QUESTION PAPER SOLUTION

UNIT 1

INTRODUCTION TO JAVA

1. **How 'compile once and run anywhere' is implemented in Java, Explain. (4M) [July 2014, July 2016]**

   Architecture-Neutral

   A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, any time, forever." To a great extent, this goal was accomplished.

2. **List and explain the Java buzzwords. (6M) [July 2014, July 2016]**

   Sun micro system officially describes java with a list of buzz words or attributes. They

   - Simple & powerful
   - Safe
   - Object oriented
   - Robust
   - Architecture neutral
   - Compiled & Interpreted
   - Multithreaded
   - Easy to learn

The salient features of Java are as follows:

- Simple & Powerful: To make the language familiar to the existing programming, java is modeled on C & C++. Java empowers you to express every idea you have in a clean object oriented way.

- Safe: Threat of viruses and abuse of resources are every where, java system not only verify the memory resources but also ensures that no viruses are communicated with the applet. The absence of pointers in java ensures that program can not gain access to memory location.
- Object-oriented: Java is an object-oriented language. Almost every thing in java is object. All the program codes & data reside within object & classes.
- Robust: Java is a strictly typed language, because the types must match exactly. It checks your code at compile time as well as at run time. Java is a garbage collected language, relieving the programmers all memory management problems (i.e., deallocation is completely automatic).
- Java incorporates exception handling which captures series of errors and eliminates any risk of crashing the system.
- Architecture neutral: Java is the language that is not tied to any particular hardware or operating system. Program developed in java can be executed anywhere on any system. You can "write once, run anywhere, anytime forever". Changes & upgrades in operating system, processors will not force any changes in java program. It works on Macintosh PC, UNIX & whatever the future platforms can offer.
- Interpreted: Java accomplishes architecture neutrality by compiling the java source code into an intermediate code called "byte codes", which can be interpreted on any system that has a proper java runtime on it. Multithreaded: Java supports multithreaded programming which allows you to write programs that do many things simultaneously.
- Easy to learn: The language features feel like the natural way to do things & encourage good programming style. Since object model is both mandatory & simple, you will quickly become acquainted with object oriented style of programming.

3. Explain : i) >>> ii) short circuit logical operators iii) for each. (6M) [July 2014]

i) int num,den;

if(den I= 0 && num |den >2){

}

ii) int num,den;

if(den I=0 & num | den == 0){

}

An object is a single instance of a class that retains the structure and behavior as defined by the class. These objects are some times called *instances of a class*

4. **Describe the process of building and running Java program. (4M) [July 2014, July 2016]**

   BYTE Code is intermediate level code, which is interpreted by the JVM. It is not directly executable on the machine. This gives java it's "write once and run anywhere" nature. When java program is written and compiled then it will create a .class file which consists of byte code instructions, understandable to JVM. This class file is system independent. Every system has it's own JVM.so jvm will convert this byte code into machine language understandable to that system. So it has run write once and run anywhere nature.

   Java achieves architecture neutrality in the following way. Being platform independent was one of the major objectives for java. Java achieves this independence by introducing an intermediate code representation of compiled java programs. Programs are compiled into a byte code which is then interpreted by platform specific interpreter. The byte code is same for any architecture, IBM compatible, Apple, Sparc, Sun Solaris.

   Java program

   Java compiler

   The Java Virtual Machine, or JVM, is an abstract computer that runs compiled Java programs. The JVM is "virtual" because it is generally implemented in software on top of a "real" hardware platform and operating system. All Java programs are compiled for the JVM. Therefore, the JVM must be implemented on a particular platform before compiled Java programs will run on that platform

5. **Explain arrays in java with examples(6M) [July 2016,July 2014 Jan 2015]**

   Arrays in Java are actual objects that can be passed around and treated just like other objects.

Arrays are a way to store a list of items. Each slot of the array holds an individual element, and you can place elements into or change the contents or those slots as you need to.

Three steps to create an array:

1. Declare a variable to hold the array.
2. Create a new array object and assign it to the array variable.
3. Store things in that array.

E.g.

```
String[] names;
names = new String[10];
names [1] =     "n1";
names[2] =      'n2';
```

6. What is jump statement? (4M) [July 2014,July 15]

- The if conditional, which enables you to execute different bits of code based on a simple test in Java, is nearly identical to if statements in C.
- if conditionals contain the keyword if, followed by a boolean test, followed by a statement (often a block statement) to execute if the test is true:
- if (x < y)

  System.out.println("x is smaller than y");

An optional else keyword provides the statement to execute if the test is false:

```
if (x < y)
System.out.println("x is smaller than y"); else
System.out.println("y is bigger");
```

7. Discuss break and continue(5M) [Jan 2015,Jan 2016]

An alternative to using the if and else keywords in a conditional statement is to use the conditional operator, sometimes called the ternary operator.

The *conditional operator* is a *ternary operator* because it has three terms.

Syntax : test ? trueresult : falseresult

The *test* is an expression that returns true or false, just like the test in the if statement. If the test is true, the conditional operator returns the value of *trueresult;* if it's false, it returns the value of *falseresult.* For example, the following conditional tests the values of x and y, returns the smaller of the two, and assigns that value to the variable smaller:

int smaller = x < y ? x : y;

The conditional operator has a very low precedence; that is, it's usually evaluated only after all its subexpressions are evaluated. The only operators lower in precedence are the assignment operators..

8. Explain about JDK(7M) [July 2014,July 15,Jan 2016]

The Java Virtual Machine, or JVM, is an abstract computer that runs compiled Java programs. The JVM is "virtual" because it is generally implemented in software on top of a "real" hardware platform and operating system. All Java programs are compiled for the JVM. Therefore, the JVM must be implemented on a particular platform before compiled Java programs will run on that platform

Java achieves architecture neutrality in the following way. Being platform independent was one of the major objectives for java. Java achieves this independence by introducing an intermediate code representation of compiled java programs. Programs are compiled into a byte code which is then interpreted by platform specific interpreter. The byte code is same for any architecture, IBM compatible, Apple, Sparc, Sun Solaris.

Java program

Java compiler

9. Discuss three OOP principles(6M) [Jan 2015,July 15]

In java basis of encapsulation is a *class*. You create a class that represents an abstraction for a set of objects that share the same structure and behavior. An object is a single instance of a

class that retains the structure and behavior as defined by the class. These objects are some times called *instances of a class*. The individual or data representation of a class is defined by a set of *instance variables*. These variables hold the dynamic state of each instance of a class. The behavior of a class is defined by methods that operate on that instance data. A method is a message to take some action on an object. Since the goal is to encapsulate complexity, there are mechanisms for hiding the data declared inside class. Each method or variable in a class may be marked private or public. You can declare private methods and instance data that can not be accessed by any other code outside the implementation of your class.

Inheritance: Inheritance is the process by which object of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. For example, the bird robin is a part of the class flying bird, which is again part of a class bird as illustrated in figure below

The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived. In OOP, the concept of inheritance provides the idea of reusability. The new class will have the combined features of both the classes. Java is said to be single inheritance language. Multiple inheritance which is explicitly not a feature of java

Polymorphism: Polymorphism means ability take more than one form or polymorphism means one object, many shapes, a simple concept that allows a method to have multiple implementations that are selected based on the number & type of the arguments passed into the method invocation. This is known as method overloading. Figure illustrate the method overloading

Listed below are some major C++ features that were intentionally omitted from java

· Java does not support global variables. it is impossible to create a global variable that is outside of all the classes.

· Java has no goto statement

· Java does not use pointers or addresses in memory, because pointers are unsafe. Improper pointer arithmetic causes most of the bugs in today's code.

UNIT 2

CLASSES, INHERITANCE, EXCEPTIONS, APPLETS

1. Describe the significance of final and super, with examples. (6M) [Jan 2014, Jan 2015,Jan 2016]

When Sun was designing Java, it omitted multiple inheritance - or more precisely multiple implementation inheritance - on purpose. Yet multiple inheritance can be useful, particularly when the potential ancestors of a class have orthogonal concerns. This article presents a utility class that not only allows multiple inheritance to be simulated, but also has other far-reaching applications.

Here, Person is a concrete class that represents a person, while Employment is another concrete class that represents the details of a person who is employed. If you could only put them together, you would have everything necessary to define and implement an Employee class. Except in Java - you can't. Inheriting implementation from more than one superclass - multiple implementation inheritance - is not a feature of the language. Java allows a class to have a single superclass and no more.

2. What is an exception? Explain the different exception handling mechanisms, with an example, (8M) [Jan 2014, July 2015]

This is the general form of an exception-handling block:

```
try
{
//block of code to be monitored for errors
}
catch (ExceptionType1 exOb )
{
//exception handler for ExceptionType1
}
catch (ExceptionType2 exOb )
{
```

```
//exception handler for ExceptionType2
}
//...
finally
{
//block of code to be executed before try block ends
}
```

DIVIDE-BY-ZERO EXCEPTION

This small program has an expression that causes a divide-by-zero error.

```
class DivideByZero
{
public static void main (String args[])
{
int d = 0;
int a = 42 / d;
}
}
```

The Java run-time system constructs a new exception object when it detects an attempt to divide-by-zero. It then throws this exception. In the above example, there is no exceptional handler to catch the exception. The default handler provided by the Java run-time system will process any exception that is not caught by a program. The output of the above program when executed by the standard Java JDK runtime interpreter:

```
java.lang.ArithemeticException:/ by zero
at DivideByZero.main(DivideByZero.java:4)
```

Although the default exception handler is provided by Java run-time system, you will usually like to handle exception yourself.

The following program includes a try block and a catch clause, which processes the ArithmeticException generated by the division-by-zero:

```
class DivideByZero
{
```

```
public static void main (String args[])
{
int d, a;
try //monitor a block of code
{
d = 0;
a = 42 / d;
System.out.println("This will not be printed. ");
}
catch(ArithmeticException e)
{
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
```

This program generates the following output:

Division by zero.

After catch statement

The call to println() inside the try block is never executed .Once an exception is thrown, program control transfers out of the try block into the catch block. Once the catch statement has executed, the program continues with the next line in the program following the entire try/

catch mechanism.

3. **Write an applet program to display the message "VTU BELGAUM". Set the background color to cyan and foreground color to red. (6M) [Jan 2014,Jan 2016, July 2016]**

The applet parameter "Message" is the string to be drawn.

```
import java.applet. *;
import java.awt.*;
public class DrawStringApplet extends Applet {
 private String defaultMessage = "Hello!"; public
 void paint(Graphics g) {
String inputFromPage = this .getParameter("Mes
 sage");
   if (inputFromPage == null) inputFromPage = defaultMessage;
   g.drawString(inputFromPage, 50, 25);
  }
}
```

HTML file that references the above applet.

```
<HTML> <HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="3 00" height="50">
   <PARAM name="Message" value="Howdy, there!"> This page will be very boring if your
   browser doesn't understand Java.
   </APPLET>
   </BODY> </HTML>
```

4. Why overriding methods are used in java? Example(8M) [July 2016, July 2015]

Overriding Methods

- When a method is called on an object, Java looks for that method definition in the class of that object, and if it doesn't find one, it passes the method call up the class hierarchy until a method definition is found.

- Method inheritance enables you to define and use methods repeatedly in subclasses without having to duplicate the code itself.
- However, there may be times when you want an object to respond to the same methods but have different behavior when that method is called. In this case, you can override that method. Overriding a method involves defining a method in a subclass that has the same signature as a method in a superclass. Then, when that method is called, the method in the subclass is found and executed instead of the one in the superclass.

5. What is meant by instance variable hiding? How to overcome it?

public static void main (String args[]) { Motorcycle m = new Motorcycle(); m.make = "Yamaha RZ350"; (4M) [July 2014,Jan 2016]

```
m.color = "yellow";
System.out.println("Calling showAtts..."); m.showAtts(); System.out.println("———
— "); System.out.println("Starting
engine..."); m.startEngine();
System.out.println("————————");
System.out.println("Calling showAtts..."); m.showAtts(); System.out.println("———
— "); System.out.println("Starting
engine..."); m.startEngine();
}
```

With the main() method, the Motorcycle class is now an application, and you can compile it again and this time it'll run. Here's how the output should look:

```
Calling showAtts...
This motorcycle is a yellow Yamaha RZ350 The engine is off.
Starting engine... The engine is now on.
Calling showAtts...
This motorcycle is a yellow Yamaha RZ350 The engine is on.
Starting engine...
The engine is already on.
```

6. Explain constructor method.how it differs from other member function(5M) [Jan 2014,July 2014]

The program looks like this now :

```
class Motorcycle {                engine is already
String make;                      on.");
String color;                     engine is now
boolean engineState;              on.");
void startEngine() {
if (engineS tate == true)
System.out.println("The
else {
}
```

The showAtts method prints the current values of the instance variables in an instance of your Motorcycle class. Here's what it looks like:

```
void showAtts() {
System. out .println ("This motorcycle is a " + color + " " + make);
if (engineState == true)
System.out.println("The engine is on."); else System.out.println("The engine is off.");
```

The showAtts method prints two lines to the screen: the make and color of the motorcycle object, and whether or not the engine is on or off.

7. What is an applet? Different stages of an applet(8M) [July 2014,Jan 2015 Jan 2016]

Applets are used to provide interactive features to web applications that cannot be provided by HTML. Since Java's bytecode is platform independent, Java applets can be executed by browsers for many platforms, including Windows, Unix, Mac OS and Linux. There are open source tools like applet2app which can be used to convert an applet to a stand alone Java application/windows executable/linux executable. This has the advantage of running a Java applet in offline mode without the need for internet browser software. Life Cycle of an

Applet: Basically, there are four methods in the Applet class on which any applet is built.
init: This method is intended for whatever initialization is needed for your applet.
It is called after the param attributes of the applet tag.
· start: This method is automatically called after init method. It is also called
whenever user returns to the page containing the applet after visiting other pages
· stop: This method is automatically called whenever the user moves away from the
page containing applets. You can use this method to stop an animation.
· destroy: This method is only called when the browser shuts down normally

8. Difference between method overloading & overriding(6M) [July 2015]

Overriding Methods

- When a method is called on an object, Java looks for that method definition in the class of that object, and if it doesn't find one, it passes the method call up the class hierarchy until a method definition is found.
- Method inheritance enables you to define and use methods repeatedly in subclasses without having to duplicate the code itself.

overriding

However, there may be times when you want an object to respond to the same methods but have different behavior when that method is called. In this case, you can override that method. Overriding a method involves defining a method in a subclass that has the same signature as a method in a superclass. Then, when that method is called, the method in the subclass is found and executed instead of the one in the superclass

UNIT 3

MULTI THREADED PROGRAMMING, EVENT HANDLING

1. What is synchronization? Explain with an example, how synchronization is implemented in Java. (6M) [Jan 2014, July 2016, Jan 2015]

One further area of concern within threads is known as "busy waiting." This is the situation where a thread is conceptually idle, perhaps waiting for some other synchronous processing to complete, but yet it is still occupying the CPU. To illustrate, consider a spell-checking thread.

It reads a text file and searches for each word in its database. If a word is not found, the thread composes a list of suggested corrections and notifies the calling process. The calling process displays a list of the while the thread waits. Eventually the user makes a selection and the calling process allows the thread to continue. The thread eventually terminates, once the whole file has been processed.

We might write such code by using a public boolean variable, 'paused,' like so:

```
paused = true;
parent.processWordList (aListOfWords);
// loop until parent process clears 'paused'
while (paused && !terminated) {}
// Continue
```

The thread simply loops continually until the paused variable is set to false. Although this seems intuitive, the ramifications are that the CPU will continue to spend significant time processing the loopingtime that could best be spent servicing other threads. Indeed, any attempt to move windows around on the screen will be noticeably jerky.

```
void sleep (long milliseconds)
void sleep (long milliseconds, int nanoseconds)
```

These methods make the thread pause for the specified number of milliseconds and/or nanoseconds. What is important here is that the thread really does pause and takes no CPU time. Not all operating systems support time resolutions as small as nanoseconds, and in

these cases the second method simply rounds the number of nanoseconds to the nearest millisecond and calls the first method.

The sleep methods have been defined in the Java class libraries as being able to throw an InterruptedException. This exception is generated if the sleep method is disturbed in some way before it completes, such as if System.exit() is called from elsewhere in the application, shutting the program down. Whether or not you wish to perform any processing to respond specifically to a sleep being interrupted, Java mandates the exception be caught, hence your calls to the sleep methods should be wrapped in a try/catch block. This exception was designed to provide a general mechanism to allow one thread to implement another. Unfortunately this has not yet been fully implemented.

The thread class provides a method, interrupt(), which sends an interruption to a specified threadpresently this amounts to nothing more than setting a boolean flag. The boolean function isInterrupted() may be used to query the status of this flag, but unfortunately there is not presently any way to actually interrupt a thread and throw the InterruptedException. So despite the fact that the exception must be caught it currently isn't useful for anything. Eventually, it will permit threads to be woken up from their sleep. Because the sleep method is not presently interruptible, the alternative is to have brief periods of inactivity (sleeping), before querying the paused status. Our code fragment thus becomes:

```
paused = true;
parent.processWordList (aListOfWords);
// loop until parent process clears 'paused'
while (paused && !terminated)
try { sleep (2000); }
catch (InterruptedException e) {}
// Continue
```

This code tells the thread to sleep for two seconds (2000 milliseconds) in the body of the while loop. The thread continues to loop but puts much less strain on the CPU. In fact, the thread's awake time is greatly reduced.

You have to be careful, though, not to make the sleep time too long or the thread will not respond swiftly once the pause flag has been cleared. Because you are using the thread's

sleep method, you have to catch the exception that could be raised (or javac will complain), but you don't need to specify any code in the body of the exception handler.

2. What is producer – consumer problem? Explain the solution for producer – consumer problem with a program. (8M) [Jan 2014,July 2016, Jan 2015]

Problem Description :

In computer science the producer-consumer problem (also known as the **bounded-buffer problem)** is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a
common, fixed-size buffer.

The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e. removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. This is the code for solving the above stated:

```
class BufferItem {
public volatile double value = 0;  // multiple threads access public volatile boolean occupied = false; // so make these `volatile' }
class BoundedBuffer { // designed for a single producer thread and // a single consumer thread
private int numSlots = 0;
private BufferItem[] buffer = null;
private int putIn = 0, takeOut = 0;
// private int count = 0;
public BoundedBuffer(int numSlots) {
if (numSlots <= 0) throw new IllegalArgumentException("numSlots<=0"); this.numSlots = numSlots;
buffer = new BufferItem[numSlots];
for (int i = 0; i < numSlots; i++) buffer[i] = new BufferItem();
putIn = (putIn + 1) % numSlots;
```

```
// count++;// race condition!!! }
public double fetch() {
double value;
while (!buffer[takeOut].occupied) // busy wait Thread.currentThread().yield();
value = buffer[takeOut].value;     // C
buffer [takeOut] .occupied = false;          // D takeOut = (takeOut + 1) % numSlots;
//            count--;  // race condition!!! return value;
}
}
```

3. What is delegation event model? Describe the significance of adapter class, with an example. (6M) [Jan 2014, Jan 2016,July 2016]

In Java, events represent all activity that goes on between the user and the application.

Two event handling mechanisms :

Delegation event model : It defines standard and consistent mechanisms to generate and process events. Here the source generates an event and sends it to on or more listeners. The listener simply waits until it receives an event. Once it is obtained, It processes this event and returns. Listeners should register themselves with a source in order to receive an even notification. Notifications are sent only to listeners that want to receive them.

Events

In the delegation model, an *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are : pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse. Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed.

Event Classes

The classes that represent events are at the core of Java's event handling mechanism. **EventObject :** It is at the root of the Java event class hierarchy in **java.util.** It is the

superclass for all events. Its one constructor is shown here: EventObject(Object *src)* Here, *src* is the object that generates this event. EventObject contains two methods: getSource( ) and toString( ). The getSource( ) method returns the source of the event.

EventObject is a superclass of all events.

The ActionEvent Class :

An **ActionEvent** is generated when a button is pressed, a list item is double-clicked, or a menu item is selected. The **ActionEvent** class defines four integer constants that can be used to identify any modifiers associated with an action event: **ALT_MASK, CTRL_MASK, META_MASK,** and **SHIFT_MASK.**

ActionEvent has these three constructors: ActionEvent(Object *src,*

int *type,* String *cmd)* ActionEvent(Object *src,* int *type,*

String *cmd,* int *modifiers)*

ActionEvent(Object *src,* int *type,* String *cmd,* long *when,* int *modifiers)* Here, *src* is a reference to the object that generated this event. The type of the event is specified by *type,* and its command string is *cmd.* The argument *modifiers* indicates which modifier keys (ALT, CTRL, META, and/or SHIFT) were pressed when the event was generated. The *when* parameter specifies when the event occurred

- **The AdjustmentEvent Class** An **AdjustmentEvent** is generated by a scroll bar
- **The ComponentEvent Class** A **ComponentEvent** is generated when the size, position, or visibility of a component is changed. There are four types of component events
- **The ContainerEvent Class** A **ContainerEvent** is generated when a component is added to or removed from a container
- **The FocusEvent Class :** A **FocusEvent** is generated when a component gains or loses input focus
- **The InputEvent Class :** The abstract class **InputEvent** is a subclass of **ComponentEvent** and is the superclass for component input events. Its subclasses are **KeyEvent** and **MouseEvent.**
- **The ItemEvent Class :** An **ItemEvent** is generated when a check box or a list item is clicked or when a checkable menu item is selected or deselected
- **The KeyEvent Class** A **KeyEvent** is generated when keyboard input occurs.

- **The MouseEvent Class** There are eight types of mouse events
- **The MouseWheelEvent Class** The **MouseWheelEvent** class encapsulates a mouse wheel event.
- **The TextEvent Class** Instances of this class describe text events. These are generated by text fields and text areas when characters are entered by a user or program.
- **The WindowEvent Class** There are ten types of window events. The **WindowEvent** class defines integer constants that can be used to identify them.

4. What is meant by multithreaded programming? Explain with an example interthread communication(10M) [July 2014, Jan 2014,Jan 2016]

A thread is a single path of execution of code in a program.

- A Multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a Thread.
- Each thread defines a separate path of execution. Multithreading is a specialized form of Multitasking.

to make the classes threadable

A class can be made threadable in one of the following ways

(1) implement the Runnable Interface and apply its run() method.

(2) extend the Thread class itself.

1. Implementing Runnable Interface: The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run().

The Format of that function is public void run().

2. Extending Thread: The second way to create a thread is to create a new class that extends the Thread class and then to create an instance of this class. This class must override the run() method which is the entry point for the new thread.

5. What is meant by thread priority? How it is assigned(6M) [July 2014 Jan 2015]

The example in the next segment demonstrates the use of Runnable and its implementation.

Synchronization

Two or more threads accessing the same data simultaneously may lead to loss of data integrity. In order to avoid this java uses the concept of monitor. A monitor is an object used as a mutually exclusive lock.

At a time only one thread can access the Monitor. A second thread cannot enter the monitor until the first comes out. Till such time the other thread is said to be waiting.

The keyword Synchronized is use in the code to enable synchronization and it can be used along with a method.

Changing the state of thread

There might be times when you need to temporarily stop a thread from processing and then resume processing, such as when you want to let another thread use the current resource. You can achieve this objective by defining your own suspend and resume methods, as shown in the following example. This example defines a MyThread class. The MyThread class defines three methods: the run() method, the suspendThread() method, and the resumeThread() method. In addition, the MyThread class declares the instance variable suspended, whose value is used to indicate whether or not the thread is suspended.

```
class MyThread implements Runnable {
String name;
Thread t;
boolean suspended;
MyThread() {
t = new Thread(this, "Thread");
suspended = false ; t.start();
}
public void run() {
try {
for (int i = 0; i < 10; i++) { System.out.println("Thread: " + i ); Thread.sleep(200);
synchronized (this) {
while (suspended) {
```

```
wait();
}
}
}
} catch (InterruptedException e ) { System.out.println("Thread: interrupted."); }
System.out.println("Thread exiting.");
}
void suspendThread() { suspended = true;
}
synchronized void resumeThread() {
suspended = false;
notify();
}
}
class Demo {
public static void main (String args [] ) { MyThread t1 = new MyThread();
try{
Thread.sleep(1000); t1.suspendThread(); System.out.println("Thread: Suspended");
Thread.sleep(1000);
t1.resumeThread(); System.out.println("Thread: Resume");
} catch ( InterruptedException e) {
}
try {
t1.t.join();
} catch ( InterruptedException e) { System.out.println (
"Main Thread: interrupted"); }
}
}
```

6. Explain action event class & adjustment event class(4M) [July 2014 Jan 2015]

Event Listener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package.

When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

Interface Description

ActionListener - Defines one method to receive action events.

AdjustmentListener - Defines one method to receive adjustment events.

ComponentListener - Defines four methods to recognize when a component is hidden, moved, resized, or shown.

ContainerListener - Defines two methods to recognize when a component is added to or removed from a container.

FocusListener - Defines two methods to recognize when a component gains or loses keyboard focus.

ItemListener - Defines one method to recognize when the state of an item changes.

KeyListener - Defines three methods to recognize when a key is pressed, released, or typed.

MouseListener - Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.

MouseMotionListener - Defines two methods to recognize when the mouse is dragged or moved.

MouseWheelListener - Defines one method to recognize when the mouse wheel is moved.

TextListener - Defines one method to recognize when a text value changes.

WindowFocusListener - Defines two methods to recognize when a window gains or loses input focus.

WindowListener Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened

7. Short notes on static, Final, Abstract, Native (10M) [July 2015,Jan 2016]

A nested class has the same behavior as any static member of a class. You can have access to it without initializing the *parent* class, and they can access the*parent* class static methods and variables also. Nested classes are always define with the keyword **static** (and we will see later that this is what differentiate them from the inner classes). An access tag (i.e. : public, protected or private) can be defined, but by default a nested class takes the default package access. Sun considers nested classes to be top-level classes (I find this at times to be confusing). Here is an example of how to define a nested class MyInner in the class enclosing class MyOuter.

Notice that when you compile this code, you will have two .class file as the output :

MyOuter.class : being the enclosing class.

MyOuter$MyInner.class : being the inner class. Since class MyInner is a static member of MyOuter, it can be access from anywhere in your code using MyOuter.MyInner (The same way that you access classes in packages, you can even use the import statement with nested classes : import MyOuter.MyInner).

8. What is a thread ? explain 2 ways of creating thread(10M) [July 2014,July 2015]

A thread is a single path of execution of code in a program.

- A Multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a Thread.
- Each thread defines a separate path of execution. Multithreading is a specialized form of Multitasking.

to make the classes threadable

A class can be made threadable in one of the following ways

(3) implement the Runnable Interface and apply its run() method.

(4) extend the Thread class itself.

3. Implementing Runnable Interface: The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called run().

The Format of that function is public void run().

4. Extending Thread: The second way to create a thread is to create a new class that extends the Thread class and then to create an instance of this class. This class must override the run() method which is the entry point for the new thread.

UNIT 4

SWINGS

1. **What is swing? List the main swing features. Explain the different types of panes of swing containers. (10M) [Jan 2014, July 2016, July 2015]**

Swing is built on top of AWT and is entirely written in Java, using AWT's lightweight component support. In particular, unlike AWT, t he architecture of Swing components makes it easy to customize both their appearance and behavior. Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs. For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars.

Component set (subclasses of JComponent) Support classes, Interfaces

Swing Components and Containers

Swing components are basic building blocks of an application. Swing toolkit has a wide range of various widgets. Buttons, check boxes,sliders, list boxes etc. Everything a programmer needs for his job. In this section of the tutorial, we will describe several useful components.

JLabel Component

JLabel is a simple component for displaying text, images or both. It does not react to input events.

JCheckBox

JCheckBox is a widget that has two states. On and Off. It is a box with a label JSlider

JSlider is a component that lets the user graphically select a value by sliding a knob within a bounded interval

JComboBox

Combobox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request.

JProgressBar

A progress bar is a widget that is used, when we process lengthy tasks. It is animated so that the user knows, that our task is progressing

JToggleButton

2. **Create a swing application having two buttons named alpha and beta. When either of the buttons pressed, it should display "alpha pressed" and "beta pressed" respectively. (10M) [Jan 2014,July 2015,Jan 2016]**

Create a JLabel with an image icon

```
import java.awt.FlowLayout;
import java.awt.HeadlessException;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class Main extends JFrame {
public Main() throws HeadlessException {
setSize(300, 300);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new FlowLayout(FlowLayout.LEFT));
Icon icon = new ImageIcon("a.png");
JLabel label1 = new JLabel("Full Name :", icon, JLabel.LEFT);
JLabel label2 = new JLabel("Address :", JLabel.LEFT);
label2.setIcon(new ImageIcon("b.png"));
getContentPane().add(label1);
getContentPane().add(label2);
}
```

```
public static void main(String[] args) {
new Main().setVisible(true);
}
}
```

3. **List the different types of swing buttons. Write a program to create four types of buttons on JApplet. Use suitable events to show actions on the buttons and use JLabel to display the action invoked. (10M) [Jan 2014, July 2016]**

```
package com.ack.gui.swing.simple;
import java.awt.*;
import java.awt.event.WindowAdapter; import java.awt.event.WindowEvent; import javax.swing.*;
public class SimpleSwingButtons extends JFrame {
public static void main( String[] argv ) {
SimpleSwingButtons myExample = new SimpleSwingButtons( "Simple Swing Buttons" );
}
public SimpleSwingButtons( String title ) {
super( title );
setSize( 150, 150 );
add WindowListener( new WindowAdapter() { public void windowClosing( WindowEvent we ) { dispose();
System.exit( 0 );
}
} );
init();
setVisible( true );
}
private void init() {
JPanel my_panel = new JPanel();
```

```
my_panel.setLayout( new GridLayout( 3, 3 ) ); for( int i = 1; i < 10; i++ ) {
ImageIcon icon = new ImageIcon( i + ".gif" ); JButton jb = new JButton( icon );
jb.setToolTipText( i + ".gif" );
my_panel.add( jb );
}
getContentPane().add( my_panel );
my_panel.setBorder( BorderFactory.createEtchedBorder() );
}
}
```

4. **Write the steps to create Jtable. WAP to create a table with the column headings Name, USN, age, address & insert records and display(10M) [July 2014,Jan 2015]**

The JTable is used to display and edit regular two-dimensional tables of cells. The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily. For example, to set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel() {
   public int getColumnCount() { return 10; }
   public int getRowCount() { return 10;}
   public Object getValueAt(int row, int col) { return new Integer(row*col); }
};
JTable table = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(table);
```

Note that if you wish to use a JTable in a standalone view (outside of a JScrollPane) and want the header displayed, you can get it using getTableHeader() and display it separately.

To enable sorting and filtering of rows, use a RowSorter. You can set up a row sorter in either of two ways:

- Directly set the RowSorter. For example: table.setRowSorter(new TableRowSorter(model)).

- Set the autoCreateRowSorter property to true, so that the JTable creates a RowSorter for you. For example: setAutoCreateRowSorter(true).

When designing applications that use the JTable it is worth paying close attention to the data structures that will represent the table's data. The DefaultTableModel is a model implementation that uses a Vector ofVectors of Objects to store the cell values. As well as copying the data from an application into the DefaultTableModel, it is also possible to wrap the data in the methods of the TableModel interface so that the data can be passed to the JTable directly, as in the example above. This often results in more efficient applications because the model is free to choose the internal representation that best suits the data. A good rule of thumb for deciding whether to use the AbstractTableModel or the DefaultTableModel is to use the AbstractTableModel as the base class for creating subclasses and the DefaultTableModel when subclassing is not required.

The "TableExample" directory in the demo area of the source distribution gives a number of complete examples of JTable usage, covering how the JTable can be used to provide an editable view of data taken from a database and how to modify the columns in the display to use specialized renderers and editors.

The JTable uses integers exclusively to refer to both the rows and the columns of the model that it displays. The JTable simply takes a tabular range of cells and uses getValueAt(int, int) to retrieve the values from the model during painting. It is important to remember that the column and row indexes returned by various JTable methods are in terms of the JTable (the view) and are not necessarily the same indexes used by the model.

By default, columns may be rearranged in the JTable so that the view's columns appear in a different order to the columns in the model. This does not affect the implementation of the model at all: when the columns are reordered, the JTable maintains the new order of the columns internally and converts its column indices before querying the model.

So, when writing a TableModel, it is not necessary to listen for column reordering events as the model will be queried in its own coordinate system regardless of what is happening in the view. In the examples area there is a demonstration of a sorting algorithm making use of exactly this technique to interpose yet another coordinate system where the order of the rows is changed, rather than the order of the columns.

5. Difference between swings and AWT(10M) [Jan 2015,July 2014,Jan 2016]

public class JTextField

extends JTextComponent

implements SwingConstants

JTextField is a lightweight component that allows the editing of a single line of text. For information on and examples of using text fields, JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class. The superclass should be consulted for additional capabilities.

JTextField has a method to establish the string used as the command string for the action event that gets fired. The java.awt.TextField used the text of the field as the command string for the ActionEvent.JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

The method setEchoChar and getEchoChar are not provided directly to avoid a new implementation of a pluggable look-and-feel inadvertently exposing password characters. To provide password-like services a separate class JPasswordField extends JTextField to provide this service with an independently pluggable look-and-feel.

The java.awt.TextField could be monitored for changes by adding a TextListener for TextEvent's. In the JTextComponent based components, changes are broadcasted from the model via a DocumentEvent toDocumentListeners. The DocumentEvent gives the location of the change and the kind of change if desired. The code fragment might look something like:

```
DocumentListener myListener = ??;
JTextField myArea = ??;
myArea.getDocument().addDocumentListener(myListener);
```

The horizontal alignment of JTextField can be set to be left justified, leading justified, centered, right justified or trailing justified. Right/trailing justification is useful if the required size of the field text is smaller than the size allocated to it. This is determined by

the setHorizontalAlignment and getHorizontalAlignment methods. The default is to be leading justified.

How the text field consumes VK_ENTER events depends on whether the text field has any action listeners. If so, then VK_ENTER results in the listeners getting an ActionEvent, and the VK_ENTER event is consumed. This is compatible with how AWT text fields handle VK_ENTER events. If the text field has no action listeners, then as of v 1.3 the VK_ENTER event is not consumed. Instead, the bindings of ancestor components are processed, which enables the default button feature of JFC/Swing to work.

Customized fields can easily be created by extending the model and changing the default model provided. For example, the following piece of code will create a field that holds only upper case characters. It will work even if text is pasted into from the clipboard or it is altered via programmatic changes.

UNIT-5

J2EE OVERVIEW, DATABASE ACCESS

1. Explain the four types of JDBC drivers. (10M) [July 2014,July 2016]

There are many possible implementations of JDBC drivers. These implementations are categorized as follows:

- **Type 1**: Drivers that implement the JDBC API as a mapping to another data access API, such as ODBC (Open Database Connectivity). Drivers of this type are generally dependent on a native library, which limits their portability. The JDBC-ODBC Bridge is an example of a Type 1 driver.

Note: The JDBC-ODBC Bridge should be considered a transitional solution. It is not supported by Oracle. Consider using this only if your DBMS does not offer a Java-only JDBC driver.

- **Type 2**: Drivers that are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect. Again, because of the native code, their portability is limited. Oracle's OCI (Oracle Call Interface) client-side driver is an example of a Type 2 driver.
- **Type 3**: Drivers that use a pure Java client and communicate with a middleware server using a database-independent protocol. The middleware server then communicates the client's requests to the data source.
- **Type 4**: Drivers that are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

Check which driver types comes with your DBMS. Java DB comes with two Type 4 drivers, an Embedded driver and a Network Client Driver. MySQL Connector/J is a Type 4 driver.

Installing a JDBC driver generally consists of copying the driver to your computer, then adding the location of it to your class path. In addition, many JDBC drivers other than Type 4 drivers require you to install a client-side API. No other special configuration is usually needed.

2. Describe the various steps of JDBC with code snippets. (10M) [Jan 2015, July 2014].

A JDBC driver allows a Java application/client to communicate with a SQL database.

2. A JDBC driver is a Java class that implements the JDBC's java.sql.Driver interface.

3. A JDBC driver converts program (and typically SQL) requests for a particular database.

Loading a JDBC Driver: Using Class.forName()

```
String className = "org.gjt.mm.mysql.Driver";
Class driverObject = Class.forName(className);
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
public class Main {
public static void main(String[] args) throws Exception {
Connection conn = getHSQLConnection();
Statement st = conn.createStatement();
st.executeUpdate("create table survey (id int,name varchar(30));");
st.executeUpdate("insert into survey (id,name ) values (1,'nameValue')");
st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM survey");
ResultSetMetaData rsMetaData = rs.getMetaData();
int numberOfColumns = rsMetaData.getColumnCount();
System.out.println("resultSet MetaData column Count=" + numberOfColumns);
rs.close();
st.close();
conn.close();
}
private static Connection getHSQLConnection() throws Exception {
Class.forName("org.hsqldb.jdbcDriver");
String url = "jdbc:hsqldb:mem:data/tutorial";
```

```
return DriverManager.getConnection(url, "sa", "");
}
}
```

Using DriverManager.registerDriver()

```
//String className = "org.gjt.mm.mysql.Driver";
try {
// Registers the given driver with the DriverManager.
DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());
// here the class is loaded
}
catch (SQLException e) {
e.printStackTrace();
}
```

To test a JDBC driver installation using Oracle

```
public class MainClass {
public static void main(String[] args) {
try {
String className = "oracle.jdbc.driver.OracleDriver";
Class driverObject = Class.forName(className);
System.out.println("driverObject=" + driverObject);
System.out.println("your installation of JDBC Driver OK.");
}
catch (Exception e) {
System.out.println("Failed: JDBC Driver Error: " + e.getMessage());
}
}
}
```

c. Write a note on database meta interface. (04 Marks)

```
public class MainClass {
public static void main(String[] args) {
```

```
try {
String className = "org.gjt.mm.mysql.Driver";
Class driverObject = Class.forName(className);
System.out.println("driverObject=" + driverObject);
System.out.println("your installation of JDBC Driver OK.");
} catch (Exception e) {
System.out.println("Failed: JDBC Driver Error: " + e.getMessage());
}
}
}
```

3.Explain : i) callable statement ii) prepared statement. (10M)[Jan 2014,Jan 2016]

The following simple code fragment gives a simple example of these three steps:

```
public void connectToAndQueryDatabase(String username, String password) {

   Connection con = DriverManager.getConnection(
                    "jdbc:myDriver:myDatabase",
                    username,
                    password);

   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

   while (rs.next()) {
      int x = rs.getInt("a");
      String s = rs.getString("b");
      float f = rs.getFloat("c");
   }
}
```

This short code fragment instantiates a DriverManager object to connect to a database driver and log into the database, instantiates a Statement object that carries your SQL language query to the database; instantiates a ResultSet object that retrieves the results of your query, and executes a simple while loop, which retrieves and displays those results. It's that simple.

4. Explain J2EE architecture(10M) [July 2016 ,Jan 2014,Jan 2016]

A tier is an abstract concept that defines a group of technologies that provides one or more services to its clients. In multi-tier architecture each tier contains services that include software object or DBMS. Multi-tier architecture is composed of clients, resources, components (service), and containers.

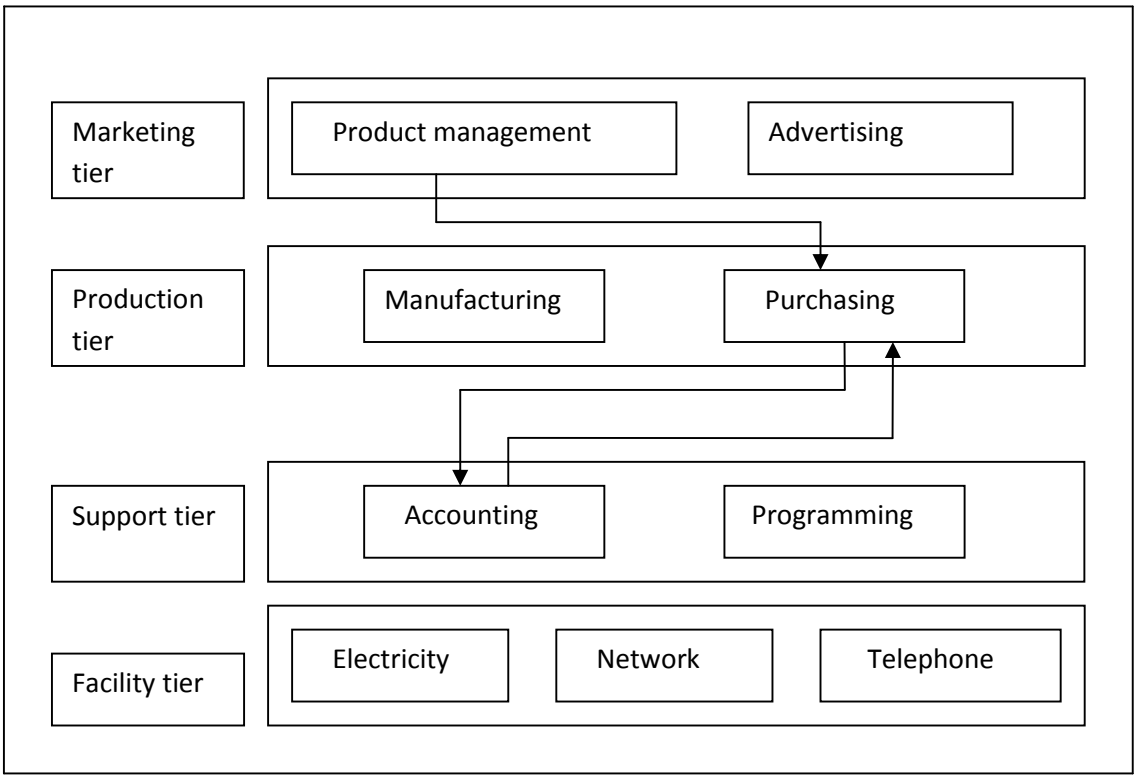
Clients, Resources and Components

A client refers to a program that requests a service from a component. A resource is anything a component needs to provide a service, and a component is part of tier that consists of a collection of classes or a program that performs a function to provide the service. A container is software that manages a component and provides a component with system services. The relationship between a container and a component is sometimes referred to as a contract, whose terms are governed by an application programming interface (API). An API defines rules a component must follow and the services a component will receive from the container.

A container handles persistence, resource management, security, threading and other system-level services for components that are associated with the container. Components are responsible for implementation of business logic. It helps the programmer to focus on coding business rules into components without becoming concerned about low-level system services. The relationship between client, component and resource is shown below.

Normally large organizations employ multi-tier architecture because it is easy to build an application that is flexible, scalable and responsive to the expectation of clients. Considering an organization that groups its services as marketing tier, production tier, support tier and facility services tier. At the lowest level facility services contains variety of resources that include electricity, elevator, computer network, and telephone services. The next tier in the organization contains support resources like computer programming, accounting, counseling etc. Production tier has the resources necessary to produce products and services sold by the

company. The highest tier consists of resources for product management and advertising. All the tiers should interact with each other for the proper functioning of the enterprise. This is similar to the tier structure in distributed systems. The tier relationship in an enterprise is given below.



5. Write a note on resultset(10M) [July 2016 Jan 2015]

A ResultSet object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database. For example, theCoffeeTables.viewTable method creates a ResultSet, rs, when it executes the query through the Statement object, stmt. Note that a ResultSet object can be created through any object that implements the Statement interface, including PreparedStatement, CallableStatement, and RowSet.

You access the data in a ResultSet object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in theResultSet.

Initially, the cursor is positioned before the first row. The method ResultSet.next moves the cursor to the next row. This method returns false if the cursor is positioned after the last row. This method repeatedly calls the ResultSet.next method with a while loop to iterate through all the data in the ResultSet. The following method, CoffeesTable.viewTable outputs the contents of the COFFEES tables, and demonstrates the use of ResultSet objects and cursors:

```
public static void viewTable(Connection con, String dbName)
throws SQLException {
    Statement stmt = null;
    String query =
        "select COF_NAME, SUP_ID, PRICE, " +
        "SALES, TOTAL " +
        "from " + dbName + ".COFFEES";

    try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + "\t" + supplierID +
                               "\t" + price + "\t" + sales +
                               "\t" + total);        }
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
    } finally {
        if (stmt != null) { stmt.close(); }
    }
```

6. Explain different type of statement object.(10M) [July 2015,Jan 2016]

JDBC Product Components

JDBC includes four components:

1. **The JDBC API** — The JDBC™ API provides programmatic access to relational data from the Java™ programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.

The JDBC API is part of the Java platform, which includes the *Java™ Standard Edition* (Java™ SE) and the *Java™ Enterprise Edition* (Java™ EE). The JDBC 4.0 API is divided into two packages: java.sql and javax.sql. Both packages are included in the Java SE and Java EE platforms.

2. **JDBC Driver Manager** — The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.

The Standard Extension packages javax.naming and javax.sql let you use a DataSource object registered with a *Java Naming and Directory Interface*™ (JNDI) naming service to establish a connection with a data source. You can use either connecting mechanism, but using a DataSource object is recommended whenever possible.

3. **JDBC Test Suite** — The JDBC driver test suite helps you to determine that JDBC drivers will run your program. These tests are not comprehensive or exhaustive, but they do exercise many of the important features in the JDBC API.
4. **JDBC-ODBC Bridge** — The Java Software bridge provides JDBC access via ODBC drivers. Note that you need to load ODBC binary code onto each client machine that uses this driver. As a result, the ODBC driver is most appropriate on a corporate network where client installations are not a major problem, or for application server code written in Java in a three-tier architecture.

This Trail uses the first two of these these four JDBC components to connect to a database and then build a java program that uses SQL commands to communicate with a

test Relational Database. The last two components are used in specialized environments to test web applications, or to communicate with ODBC-aware DBMSs

UNIT-6

SERVLETS

1. **Explain the different stages in the life cycle of a servlet. (6M) [Jan 2014, July 2016 , Jan 2015]**

Servlet Life Cycle

The life cycle of a servlet is controlled by the container in which the servlet has been deployed.

When a request is mapped to a servlet, the container performs the following steps.

1. If an instance of the servlet does not exist, the Web container

a. Loads the servlet class.

b. Creates an instance of the servlet class.

c. Initializes the servlet instance by calling the init method.

Initialization is covered in Initializing a Servlet.

2. Invokes the service method, passing a request and response object.

Service methods are discussed in the section Writing Service Methods.

If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method. Finalization is discussed in Finalizing a Servlet.

2. **What is a cookie? List out the methods defined by cookie. Write a program to add a cookie. (8M) [Jan 2014, July 2016, July 2015, Jan 2015]**

An individual developing servlet for handling HTTP Requests needs to override one of these methods in order to process the request and generate a response. The servlet is invoked dynamically when an end-user submits a form.

Example:

<form name="F1" action=/servlet/ColServlet> Select the color:

<select name = "col" size = "3">

<option value = "blue">Blue</option> <option value = "orange">Orange</option>

</select>

<input type = "submit" value = "Submit"> </form>

Here's the code for ColServlet.java that overrides the doGet() method to retrieve data

from the HTTP Request and it then generates a response as well.

```
// import the java packages that are needed for the servlet to work
import java.io .*;
import javax.servlet. *;
import javax.servlet.http. *;
// defining a class
public class ColServlet extends HttpServlet {
public void doGet(HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException
// request is an object of type HttpServletRequest and it's used to obtain information
// response is an object of type HttpServletResponse and it's used to generate a response // throws is used to specify the exceptions than a method can throw
{
String colname = request.getParameter("col");
// getParameter() method is used to retrieve the selection made by the user
response.setContentType("text/html");
PrintWriter info = response.getWriter();
info .println("The color is: ");
```

```
info .println(col);
info.close();
}
}
```

3. Write a program to describe parameter reading using servlets.(6M) [Jan 2014,Jan 2015,July 2016]

The main difference between them is, In servlets both the presentation and business logic are place it together. Where as in jsp both are saparated by defining by java beans . In jsp's the overall code is modulated so the developer who deesn't know about java can write jsp pages by simply knowing the additional tages and class names. One more important to be considered is servlet take less time to compile. Jsp is a tool to simplify theprocess and make the process automate.Both are webapplications used to produce web content that mean dynamic web pages. Bot are used to take the requests and service the clients.When the JSP engine encounters a tag extension in a JSP attranslation time, it parses the tag library descriptor to find the required tag handler class and generates code to obtain, and interact with, the taghandler.

The Tag or BodyTag interfaces, one of which must be implemented by any tag handler,

For performance reasons, JSP engines will not necessarily instantiate a new tag handler instance every time a tag is encountered in a JSP. Instead, they may maintain a pool of tag instances, reusing them where possible. When a tag is encountered in a JSP, the JSP engine will try to find a Tag instance that is not being used, initialize it, use it and release it (but not destroy it), making it available for further use. The programmer has no control over any pooling that may occur. The repeated use model is similar to a servlet lifecycle, but note one very important difference: tag handler implementations don't need to concern themselves with thread safety. The JSP engine will not use an instance of a tag handler to handle a tag unless it is free. This is good news: as with JSP authoring in general, developers need to worry about threading issues less often than when developing servlets.

4. Write a note on HTTP status codes.(4M) [July 2016,July 2015,Jan 2016]

Tomcat

· Tomcat is the Servlet Engine than handles servlet requests for Apache

o Tomcat is a “helper application” for Apache

o It’s best to think of Tomcat as a “servlet container”

· Apache can handle many types of web services

o Apache can be installed without Tomcat

o Tomcat can be installed without Apache

· It’s easier to install Tomcat standalone than as part of Apache

o By itself, Tomcat can handle web pages, servlets, and JSP

· Apache and Tomcat are open source (and therefore free)

c. With a code snippet, explain how session tracking is handled in Java with servlets. (04 Marks)

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String docType =
"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
"Transitional//EN\">\n";
out.println(docType +
"<HTML>\n" +
"<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1>Hello World</H1>\n" +
"</BODY></HTML>");
}
}
```

The superclass

· public class HelloServlet extends HttpServlet {

· Every class must extend GeneriISErvlet or a subclass of GeneriISErvlet

o GeneriISErvlet is “protocol independent,” so you could write a servlet to process any protocol

o In practice, you almost always want to respond to an HTTP request, so you extend HttpServlet

5. Explain servlet interface, generic class, cookie class (6M) [Jan 2014, July 2015,Jan 2016]

```
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String docType =
"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
"Transitional//EN\">\n";
out.println(docType +
"<HTML>\n" +
"<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1>Hello World</H1>\n" +
"</BODY></HTML>");
}
}
```

The superclass

· public class HelloServlet extends HttpServlet {

· Every class must extend GeneriISErvlet or a subclass of GeneriISErvlet

o GeneriISErvlet is “protocol independent,” so you could write a servlet to process any protocol

o In practice, you almost always want to respond to an HTTP request, so you extend HttpServlet

6. **Explain use of session information in servlets(6M) [July 2014,Jan 2016]**

In servlets both the presentation and business logic are place it together. Where as in jsp both are saparated by defining by java beans . In jsp's the overall code is modulated so the developer who deesn't know about java can write jsp pages by simply knowing the additional tages and class names. One more important to be considered is servlet take less time to compile. Jsp is a tool to simplify theprocess and make the process automate.Both are webapplications used to produce web content that mean dynamic web pages. Bot are used to take the requests and service the clients.When the JSP engine encounters a tag extension in a JSP attranslation time, it parses the tag librarydescriptor to find the required tag handler class,and generates code to obtain, and interact with, the taghandler.

The Tag or BodyTag interfaces, one of which must be implemented by any tag handler,

For performance reasons, JSP engines will not necessarily instantiate a new tag handler instance every time a tag is encountered in a JSP. Instead, they may maintain a pool of tag instances, reusing them where possible. When a tag is encountered in a JSP, the JSP engine will try to find a Tag instance that is not being used, initialize it, use it and release it (but not destroy it), making it available for further use.The programmer has no control over any pooling that may occur. The repeated use model is similar to a servlet lifecycle, but note one very important difference: tag handler implementations don't need to concern themselves with thread safety. The JSP engine will not use an instance of a tag handler to handle a tag unless it is free. This is good news: as with JSP authoring in general, developers need to worry about threading issues less often than when developing servlets.

7. **Write short nots on HTTP request & HTTP response (5M) [July 2014]**

- Servlets can be used for handling both the GET Requests and the POST Requests.

- The HttpServlet class is used for handling HTTP GET Requests as it has som specialized methods that can efficiently handle the HTTP requests. These methods are;

  doGet()

  doPost()

  doPut()

  doDelete() doOptions() doTrace() doHead()

An individual developing servlet for handling HTTP Requests needs to override one of these methods in order to process the request and generate a response. The servlet is invoked dynamically when an end-user submits a form.

Example:

<form name="F1" action=/servlet/ColServlet> Select the color:

<select name = "col" size = "3">

<option value = "blue">Blue</option> <option value = "orange">Orange</option> </select>

<input type = "submit" value = "Submit"> </form>

Here's the code for ColServlet.java that overrides the doGet() method to retrieve data from the HTTP Request and it then generates a response as well.

// import the java packages that are needed for the servlet to work

import java.io .*;

import javax.servlet. *;

import javax.servlet.http. *;

// defining a class

public class ColServlet extends HttpServlet {

public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException

// request is an object of type HttpServletRequest and it's used to obtain information

// response is an object of type HttpServletResponse and it's used to generate a response // throws is used to specify the exceptions than a method can throw

{

String colname = request.getParameter("col");

```
// getParameter() method is used to retrieve the selection made by the user
response.setContentType("text/html");
PrintWriter info = response.getWriter();
info .println("The color is: ");
info .println(col);
info.close();}}
```

UNIT-7

JSP, RMI

1. **Define JSP. Explain the different types of JSP tags by taking suitable examples. (10 Marks) [Jan 2014, July 2016 , Jan 2015]**

In JSP tags can be divided into 4 different types. These are:

Directives: In the directives we can import packages, define error handling pages or the session information of the JSP page.

Declarations: This tag is used for defining the functions and variables to be used in the JSP.

Scriplets: In this tag we can insert any amount of valid java code and these codes are placed in _jspService method by the JSP engine.

Expressions: We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

DIRECTIVES

Syntax of JSP directives is:

<%@directive attribute="value" %>

Where directive may be:

page: page is used to provide the information about it.
Example: <%@page language="java" %>

include: include is used to include a file in the JSP page.
Example: <%@ include file="/header.jsp" %>

taglib: taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags).
Example: <%@ taglib uri="tlds/taglib.tld" prefix="mytag" %> and attribute may be:
language="java"This tells the server that the page is using the java language. Current JSP specification supports only java language.
Example: <%@page language="java" %>

extends="mypackage.myclass"
This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages.
Example: <%@page language="java" import="java.sql.*,mypackage.myclass" %>

session="true"
When this value is true session data is available to the JSP page otherwise not. By default this value is true.
Example:<%@pagelanguage="java"session="true" %>

errorPage="error.jsp"
errorPage is used to handle the un-handled exceptions in the page.
Example: <%@page language="java" session="true" errorPage="error.jsp" %>

contentType="text/html;charset=ISO-8859-1"

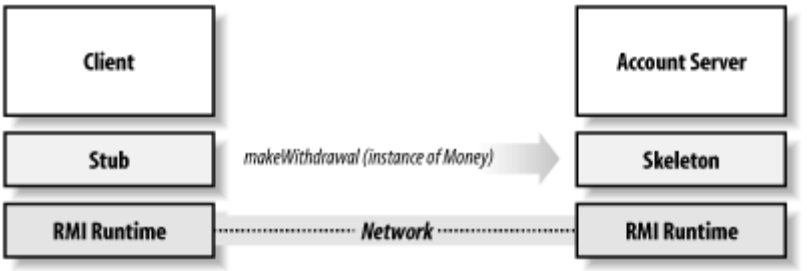
Use this attribute to set the mime type and character set of the JSP.

Example: <%@page language="java" session="true"

contentType="text/html;charset=ISO-8859-1" %>

2. **What is RMI? Describe with code snippet RMI at server side. (10M) [Jan 2014, July 2014, Jan 2015,Jan 2016]**

Serialization is the process of converting a set of object instances that contain references to each other into a linear stream of bytes, which can then be sent through a socket, stored to a file, or simply manipulated as a stream of data. Serialization is the mechanism used by RMI to pass objects between JVMs, either as arguments in a method invocation from a client to a server or as return values from a method invocation.



Server Implementations

Once the remote object's Java interface is defined, a server implementation of the interface can be written. In addition to implementing the object's interface, the server also typically extends thejava.rmi.server.UnicastRemoteObject class. UnicastRemoteObject is an extension of theRemoteServer class, which acts as a base class for server implementations of objects in RMI. Subclasses of RemoteServer can implement different kinds of object distribution schemes, like replicated objects, multicast objects, or point-to-point communications. The current version of RMI (1.1) only supports remote objects that use point-to-point communication, andUnicastRemoteObject is the only subclass of RemoteServer provided. RMI doesn't require your server classes to derive from a RemoteServer subclass, but doing so lets your server inherit specialized

implementations of some methods from Object (hashCode(), equals(), andtoString()) so that they do the right thing in a remote object scenario. If you decide that you don't want to subclass from a RemoteServer subclass for some reason, then you have to either provide your own special implementations for these methods or live with the fact that these methods may not behave consistently on your remote objects. For example, if you have two client stubs that refer to the same remote object, you would probably want their hashCode() methods to return the same value, but the standard Object implementation will return independent hash codes for the two stubs. The same inconsistency applies to the standard equals() and toString() methods.

3. **Difference between servlet and JSP?(6M) [July 2016, July 2015,Jan 2016]**

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
It's easier to code in JSP than in Java Servlets.	Its little much code to write here.
In MVC, jsp act as a view.	In MVC, servlet act as a controller.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
The advantage of JSP programming over servlets	There is no such custom tag facility in servlets.

4. **Write a note on request string(4M) [July 2014,July 2015]**

```
<html>
<head><title>First JSP</title></head>
<body>
```

```
<%
double num = Math.random();
if (num > 0.95) {
%>
<h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
<%
} else {
%>
<h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
<%
}
%>
<a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

```
<html>
<h2>You'll have a luck day!</h2>
<p>(0.987)</p>
<a href="first.jsp"><h3>Try Again</h3></a></html>
```

5. **WAP to create & read cookie called EMPIDthat has a value of AN2356 (10M) [July 2015,Jan 2016]**

Create Table

To create the Employees table in EMP database, use the following steps:

Step 1:

Open a Command Prompt and change to the installation directory as follows:

C:\>

C:\>cd Program Files\MySQL\bin

C:\Program Files\MySQL\bin>

Step 2:

Login to database as follows

C:\Program Files\MySQL\bin>mysql -u root -p

Enter password: ********

mysql>

Step 3:

Create the table Employee in TEST database as follows:

```
mysql> use TEST;
mysql> create table Employees
   (
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
   );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

Create Data Records

Finally you create few records in Employee table as follows:

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
```

Query OK, 1 row affected (0.00 sec)

mysql>

SELECT Operation:

Following example shows how we can execute SQL SELECT statement using JTSL in JSP programming:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>SELECT Operation</title>
</head>
<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/TEST"
    user="root"  password="pass123"/>

<sql:query dataSource="${snapshot}" var="result">
SELECT * from Employees;
</sql:query>

<table border="1" width="100%">
<tr>
  <th>Emp ID</th>
  <th>First Name</th>
  <th>Last Name</th>
  <th>Age</th>
```

```
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
  <td><c:out value="${row.id}"/></td>
  <td><c:out value="${row.first}"/></td>
  <td><c:out value="${row.last}"/></td>
  <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html
```

UNIT-8

ENTERPRISE JAVA BEANS

- **What is deployment descriptor? List the deployment descriptor for EJB1. (6M) [Jan 2014, July 2016, Jan 2015]**
  - **EJB structural information**, such as the EJB name, class, home and remote interfaces, bean type (session or entity), environment entries, resource factory references, EJB references, security role references, as well as additional information based on the bean type.
  - **Application assembly information**, such as EJB references, security roles, security role references, method permissions, and container transaction attributes. Specifying assembly descriptor information is an optional task that an Application Assembler performs.

Specifying deployment descriptor information is a required task that a Bean Provider performs. The Bean Provider creates a standard EJB deployment descriptor file using the XML markup conventions in accordance with the syntax described in the Enterprise JavaBeans Specification 1.1. Multiple EJBs can be defined in a single deployment descriptor.

DOCTYPE Declaration

All valid EJB JAR deployment descriptors must contain the following DOCTYPE declaration:

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

Security

The following ejb-jar.xml file overrides any @RolesAllowed, @PermitAll, or @DenyAll source code annotations for the bar method of the FooB EJB and adds a @RolesAllowed annotation for the allowed role.

<ejb-jar>

```
<assembly-descriptor>
<method-permission>
<role-name>allowed</role-name>
<method>
<ejb-name>FooB</ejb-name>
<method-name>bar</method-name>
</method>
</method-permission>
</assembly-descriptor>
</ejb-jar>
```

Transactions

The following ejb-jar.xml file overrides any @TransactionAttribute annotations for the bar method of the FooA EJB and adds a@TransactionAttribute annotation for NOT SUPPORTED.

```
<ejb-jar>
<assembly-descriptor>
<container-transaction>
<method>
<ejb-name>FooA</ejb-name>
<method-name>bar</method-name>
</method>
<trans-attribute>NotSupported</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

2. **With a skeleton, explain entity Java bean. (6M) [Jan 2014, July 2016 , Jan 2016]**

A enterprise bean is written in Java is a server side component that encapsulates business logic of an application. For example in an inventory control system, an enterprise bean might have a method named orderProduct or checkInventory level.

Session bean represents a single client inside the J2EE server. To access the application deployed in the server the client invokes methods on the session bean. The session bean performs the task shielding the client from the complexity of the business logic. Thereare two types of session beans, namely: Stateful and Stateless.

Stateful Session Bean: A state of a bean is represented by the values of its instance variables. In a stateful session bean the values of its instance variables represent the state of a client-bean session. When the client quits or leaves the bean the state is terminated.

There are only two stages in the life cycle of a stateless bean, namely i. does not exists and ii. ready. ejbActivate and ejbPassivate will never be called hence their state will nover be saved

setEntityState

create

ejbCreate

ejbPostCreate

ejbActivate

ejbPassivate

remove

ejbRemove

unsetEntityState

3. **Explain : i) JAR file ii) Stateless bean versus stateful bean. (8M) [Jan 2014, July 2016, Jan 2015]**

A JAR file allows Java runtimes to efficiently deploy a set of classes and their associated resources. The elements in a JAR file can be compressed, which, together with the ability to download an entire application in a single request, makes downloading a JAR file much more convenient than separately downloading the many uncompressed files which would form a single Java Application. The package java.util.zip contains classes that read and write JAR files.

A JAR file has an optional manifest file located in the path META-INF/MANIFEST.MF. The entries in the manifest file determine how one can use the JAR file. For instance, a Classpath entry can be used to specify other JAR files for loading with the JAR. This

entry consists of a list of absolute or relative paths to other JAR files. Although intended to simplify JAR use, in practice it turns out to be notoriously brittle, as it depends on all the relevant JARs being in the exact locations specified when the entry-point JAR was built.

Stateless Session Beans

A session bean represents work performed by a single client. That work can be performed within a single method invocation, or it may span multiple method invocations. If the work does span more than one method, the object must retain the user's object state across the method calls, and a stateful session bean would therefore be required.

Generally, stateless beans are intended to perform individual operations automatically and don't maintain state across method invocations. They're also amorphous, in that any client can use any instance of a stateless bean at any time at the container's discretion. They are the lightest weight and easiest to manage of the various EJB component configurations.

Stateful Session Beans

Stateful session beans maintain state both within and between transactions. Each stateful session bean is therefore associated with a specific client. Containers are able to save and retrieve a bean's state automatically while managing instance pools (as opposed to bean pools) of stateful session beans.

Stateful session beans maintain data consistency by updating their fields each time a transaction is committed. To keep informed of changes in transaction status, a stateful session bean implements the SessionSynchronization interface. The container calls methods of this interface while it initiates and completes transactions involving the bean.

Session beans, whether stateful or stateless, are not designed to be persistent. The data maintained by stateful session beans is intended to be transitional. It is used solely for a particular session with a particular client. A stateful session bean instance typically can't survive system failures and other destructive events. While a session bean has a container-provided identity (called its handle), that identity passes when the client removes the session bean at the end of a session. If a client needs to revive a stateful bean that has disappeared, it must provide its own means to reconstruct the bean's state.

4. List & explain EJB transaction attributes(10M) [July 2014, Jan 2013, Jan 2015]

EJB transactions are a set of concepts and a set of mechanisms that attempt to insure the integrity and consistency of a database for which multiple clients may attempt to access it and/or update it simultaneously.

An Entity EJB is a representation of business data. Its state is represented completely by the values of its persistent instance variables, that is, those that are synchronized with the persistent store (database). If a transactional operation in such an Entity EJB fails, then the system considers it to have been successfully rolled back if its underlying business data is put back to its pre-transactional state. Instance variables that do not represet data in the persistent store cannot be restored. If the EJB has container-managed persistence, then all the synchronization of the variables with the persistent store will be automatic, even after a rollback. This takes some work away from the developer, but it does imply that the EJB must be no more than an object representing a database row.

Because of the assumption above, transaction management in Entity EJBs should never be under programmer control. The EJB 2.0 specification prohibits an Entity EJB from managing its own transactions; the container must do everything. Therefore, if your transaction management requirements are beyond those offered by the container, then you can't use Entity EJBs at all. Of course, the EJB paradigm says that an Entity EJB is simply a model of business data and, as such, will never need a more complicated transaction mechanism than that offered by the container.

Basic properties of transactions can be summarized using the **ACID** mnemonic:

Atomic

All or nothing. If a transaction is interrupted, all previous steps within that transaction are undone.

Consistent

The state of objects and/or the state of tables within a database move from one consistent state to another consistent state.

Isolated

What happens within one transaction should not affect or be visible within another transaction.

Durable

The effects of a transaction are persistent.

5. **Difference between stateless and stateful session bean(4M) [July 2014, Jan 2016]**

Stateless Session Beans

A session bean represents work performed by a single client. That work can be performed within a single method invocation, or it may span multiple method invocations. If the work does span more than one method, the object must retain the user's object state across the method calls, and a stateful session bean would therefore be required.

Generally, stateless beans are intended to perform individual operations automatically and don't maintain state across method invocations. They're also amorphous, in that any client can use any instance of a stateless bean at any time at the container's discretion. They are the lightest weight and easiest to manage of the various EJB component configurations.

Stateful Session Beans

Stateful session beans maintain state both within and between transactions. Each stateful session bean is therefore associated with a specific client. Containers are able to save and retrieve a bean's state automatically while managing instance pools (as opposed to bean pools) of stateful session beans.

Stateful session beans maintain data consistency by updating their fields each time a transaction is committed. To keep informed of changes in transaction status, a stateful session bean implements the SessionSynchronization interface. The container calls methods of this interface while it initiates and completes transactions involving the bean.

Session beans, whether stateful or stateless, are not designed to be persistent. The data maintained by stateful session beans is intended to be transitional. It is used solely for a particular session with a particular client. A stateful session bean instance typically can't survive system failures and other destructive events. While a session bean has a container-provided identity (called its handle), that identity passes when the client removes the session bean at the end of a session. If a client needs to revive a stateful

session bean that has disappeared, it must provide its own means to reconstruct the bean's state.

6. **Write a note on message driven bean(4M) [July 2014,Jan 2016]**

A message-driven bean is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

- A message-driven bean's instances retain no data or conversational state for a specific client.
- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.
- A single message-driven bean can process messages from multiple clients.