
    int[ ] nums2 = new int[10];

    for(i=0; i < 10; i++) nums1[i] = i;

    for(i=0; i < 10; i++) nums2[i] = -i;

    Console.Write("Here is nums1: ");
    for(i=0; i < 10; i++)
      Console.Write(nums1[i] + " ");
    Console.WriteLine();
    Console.Write("Here is nums2: ");
    for(i=0; i < 10; i++)
      Console.Write(nums2[i] + " ");
    Console.WriteLine();

    nums2 = nums1; // now nums2 refers to nums1
    Console.Write("Here is nums2 after assignment: ");
    for(i=0; i < 10; i++)

      Console.Write(nums2[i] + " ");
    Console.WriteLine();

   // now operate on nums1 array through nums2
   nums2[3] = 99;
    Console.Write("Here is nums1 after change through nums2: ");
    for(i=0; i < 10; i++)
      Console.Write(nums1[i] + " ");
    Console.WriteLine();
  }
}
```

The output from the program is shown here:

```
Here is nums1: 0 1 2 3 4 5 6 7 8 9
Here is nums2: 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
Here is nums2 after assignment: 0 1 2 3 4 5 6 7 8 9
Here is nums1 after change through nums2: 0 1 2 99 4 5 6 7 8 9
```

As the output shows, after the assignment of **nums1** to **nums2**, both array reference variables refer to the same object.

Using the Length Property

A number of benefits result because C# implements arrays as objects. One comes from the fact that each array has associated with it a **Length** property that contains the number of elements that an array can hold. Thus, each array carries with it a field that contains the array's length. Here is a program that demonstrates this property:

```
// Use the Length array property.

using System;

class LengthDemo {
  public static void Main() {
    int[ ] nums = new int[10];

    Console.WriteLine("Length of nums is " + nums.Length);
    // use Length to initialize nums
    for(int i=0; i < nums.Length; i++)
      nums[i] = i * i;

    // now use Length to display nums
    Console.Write("Here is nums: ");
    for(int i=0; i < nums.Length; i++)
      Console.Write(nums[i] + " ");

    Console.WriteLine();
  }
}
```

This program displays the following output:

```
Length of nums is 10
Here is nums: 0 1 4 9 16 25 36 49 64 81
```

In **LengthDemo** notice the way that **nums.Length** is used by the **for** loops to govern the number of iterations that take place. Since each array carries with it its own length, you can use this information rather than manually keeping track of an array's size. Keep in mind that the value of **Length** has nothing to do with the number of elements that are actually in use. It contains the number of elements that the array is capable of holding.

When the length of a multidimensional array is obtained, the total number of elements that can be held by the array is returned. For example:

```
// Use the Length array property on a 3-D array.

using System;

class LengthDemo3D {
  public static void Main() {
    int[,,] nums = new int[10, 5, 6];

    Console.WriteLine("Length of nums is " + nums.Length);
  }
}
```

The output is shown here:

```
Length of nums is 300
```

As the output verifies, **Length** obtains the number of elements that **nums** can hold, which is 300 (10×5×6) in this case. It is not possible to use **Length** to obtain the length of a specific dimension.

The inclusion of the **Length** property simplifies many algorithms by making certain types of array operations easier—and safer—to perform. For example, the following program uses **Length** to reverse the contents of an array by copying it back-to-front into another array:

```
// Reverse an array.

using System;

class RevCopy {
  public static void Main() {
    int i,j;
    int[ ] nums1 = new int[10];
    int[ ] nums2 = new int[10];

    for(i=0; i < nums1.Length; i++) nums1[i] = i;

    Console.Write("Original contents: ");
    for(i=0; i < nums2.Length; i++)
      Console.Write(nums1[i] + " ");
    Console.WriteLine();

    // reverse copy nums1 to nums2
    if(nums2.Length >= nums1.Length) // make sure nums2 is long
enough
      for(i=0, j=nums1.Length-1; i < nums1.Length; i++, j--)
        nums2[j] = nums1[i];

    Console.Write("Reversed contents: ");
    for(i=0; i < nums2.Length; i++)
      Console.Write(nums2[i] + " ");

    Console.WriteLine();
  }
}
```

Here is the output:

```
Original contents: 0 1 2 3 4 5 6 7 8 9
Reversed contents: 9 8 7 6 5 4 3 2 1 0
```

Here, **Length** helps perform two important functions. First, it is used to confirm that the target array is large enough to hold the contents of the source array. Second, it provides the termination condition of the **for** loop that performs the reverse copy. Of course, in this simple example, the size of the arrays is easily known, but this same approach can be applied to a wide range of more challenging situations.

Using Length with Jagged Arrays

A special case occurs when **Length** is used with jagged arrays. In this situation, it is possible to obtain the length of each individual array. For example, consider the following program, which simulates the CPU activity on a network with four nodes:

```
// Demonstrate Length with jagged arrays.

using System;

class Jagged {
  public static void Main() {
    int[ ][ ] network_nodes = new int[4][ ];
    network_nodes[0] = new int[3];
    network_nodes[1] = new int[7];
    network_nodes[2] = new int[2];
    network_nodes[3] = new int[5];

    int i, j;

    // fabricate some fake CPU usage data
    for(i=0; i < network_nodes.Length; i++)
      for(j=0; j < network_nodes[i].Length; j++)
        network_nodes[i][j] = i * j + 70;
    Console.WriteLine("Total number of network nodes: " +
                  network_nodes.Length + "\n");

    for(i=0; i < network_nodes.Length; i++) {
      for(j=0; j < network_nodes[i].Length; j++) {
        Console.Write("CPU usage at node " + i +
                  " CPU " + j + ": ");
        Console.Write(network_nodes[i][j] + "% ");
        Console.WriteLine();
      }
      Console.WriteLine();
    }
  }
}
```

The output is shown here:

```
Total number of network nodes: 4

CPU usage at node 0 CPU 0: 70%
CPU usage at node 0 CPU 1: 70%
CPU usage at node 0 CPU 2: 70%

CPU usage at node 1 CPU 0: 70%
CPU usage at node 1 CPU 1: 71%
CPU usage at node 1 CPU 2: 72%
CPU usage at node 1 CPU 3: 73%
CPU usage at node 1 CPU 4: 74%
CPU usage at node 1 CPU 5: 75%
CPU usage at node 1 CPU 6: 76%

CPU usage at node 2 CPU 0: 70%
CPU usage at node 2 CPU 1: 72%


CPU usage at node 3 CPU 0: 70%
CPU usage at node 3 CPU 1: 73%
CPU usage at node 3 CPU 2: 76%
CPU usage at node 3 CPU 3: 79%
CPU usage at node 3 CPU 4: 82%
```

Pay special attention to the way **Length** is used on the jagged array **network_nodes**. Recall, a two-dimensional jagged array is an array of arrays. Thus, when the expression

```
network_nodes.Length
```

is used, it obtains the number of *arrays* stored in **network_nodes**, which is 4 in this case. To obtain the length of any individual array in the jagged array, you will use an expression such as this:

```
network_nodes[0].Length
```

which, in this case, obtains the length of the first arrays

6.6 CLASSES AND OBJECTS:-

The class is the foundation of C# because it defines the nature of an object. Furthermore, the class forms the basis for object-oriented programming. Within a class are defined both code and data. Because classes and objects are fundamental to C#, they constitute a large topic, which spans several chapters. This chapter begins the discussion by covering their main features.

Class Fundamentals

Since all C# program activity occurs within a class, we have been using classes since the start of this book. Of course, only extremely simple classes have been used, and we have not taken advantage of the majority of their features. Classes are substantially more powerful than the limited ones presented so far.

Let's begin by reviewing the basics. A class is a template that defines the form of an object. It specifies both the data and the code that will operate on that data. C# uses a class specification to construct *objects.* Objects are *instances* of a class. Thus, a class is essentially a set of plans that specify how to build an object. It is important to be clear on one issue: A class is a logical abstraction. It is not until an object of that class has been created that a physical representation of that class exists in memory.

The General Form of a Class

When you define a class, you declare the data that it contains and the code that operates on it. While very simple classes might contain only code or only data, most real-world classes contain both.

In general terms, data is contained in *data members* defined by the class, and code is contained in *function members*. It is important to state at the outset that C# defines several specific flavors of data and function members. For example, data members (also called *fields*) include instance variables and static variables. Function members include methods, constructors, destructors, indexers, events, operators, and properties. For now, we will limit our discussion of the class to its essential elements: instance variables and methods. Later in this chapter, constructors and destructors are discussed. The other types of members are described in later chapters.

A class is created by use of the keyword **class**. Here is the general form of a simple **class** definition that contains only instance variables and methods:

```
class classname{
  // declare instance variables
  access type var1;
  access type var2;
  // ...
  access type varN;

  // declare methods
  access ret-type method1(parameters) {
      // body of method
  }
  access ret-type method2(parameters) {
      // body of method
  }
      // ...
  access ret-type methodN(parameters) {
      // body of method
  }
}
```

Notice that each variable and method declaration is preceded with *access.* Here, *access* is an access specifier, such as **public**, which specifies how the member can be accessed. Class members can be private to a class or more accessible. The access specifier determines what type of access is allowed. The access specifier is optional and if absent, then the member is private to the class. Members with private access can be used only by other members of their class. For the examples in this chapter, all members will be specified as **public**, which means that they can be used by all other code—even code defined outside the class.

Note *In addition to an access specifier, the declaration of a class member can also contain one or more type modifiers. These modifiers are discussed later in this book.*

Although there is no syntactic rule that enforces it, a well-designed class should define one and only one logical entity. For example, a class that stores names and telephone numbers will not normally also store information about the stock market, average rainfall, sunspot cycles, or other unrelated information. The point here is that a well-designed class groups logically connected information. Putting unrelated information into the same class will quickly destructure your code.

Up to this point, the classes that we have been using have only had one method: **Main( )**. However, notice that the general form of a class does not specify a **Main( )** method. A **Main( )** method is required only if that class is the starting point for your program.

Defining a Class

To illustrate classes, we will be evolving a class that encapsulates information about buildings, such as houses, stores, offices, and so on. This class is called **Building**, and it will store three items of information about a building: the number of floors, the total area, and the number of occupants.

The first version of **Building** is shown here. It defines three instance variables: **floors**, **area**, and **occupants**. Notice that **Building** does not contain any methods. Thus, it is currently a data-only class. (Subsequent sections will add methods to it.)

```
class Building {
  public int floors;    // number of floors
  public int area;      // total square footage of building
  public int occupants; // number of occupants
}
```

The instance variables defined by **Building** illustrate the way that instance variables are declared in general. The general form for declaring an instance variable is shown here:

access type var-name;

Here, *access* specifies the access, *type* specifies the type of variable, and *var-name* is the variable's name. Thus, aside from the access specifier, you declare an instance variable in the same way that you declare local variables. For **Building**, the variables are preceded by the **public** access modifier. As explained, this allows them to be accessed by code outside of **Building**.

A **class** definition creates a new data type. In this case, the new data type is called **Building**. You will use this name to declare objects of type **Building**. Remember that a **class** declaration is only a type description; it does not create an actual object. Thus, the preceding code does not cause any objects of type **Building** to come into existence.

To actually create a **Building** object, you will use a statement like the following:

```
Building house = new Building(); // create an object of type building
```

After this statement executes, **house** will be an instance of **Building**. Thus, it will have "physical" reality. For the moment, don't worry about the details of this statement.

Each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class. Thus, every **Building** object will contain its own copies of the instance variables **floors**, **area**, and **occupants**. To access these variables, you will use the *dot* (**.**) operator. The dot operator links the name of an object with the name of a member. The general form of the dot operator is shown here:

object.member

Thus, the object is specified on the left, and the member is put on the right. For example, to assign the **floors** variable of **house** the value 2, use the following statement:

```
house.floors = 2;
```

In general, you can use the dot operator to access both instance variables and methods.

Here is a complete program that uses the **Building** class:

```
// A program that uses the Building class.

using System;

class Building {
  public int floors;    // number of floors
  public int area;      // total square footage of building
  public int occupants; // number of occupants
}

// This class declares an object of type Building.
class BuildingDemo {
  public static void Main( ) {
    Building house = new Building(); // create a Building object
    int areaPP; // area per person

    // assign values to fields in house
    house.occupants = 4;
    house.area = 2500;
    house.floors = 2;

    // compute the area per person
    areaPP = house.area / house.occupants;

    Console.WriteLine("house has:\n  " +
                  house.floors + " floors\n  " +
                  house.occupants + " occupants\n  " +
                  house.area + " total area\n  " +
                  areaPP + " area per person");
  }
}
```

This program consists of two classes: **Building** and **BuildingDemo**. Inside **BuildingDemo**, the **Main( )** method creates an instance of **Building** called **house**. Then the code within **Main( )** accesses the instance variables associated with **house**, assigning them values and using those values. It is important to understand that **Building** and **BuildingDemo** are two separate classes. The only relationship they have to each other is that one class creates an instance of the other. Although they are separate classes, code inside **BuildingDemo** can access the members of **Building** because they are declared **public**. If they had not been given the **public** access specifier, their access would have been limited to the **Building** class, and **BuildingDemo** would not have been able to use them.

Assume that you call the preceding file **UseBuilding.cs**. Compiling this program creates a file called **UseBuilding.exe**. Both the **Building** and **BuildingDemo** classes are automatically part of the executable file. The program displays the following output:

```
house has:
  2 floors
  4 occupants
  2500 total area
  625 area per person
```

Actually, it is not necessary for the **Building** and the **BuildingDemo** class to be in the same source file. You could put each class in its own file, called **Building.cs** and **BuildingDemo.cs**, for example. Just tell the C# compiler to compile both files and link them together. For example, you could use this command line to compile the program if you split it into two pieces as just described:

```
csc Building.cs BuildingDemo.cs
```

If you are using the Visual Studio IDE, you will need to add both files to your project and then build.

Before moving on, let's review a fundamental principle: each object has its own copies of the instance variables defined by its class. Thus, the contents of the variables in one object can differ from the contents of the variables in another. There is no connection between the two objects except for the fact that they are both objects of the same type. For example, if you have two **Building** objects, each has its own copy of **floors**, **area**, and **occupants**, and the contents of these can differ between the two objects. The following program demonstrates this fact:

```
// This program creates two Building objects.

using System;

class Building {
  public int floors;    // number of floors
  public int area;      // total square footage of building
  public int occupants; // number of occupants
}

// This class declares two objects of type Building.
class BuildingDemo {
  public static void Main( ) {
    Building house = new Building();
    Building office = new Building();

    int areaPP; // area per person

    // assign values to fields in house
    house.occupants = 4;
    house.area = 2500;
    house.floors = 2;

    // assign values to fields in office
    office.occupants = 25;
    office.area = 4200;
    office.floors = 3;

    // compute the area per person in house
    areaPP = house.area / house.occupants;

    Console.WriteLine("house has:\n  " +
                      house.floors + " floors\n  " +
                      house.occupants + " occupants\n  " +
                      house.area + " total area\n  " +
                           areaPP + " area per person");

    Console.WriteLine( );

    // compute the area per person in office
    areaPP = office.area / office.occupants;

    Console.WriteLine("office has:\n  " +
                      office.floors + " floors\n  " +
                      office.occupants + " occupants\n  " +
                      office.area + " total area\n  " +
                      areaPP + " area per person");
  }
}
```

The output produced by this program is shown here:

```
house has:
  2 floors
  4 occupants
  2500 total area
  625 area per person

office has:
  3 floors
  25 occupants
  4200 total area
  168 area per person
```

How Objects are Created

In the preceding programs, the following line was used to declare an object of type **Building**:

```
Building house = new Building();
```

This declaration performs two functions. First, it declares a variable called **house** of the class type **Building**. This variable does not define an object. Instead, it is simply a variable that can *refer to* an object. Second, the declaration creates an actual, physical copy of the object and assigns to **house** a reference to that object. This is done by using the **new** operator. Thus, after the line executes, **house** refers to an object of type **Building**.

The **new** operator dynamically allocates (that is, allocates at runtime) memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by **new**. This reference is then stored in a variable. Thus, in C#, all class objects must be dynamically allocated.

The two steps combined in the preceding statement can be rewritten like this to show each step individually:

```
Building house; // declare reference to object
house = new Building(); // allocate a Building object
```

The first line declares **house** as a reference to an object of type **Building**. Thus, **house** is a variable that can refer to an object, but it is not an object, itself. The next line creates a new **Building** object and assigns a reference to it to **house**. Now, **house** is linked with an object.

The fact that class objects are accessed through a reference explains why classes are called *reference types.* The key difference between value types and reference types is what a variable of each type means. For a variable of a value type, the variable, itself, contains the value. For example, given

```
int x;
x = 10;
```

x contains the value 10 because **x** is a variable of type **int**, which is a value type. However, in the case of

```
Building house = new Building();
```

house does not, itself, contain the object. Instead, it contains a reference to the object.

6.7 EXCEPTION HANDLING:-

In C#, exceptions are represented by classes. All exception classes must be derived from the built-in exception class **Exception**, which is part of the **System** namespace. Thus, all exceptions are subclasses of **Exception**.

From **Exception** are derived **SystemException** and **ApplicationException**. These support the two general categories of exceptions defined by C#: those generated by the C# runtime system (that is, the CLR) and those generated by application programs. Neither **SystemException** nor **ApplicationException** adds anything to **Exception**. They simply define the tops of two different exception hierarchies.

C# defines several built-in exceptions that are derived from **SystemException**. For example, when a division-by-zero is attempted, a **DivideByZeroException** is generated. As you will see later in this chapter, you can create your own exception classes by deriving them from **ApplicationException**.

Exception Handling Fundamentals

C# exception handling is managed via four keywords: **try**, **catch**, **throw**, and **finally**. They form an interrelated subsystem in which the use of one implies the use of another. Throughout the course of this chapter, each keyword is examined in detail. However, it is useful at the outset to have a general understanding of the role each plays in exception handling. Briefly, here is how they work.

Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the **try** block, it is *thrown.* Your code can catch this exception using **catch** and handle it in some rational manner. System-generated exceptions are automatically thrown by the C# runtime system. To manually throw an exception, use the keyword **throw**. Any code that absolutely must be executed upon exiting from a **try** block is put in a **finally** block.

Using try and catch

At the core of exception handling are **try** and **catch**. These keywords work together; you can't have a **catch** without a **try**. Here is the general form of the **try**/**catch** exception handling blocks:

```
try {
  // block of code to monitor for errors
}

catch (ExcepType1 exOb) {
  // handler for ExcepType1
}

catch (ExcepType2 exOb) {
  // handler for ExcepType2
}
.
.
.
```

Here, *ExcepType* is the type of exception that has occurred. When an exception is thrown, it is caught by its corresponding **catch** statement, which then processes the exception. As the general form shows, there can be more than one **catch** statement associated with a **try**. The type of the exception determines which **catch** statement is executed. That is, if the exception type specified by a **catch** statement matches that of the exception, then that **catch** statement is executed (and all others are bypassed). When an exception is caught, *exOb* will receive its value.

Actually, specifying *exOb* is optional. If the exception handler does not need access to the exception object (as is often the case), there is no need to specify *exOb.* For this reason, many of the examples in this chapter will not specify *exOb.*

Here is an important point: If no exception is thrown, then a **try** block ends normally, and all of its **catch** statements are bypassed. Execution resumes with the first statement following the last **catch**. Thus, **catch** statements are executed only if an exception is thrown.

A Simple Exception Example

Here is a simple example that illustrates how to watch for and catch an exception. As you know, it is an error to attempt to index an array beyond its boundaries. When this occurs, the C# runtime system throws an **IndexOutOfRangeException**, which is a standard exception defined by C#. The following program purposely generates such an exception and then catches it:

```
// Demonstrate exception handling.

using System;

class ExcDemo1 {
  public static void Main() {
    int[ ] nums = new int[4];

    try {
      Console.WriteLine("Before exception is generated.");

      // Generate an index out-of-bounds exception.
      for(int i=0; i < 10; i++) {
        nums[i] = i;
        Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
      }

      Console.WriteLine("this won't be displayed");
    }
    catch (IndexOutOfRangeException) {
      // catch the exception
      Console.WriteLine("Index out-of-bounds!");
    }
    Console.WriteLine("After catch statement.");
  }
}
```

This program displays the following output:

```
Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3
Index out-of-bounds!
After catch statement.
```

Notice that **nums** is an **int** array of four elements. However, the **for** loop tries to index **nums** from 0 to 9, which causes an **IndexOutOfRangeException** to occur when an index value of 4 is tried.

Although quite short, the preceding program illustrates several key points about exception handling. First, the code that you want to monitor for errors is contained within a **try** block. Second, when an exception occurs (in this case, because of the attempt to index **nums** beyond its bounds inside the **for** loop), the exception is thrown out of the **try** block and caught by the **catch** statement. At this point, control passes to the **catch**, and the **try** block is terminated. Therefore, **catch** is *not* called. Rather, program execution is transferred to it. Thus, the **WriteLine( )** statement following the out-of-bounds index will never execute. After the **catch** statement executes, program control continues with the statements following the **catch**. It is the job of your exception handler to remedy the problem that caused the exception so that program execution can continue normally.

Notice that no parameter is specified in the **catch** clause. As mentioned, a parameter is needed only when access to the exception object is required. In some cases, the value of the exception object can be used by the exception handler to obtain additional information about the error, but in many cases it is sufficient to simply know that an exception occurred. Thus, it is not unusual for the **catch** parameter to be absent in the exception handler, as is the case in the preceding program.

As explained, if no exception is thrown by a **try** block, no **catch** statements will be executed, and program control resumes after the **catch** statement. To confirm this, in the preceding program, change the **for** loop from

```
for(int i=0; i < 10; i++) {
```

to

```
for(int i=0; i < nums.Length; i++) {
```

Now, the loop does not overrun **nums** boundary. Thus, no exception is generated, and the **catch** block is not executed.

A Second Exception Example

It is important to understand that all code executed within a **try** block is monitored for exceptions. This includes exceptions that might be generated by a method called from within the **try** block. An exception thrown by a method called from within a **try** block can be caught by that **try** block, assuming, of course, that the method itself did not catch the exception.

For example, consider the following program. **Main( )** establishes a **try** block from which the method **genException( )** is called. Inside **genException( )**, an **IndexOutOfRangeException** is generated. This exception is not caught by **genException( )**. However, since **genException( )** was called from within a **try** block in **Main( )**, the exception is caught by the **catch** statement associated with that **try**.

```
/* An exception can be generated by one
   method and caught by another. */

using System;

class ExcTest {
  // Generate an exception.
  public static void genException() {
    int[ ] nums = new int[4];

    Console.WriteLine("Before exception is generated.");

    // Generate an index out-of-bounds exception.
    for(int i=0; i < 10; i++) {
      nums[i] = i;
      Console.WriteLine("nums[{0}]: {1}", i, nums[i]);
    }
    Console.WriteLine("this won't be displayed");
  }
}

class ExcDemo2 {
  public static void Main() {

    try {
      ExcTest.genException();
    }
    catch (IndexOutOfRangeException) {
      // catch the exception
      Console.WriteLine("Index out-of-bounds!");
    }
    Console.WriteLine("After catch statement.");
  }
}
```

This program produces the following output, which is the same as that produced by the first version of the program shown earlier:

```
Before exception is generated.
nums[0]: 0
nums[1]: 1
nums[2]: 2
nums[3]: 3
Index out-of-bounds!
After catch statement.
```

As explained, since **genException( )** is called from within a **try** block, the exception that it generates (and does not catch) is caught by the **catch** in **Main( )**. Understand, however, that if **genException( )** had caught the exception, then it would never have been passed back to **Main( )**.

6.8 GENERICS:-

Generics are a new feature in version 2.0 of the C# language and the common language runtime (CLR). Generics introduce to the .NET Framework the concept of type parameters, which make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared and instantiated by client code. For example, by using a generic type parameter T you can write a single class that other client code can use without incurring the cost or risk of runtime casts or boxing operations, as shown here:

```
// Declare the generic class

public class GenericList<T>
{
   void Add(T input) { }
}
class TestGenericList
{
   private class ExampleClass { }
   static void Main()
   {
      // Declare a list of type int
      GenericList<int> list1 = new GenericList<int>();

      // Declare a list of type string
      GenericList<string> list2 = new GenericList<string>();

      // Declare a list of type ExampleClass
      GenericList<ExampleClass> list3 = new
GenericList<ExampleClass>();
   }
}
```

Generics Overview

- Use generic types to maximize code reuse, type safety, and performance.
- The most common use of generics is to create collection classes.

- The .NET Framework class library contains several new generic collection classes in the System.Collections.Generic namespace. These should be used whenever possible in place of classes such as ArrayList in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events and delegates.
- Generic classes may be constrained to enable access to methods on particular data types.
- Information on the types used in a generic data type may be obtained at run-time by means of reflection

6.9 FILE HANDLING IN C#

File handling is an unmanaged resource in your application system. It is outside your application domain (unmanaged resource). It is not managed by CLR.

Data is stored in two ways, persistent and non-persistent manner.

When you open a file for reading or writing, it becomes stream.

Stream: Stream is a sequence of bytes traveling from a source to a destination over a communication path.

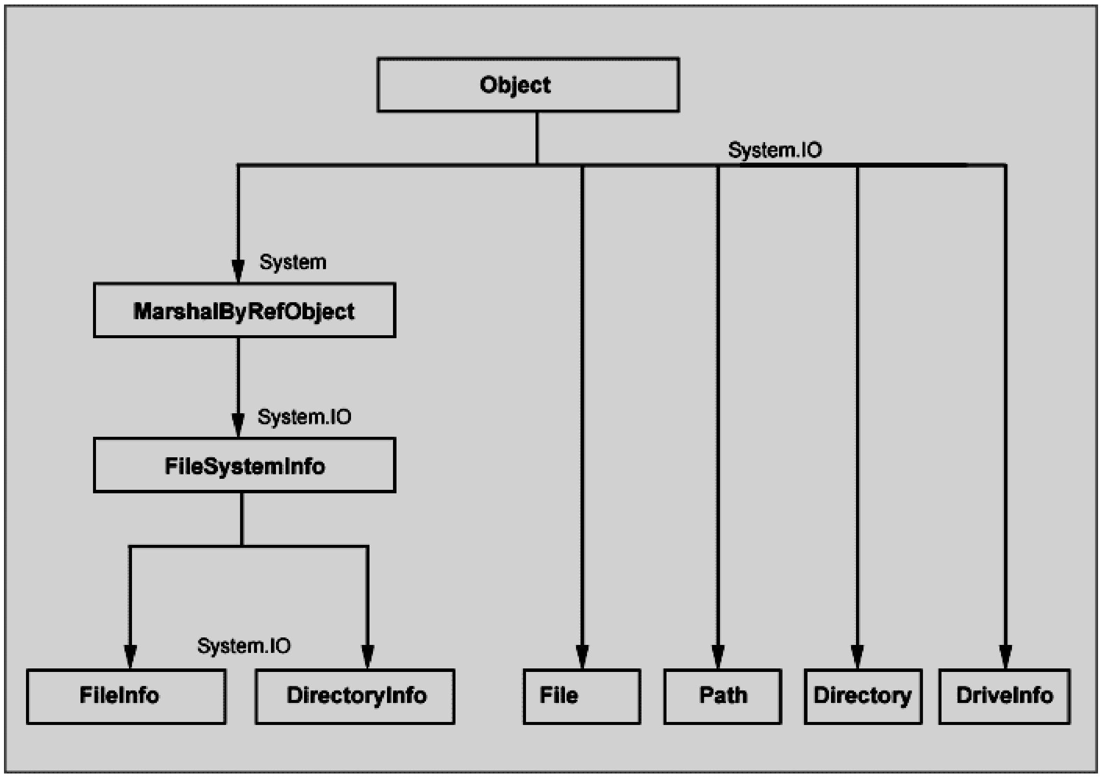
The two basic streams are input and output streams. Input stream is used to read and output stream is used to write.

The **System.IO** namespace includes various classes for file handling.

The parent class of file processing is stream. Stream is an abstract class, which is used as the parent of the classes that actually implement the necessary operations.

The primary support of a file as an object is provided by a .NET Framework class called File. This static class is equipped with various types of (static) methods to create, save, open, copy, move, delete, or check the existence of a file.

Diagram to represent file-handling class hierarchy



Note: FileIno, DirectoryInfo and DriveInfo classes have instance methods. File, Directory, Path classes have static methods.

The following table describes some commonly used classes in the System.IO namespace.

FileStream	It is used to read from and write to any location within a file
BinaryReader	It is used to read primitive data types from a binary stream
BinaryWriter	It is used to write primitive data types in binary format
StreamReader	It is used to read characters from a byte Stream
StreamWriter	It is used to write characters to a stream.
StringReader	It is used to read from a string buffer
StringWriter	It is used to write into a string buffer
DirectoryInfo	It is used to perform operations on directories
FileInfo	It is used to perform operations on files

Reading and writing in the text file
StreamWriter Class

The StreamWriter class in inherited from the abstract class TextWriter. The TextWriter class represents a writer, which can write a series of characters.

The following table describes some of the methods used by StreamWriter class.

Methods	***Description***
Close	Closes the current StreamWriter object and the underlying stream
Flush	Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream
Write	Writes to the stream
WriteLine	Writes data specified by the overloaded parameters, followed by end of line

Program to write user input to a file using StreamWriter Class

```
using System;
using System.Text;
using System.IO;

namespace FileWriting_SW
{
   class Program
   {
      class FileWrite
      {
         public void WriteData()
         {
               FileStream fs = new FileStream("c:\\test.txt", FileMode.Append, FileAccess.Write);
            StreamWriter sw = new StreamWriter(fs);
            Console.WriteLine("Enter the text which you want to write to the file");
            string str = Console.ReadLine();
            sw.WriteLine(str);
            sw.Flush();
            sw.Close();
            fs.Close();
         }
      }
      static void Main(string[] args)
```

```
        {
            FileWrite wr = new FileWrite();
            wr.WriteData();
        }
    }
}
```

StreamReader Class

The StreamReader class is inherited from the abstract class TextReader. The TextReader class represents a reader, which can read series of characters.

The following table describes some methods of the StreamReader class.

Methods	***Description***
Close	Closes the object of StreamReader class and the underlying stream, and release any system resources associated with the reader
Peek	Returns the next available character but doesn't consume it
Read	Reads the next character or the next set of characters from the stream
ReadLine	Reads a line of characters from the current stream and returns data as a string
Seek	Allows the read/write position to be moved to any position with the file

Program to read from a file using StreamReader Class

```
using System;
using System.IO;

namespace FileReading_SR
{
    class Program
    {
        class FileRead
        {
            public void ReadData()
            {
                FileStream fs = new FileStream("c:\\test.txt",
FileMode.Open, FileAccess.Read);
                StreamReader sr = new StreamReader(fs);
                Console.WriteLine("Program to show content of test file");
                sr.BaseStream.Seek(0, SeekOrigin.Begin);
                string str = sr.ReadLine();
                while (str != null)
                {
                    Console.WriteLine(str);
                    str = sr.ReadLine();
                }
                Console.ReadLine();
                sr.Close();
                fs.Close();
            }
        }
        static void Main(string[] args)
        {
            FileRead wr = new FileRead();
            wr.ReadData();

        }
    }
}
```

6.10 INHERITANCE AND POLYMORPHISM

Inheritance

One of the key concepts of Object Oriented Programming is nothing but inheritance. By using the concept of inheritance, it is possible to create a new class from an existing one and add new features to it. Thus inheritance provides a mechanism for class level re usability. The new programming language C# also supports inheritance. The syntax of inheritance is very simple and straightforward.

```
class Base
{
}
class Derived : Base
{
}
```

The operator ':'is used to indicate that a class is inherited from another class. Remember that in C#, a derived class can't be more accessible than it's base class. That means that it is not possible to declare a derived class as public, if it inherits from a private class. For example the following code will generate a compile time error.

```
class Base
{
}
```

```
public class Derived : Base
{
}
```

In the above case the Base class is private. We try to inherit a public class from a private class.

Let us see a concrete example.

In this case Derived class inherits public members of the Base class x,y and Method().The objects of the Derived class can access these inherited members along with its own member z.

```
using System;
class Base
{
public int x = 10;
public int y = 20;
public void Method()
{
Console.WriteLine("Base Method");
}
}
class Derived : Base
{
public int z = 30;
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
Console.WriteLine("{0},{1},{2}",d1.x,d1.y,d1.z); // displays 10,20,30
d1.Method();// displays 'Base Method'
}
}
```

Inheritance & Access Modifiers

A derived class inherits everything from the base class except constructors and destructors. The public members of the Base class becomes the public members of the Derived class also. Similarly the protected members of the base class become protected members of the derived class and internal member becomes internal members of the derived class. Even the private members of the base class are inherited to the derived class, even though derived class can't access them.

Inheritance & Data Members

We know all base class data members are inherited to the derived, but their accessibility remains unchanged in the derived class. For example in the program given below

```
using System;
class Base
{
public int x = 10;
public int y = 20;
}
class Derived : Base
{
public int z = 30;
public void Sum()
{
int sum = x+y+z;
Console.WriteLine(sum);
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.Sum();// displays '60'
}
}
```

Here class Derived have total three data members, two of them are inherited from the Base class.

In C#, even it is possible to declare a data member with the same name in the derived class as shown below. In this case, we are actually hiding a base class data member inside the Derived class. Remember that, still the Derived class can access the base class data member by using the keyword base.

```
using System;
class Base
{
public int x = 10;
public int y = 20;
}
class Derived : Base
{
public int x = 30;
public void Sum()
{
```

```
int sum = base.x+y+x;
Console.WriteLine(sum);
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.Sum();// displays '60'
}
}
```

But when we compile the above program, the compiler will show a warning, since we try to hide a Base class data member inside the Derived class. By using the keyword new along with the data member declaration inside the Derived class, it is possible to suppress this compiler warning. The keyword new tells the compiler that we are trying to explicitly hiding the Base class data member inside the Derived class. Remember that we are not changing the value of the Base class data member here. Instead we are just hiding or shadowing them inside the Derived class. However the Derived class can access the base class data member by using the base operator.

```
using System;
class Base
{
public int x = 10;
public int y = 20;
}
class Derived : Base
{
public new int x = 30;
public void Sum()
{
int sum = base.x+y+x;
Console.WriteLine(sum);
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.Sum();// displays '60'
}
}
```

Inheritance & Member Functions

A derived class member function can call the base class member function by using the base operator. It is possible to hide the implementation of a Base class member function inside a Derived class by using the new operator. When we declare a method in the Derived class with exactly same name and signature of a Base class method, it is known as 'method hiding'. But during the compilation time, the compiler will generate a warning. But during run-time the objects of the Derived class will always call the Derived class version of the method. By declaring the derived class method as new, it is possible to suppress the compiler warning.

```
using System;
class Base
{
public void Method()
{
Console.WriteLine("Base Method");
}
}
class Derived : Base
{
public void Method()
{
Console.WriteLine("Derived Method");
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.Method(); // displays "Derived Method'
}
}
```

Uses of new and base operators are given in the following program.

```
using System;
class Base
{
public void Method()
{
Console.WriteLine("Base Method");
}
}
class Derived : Base
{
```

```
public new void Method()
{
Console.WriteLine("Derived Method");
base.Method();
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.Method(); // displays 'Derived Method' followed by 'Base Method'
}
}
```

Inheritance & Constructors

The constructors and destructors are not inherited to a Derived class from a Base class. However when we create an object of the Derived class, the derived class constructor implicitly call the Base class default constructor. The following program shows this.

```
using System;
class Base
{
public Base()
{
Console.WriteLine("Base class default constructor");
}
}
class Derived : Base
{
}
class MyClient
{
public static void Main()
{
Derived d1 =new Derived();// Displays 'Base class default
constructor'
}
}
```

Remember that the Derived class constructor can call only the default constructor of Base class explicitly. But they can call any Base class constructor explicitly by using the keyword base.

```
// Inheritance : constructor chaining

using System;
class Base
{
public Base()
{
Console.WriteLine("Base constructor1");
}
public Base(int x)
{
Console.WriteLine("Base constructor2");
}
}
class Derived : Base
{
public Derived() : base(10)// implicitly call the Base(int x)
{
Console.WriteLine("Derived constructor");
}
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();// Displays 'Base constructor2 followed
by 'Derived Constructor''
}
}
```

Note that by using base() the constructors can be chained in an inheritance hierarchy.

In this article I will explain polymorphism. What are different types of polymorphism? The use of method overloading, virtual method, method hiding, method shadowing and method overriding.

Inheritance is one of the primary concepts of object-oriented programming. It allows you to reuse existing code. Through effective employment of reuse, you can save time in your programming. Inheritance is transitive in nature.

Types of polymorphism

There are two types of polymorphism:

1. Compile time polymorphism
2. Run time polymorphism.

Compile Time Polymorphism

Compile time polymorphism is method and operators overloading. It is also called early binding.

In method overloading method performs the different task at the different input parameters.

Runtime Time Polymorphism

Runtime time polymorphism is done using inheritance and virtual functions. Method overriding is called runtime polymorphism. It is also called late binding.

When overriding a method, you change the behavior of the method for the derived class. Overloading a method simply involves having another method with the same prototype.

Note: C# supports single class inheritance only. Therefore, you can specify only one base class to inherit from. However, it does allow multiple interface inheritance.

Practical example of Method Overloading (Compile Time Polymorphism)

```
using System;
namespace method_overloading
{
class Program
{
public class Print
{

public void display(string name)
{
Console.WriteLine("Your name is : " + name);
}
public void display(int age, float marks)
{
Console.WriteLine("Your age is : " + age);
Console.WriteLine("Your marks are :" + marks);
}

}

static void Main(string[] args)
{
Print obj = new Print();
obj.display("George");
obj.display(34, 76.50f);
Console.ReadLine();

}
}
}
```

In the code if you observe display method is called two times. Display method will work according to the number of parameters and type of parameters.

Inheritance can be seen in following context:

Virtual Method
Method Hiding
Method Shadowing
Method Overloading

Virtual Method

Virtual means the method can be over-ridden in classes that derive from the base-class with the virtual method in.

You could derive from a class with a virtual method, and re-define the virtual method with **new** instead of the **override** keyword.

Method Hiding

In this process derived class method will hide the method of base. Method hiding is implicit process. (It can be by mistake and will give you warning)

Practical example of Method Hiding

```
using System;
namespace Method_hiding
{
class Program
{
public class BaseClass
{
string name;
public BaseClass(string name)
{
this.name = name;
}
public void display()
{
Console.WriteLine("Base class name is : " + this.name);
}
}
public class DerivedClass : BaseClass
{
string dname;
public DerivedClass(string dname) : base ("First")
{
this.dname = dname;
}
public void display()
{
Console.WriteLine("Derived Class name is : " + this.dname);
}
}
public static void Main(string[] args)
{
BaseClass ob1 = new BaseClass("First");
DerivedClass ob2 = new DerivedClass("Second");
ob1.display();
ob2.display();

Console.ReadLine();
}
}
}
```

Method Shadowing

Method shadowing is an explicit process. Shadowing has existence of both the methods that is of base class as well as derived class.

You can't shadow more than one time (one time inheritance)

Practical example of Method Shadowing

```
using System;
namespace Method_shadowing
{
class Program
{
public class BaseClass
{
string name;
public BaseClass(string name)
{
this.name = name;
}
public void display()
{
Console.WriteLine("Base class name is : " + name);
}
}
public class DerivedClass : BaseClass
{
string derivedName;
public DerivedClass(string derivedName) : base ("First")
{
this.derivedName = derivedName;
}
public new void display()
{
Console.WriteLine("Derviced class new name is : " +
derivedName);
}
}

static void Main(string[] args)
{
BaseClass ob1 = new BaseClass("First");
ob1.display();
DerivedClass ob2 = new DerivedClass("Second");
ob2.display();
Console.ReadLine();
}
}
}
```

You mark derived class method with **new** keyword in method shadowing.

In method shadowing you have both the methods available. You are only changing the functionality prototype remains same. .

You can call the new method and the method defined in the base class. Existence of both base class and derived class method in managed heap.

Method Overriding

Overriding is a way to optimize code. Overriding process overwrites the method of base class and only one method exists in the managed heap.

You can override to N level. (To stop overriding we use sealed method) In overriding we can't change the method prototype but can change its functionality.

Overriding is runtime polymorphism.

Sealed method is used to define overriding level of a virtual method. Sealed keyword is always used with override key word.

Practical example of Method Overriding

```
using System;
namespace method_overriding
{
class Program
{
public class BaseClass
{
string name;
public BaseClass(string name)
{
this.name = name;
}
public virtual void display()
{
Console.WriteLine("Base class method " + this.name);
}
}
public class DerivedClass : BaseClass
{
string derivedName;
public DerivedClass(string derivedName)
: base("First")
{
this.derivedName = derivedName;
}
public override void display()
{
Console.WriteLine("Derviced class method name : " +
derivedName);
}
}
static void Main(string[] args)
{
BaseClass ob1 = new BaseClass("First");
ob1.display();
DerivedClass ob2 = new DerivedClass("Second");
ob2.display();
Console.ReadLine();
}
}
}
```

Note: virtual method only changes functionality but **new** method creates a new method.

6.11 DATABASE PROGRAMMING

Definition

Database programming is a reference to the methods used to establish and then configure a database. The correct configuration of your new or existing database can greatly increase performance, speed, and longevity and the ability to expand.

Database

A database is a collection of data for one or more multiple uses. Integrated data files organized and stored electronically in a uniform file structure that allows data elements to be manipulated, correlated, or extracted to satisfy diverse analytical and reporting needs.

Why use a database?

The main advantage is fast and efficient data retrieval. A database helps you to organize your data in alogical manner. Database management systems are fine-tuned to rapidly retrieve the data you want in the way you want it. Databases also enable you to break data into specific parts. Retrieving data from a database is called querying.

Databases also allow you to set up rules that ensure that data remains consistent when you add, update, or delete data.

Relational Vs. Hierarchical

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

(RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant
- Each Column Has a Unique Name

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retreival time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

Hence C# programmers need to know about relational databases.

- Using Dataset class

A DataSet is an in-memory data store that can hold numerous tables. DataSets only hold data and do not interact with a data source.

Creating a DataSet Object

There isn't anything special about instantiating a DataSet. You just create a new instance, just like any other object:

```
DataSet dsCustomers = new DataSet();
```

The DataSet constructor doesn't require parameters. However there is one overload that accepts a string for the name of the DataSet, which is used if you were to serialize the data to XML.

Since that isn't a requirement for this example, I left it out. Right now, the DataSet is empty and you need a SqlDataAdapter to load it.

Creating A SqlDataAdapter

The SqlDataAdapter holds the SQL commands and connection object for reading and writing data. You initialize it with a SQL select statement and connection object:

```
SqlDataAdapter daCustomers = new SqlDataAdapter(
   "select CustomerID, CompanyName from Customers", conn);
```

The code above creates a new SqlDataAdapter, *daCustomers*. The SQL select statement specifies what data will be read into a DataSet. The connection object, *conn*, should have already been instantiated, but not opened. It is the Sql Data Adapter's responsibility to open and close the connection during Fill and Update method calls.

As indicated earlier, the SqlDataAdapter contains all of the commands necessary to interact with the data source. The code showed how to specify the select statment, but didn't show the insert, update, and delete statements. These are added to the SqlDataAdapter after it is instantiated.

There are two ways to add insert, update, and delete commands: via SqlDataAdapter properties or with a SqlCommandBuilder. In this lesson, I'm going to show you the easy way of doing it with the SqlCommandBuilder. In a later lesson, I'll show you how to use the SqlDataAdapter properties, which takes more work but will give you more capabilities than what the SqlCommandBuilder does. Here's how to add commands to the SqlDataAdapter with the SqlCommandBuilder:

```
SqlCommandBuilder cmdBldr = new
SqlCommandBuilder(daCustomers);
```

Notice in the code above that the SqlCommandBuilder is instantiated with a single constructor parameter of the SqlDataAdapter, *daCustomers*, instance. This tells the SqlCommandBuilder what SqlDataAdapter to add commands to. The SqlCommandBuilder will read the SQL select statement (specified when the SqlDataAdapter was instantiated), infer the insert, update, and delete commands, and assign the new commands to the Insert, Update, and Delete properties of the SqlDataAdapter, respectively.

As I mentioned earlier, the SqlCommandBuilder has limitations. It works when you do a simple select statement on a single table. However, when you need a join of two or mor tables or must do a stored procedure, it won't work. I'll describe a work-around for these scenarios in future lessons.

Filling the DataSet

Once you have a DataSet and SqlDataAdapter instances, you need to fill the DataSet. Here's how to do it, by using the Fill method of the SqlDataAdapter:

```
daCustomers.Fill(dsCustomers, "Customers");
```

The *Fill* method, in the code above, takes two parameters: a DataSet and a table name. The DataSet must be instantiated before trying to fill it with data. The second parameter is the name of the table that will be created in the DataSet. You can name the table anything you want. Its purpose is so you can identify the table with a meaningful name later on. Typically, I'll give it the same name as the database table. However, if the SqlDataAdapter's select command contains a join, you'll need to find another meaningful name.

The *Fill* method has an overload that accepts one parameter for the DataSet only. In that case, the table created has a default name of "table1" for the first table. The number will be incremented (table2, table3, ..., tableN) for each table added to the DataSet where the table name was not specified in the Fill method.

Using the DataSet

A DataSet will bind with both ASP.NET and Windows forms DataGrids. Here's an example that assigns the DataSet to a Windows forms DataGrid:

```
dgCustomers.DataSource=dsCustomers;
dgCustomers.DataMember = "Customers";
```

The first thing we do, in the code above, is assign the DataSet to the DataSource property of the DataGrid. This lets the DataGrid know that it has something to bind to, but you will get a '+' sign in the GUI because the DataSet can hold multiple tables and this would allow you to expand each available table. To specify exactly which table to use, set the DataGrid's *DataMember* property to the name of the table. In the example, we set the name to *Customers*, which is the same name used as the second parameter to the SqlDataAdapter Fill method. This is why I like to give the table a name in the *Fill* method, as it makes subsequent code more readable.

Updating Changes

After modifications are made to the data, you'll want to write the changes back to the database. Refer to previous discussion in the Introduction of this article on update guidance. The following code shows how to use the *Update* method of the SqlDataAdapter to push modifications back to the database.

```
daCustomers.Update(dsCustomers, "Customers");
```

The *Update* method, above, is called on the SqlDataAdapter instance that originally filled the *dsCustomers* DataSet. The second parameter to the *Update* method specifies which table, from the DataSet, to update. The table contains a list of records that have been modified and the Insert, Update, and Delete properties of the SqlDataAdapter contain the SQL statements used to make database modifications.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Windows.Forms;

class DisconnectedDataform : Form
{
	private SqlConnection  conn;
	private SqlDataAdapter daCustomers;

	private DataSet  dsCustomers;
	private DataGrid dgCustomers;

	private const string tableName = "Customers";

	// initialize form with DataGrid and Button
	public DisconnectedDataform()
	{
		// fill dataset
		Initdata();

		// set up datagrid
		dgCustomers = new DataGrid();
		dgCustomers.Location = new Point(5, 5);
		dgCustomers.Size = new Size(
			this.ClientRectangle.Size.Width - 10,
			this.ClientRectangle.Height - 50);
```

```
                    dgCustomers.DataSource = dsCustomers;
                    dgCustomers.DataMember = tableName;

                    // create update button
                    Button btnUpdate = new Button();
                    btnUpdate.Text = "Update";
                    btnUpdate.Location = new Point(
                              this.ClientRectangle.Width/2 -
btnUpdate.Width/2,
                              this.ClientRectangle.Height -
(btnUpdate.Height + 10));
                    btnUpdate.Click += new
EventHandler(btnUpdateClicked);

                    // make sure controls appear on form
                    Controls.AddRange(new Control[] { dgCustomers,
btnUpdate });
          }

          // set up ADO.NET objects
          public void Initdata()
          {
                    // instantiate the connection
                    conn = new SqlConnection(

          "Server=(local);DataBase=Northwind;Integrated
Security=SSPI");

                    // 1. instantiate a new DataSet
                    dsCustomers = new DataSet();

                    // 2. init SqlDataAdapter with select command and
connection
                    daCustomers = new SqlDataAdapter(
                    "select CustomerID, CompanyName from
Customers", conn);

                    // 3. fill in insert, update, and delete commands
                    SqlCommandBuilder cmdBldr = new
SqlCommandBuilder(daCustomers);

                    // 4. fill the dataset
                    daCustomers.Fill(dsCustomers, tableName);
          }

          // Update button was clicked
          public void btnUpdateClicked(object sender, EventArgs e)
          {
                    // write changes back to DataBase
```

```
                daCustomers.Update(dsCustomers, tableName);
        }

        // start the Windows form
        static void Main()
        {
                Application.Run(new DisconnectedDataForm());
        }

}
```

Summary

DataSets hold multiple tables and can be kept in memory and reused. The SqlDataAdapter enables you to fill a DataSet and Update changes back to the database. You don't have to worry about opening and closing the SqlConnection because the SqlDataAdapter does it automatically. A SqlCommandBuilder populates insert, update, and delete commands based on the SqlDataAdapter's select statement. Use the *Fill* method of the SqlDataAdapter to fill a DataSet with data. Call the SqlDataAdapter's *Update* method to push changes back to a database.

- **Using Datatable**

Simple DataTable example

First, the DataTable type is probably the most convenient and powerful way to store data in memory. You may have fetched this data from a database, or you may have generated it dynamically. In this example, we get a DataTable with four columns of type int, string and DateTime. This DataTable could then be persisted or displayed.

```
using System;
using System.Data;

class Program
{
   static void Main()
   {
      //
      // Get the DataTable.
      //
      DataTable table = GetTable();
      //
      // Use DataTable here with SQL, etc.
      //
```

```
    }

    /// <summary>
    /// This example method generates a DataTable.
    /// </summary>
    static DataTable GetTable()
    {
        //
        // Here we create a DataTable with four columns.
        //
        DataTable table = new DataTable();
        table.Columns.Add("Dosage", typeof(int));
        table.Columns.Add("Drug", typeof(string));
        table.Columns.Add("Patient", typeof(string));
        table.Columns.Add("Date", typeof(DateTime));

        //
        // Here we add five DataRows.
        //
        table.Rows.Add(25, "Indocin", "David", DateTime.Now);
        table.Rows.Add(50, "Enebrel", "Sam", DateTime.Now);
        table.Rows.Add(10, "Hydralazine", "Christoff", DateTime.Now);
        table.Rows.Add(21, "Combivent", "Janet", DateTime.Now);
        table.Rows.Add(100, "Dilantin", "Melanie", DateTime.Now);
        return table;
    }
}
```

Understanding DataView methods

First, the DataTable you are using stores the physical data, while the **DataView** is only a view of that data. This means that you cannot easily sort a DataTable without using a DataView. Fortunately, DataView offers a convenient Sort string, which you can specify the column to sort with.

DataTable
Where you populate your data, from the user or database.

DataView
Accessed with the DefaultView property on DataTable. DataViews allow you to filter and sort data, not store it. Use DataView for inputting filtered data to your database. It is useful for displaying on a window or web page.

DefaultView
Access this property on your DataTable instance. This is an instance of DataView.

Count

This is an instance property on all DataView instances. You can use this in a for loop on the DataView.

Sort

This is a string property on every DataView. Assign this to a string containing the name of a column. After you specify this, you can loop over the DataView.

Using DataView for sorting

Here we see an example of how you can use **DataView** to sort one of four columns on a DataTable. In your program, the DataTable may be generated from user input or a database, but here we create it programmatically. The GetTable method returns a table with four columns.

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        //
        // Specify the column to sort on.
        //
        DataTable table = GetTable();
        table.DefaultView.Sort = "Weight";

        //
        // Display all records in the view.
        //
        DataView view = table.DefaultView;
        Console.WriteLine("=== Sorted by weight ===");
        for (int i = 0; i < view.Count; i++)
        {
            Console.WriteLine("{0}, {1}, {2}, {3}",
                view[i][0],
                view[i][1],
                view[i][2],
                view[i][3]);
        }

        //
        // Now sort on the Name.
        //
        view.Sort = "Name";
        //
        // Display all records in the view.
        //
        Console.WriteLine("=== Sorted by name ===");
        for (int i = 0; i < view.Count; i++)
        {
            Console.WriteLine("{0}, {1}, {2}, {3}",
                view[i][0],
                view[i][1],
                view[i][2],
                view[i][3]);
        }
    }

    /// <summary>
    /// This example method generates a DataTable.
    /// </summary>
    static DataTable GetTable()
    {
        //
        // Here we create a DataTable with four columns.
        //
        DataTable table = new DataTable();
        table.Columns.Add("Weight", typeof(int));
        table.Columns.Add("Name", typeof(string));
```

```
        table.Columns.Add("Breed", typeof(string));
        table.Columns.Add("Date", typeof(DateTime));

        //
        // Here we add unsorted data to the DataTable and return.
        //
        table.Rows.Add(57, "Koko", "Shar Pei", DateTime.Now);
        table.Rows.Add(130, "Fido", "Bullmastiff", DateTime.Now);
        table.Rows.Add(92, "Alex", "Anatolian Shepherd Dog",
DateTime.Now);
        table.Rows.Add(25, "Charles", "Cavalier King Charles
Spaniel", DateTime.Now);
        table.Rows.Add(7, "Candy", "Yorkshire Terrier",
DateTime.Now);
        return table;
    }
}
```

- **Using Stored procedures**

Why Use Stored Procedures?

There are several advantages of using stored procedures instead of standard SQL. First, stored procedures allow a lot more flexibility offering capabilities such as conditional logic. Second, because stored procedures are stored within the DBMS, bandwidth and execution time are reduced. This is because a single stored procedure can execute a complex set of SQL statements. Third, SQL Server pre-compiles stored procedures such that they execute optimally. Fourth, client developers are abstracted from complex designs. They would simply need to know the stored procedure's name and the type of data it returns.

Creating a Stored Procedure

Enterprise Manager provides an easy way to create stored procedures. First, select the database to create the stored procedure on. Expand the database node, right-click on "Stored Procedures" and select "New Stored Procedure...". You should see the following:

CREATE PROCEDURE [dbo].[GetProducts] AS

Calling a Stored Procedure

A very nice aspect of ADO.NET is that it allows the developer to call a stored procedure in almost the exact same way as a standard SQL statement.

1. Create a new C# Windows Application project.

2. From the Toolbox, drag and drop a DataGrid onto the Form. Resize it as necessary.

3. Double-click on the Form to generate the Form_Load event handler. Before entering any code, add "using System.Data.SqlClient" at the top of the file.

Enter the following code:

```
private void Form1_Load(object sender, System
.EventArgs e)
{
   SqlConnection conn = new SqlConnection("Data
Source=localhost;Database=Northwind;Integrated Security=SSPI");
   SqlCommand command = new SqlCommand("GetProducts",
conn);
   SqlDataAdapter adapter = new SqlDataAdapter(command);
   DataSet ds = new DataSet();
   adapter.Fill(ds, "Products");
   this.dataGrid1.DataSource = ds;
   this.dataGrid1.DataMember = "Products";
}
```

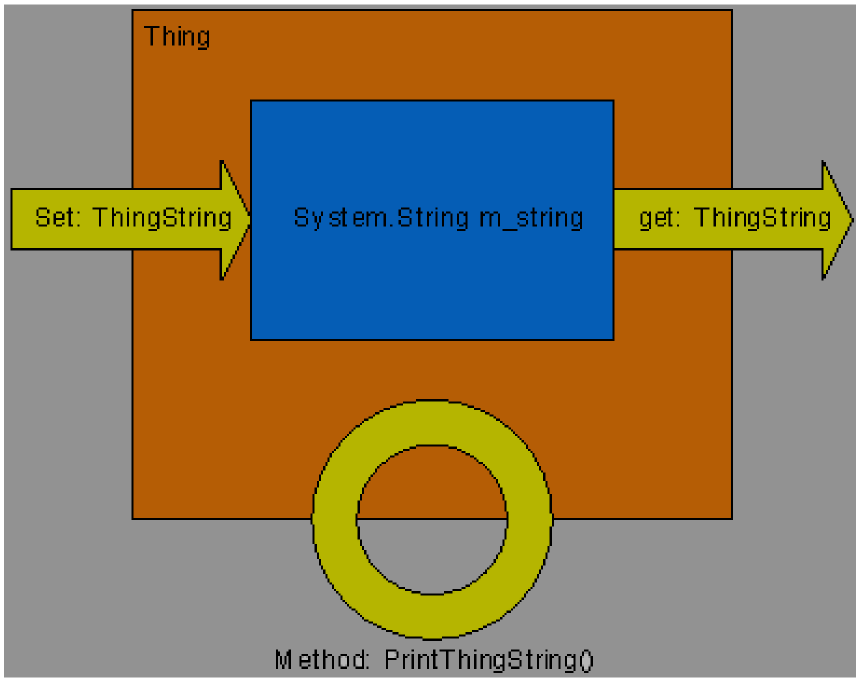
Creating a Basic Wrapper.

These three patterns, proxy, decorator, and adapter are all implemented identically. They are all wrappers at the basic level. The functionality they provide is how we can make the distinction between them. Let's first look at a template for a basic wrapper and then at how to use a wrapper to implement each of these patterns can help us in our quest for the bizarre.

Before we begin to understand how to perform super-feats, we need to understand the basic super-power all C# developers are born with - our secret ability to wrap (not to be confused with rap). Here is how a wrapper works:

Let's say we have a Thing with a ThingString property and a method PrintThingString() as follows:

```
class Thing
{
        private string m_string;
        public string ThingString
        {
                get { return m_string; }
                set { m_string = value; }
        }
        public void PrintThingString()
        {
                Console.WriteLine(m_string);
        }
}
```



Here is how we create a basic ThingWrapper to wrap the Thing. It contains a reference to the Thing being wrapped and has the same signature. The ThingWrapper just passes requests into the wrapped Thing.

```
class ThingWrapper
{
        private Thing m_thing;
        public ThingWrapper(Thing pThing)
        {
                m_thing = pThing;
        }
        public string ThingString
        {
                get { return m_thing.ThingString; }
                set { m_thing.ThingString = value; }
        }
        public void PrintThingString()
        {
                m_thing.PrintThingString();}}
```

Data-bound Controls

Data-bound controls are WinForms controls those can easily bind with data components. Microsoft Visual Studio.NET is a rich IDE for ADO.NET data components. Im going to talk about these controls in a moment. In this article, Im going to talk about three main data-bound controls DataGrid, ListBox, and a ComboBox.

Data-bound controls have properties, which you can set as a data component and theyre ready to present your data in WinForms. DataSource and DisplayMemeber are two important properties.

DataSource property of these controls plays a major role. You can set different kind of data components as datasource property of a control. For example, you can set a DefaultViewManager or a DataView as this property.

```
DataSet ds = new DataSet();
dataGrid1.DataSource = ds.DefaultViewManager;
```

DisplayMember property can be set to a database table field name if you want to bind a particular field to the control.

```
DataSet ds = new DataSet();
// Attach dataset's DefaultView to the datagrid control
DataView dv = ds.Tables["Employees"].DefaultView;
listBox1.DataSource = dv;
listBox1.DisplayMember = "FirstName";
```

ADO.NET Data Components in VS.NET

Microsoft Visual Studio.NET provides a rich set of ADO.NET data components. These components sit between WinForms data-bound controls and the data source and passes data back and forth to the controls. These components are:

Data Connection
DataSet
DataView
Data Adapters
Data Commands

You can create these components in either at design-time or at run-time. Creating these components at design-time is pretty simple task. You just drag these components on a form and set properties and youre all set.

Connection, data adapter, and command components are specific to a data provider and dataview and dataset are common components.

ADO.NET Data Providers

In Microsoft .NET Beta 2, ADO.NET has three types of data providers. Each data provider is designed to work with different types of data sources. All of these data providers provide same classes for a connection, data adapter and command classes to work with and work in similar fashion.

These data providers are:

SQL Data Providers:

SQL data providers are designed to work with SQL Server 7 or later databases. The connection, command and data adapter classes are SqlConnection, SqlCommand, and SqlDataAdapter.

OLE DB Data Providers

Ole-db data providers are designed to work with any OLE-DB data source. You need to have an OLE-DB provider to work with a data source. The connection, command and data adapter classes are OleDbConnection, OleDbCommand, and OleDbDataAdapter.

ODBC Data Providers

ODBC data providers is a recent addition to the .NET SDK. This API doesnt ship with .NET Beta 2. You need to download it separately than .NET SDK. You can download it Microsofts site at ODBC SDK. ODBC providers are designed to work with any ODBC data source. You need to have an ODBC driver to work with a data source. The connection, command and data adapter classes are ODBCConnection, ODBCCommand, and ODBCDataAdapter.

As mentioned earlier, working with all of these data providers is similar accept the class names and data sources. So if you know one of them, you can just replace data source and the class names.

Working with Data Components

There are few simple steps include to work with data components. Just follow these steps one by one.

Step 1: Connect to a data source

First step is to create a connection to the data source. You use a Connection object to connect to a data source. You need to create a connection string and create connection object. Here u MS-Access 2000 is usedas data source and OleDB Data Adapters to work with the data source.

```
// Creating connection and command sting
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\northwind.mdb";
// Create connection object
OleDbConnection conn = new OleDbConnection(conStr);
```

Step 2: Creating a Data Adapter

Now you create a data adapter. A data adapter constructor takes two arguments A SQL
string and a connection object.

```
string sqlStr = "SELECT * FROM Employees";
// Create data adapter object
OleDbDataAdapter da = new OleDbDataAdapter(sqlStr, conn);
```

Step 3: Creating and Filling a DataSet

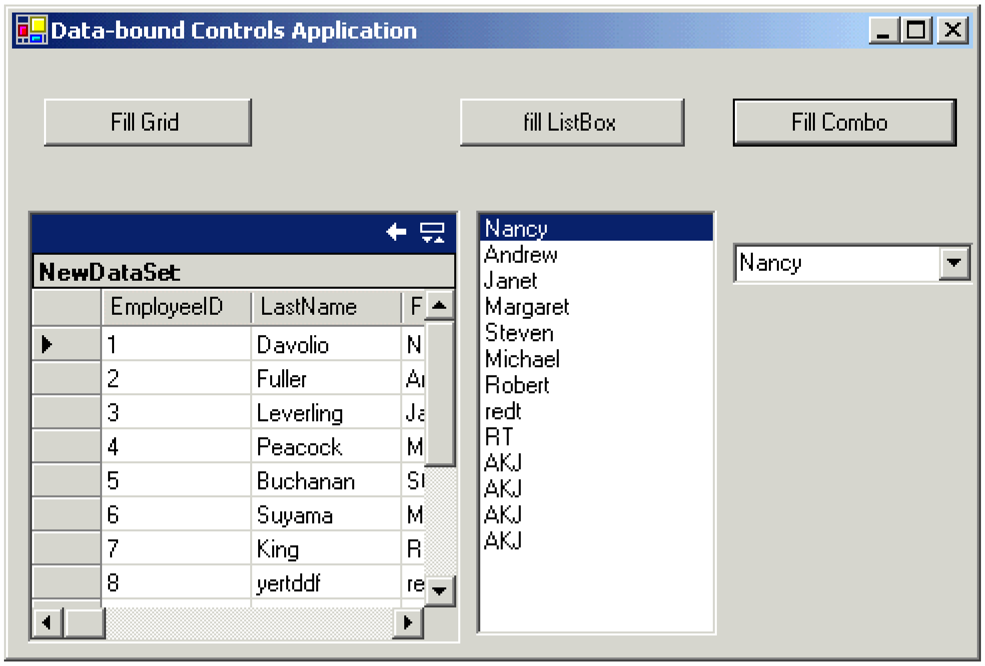
Now next step is to create a dataset and fill it by using data adapters Fill method.

```
// Create a dataset object and fill with data using data adapter's Fill
method
DataSet ds = new DataSet();
da.Fill(ds, "Employees");
```

Step 4: Bind to a data-bound control

The last step is to bind the data set to a data-bound control using above discussed
methods.

```
// Attach dataset's DefaultView to the datagrid control
dataGrid1.DataSource = ds.DefaultViewManager;
```



	EmployeeID	LastName	F
▶	1	Davolio	N
	2	Fuller	A
	3	Leverling	Ja
	4	Peacock	M
	5	Buchanan	St
	6	Suyama	M
	7	King	R
	8	yertddf	re

```
Nancy
Andrew
Janet
Margaret
Steven
Michael
Robert
redt
RT
AKJ
AKJ
AKJ
AKJ
```

This sample application is a Windows application which three controls a DataGrid, a ListBox, and a ComboBox and three buttons Fill DataGrid, Fill ListBox, and Fill ComboBox respectively.

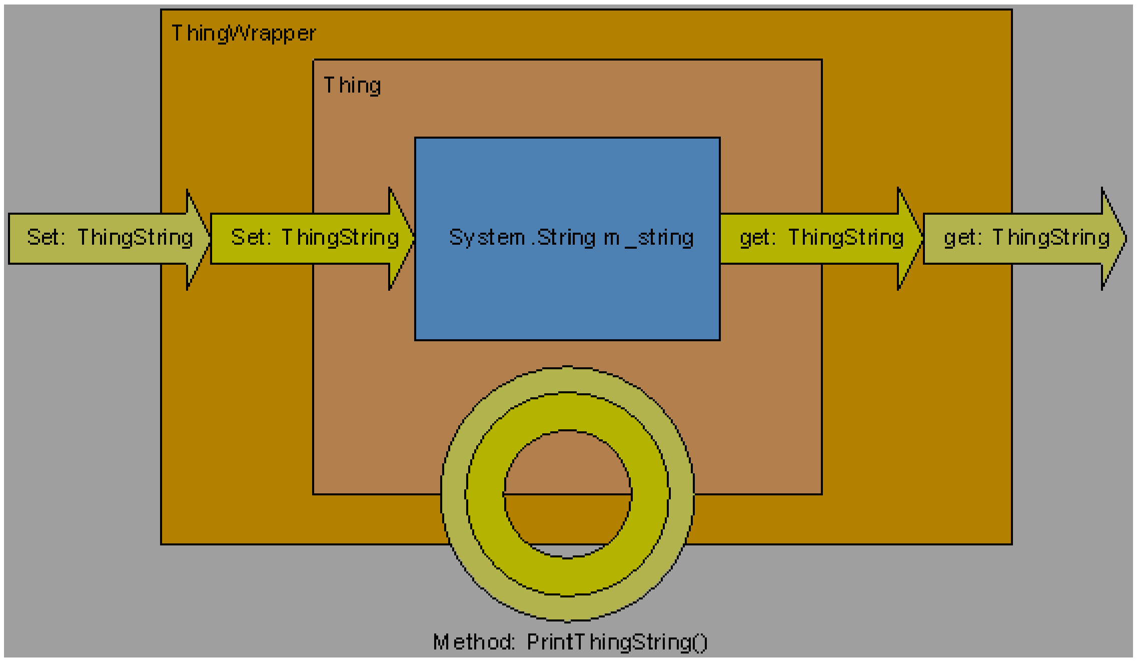
When you click on these buttons, the fill the data from the data source to the control. The code is shown in the below table -

```
private void button1_Click(object sender, System.EventArgs e)
{
// Creating connection and command sting
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\northwind.mdb";
string sqlStr = "SELECT * FROM Employees";
// Create connection object
OleDbConnection conn = new OleDbConnection(conStr);
// Create data adapter object
OleDbDataAdapter da = new OleDbDataAdapter(sqlStr,conn);
// Create a dataset object and fill with data using data adapter's Fill
method
DataSet ds = new DataSet();
da.Fill(ds, "Employees");
// Attach dataset's DefaultView to the datagrid control
dataGrid1.DataSource = ds.DefaultViewManager;
```

```
}
private void button2_Click(object sender, System.EventArgs e)
{
// Creating connection and command sting
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data
source=c:\\northwind.mdb";
string sqlStr = "SELECT * FROM Employees";
// Create connection object
OleDbConnection conn = new OleDbConnection(conStr);
// Create data adapter object
OleDbDataAdapter da = new OleDbDataAdapter(sqlStr,conn);
// Create a dataset object and fill with data using data adapter's Fill
method
DataSet ds = new DataSet();
da.Fill(ds, "Employees");
// Attach dataset's DefaultView to the datagrid control
DataView dv = ds.Tables["Employees"].DefaultView;
listBox1.DataSource = dv;
listBox1.DisplayMember = "FirstName";
}
private void button3_Click(object sender, System.EventArgs e)
{
// Creating connection and command sting
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data a
source=c:\\northwind.mdb";
string sqlStr = "SELECT * FROM Employees";
// Create connection object
OleDbConnection conn = new OleDbConnection(conStr);
// Create data adapter object
OleDbDataAdapter da = new OleDbDataAdapter(sqlStr,conn);
// Create a dataset object and fill with data using data adapter's Fill
method
DataSet ds = new DataSet();
```

```
da.Fill(ds, "Employees");
// Attach dataset's DefaultView to the datagrid control
DataView dv = ds.Tables["Employees"].DefaultView;
comboBox1.DataSource = dv;

comboBox1.DisplayMember = "FirstName";}
```



Because the ThingWrapper has the same signature as the Thing, it is really no different dealing with the Thing or the ThingWrapper. We can work with either the Thing or the ThingWrapper access our internal string m_string, or perform the base functionality which is to print a line to the console.

```
Thing t = new Thing();
t.ThingString = "This is the ThingString";
t.PrintThingString();

ThingWrapper tWrap = new ThingWrapper(t);
tWrap.PrintThingString();
tWrap.ThingString = "This is still the ThingString";
tWrap.PrintThingString();
```

We can wrap any class this way.

Exercise:

1. Comment C# is an object oriented programming.
2. Explain the features of C3# language.
3. Explain the different data types in C#.
4. What is inheritance? Explain how it is implemented.
5. What is virtual function?
6. Explain how polymorphism is implemented in C#.
7. What is ADO.Net?
8. List and exaplin the steps used for establishing the database connectivity with SQL.
9. What is wrapper class?

❖❖❖❖

7

WEB APPLICATIONS IN ASP.NET

ASP.NET Coding Modules, ASP.NET Page directives, Page Events and Page Life Cycle, PostBack and CrossPage Posting, ASP.NET Application Compilation Models, ASP.Net, Server Controls, HTML Controls, Validation Controls, Building Databases.

7.1 ASP.NET Coding Modules:

HTTP Modules

HTTP modules are .NET components that implement the System.Web.IHttpModule interface. These components plug themselves into the ASP.NET request processing pipeline by registering themselves for certain events. Whenever those events occur, ASP.NET invokes the interested HTTP modules so that the modules can play with the request.

An HTTP module is supposed to implement the following methods of the IHttpModule interface:

Method Name	Description
Init	This method allows an HTTP module to register its event handlers to the events in the HttpApplication object.
Dispose	This method gives HTTP module an opportunity to perform any clean up before the object gets garbage collected.

An HTTP module can register for the following events exposed by the System.Web.HttpApplication object.

EVENT NAME	**DESCRIPTION**
AcquireRequestState	This event is raised when ASP.NET runtime is ready to acquire the Session state of the current HTTP request.
AuthenticateRequest	This event is raised when ASP.NET runtime is ready to authenticate the identity of the user.
AuthorizeRequest	This event is raised when ASP.NET runtime is ready to authorize the user for the resources user is trying to access.
BeginRequest	This event is raised when ASP.NET runtime receives a new HTTP request.
Disposed	This event is raised when ASP.NET completes the processing of HTTP request.
EndRequest	This event is raised just before sending the response content to the client.
Error	This event is raised when an unhandled exception occurs during the processing of HTTP request.
PostRequestHandlerExecute	This event is raised just after HTTP handler finishes execution.
PreRequestHandlerExecute	This event is raised just before ASP.NET begins executing a handler for the HTTP request. After this event, ASP.NET will forward the request to the appropriate HTTP handler.
PreSendRequestContent	This event is raised just before ASP.NET sends the response contents to the client. This event

	allows us to change the contents before it gets delivered to the client. We can use this event to add the contents, which are common in all pages, to the page output. For example, a common menu, header or footer.
PreSendRequestHeaders	This event is raised just before ASP.NET sends the HTTP response headers to the client. This event allows us to change the headers before they get delivered to the client. We can use this event to add cookies and custom data into headers.
ReleaseRequestState	This event is raised after ASP.NET finishes executing all request handlers.
ResolveRequestCache	This event is raised to determine whether the request can be fulfilled by returning the contents from the Output Cache. This depends on how the Output Caching has been setup for your web application.
UpdateRequestCache	This event is raised when ASP.NET has completed processing the current HTTP request and the output contents are ready to be added to the Output Cache. This depends on how the Output Caching has been setup for your Web application.

Apart from these events, there are four more events that we can use. We can hook up to these events by implementing the methods in the global.asax file of our Web application.

These events are as follows:

- Application_OnStart
  This event is raised when the very first request arrives to the Web application.
- Application_OnEnd
  This event is raised just before the application is going to terminate.
- Session_OnStart
  This event is raised for the very first request of the user's session.
- Session_OnEnd
  This event is raised when the session is abandoned or expired.

Registering HTTP Modules in Configuration Files

Once an HTTP module is built and copied into the bin directory of our Web application or copied into the Global Assembly Cache, then we will register it in either the web.config or machine.config file.

We can use <httpModules> and <add> nodes for adding HTTP modules to our Web applications. In fact the modules are listed by using <add> nodes in between <httpModules> and </httpModules> nodes.

Since configuration settings are inheritable, the child directories inherit configuration settings of the parent directory. As a consequence, child directories might inherit some unwanted HTTP modules as part of the parent configuration; therefore, we need a way to remove those unwanted modules. We can use the <remove> node for this.

If we want to remove all of the inherited HTTP modules from our application, we can use the <clear> node.

The following is a generic example of adding an HTTP module:

```
<httpModules>
        <add type="classname, assemblyname"
name="modulename"  />
<httpModules>
```

The following is a generic example of removing an HTTP module from your application.

```
<httpModules>
        <remove name="modulename"  />
<httpModules>
```

In the above XML,

- The type attribute specifies the actual type of the HTTP module in the form of class and assembly name.
- The name attribute specifies the friendly name for the module. This is the name that will be used by other applications for identifying the HTTP module.

Use of HTTP Modules by the ASP.NET Runtime

ASP.NET runtime uses HTTP modules for implementing some special features. The following snippet from the machine.config file shows the HTTP modules installed by the ASP.NET runtime.

```
<httpModules>
 <add name="OutputCache"
type="System.Web.Caching.OutputCacheModule"/>
 <add name="Session"
type="System.Web.SessionState.SessionStateModule"/>
 <add name="WindowsAuthentication"
   type="System.Web.Security.WindowsAuthenticationModule"/>
 <add name="FormsAuthentication"
   type="System.Web.Security.FormsAuthenticationModule"/>
 <add name="PassportAuthentication"
   type="System.Web.Security.PassportAuthenticationModule"/>
 <add name="UrlAuthorization"
   type="System.Web.Security.UrlAuthorizationModule"/>
 <add name="FileAuthorization"
   type="System.Web.Security.FileAuthorizationModule"/>
</httpModules>
```

All of the above HTTP modules are used by ASP.NET to provide services like authentication and authorization, session management and output caching. Since these modules have been registered in machine.config file, these modules are automatically available to all of the Web applications.

Implementing an HTTP Module for Providing Security Services

Now we will implement an HTTP module that provides security services for our Web application. Our HTTP module will basically provide a custom authentication service. It will receive authentication credentials in HTTP request and will determine whether those credentials are valid. If yes, what roles are the user associated with? Through the User.Identity object, it will associate those roles that are accessible to our Web application pages to the user's identity.

Following is the code of our HTTP module.

```
using System;
using System.Web;
using System.Security.Principal;

namespace SecurityModules
{
 /// <summary>
 /// Summary description for Class1.
 /// </summary>

 public class CustomAuthenticationModule : IHttpModule
 {
  public CustomAuthenticationModule()
  {
  }
  public void Init(HttpApplication r_objApplication)
  {
   // Register our event handler with Application object.
   r_objApplication.AuthenticateRequest +=
             new EventHandler(this.AuthenticateRequest) ;
  }

  public void Dispose()
  {
   // Left blank because we dont have to do anything.
  }

  private void AuthenticateRequest(object r_objSender,
                                EventArgs r_objEventArgs)
  {
   // Authenticate user credentials, and find out user roles.
   1. HttpApplication objApp = (HttpApplication) r_objSender ;
   2. HttpContext objContext = (HttpContext) objApp.Context ;
   3. if ( (objApp.Request["userid"] == null) ||
   4.  (objApp.Request["password"] == null) )
  5. {
  6.  objContext.Response.Write("<H1>Credentials not
      provided</H1>") ;
  7.  objContext.Response.End() ;
  8. }
  9. string userid = "" ;
  10. userid = objApp.Request["userid"].ToString() ;
  11. string password = "" ;
  12. password = objApp.Request["password"].ToString() ;
  13. string[] strRoles ;
  14. strRoles = AuthenticateAndGetRoles(userid, password) ;
  15. if ((strRoles == null) || (strRoles.GetLength(0) == 0))
  16. {
  17. objContext.Response.Write("<H1>We are sorry but we could
      not find this user id and password in our database</H1>") ;
  18. objApp.CompleteRequest() ;
  19. }

  20. GenericIdentity objIdentity = new GenericIdentity(userid,
                                  "CustomAuthentication") ;
  21. objContext.User = new GenericPrincipal(objIdentity, strRoles)
        ;
}

 private string[] AuthenticateAndGetRoles(string r_strUserID,
                                    string r_strPassword)
 {
  string[] strRoles = null ;
  if ((r_strUserID.Equals("Steve")) &&
                        (r_strPassword.Equals("15seconds")))
  {
   strRoles = new String[1] ;
   strRoles[0] = "Administrator" ;
  }
  else if ((r_strUserID.Equals("Mansoor")) &&
                            (r_strPassword.Equals("mas")))
  {
    strRoles = new string[1] ;
    strRoles[0] = "User" ;
   }
   return strRoles ;
  }
 }
}
```

Let's explore the code.

We start with the Init function. This function plugs in our handler for the AuthenticateRequest event into the Application object's event handlers list. This will cause the Application object to call this method whenever the AuthenticationRequest event is raised.

Once our HTTP module is initialized, its AuthenticateRequest method will be called for authenticating client requests. AuthenticateRequest method is the heart of the security/authentication mechanism. In that function:

Line 1 and Line 2 extract the HttpApplication and HttpContext objects. Line 3 through Line 7 checks whether any of the userid or password is not provided to us. If this is the case, error is displayed and the request processing is terminated.

Line 9 through Line 12 extract the user id and password from the HttpRequest object.

Line 14 calls a helper function, named AuthenticateAndGetRoles. This function basically performs the authentication and determines the user role. This has been hard-coded and only two users are allowed, but we can generalize this method and add code for interacting with some user database to retrieve user roles.

Line 16 through Line 19 checks whether the user has any role assigned to it. If this is not the case that means the credentials passed to us could not be verified; therefore, these credentials are not valid. So, an error message is sent to the client and the request is completed.

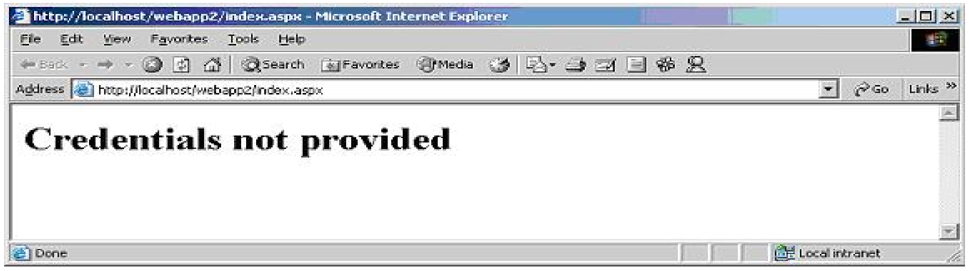
Line 20 and Line 21 are very important because these lines actually inform the ASP.NET HTTP runtime about the identity of the logged-in user. Once these lines are successfully executed, our aspx pages will be able to access this information by using the User object.

Now let's see this authentication mechanism in action. Currently we are only allowing the following users to log in to our system:

- User id = Steve, Password = 15seconds, Role = Administrator
- User id = Mansoor, Password = mas, Role = User

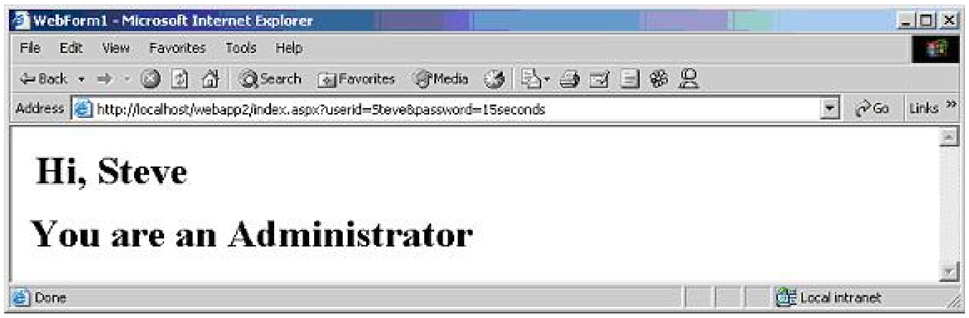
Note that user id and password are case-sensitive.

First try logging-in without providing credentials. Go to http://localhost/webapp2/index.aspx and you should see the following message.


http://localhost/webapp2/index.aspx - Microsoft Internet Explorer
Address http://localhost/webapp2/index.aspx

Credentials not provided

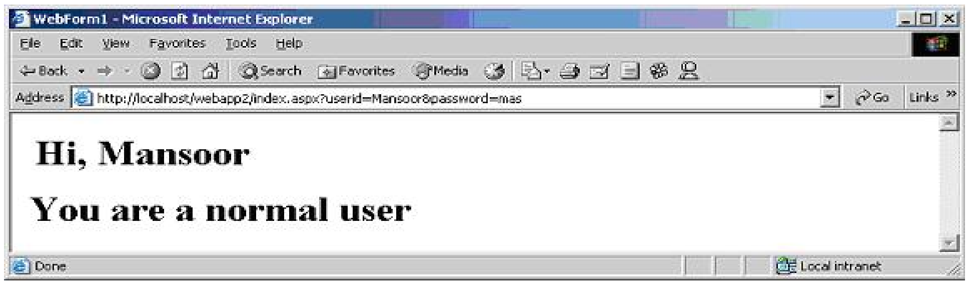
Now try logging-in with the user id "Steve" and password "15seconds". Go to http://localhost/webapp2/index.aspx?userid=Steve&password=15seconds and you should see the following welcome message.


WebForm1 - Microsoft Internet Explorer
Address http://localhost/webapp2/index.aspx?userid=Steve&password=15seconds

Hi, Steve

You are an Administrator

Now try to log-in with the user id "Mansoor" and password "15seconds". Go to http://localhost/webapp2/index.aspx?userid=Mansoor&password=mas and you should see the following welcome page.

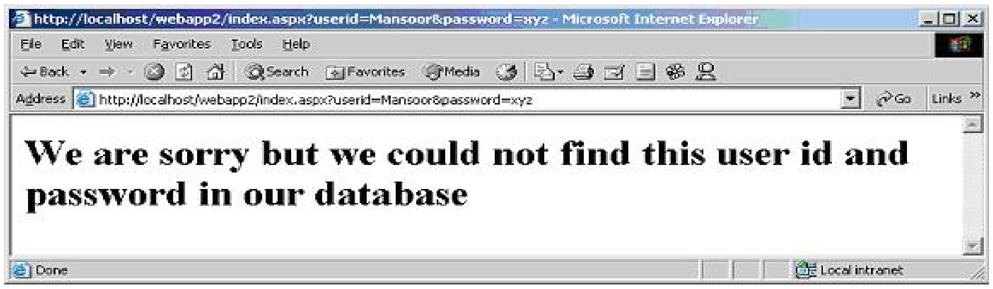

WebForm1 - Microsoft Internet Explorer
Address http://localhost/webapp2/index.aspx?userid=Mansoor&password=mas

Hi, Mansoor

You are a normal user

Now try to log-in with the wrong combination of user id and password. Go to

http://localhost/webapp2/index.aspx?userid=Mansoor&password=xyz and you should see the following error message.


http://localhost/webapp2/index.aspx?userid=Mansoor&password=xyz - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/webapp2/index.aspx?userid=Mansoor&password=xyz
Go Links
Done
Local intranet

We are sorry but we could not find this user id and password in our database

This shows our security module in action. You can generalize this security module by using database-access code in the AuthenticateAndGetRoles method.

For all of this to work, we have to perform some changes in our web.config file. First of all, since we are using our own custom authentication, we don't need any other authentication mechanism. To specify this, change the <authentication> node in web.config file of webapp2 to look like this:

```
<authentication mode="None"/>
```

Similarly, don't allow anonymous users to our Web site. Add the following to web.config file:

```
<authorization>
 <deny users="?"/>
</authorization>
```

Users should at least have anonymous access to the file that they will use for providing credentials. Use the following configuration setting in the web.config file for specifying index.aspx as the only anonymously accessible file:

```
<location path="index.aspx">
 <system.web>
  <authorization>
   <allow users="*"/>
  </authorization>
 </system.web>
</location>
```

ASP.NET Page directives:

Asp.Net web form page framework supports the following directives

1. @Page
2. @Master
3. @Control

4. @Register
5. @Reference
6. @PreviousPageType
7. @OutputCache
8. @Import
9. @Implements
10. @Assembly
11. @MasterType
12. @Page Directive

The @Page directive enables you to specify attributes and values for an Asp.Net Page to be used when the page is parsed and compiled. Every .aspx files should include this @Page directive to execute. There are many attributes belong to this directive. We shall discuss some of the important attributes here.

a. AspCompat: When set to True, this allows to the page to be executed on a single-threaded apartment. If you want to use a component developed in VB 6.0, you can set this value to True. But setting this attribute to true can cause your page's performance to degrade.

b. Language: This attribute tells the compiler about the language being used in the code-behind. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET.

c. AutoEventWireup: For every page there is an automatic way to bind the events to methods in the same .aspx file or in code behind. The default value is true.

d. CodeFile: Specifies the code-behid file with which the page is associated.

e. Title: To set the page title other than what is specified in the master page.

f. Culture: Specifies the culture setting of the page. If you set to auto, enables the page to automatically detect the culture required for the page.

g. UICulture: Specifies the UI culture setting to use for the page. Supports any valid UI culture value.

h. ValidateRequest: Indicates whether request validation should occur. If set to true, request validation checks all input data against a hard-coded list of potentially dangerous values. If a match occurs, an HttpRequestValidationException Class is thrown. The default is true. This feature is enabled in the machine configuration file (Machine.config). You can disable it in your application configuration file (Web.config) or on the page by setting this attribute to false.

i. Theme: To specify the theme for the page. This is a new feature available in Asp.Net 2.0.

j. SmartNavigation: Indicates the smart navigation feature of the page. When set to True, this returns the postback to current position of the page. The default value is false.

k. MasterPageFile: Specify the location of the MasterPage file to be used with the current Asp.Net page.

l. EnableViewState: Indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true.

m. ErrorPage: Specifies a target URL for redirection if an unhandled page exception occurs.

n. Inherits: Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.

There are also other attributes which are of seldom use such as Buffer, CodePage, ClassName, EnableSessionState, Debug, Description, EnableTheming, EnableViewStateMac, TraceMode, WarningLevel, etc. Here is an example of how a @Page directive looks

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Sample.aspx.cs" Inherits="Sample" Title="Sample
Page Title" %>
```

@Master Directive

The @Master directive is quite similar to the @Page directive. The @Master directive belongs to Master Pages that is

.master files. The master page will be used in conjunction of any number of content pages. So the content pages can the inherits the attributes of the master page. Even though, both @Page and @Master page directives are similar, the @Master directive has only fewer attributes as follows

a. Language: This attribute tells the compiler about the language being used in the code-behind. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET.

b. AutoEventWireup: For every page there is an automatic way to bind the events to methods in the same master file or in code behind. The default value is True.

c. CodeFile: Specifies the code-behid file with which the MasterPage is associated

d. Title: Set the MasterPage Title.

e. MasterPageFile: Specifies the location of the MasterPage file to be used with the current MasterPage. This is called as Nested Master Page.

f. EnableViewState: Indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true.

g. Inherits: Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.

Here is an example of how a @Master directive looks

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="WebMaster.master.cs" Inherits="WebMaster" %>
```

@Control Directive

The @Control directive is used when we build an Asp.Net user controls. The @Control directive helps us to define the properties to be inherited by the user control. These values are assigned to the user control as the page is parsed and compiled. The attributes of @Control directives are

a. Language: This attribute tells the compiler about the language being used in the code-behind. Values can represent any .NET-supported language, including Visual Basic, C#, or JScript .NET.

b. AutoEventWireup: For every page there is an automatic way to bind the events to methods in the same .ascx file or in code behind. The default value is true.

c. CodeFile: Specifies the code-behid file with which the user control is associated.

d. EnableViewState: Indicates whether view state is maintained across page requests. true if view state is maintained; otherwise, false. The default is true.

e. Inherits: Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.

f. Debug: Indicates whether the page should be compiled with debug symbols.

g. Src: Points to the source file of the class used for the code behind of the user control.

The other attributes which are very rarely used is ClassName, CompilerOptions, ComplieWith, Description, EnableTheming, Explicit, LinePragmas, Strict and WarningLevel.

Here is an example of how a @Control directive looks

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="MyControl.ascx.cs" Inherits=" MyControl " %>
```

@Register Directive

The @Register directive associates aliases with namespaces and class names for notation in custom server control syntax. When you drag and drop a user control onto your .aspx pages, the Visual Studio 2005 automatically creates an @Register directive at the top of the page. This register the user control on the page so that the control can be accessed on the .aspx page by a specific name.

The main atttribues of @Register directive are

a. Assembly: The assembly you are associatin with the TagPrefix.

b. amespace: The namspace to relate with TagPrefix.

c. Src: The location of the user control.

d. TagName: The alias to relate to the class name.

e. TagPrefix: The alias to relate to the namespace.

Here is an example of how a @Register directive looks

```
<%@ Register Src="Yourusercontrol.ascx" TagName="
Yourusercontrol " TagPrefix="uc1"
Src="~\usercontrol\usercontrol1.ascx" %>
```

@Reference Directive

The @Reference directive declares that another asp.net page or user control should be complied along with the current page or user control. The 2 attributes for @Reference direcive are

a. Control: User control that ASP.NET should dynamically compile and link to the current page at run time.

b. Page: The Web Forms page that ASP.NET should dynamically compile and link to the current page at run time.

c. VirutalPath: Specifies the location of the page or user control from which the active page will be referenced.

Here is an example of how a @Reference directive looks

```
<%@ Reference VirutalPath="YourReferencePage.ascx" %>
```

@PreviousPageType Directive

The @PreviousPageType is a new directive makes excellence in asp.net 2.0 pages. The concept of cross-page posting between Asp.Net pages is achieved by this directive. This directive is used to specify the page from which the cross-page posting initiates. This simple directive contains only two attibutes

a. TagName: Sets the name of the derived class from which the postback will occur.

b. VirutalPath: sets the location of the posting page from which the postback will occur.

Here is an example of @PreviousPageType directive

```
<%@ PreviousPageType
VirtualPath="~/YourPreviousPageName.aspx" %>
```

@OutputCache Directive

The @OutputCache directive controls the output caching policies of the Asp.Net page or user control. You can even cache programmatically through code by using Visual Basic .NET or Visual C# .NET. The very important attributes for the @OutputCache directive are as follows

Duration: The duration of time in seconds that the page or user control is cached.

Location: To specify the location to store the output cache. To store the output cache on the browser client where the request originated set the value as 'Client'. To store the output cache on any HTTP 1.1 cache-capable devices including the proxy servers and the client that made request, specify the Location as Downstream. To store the output cache on the Web server, mention the location as Server.

VaryByParam: List of strings used to vary the output cache, separated with semi-colon.

VaryByControl: List of strings used to vary the output cache of a user Control, separated with semi-colon.

VaryByCustom: String of values, specifies the custom output caching requirements.

VaryByHeader: List of HTTP headers used to vary the output cache, separated with semi-colon.

The other attribues which is rarely used are CacheProfile, DiskCacheable, NoStore, SqlDependency, etc.

```
<%@ OutputCache Duration="60" Location="Server"
VaryByParam="None" %>
```

To turn off the output cache for an ASP.NET Web page at the client location and at the proxy location, set the Location attribute value to none, and then set the VaryByParam value to none in the @ OutputCache directive. Use the following code samples to turn off client and proxy caching.

```
<%@ OutputCache Location="None" VaryByParam="None"
%>
```

@Import Directive

The @Import directive allows you to specify any namespaces to the imported to the Asp.Net pages or user controls. By importing, all the classes and interfaces of the namespace are made available to the page or user control. The example of the @Import directive

```
<%@ Import namespace=”System.Data” %>
<%@ Import namespace=”System.Data.SqlClient” %>
```

@Implements Directive

The @Implements directive gets the Asp.Net page to implement a specified .NET framework interface. The only single attribute is Interface, helps to specify the .NET Framework interface. When the Asp.Net page or user control implements an interface, it has direct access to all its events, methods and properties.

```
<%@ Implements Interface=”System.Web.UI.IValidator” %>
```

@Assembly Directive

The @Assembly directive is used to make your ASP.NET page aware of external components. This directive supports two attributes:

a. Name: Enables you specify the name of an assembly you want to attach to the page. Here you should mention the filename without the extension.

b. Src: represents the name of a source code file

```
<%@ Assembly Name="YourAssemblyName" %>
```

@MasterType Directive

To access members of a specific master page from a content page, you can create a strongly typed reference to the master page by creating a @MasterType directive. This directive supports of two attributes such as TypeName and VirtualPath.

a. TypeName: Sets the name of the derived class from which to get strongly typed references or members.

b. VirtualPath: Sets the location of the master page from which the strongly typed references and members will be retrieved.

If you have public properties defined in a Master Page that you'd like to access in a strongly-typed manner you can add the MasterType directive into a page as shown next

- ***Questions:***
  - 2.1: What is the use of @ Register directives?
  - 2.2: What are directives ? Which are the directives used in ASP ?
  - 2.3: What is Page Directive?

Page Event and Page Life Cycle.

General Page Life-cycle Stages
Stage
Description

Page request
The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled or whether a cached version of the page can be sent in response without running the page.

Start

In the start step, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. Additionally, during the start step, the page's UICulture property is set.

Page initialization

During page initialization, controls on the page are available and each control's UniqueID property is set. Any themes are also applied to the page. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.

Load

During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.

Validation

During validation, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page.

Postback event handling

If the request is a postback, any event handlers are called.

Rendering

Before rendering, view state is saved for the page and all controls. During the rendering phase, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream of the page's Response property.

Unload

Unload is called after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and any cleanup is performed.

Data Binding Events for Data-Bound Controls

Control Event

Typical Use

DataBinding

This event is raised by data-bound controls before the PreRender event of the containing control (or of the Page object) and marks the beginning of binding the control to the data.

RowCreated (GridView) ItemCreated (DataList, DetailsView, SiteMapPath, DataGrid, FormView, Repeater)

Use this event to manipulate content that is not dependent on data binding. For example, at run time, you might programmatically add formatting to a header or footer row in a GridView control.

RowDataBound (GridView) ItemDataBound (DataList, SiteMapPath, DataGrid, Repeater)

When this event occurs, data is available in the row or item, so you can format data or set the FilterExpression property on child data source controls for displaying related data within the row or item.

DataBound
This event marks the end of data-binding operations in a data-bound control. In a GridView control, data binding is complete for all rows and any child controls. Use this event to format data bound content or to initiate data binding in other controls that depend on values from the current control's content.

Common Life-cycle Events
Page Event
Typical Use
PreInit

Use this event for the following:

- Check the IsPostBack property to determine whether this is the first time the page is being processed.

- Create or re-create dynamic controls.
- Set a master page dynamically.
- Set the Theme property dynamically.
- Read or set profile property values.

Note: If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event.

Init
Raised after all controls have been initialized and any skin settings have been applied. Use this event to read or initialize control properties.

InitComplete
Raised by the Page object. Use this event for processing tasks that require all initialization be complete.

PreLoad
Use this event if you need to perform processing on your page or control before the Load event. After the Page raises this event, it loads view state for itself and all controls, and then processes any postback data included with the Request instance.

Load
The Page calls the OnLoad event method on the Page, then recursively does the same for each child control, which does the same for each of its child controls until the page and all controls are loaded.

Control events
Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event. In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.

LoadComplete
Use this event for tasks that require that all other controls on the page be loaded.

PreRender
Before this event occurs:

- The Page object calls EnsureChildControls for each control and for the page.
- Each data bound control whose DataSourceID property is set calls its DataBind method.
- The PreRender event occurs for each control on the page. Use the event to make final changes to the contents of the page or its controls.

SaveStateCompleteBefore this event occurs, ViewState has been saved for the page and for all controls. Any changes to the page or controls at this point will be ignored. Use this event perform tasks that require view state to be saved, but that do not make any changes to controls.

Render
This is not an event; instead, at this stage of processing, the Page object calls this method on each control. All ASP.NET Web server controls have a Render method that writes out the control's markup that is sent to the browser. If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the Render method. A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code.

Unload
This event occurs for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections. For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks. Note: During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the Response.Write method, the page will throw an exception.

Questions:

1. List the various stages of Page-Load lifecycle.
2. What’s the sequence in which ASP.NET events are processed?
3. What is event bubbling ?

PostBack and CrossPage Posting:

PostBack

Programming model in old ASP for using POST method in form is to post the values of a Form to a second page. The second asp page will receive the data and process it for doing any validation or processing on the server side.

With ASP .Net, the whole model has changed. Each of the asp .net pages will be a separate entity with ability to process its own posted data. That is, the values of the Form are posted to the same page and the very same page can process the data. This model is called post back.

Each Asp .net page when loaded goes through a regular creation and destruction cycle like Initialization, Page load etc., in the beginning and unload while closing it. This Postback is a read only property with each Asp .Net Page (System.Web.UI.Page) class. This is false when the first time the page is loaded and is true when the page is submitted and processed. This enables users to write the code depending on if the PostBack is true or false (with the use of the function Page.IsPostBack()).

Implementation of ASP.Net Post back on the Client side:

Post back is implemented with the use javascript in the client side. The HTML page generated for each .aspx page will have the action property of the form tag set to the same page. This makes the page to be posted on to itself. If we check the entry on the HTML file, it will look something like this.

```
<form name="_ctl1" method="post"
action="pagename.aspx?getparameter1=134"
language="javascript" onsubmit="if (!ValidatorOnSubmit()) return
false;" id="_ctl1" >
```

Also, all the validation code that is written (Required Field Validation, Regular Expression validation etc.,) will all be processed at the client side using the .js(javascript) file present in the webserver_wwwroot/aspnet_client folder.

With this new ASP .Net model, even if the user wants to post the data to a different .aspx page, the web server will check for the runat='server' tag in the form tag and post the web form to the same .aspx page. A simple declaration as in the following code snippet will be enough to create such a web form.

```
<form id="form1" runat="server" >
<!-- place the controls inside -->
</form>
```

Cross Page posting or cross page postback is used to submit a form on one page (say default.aspx) and retrieve values of controls of this page on another page (say Default2.aspx)

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
First Name:
<asp:TextBox ID="txtFirstName" runat="server">
</asp:TextBox><br /><br />
Last Name:
<asp:TextBox ID="txtLastName" runat="server">
</asp:TextBox><br /><br /><br />

<asp:Button ID="btnSubmit" runat="server"
          OnClick="btnSubmit_Click"
          PostBackUrl="~/Default2.aspx"
          Text="Submit to Second Page" /><br />
</div>
</form>
</body>
</html>
```

Don't forget to set PostBackUrl Property of Button
PostBackUrl="~/Default2.aspx"

Now to retrieve values of textBoxes on Default2.aspx page, write below mentioned code in Page_Load event of second page (Default2.aspx)

C# code behind

```
01. protected void Page_Load(object sender, EventArgs e)
02. {
03. //Check whether previous page is cross page post back
or not
04. if (PreviousPage != null &&
PreviousPage.IsCrossPagePostBack)
{
TextBox txtPbFirstName = (TextBox)PreviousPage.FindControl("txtFirstName");
TextBox txtPbLastName = (TextBox)PreviousPage.FindControl("txtLastName");
Label1.Text = "Welcome " + txtPbFirstName.Text + " " + txtPbLastName.Text;
}
else
{
Response.Redirect("Default.aspx");
}
}
```

VB.NET Code behind

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
'Check whether previous page is cross page post back or not
If PreviousPage IsNot Nothing AndAlsoPreviousPage.IsCrossPagePostBack Then
Dim txtPbFirstName As TextBox =DirectCast(PreviousPage.FindControl("txtFirstName"), TextBox)
Dim txtPbLastName As TextBox =DirectCast(PreviousPage.FindControl("txtLastName"), TextBox)
Label1.Text = ("Welcome " & txtPbFirstName.Text & " ") + txtPbLastName.Text
Else
Response.Redirect("Default.aspx")
End If
End Sub
```

If you are using masterpages then you need to write code to FindControl as mentioned below

```
ContentPlaceHolder exampleHolder =(ContentPlaceHolder)Page.PreviousPage.Form.FindControl ("Content1"));
TextBox txtExample = exampleHolder.FindControl("txtFirstName");
```

Questions:

1. What is Postback?
2. What is CrossPage Posting?
3. What' is the sequence in which ASP.NET events are processed?
4. In which event are the controls fully loaded?
5. What is event bubbling?

ASP.NET Application Compilation Models:

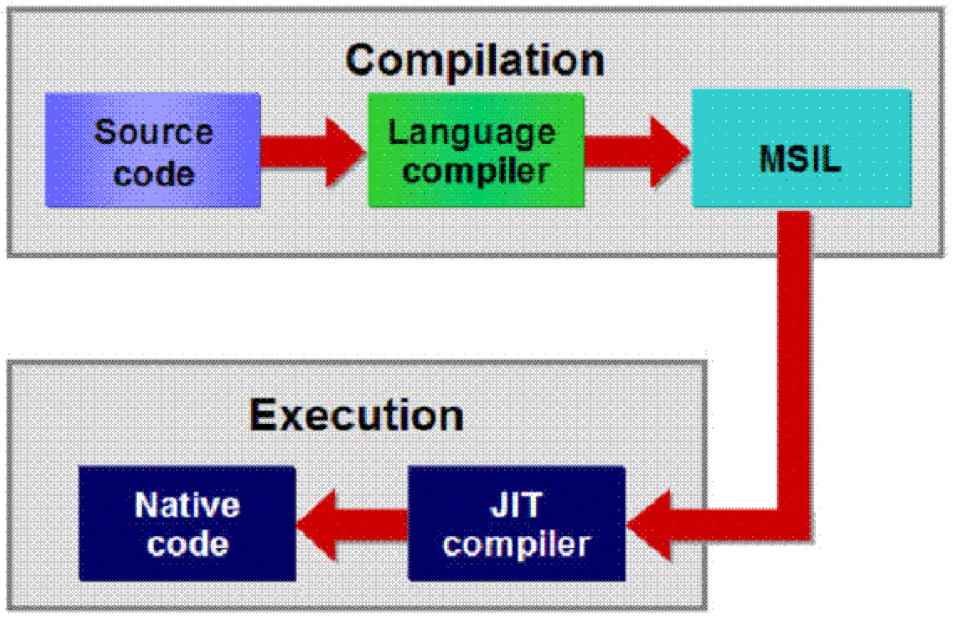
ASP.NET Compilation

This information is not vital to your success as an ASP.NET developer, but having an understanding of the architecture of your development environment always makes you a better developer.

ASP.NET is nothing like the legacy ASP with which many developers are familiar. You develop ASP pages by using VBScript or JScript, and they are interpreted, meaning that they are executed just as they are written, directly from the page. ASP.NET is entirely different in that ASP.NET pages are compiled before they are executed.

When you write ASP.NET code, you do so in human-readable text. Before ASP.NET can run your code, it has to convert it into something that the computer can understand and execute. The process of converting code from what a programmer types into what a computer can actually execute is called *compilation*.

Exactly how compilation takes place in ASP.NET depends on the compilation model that you use. Several different compilation models are available to you in ASP.NET 3.5.



The Web Application Compilation Model

The web application compilation model is the same model provided in ASP.NET 1.0 and 1.1. When you use this model, you use the Build menu in Visual Web Developer to compile your application into a single DLL file that is copied to a bin folder in the root of your application. When the first request comes into your application, the DLL from the bin folder is copied to the Temporary ASP.NET Files folder, where it is then recompiled into code that the operating system can execute in a process known as *just-in-time (JIT)* compilation. The JIT compilation causes a delay of several seconds on the first request of the application.

NOTE

The web application model is available only in Visual Studio 2008. Visual Web Developer 2008 does not enable you to create ASP.NET applications using the web application model.

NOTE

The Temporary ASP.NET Files folder is located at Windows\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files by default.

To create a new ASP.NET web application using the web application compilation model, select File, New Project, and then choose the ASP.NET Web Application template as shown in Figure 3.1.

Figure 3.1

Choose the New Project option on the File menu to create a new ASP.NET application that uses the web application compilation model.

ASP.NET Server Controls:

ASP.NET - Server Controls
ASP.NET has solved the "spaghetti-code" problem described above with server controls.
Server controls are tags that are understood by the server.
There are three kinds of server controls:

- HTML Server Controls - Traditional HTML tags
- Web Server Controls - New ASP.NET tags
- Validation Server Controls - For input validation

ASP.NET - HTML Server Controls
HTML server controls are HTML tags understood by the server.

HTML elements in ASP.NET files are, by default, treated as text. To make these elements programmable, add a runat="server" attribute to the HTML element. This attribute indicates that the element should be treated as a server control. The id attribute is added to identify the server control. The id reference can be used to manipulate the server control at run time.

Note: All HTML server controls must be within a <form> tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.

In the following example we declare an HtmlAnchor server control in an .aspx file. Then we manipulate the HRef attribute of the HtmlAnchor control in an event handler (an event handler is a subroutine that executes code for a given event). The Page_Load event is one of many events that ASP.NET understands:

```
<script runat="server">
Sub Page_Load
link1.HRef="http://www.w3schools.com"
End Sub
</script>

<html>
<body>

<form runat="server">
<a id="link1" runat="server">Visit W3Schools!</a>
</form>

</body>
</html>
```

ASP.NET - Web Server Controls

Web server controls are special ASP.NET tags understood by the server.

Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work. However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.

The syntax for creating a Web server control is:

```
<asp:control_name id="some_id" runat="server" />
```

In the following example we declare a Button server control in an .aspx file. Then we create an event handler for the Click event which changes the text on the button:

```
<script runat="server">
Sub submit(Source As Object, e As EventArgs)
button1.Text="You clicked me!"
End Sub
</script>

<html>
<body>

<form runat="server">
<asp:Button id="button1" Text="Click me!"
runat="server" OnClick="submit"/>
</form>

</body>
</html>
```

ASP.NET - Validation Server Controls

Validation server controls are used to validate user-input. If the user-input does not pass validation, it will display an error message to the user.

Each validation control performs a specific type of validation (like validating against a specific value or a range of values).

By default, page validation is performed when a Button, ImageButton, or LinkButton control is clicked. You can prevent validation when a button control is clicked by setting the CausesValidation property to false.

The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat="server" />
```

In the following example we declare one TextBox control, one Button control, and one RangeValidator control in an .aspx file. If validation fails, the text "The value must be from 1 to 100!" will be displayed in the RangeValidator control:

```
<html>
<body>

<form runat="server">
<p>Enter a number from 1 to 100:
<asp:TextBox id="tbox1" runat="server" />
```

```
<br /><br />
<asp:Button Text="Submit" runat="server" />
</p>

<p>
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
Text="The value must be from 1 to 100!"
runat="server" />
</p>
</form>

</body>
</html>
```

Questions:

1. How to set view state for server control? Enableviewstate property?

HTML Controls:

HtmlControls In ASP.NET

System.Web.UI.HtmlControls namespace is often ignored by ASP.NET developers. There is an opinion that **System.Web.UI.WebControls** classes are more natural to ASP.NET web application and I agree with that. However, HtmlControls namespace is still standard part of .Net Framework just like WebControls. You can drag it from toolbox and easily drop it to your web form. HtmlControls have its advantages in some scenarios and you **should know both namespaces** so you can decide which class to use in your specific case.

HtmlControls are just programmable HTML tags. By default these tags are literal text and you can't reference them with server side code. To "see" any HTML tag with your ASP.NET server side code you need to add runat="server" and some value to ID parameter. For example, to work with <textarea> HTML tag with server side code, you can use HTML code like this:

```
<textarea runat="server" id="TextArea1" cols="20"
rows="2"></textarea>
```

So, nothing hard here, we just set value of id property and add runat="server" part. After this, we can manipulate with this tag with C# or VB.NET server side code, like this:

[C#]

```
protected void Page_Load(object sender, EventArgs e)
{
   // set new size of textarea
   TextArea1.Cols = 15;
}
```

[VB.NET]

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
   ' set new size of textarea
   TextArea1.Cols = 15
End Sub
```

HtmlControls are much less abstract than WebControls. With HtmlControls you work directly with HTML output. WebControls are not always rendered on the same way. For example TextBox control is rendered as <input type="text" /> tag if value of its TextMode property is SingleLine but TextBox will render as <textarea > if TextMode=MultiLine.

How to apply styles to HtmlControls

HtmlControls have not styles property so you can't set style directly. To apply styles to HtmlControls you need to use Attributes property, with code like this:

[C#]

```
TextArea1.Attributes["Style"] = "FONT-FAMILY: 'Arial'; COLOR:
blue; BACKGROUND-COLOR: yellow";
```

[VB.NET]

```
TextArea1.Attributes("Style") = "FONT-FAMILY: 'Arial'; COLOR:
blue; BACKGROUND-COLOR: yellow”
```

Questions:

1. What are HTML server controls and Web controls ?

Validation Controls:

With ASP.NET, there are six(6) controls included. They are:

- **The RequiredFieldValidation Control**
- **The CompareValidator Control**
- **The RangeValidator Control**
- **The RegularExpressionValidator Control**
- **The CustomValidator Control**

Validator Control Basics

All of the validation controls inherit from the base class BaseValidator so they all have a series of properties and methods that are common to all validation controls. They are:

- **ControlToValidate** - This value is which control the validator is applied to.
- **ErrorMessage** - This is the error message that will be displayed in the validation summary.
- **IsValid** - Boolean value for whether or not the control is valid.
- **Validate** - Method to validate the input control and update the IsValid property.
- **Display** - This controls how the error message is shown. Here are the possible options:
    - None (The validation message is never displayed.)
    - Static (Space for the validation message is allocated in the page layout.)
    - Dynamic (Space for the validation message is dynamically added to the page if validation fails.)

The RequiredFieldValidation Control

The first control we have is the RequiredFieldValidation Control. As it's obvious, it make sure that a user inputs a value. Here is how it's used:

```
Required field: <asp:textbox id="textbox1" runat="server"/>

<asp:RequiredFieldValidator id="valRequired" runat="server"
ControlToValidate="textbox1"

   ErrorMessage="* You must enter a value into textbox1"
Display="dynamic">*

</asp:RequiredFieldValidator>
```

In this example, we have a textbox which will not be valid until the user types something in. Inside the validator tag, we have a single *. The text in the innerhtml will be shown in the controltovalidate if the control is not valid. It should be noted that the *ErrorMessage* attribute is not what is shown. The ErrorMessage tag is shown in the Validation Summary (see below).

The CompareValidator Control

Next we look at the CompareValidator Control. Usage of this CompareValidator is for confirming new passwords, checking if a departure date is before the arrival date, etc. We'll start of with a sample:

```
Textbox 1: <asp:textbox id="textbox1" runat="server"/><br />

Textbox 2: <asp:textbox id="textbox2" runat="server"/><br />

<asp:CompareValidator id="valCompare" runat="server"

   ControlToValidate="textbox1"
ControlToCompare="textbox2"

   Operator="Equals"

   ErrorMessage="* You must enter the same values into
textbox 1 and textbox 2"

   Display="dynamic">*

</asp:CompareValidator>
```

Here we have a sample where the two textboxes must be equal. The tags that are unique to this control is the *ControlToCompare* attribute which is the control that will be compared. The two controls are compared with the type of comparison specified in the *Operator* attribute. The *Operator* attribute can contain Equal, GreterThan, LessThanOrEqual, etc. Another usage of the ComapareValidator is to have a control compare to a value. For example:

```
Field: <asp:textbox id="textbox1" runat="server"/>

<asp:CompareValidator id="valRequired" runat="server"
ControlToValidate="textbox1"

   ValueToCompare="50"

   Type="Integer"

   Operator="GreaterThan"

   ErrorMessage="* You must enter the a number greater
than 50" Display="dynamic">*

</asp:CompareValidator>
```

The data type can be one of: Currency, Double, Date, Integer or String. String being the default data type.

The RangeValidator Control

Range validator control is another validator control which checks to see if a control value is within a valid range. The attributes that are necessary to this control are: *MaximumValue*, *MinimumValue*, and *Type*.

Sample:

```
Enter a date from 1998:

<asp:textbox id="textbox1" runat="server"/>

<asp:RangeValidator id="valRange" runat="server"

   ControlToValidate="textbox1"

   MaximumValue="12/31/1998"

   MinimumValue="1/1/1998"

   Type="Date"

   ErrorMessage="* The date must be between 1/1/1998 and
12/13/1998" Display="static">*</asp:RangeValidator>
```

The RegularExpressionValidator Control

The regular expression validator is one of the more powerful features of ASP.NET. Everyone loves regular expressions. Especially when you write those really big nasty ones... and then a few days later, look at it and say to yourself. What does this do? Again, the simple usage is:

```
E-mail: <asp:textbox id="textbox1" runat="server"/>

<asp:RegularExpressionValidator id="valRegEx"
runat="server"

   ControlToValidate="textbox1"

   ValidationExpression=".*@.*\..*"

   ErrorMessage="* Your entry is not a valid e-mail address."

   display="dynamic">*

</asp:RegularExpressionValidator>
```

The CustomValidator Control

The final control we have included in ASP.NET is one that adds great flexibility to our validation abilities. We have a custom validator where we get to write out own functions and pass the control value to this function.

```
Field: <asp:textbox id="textbox1" runat="server">

<asp:CustomValidator id="valCustom" runat="server"

   ControlToValidate="textbox1"

   ClientValidationFunction="ClientValidate"

   OnServerValidate="ServerValidate"

   ErrorMessage="*This box is not valid"
dispaly="dynamic">*

</asp:CustomValidator>
```

We notice that there are two new attributes *ClientValidationFunction* and *OnServerValidate*. These are the tell the validation control which functions to pass the controltovalidate value to. ClientValidationFunction is usually a javascript funtion included in the html to the user. OnServerValidate is the function that is server-side to check for validation if client does not support client-side validation.

Client Validation function:

```
<script language="Javascript">

<!--

   /* ... Code goes here ... */

-->

</script>
```

Server Validation function:

```
Sub ServerValidate (objSource As Object, objArgs As
ServerValidateEventsArgs)

   ' Code goes here

End Sub
```

Validation Summary

ASP.NET has provided an additional control that complements the validator controls. This is the validation summary control which is used like:

```
<asp:ValidationSummary id="valSummary" runat="server"

   HeaderText="Errors:"

   ShowSummary="true" DisplayMode="List" />
```

The validation summary control will collect all the error messages of all the non-valid controls and put them in a tidy list. The list can be either shown on the web page (as shown in the example above) or with a popup box (by specifying *ShowMessageBox="True"*)

Questions:

1. How many types of validation controls are provided by ASP.NET?
2. Which two properties on validation control?
3. What type of data validation events are commonly seen in the client-side form validation?
4. Which control is used to make sure the values in two different controls are matched?
5. How do you validate the controls in ASP.NET page?
6. Name two properties common in every validation control.

Building Databases:

Create a Database Connection

We are going to use the Northwind database in our examples.

First, import the "System.Data.OleDb" namespace. We need this namespace to work with Microsoft Access and other OLE DB database providers. We will create the connection to the database in the Page_Load subroutine. We create a dbconn variable as a new OleDbConnection class with a connection string which identifies the OLE DB provider and the location of the database. Then we open the database connection

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn
dbconn=New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
end sub
</script>
```

Create a Database Command

To specify the records to retrieve from the database, we will create a dbcomm variable as a new OleDbCommand class. The OleDbCommand class is for issuing SQL queries against database tables:

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm
dbconn=New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
end sub
</script>
```

Create a DataReader

The OleDbDataReader class is used to read a stream of records from a data source. A DataReader is created by calling the ExecuteReader method of the OleDbCommand object:.

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
end sub
</script>
```

Bind to a Repeater Control

Then we bind the DataReader to a Repeater control:

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New
```

```
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
customers.DataSource=dbread
customers.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="customers" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Address</th>
<th>City</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("companyname")%></td>
<td><%#Container.DataItem("contactname")%></td>
<td><%#Container.DataItem("address")%></td>
<td><%#Container.DataItem("city")%></td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
```

```
</form>

</body>
</html>
```

Close the Database Connection

Always close both the DataReader and database connection after access to the database is no longer required:

```
dbread.Close()
dbconn.Close()
```

Exercise:

1. What is ADO .NET and what is difference between ADO and ADO.NET?
2. Give the comparision between C# and ASP.NET.
3. List an explain the steps for loading the simple ASP.NET web application.
4. What is the role of web.config file?
5. What is container class?
6. Write the steps for implementing the Asp.NET application with Database.

❖❖❖❖

8

XML

Syntax, DTDs and XML Schema, XPath, XSLT, Sax and DOM

Unit Structure

8.1 XML

What is XML?

- XML stands for EXtensible Markup Language.
- XML is a markup language much like HTML.
- XML was designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is designed to be self-descriptive.
- XML is a W3C Recommendation.

The Difference Between XML and HTML:

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is.
- HTML was designed to display data, with focus on how data looks.

HTML is about displaying information, while XML is about carrying information.

With XML You Invent Your Own Tags:

The tags are "invented" by the author of the XML document.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

XML is Not a Replacement for HTML:

XML is a complement to HTML.

It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.

XML is a software- and hardware-independent tool for carrying information.

XML is a W3C Recommendation:

XML became a W3C Recommendation 10. February 1998.

XML is Everywhere:

XML is now as important for the Web as HTML was to the foundation of the Web.

XML is the most common tool for data transmissions between all sorts of applications.

XML is used in many aspects of web development, often to simplify data storage and sharing.

XML Separates Data from HTML:

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

XML Simplifies Data Sharing;

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

XML Simplifies Data Transport:

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML Simplifies Platform Changes:

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Makes Your Data More Available:

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

XML is Used to Create New Internet Languages:

A lot of new Internet languages are created with XML.

Here are some examples:

- XHTML
- WSDL (Web Services Description Language) for describing available web services

- WAP and WML as markup languages for handheld devices
- RSS languages for news feeds
- RDF and OWL for describing resources and ontology
- SMIL for describing multimedia for the web

XML Documents Form a Tree Structure:

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements):

```
<root>
 <child>
   <subchild>.....</subchild>
 </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

1.1. SYNTAX

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

All XML Elements Must Have a Closing Tag:

In HTML, elements do not have to have a closing tag:

```
<p>This is a paragraph
<p>This is another paragraph
```

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

XML Tags are Case Sensitive:

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

```
<Message>This is incorrect</message>
<message>This is correct</message>
```

XML Elements Must be Properly Nested:

In HTML, you might see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

XML Documents Must Have a Root Element:

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML Attribute Values Must be Quoted:

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted.

Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

Entity References:

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Comments in XML:

The syntax for writing comments in XML is similar to that of HTML.

<!-- This is a comment -->

White-space is Preserved in XML:

HTML truncates multiple white-space characters to one single white-space:

HTML:	Hello Tove
Output:	Hello Tove

With XML, the white-space in a document is not truncated.

XML Stores New Line as LF:

In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF). In Unix applications, a new line is normally stored as a LF character. Macintosh applications also use an LF to store a new line. XML stores a new line as LF.

8.2 DTDs and XML Schema

DTD

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes.

A DTD can be declared inline inside an XML document, or as an external reference.

Internal DTD Declaration

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

Example XML document with an internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element contains four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

External DTD Declaration

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
```

This is the same XML document as above, but with an external DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And this is the file "note.dtd" which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Why Use a DTD?

With a DTD, each of your XML files can carry a description of its own format.

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

Your application can use a standard DTD to verify that the data you receive from the outside world is valid.

You can also use a DTD to verify your own data.

XML Schema

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas are the Successors of DTDs

We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

XML Schemas are much more powerful than DTDs.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

Some benefits of that XML Schemas are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

```
<date type="date">2004-03-11</date>
```

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Well-Formed is not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element

- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

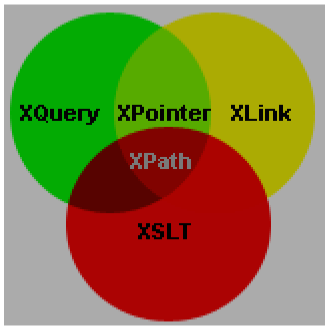
Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

8.3 XPath

- XPath is used to navigate through elements and attributes in an XML document.
- XPath is a major element in W3C's XSLT standard - and XQuery and XPointer are both built on XPath expressions.
- XPath is a language for finding information in an XML document.

What is XPath?



- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is a W3C recommendation

XPath Path Expressions

XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath Standard Functions

XPath includes over 100 built-in functions. There are functions for string values, numeric values, date and time comparison, node and

QName manipulation, sequence manipulation, Boolean values, and more.

XPath is Used in XSLT

XPath is a major element in the XSLT standard. Without XPath knowledge you will not be able to create XSLT documents.

XQuery and XPointer are both built on XPath expressions. XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

XPATH is a W3C Recommendation

XPath became a W3C Recommendation 16. November 1999.

XPath was designed to be used by XSLT, XPointer and other XML parsing software.

8.4 XSLT

XSLT(Extensible Stylesheet Language Transformation) is a language for transforming XML documents into XHTML documents or to other XML documents.

XPath is a language for navigating in XML documents.

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT = XSL Transformations

XSLT is the most important part of XSL.

XSLT is used to transform an XML document into another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. Normally XSLT does this by transforming each XML element into an (X)HTML element.

With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.

A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**.

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

How Does it Work?

In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT is a W3C Recommendation

XSLT became a W3C Recommendation 16. November 1999.

8.5 SAX and DOM

SAX (**Simple API for XML**) is a serial access parser API for XML. SAX provides a mechanism for reading data from an XML document. It is a popular alternative to the Document Object Model (DOM).

XML processing with SAX

A parser which implements SAX (ie, *a SAX Parser*) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include:

- XML Text nodes
- XML Element nodes
- XML Processing Instructions
- XML Comments

Events are fired when each of these XML features are encountered, and again when the end of them is encountered. XML attributes are provided as part of the data passed to element events.

SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

Example

Given the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement param="value">
   <FirstElement>
      Some Text
   </FirstElement>
   <?some_pi some_attr="some_value"?>
   <SecondElement param2="something">
      Pre-Text <Inline>Inlined text</Inline> Post-text.
   </SecondElement>
</RootElement>
```

This XML document, when passed through a SAX parser, will generate a sequence of events like the following:

- XML Element start, named *RootElement*, with an attribute *param* equal to "value"
- XML Element start, named *FirstElement*
- XML Text node, with data equal to "Some Text" (note: text processing, with regard to spaces, can be changed)
- XML Element end, named *FirstElement*
- Processing Instruction event, with the target *some_pi* and data *some_attr="some_value"*
- XML Element start, named *SecondElement*, with an attribute *param2* equal to "something"
- XML Text node, with data equal to "Pre-Text"
- XML Element start, named *Inline*
- XML Text node, with data equal to "Inlined text"
- XML Element end, named *Inline*
- XML Text node, with data equal to "Post-text."
- XML Element end, named *SecondElement*
- XML Element end, named *RootElement*

Note that the first line of the sample above is the XML Declaration and not a processing instruction; as such it will not be reported as a processing instruction event.

The result above may vary: the SAX specification deliberately states that a given section of text may be reported as multiple sequential text events. Thus in the example above, a SAX parser may generate a different series of events, part of which might include:

- XML Element start, named *FirstElement*
- XML Text node, with data equal to "Some "
- XML Text node, with data equal to "Text"
- XML Element end, named *FirstElement*

Benefits

SAX parsers have certain benefits over DOM-style parsers. The quantity of memory that a SAX parser must use in order to function is typically much smaller than that of a DOM parser. DOM parsers must have the entire tree in memory before any processing can begin, so the amount of memory used by a DOM parser depends entirely on the size of the input data. The memory footprint of a SAX parser, by contrast, is based only on the maximum depth of the XML file (the maximum depth of the XML tree) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the parsed tree itself.

Because of the event-driven nature of SAX, processing documents can often be faster than DOM-style parsers. Memory allocation takes time, so the larger memory footprint of the DOM is also a performance issue.

Due to the nature of DOM, streamed reading from disk is impossible. Processing XML documents larger than main memory is also impossible with DOM parsers but can be done with SAX parsers. However, DOM parsers may make use of disk space as memory to sidestep this limitation.

Drawbacks

The event-driven model of SAX is useful for XML parsing, but it does have certain drawbacks.

Certain kinds of XML validation require access to the document in full. For example, a DTD IDREF attribute requires that there be an element in the document that uses the given string as a DTD ID attribute. To validate this in a SAX parser, one would need to keep track of every previously encountered ID attribute and every previously encountered IDREF attribute, to see if any matches are made. Furthermore, if an IDREF does not match an ID, the user only discovers this after the document has been parsed; if this linkage was important to building functioning output, then time has been wasted in processing the entire document only to throw it away.

Additionally, some kinds of XML processing simply require having access to the entire document. XSLT and XPath, for example, need to be able to access any node at any time in the parsed XML tree. While a SAX parser could be used to construct such a tree, the DOM already does so by design.

DOM

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents like XML and HTML:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The DOM is separated into 3 different parts / levels:

- Core DOM - standard model for any structured document
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.

What is the HTML DOM?

The HTML DOM defines the **objects and properties** of all HTML elements, and the **methods** (interface) to access them.

If you want to study the HTML DOM, find the HTML DOM tutorial on our Home page.

What is the XML DOM?

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

QUESTIONS:

1) What does XML stands for?
2) What is the difference between XML and HTML?
3) Which are the 5 predefined entity references in XML?
4) What does DTD stands for?
5) What is an XML schema? Explain.
6) What is XPath? Explain.
7) What does XSL stands for?
8) What is XSLT? How does it work?
9) What is SAX? Explain.
10) What are different SAX events?
11) What are the benefits of SAX over DOM?
12) What are the drawbacks of SAX?
13) What are the three different parts/levels of DOM?

❖❖❖❖

MCA (Sem- V)

Advanced Web Tecnologies

Paper – IV

1. **Introduction** *2 hrs*

   - The World Wide Web:
   - Web Search Engines
   - Search engines optimization and limitations;
   - Introduction to the semantic web;

2. **Servlets** *5 hrs*

   - Introduction to servlets
   - Servlet life Cycle
   - Servlet classes
     - Servlet
     - ServletRequest
     - ServletResponse
     - Servlet Context
   - Threading Models

3. **JSP** *4 hrs*

   - JSP Development Model
   - Components of JSP page
   - Request Dispatching
   - Session and Thread Management

4. **Introduction to web services** *4 hrs*

   - What is a Web Service?
   - Software as a service
   - Web Service Architectures
   - SOA

5. Introduction to .NET framework *7 hrs*

- Evolution of .NET
- Comparison of Java and .NET
- Architecture of .NET framework
  - i. Common Language Runtime
  - ii. Common Type System
  - iii. Metadata
  - iv. Assemblies
  - v. Application Domains
  - vi. CFL
- Features of .NET
- Advantages and Application

6. C#

- Basic principles of object oriented programming
- Basic Data Types
- Building Blocks-Control Structures, operators, expressions, variables
- Reference Data Types-Strings, Data time objects Arrays
- Classes and object
- Exception Handling
- Generics
- File Handling
- Inheritance and Polymorphism
- Database programming

7. Web Applications in ASP-NET *8 hrs*

- ASP.NET Coding Modules
- ASP.NET Page Directives
- Page events and Page Life Cycle
- Post Back and Cross Page Posting
- ASP.NET Application Compilation models
- ASP.NET server Controls

- HTML Controls
- Validation Controls
- Building Databases

8. XML *5 hrs*

- Syntax
- DTDs and XML Schema
- XPath
- XSLT
- Sax and DOM

Term work/Practical : Each candidate will submit a journal in which at least 12 practical assignments based on the above syllabus along with the flow chart and program listing will be submitted with the internal test paper. Test graded for 10 marks and Practical graded for 15 marks.

Reference :

1) .NET programming – Black Book
2) Beginning C#-Wrox Publication
3) C# with Visual Studio-Vijay Mukhi, BPB
4) .NET 2008 Programming – SAMs Techmedia
5) XML Complete Reference
6) JSP complete Reference

❖❖❖❖