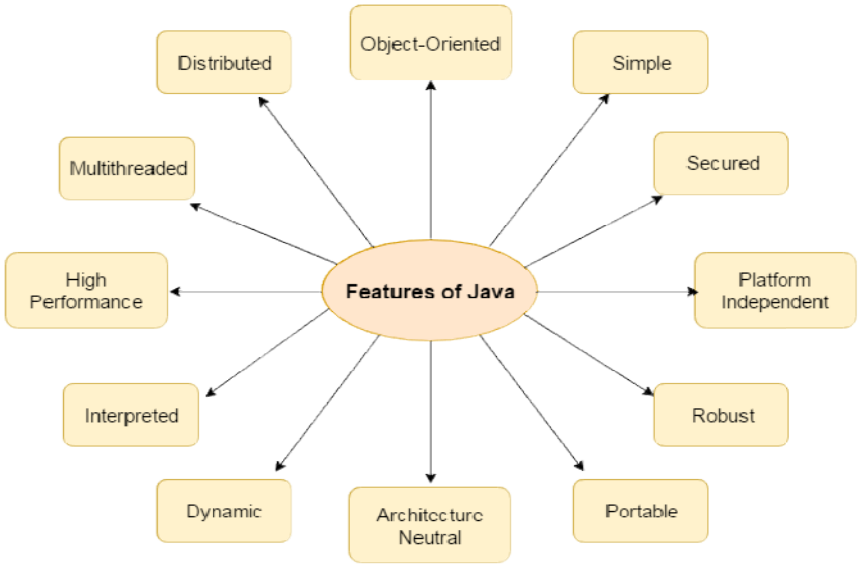
7th Sem CSE, VTU, JAVA and J2EE 10CS753
Model Answer/Solution
Dr. P. N. Singh, Professor(CSE)

1.

a. **List and explain java features/buzzwords.** **8M**

Ans:
Java features/buzzwords



Explaining the important feature:

- **Java is Object-oriented:** Everything in Java programming is discussed as an object. An object is instance of class. It is to organize our software as a combination of different types of objects that incorporates both data and behaviour
- **Java is platform independent/portable :** Executing a code irrespective of the operating system on which it is being executed.
- **Java is interpreted:** Java does not created any object code. By JVM bytecode is created on executing computer.
- **Java is simple:** Syntax is based on C++. Java is first cousin of C++ and 2nd cousin of C.
- **Java is secured:** There is no explicit pointer (but reference). Java programs inside virtual machine sandbox.
- **Java is robust:** Java uses strong memory management with automatic garbage collection.
- **Java is architecture neutral :** There is no implementation dependent features e.g. size of primitive types is fixed.
- **Java is dynamic:** Program of a computer is running on others creating virtual machine

b. Explain bitwise shift and bitwise logical operators with example. 5M

Ans:
Bitwise logical and bitwise shift operators of Java

Operator	Usage	Description
Bitwise AND	a & b	Returns a 1 in each bit position for which the corresponding bits of both operands are 1's.
Bitwise OR	a | b	Returns a 1 in each bit position for which the corresponding bits of either or both operands are 1's.
Bitwise XOR	a ^ b	Returns a 1 in each bit position for which the corresponding bits of either but not both operands are 1's.
Bitwise NOT	~ a	Inverts the bits of its operand.
Left shift	a << b	Shifts a in binary representation b (< 32) bits to the left, shifting in 0's from the right.
Sign-propagating right shift	a >> b	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off.
Zero-fill right shift	a >>> b	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off, and shifting in 0's from the left.

Important with Examples:
>>> Right shift fill zero, Internally all leading blanks will be filled with zero
25>>>2; //i.e. 6 so, 00000110 for 8 bits
Bitwise operators can be used with == assignment operator

n=25;

n>>=2; i.e. n = n>>2; //so, 6 will be stores in n

For negative number changes parity bit (MSB) to 0

System.out.println(-20>>>2);

//output should be : 1073741819 (to be checked by calculation)

c. **With the code snippets explain jump statements in Java. 7M**

Ans:

break, continue & **return** statements are also known as jump statements in Java. These statements transfer control to another part of the program.

break statement:
The break construct is used to break out of the middle of loops: for, do, or while loop. We can label a for loop by using a label name (followed by:) before the loop . It is useful if we have nested for loop so that we can break/continue specific for loop.

// code example

```
class LabeledForExample {
public static void main(String[] args) {
  aa:
    for(int i=1;i<=3;i++){
      bb:
        for(int j=1;j<=3;j++){
          if(i==2&&j==2){
            break aa;
          }
          System.out.println(i+" "+j);
        }
    }
}
}
//Output will 1  1,  1  2,   1  3, & 2  1 in separate lines
```

continue statement
This command skips the whole body of the loop and executes the loop with the next iteration. On finding continue command, control leaves the rest of the statements in the loop and goes back to the top of the loop to execute it with the next iteration (value).

```
/* Print Number from 1 to 10 Except 5 */
class NumberExcept
{
        public static void main(String args[] )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        if(i==5) continue;
                        System.out.print(i +" ");
                }
        }
}
```

return statement

This statement is mainly used in methods in order to terminate a method in between and return back to the caller method. It is an optional statement. That is, even if a method doesn't include a return statement, control returns back to the caller method after execution of the method. Return statement mayor may not return parameters to the caller method.

2.

a. **What is a constructor? Mention the properties and illustrate example to overload constructions. 8M**

Ans:

- Constructor is a special type of method having the same name as class name.
- Constructor does not have return type & does not return any value.
- Constructor is invoked implicitly as the object is created.
- The java compiler provides a default constructor if we don't have any constructor.
- **Constructor is used to initialize the object.**
- It constructs the values i.e. provides data for the object that is why it is known as constructor.

Constructor of the class can be overloaded with different number of arguments

```
// Example of overloading constructor
class Student{
  int id;
  String name;
  int age;
  Student(int i,String n){
  id = i;
  name = n;
  }
  Student(int i,String n,int a){
  id = i;
  name = n;
  age=a;
  }
  void display(){System.out.println(id+" "+name+" "+age);}
  public static void main(String args[]){
  Student5 s1 = new Student(111,"Karan");
  Student5 s2 = new Student(222,"Aryan",25);
  s1.display();
  s2.display();
 }
}
```

b. **How are exceptions handled in Java? Explain briefly with example. 6M**

Ans:

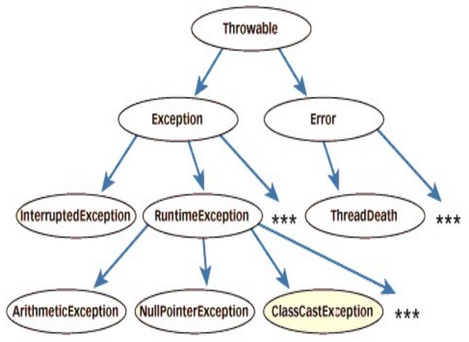
“Exceptions are run-time errors” (sometimes compile time error also)

- Exception is an abnormal condition that arises in the code sequence occur during run time.(few at compile time)

- “throwable” is the super class in exception hierarchy.
- Compile time errors occurs due to incorrect syntax.
- Examples of run-time errors(exceptions)
    - ✓ Invalid/incorrect input by user
    - ✓ Divide by zero, square root of negative value
    - ✓ Wrong resource name(example: filename)
    - ✓ Subscript out of bounds
    - ✓ Numeric overflow
    - ✓ Logical error that was not fixed

Exceptions Handling in Java

- In Java, exceptions are also objects. Objects are thrown as exception whose classes descend from *throwable*.
- throwable serves as the base class for an entire family of exception classes, declared in java.lang utility.
- Exceptions are thrown to signal abnormal conditions that can often be handled by some catcher, though it's possible they may not be caught and therefore could result in a dead thread.
- Errors are usually thrown for more serious problems, such as OutOfMemoryError, that may not be so easy to handle. In general, code you write should throw only exceptions, not errors.
- Errors are usually thrown by the methods of the Java API, or by the Java virtual machine itself.



Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

```
//Solution by try-catch block
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

o/p:
Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

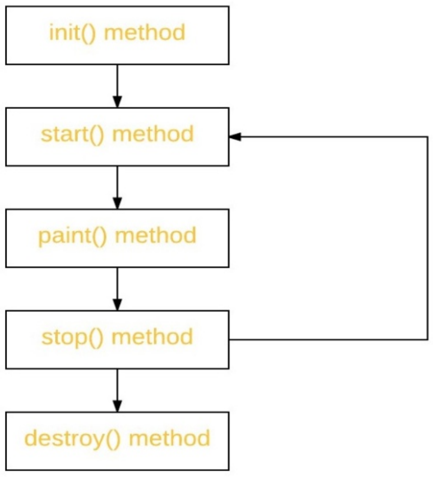
```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

o/p:
task1 completed
rest of the code...

c. What is an applet? Explain life cycle of an applet. 6M

Ans:
An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

Lifecycle of an Applet:



Life Cycle of an applet

Lifecycle methods for Applet:
The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.
java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

- public void init(): is used to initialized the Applet. It is invoked only once.
- public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
- public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

- public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

3.

a. **How do you make a class threadable? Explain with example. 7M**

Ans:

There are two approaches to making our classes behave as threads:

- **Extending the Thread class**
- **Implementing the Runnable interface**

Which approach we choose depends on our application's needs and the constraints placed on it. Giving our class the capability to run as a thread does not automatically make it run as such.
When a class is called runnable, that means it has the ability to act as a separate thread, but not necessarily that it is running as a separate thread. The reason for this is that the start method that is defined in the Runnable interface must be called to start the thread running.

Thread class provide constructors and methods to create and perform operations on a thread.

class classname extends Thread { }

Thread class extends Object class and implements Runnable interface.

class classname implements Runnable{ }

Commonly used constructor in thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

#Examples by extends:

```
class Multi1 extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi1 t1=new Multi1();
t1.start();
 }
}
```

#Example by implements

```
class Multi3 implements Runnable{
  public void run(){
    System.out.println("thread is running...");
  }
  public static void main(String args[]){
  Runnable m1=new Multi3();
  // Multi3 m1 = new Multi3();
 // If we do not extend the thread class, the class object will not be treated as
 // thread class object. We pass the object of the class that implements Runnable
 // so class run() method may execute.
  Thread t1 =new Thread(m1);
  t1.start();
 }
}
```

b. List the source of events. 5M

Ans:

Event:

- Changing the state of an object is known as an event
- Describes the change in status of source
- Generated as result f user interaction with GUI
- Example: click on button, dragging mouse etc.
- Java application written for windows, is event driven

The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event Source:

- An object on which event occurs
- Responsible for providing information of the occurred event to its handler

Here are some of the most common types of events in Java:

- **ActionEvent**: Represents a graphical element is clicked, such as a button or item in a list. Related listener: ActionListener.
- **ContainerEvent**: Represents an event that occurs to the GUI's container itself, for example, if a user adds or removes an object from the interface. Related listener: ContainerListener.
- **KeyEvent**: Represents an event in which the user presses, types or releases a key. Related listener: KeyListener.
- **WindowEvent**: Represents an event relating to a window, for example, when a window is closed, activated or deactivated. Related listener: WindowListener.
- **MouseEvent**: Represents any event related to a mouse, such as when a mouse is clicked or pressed. Related listener: MouseListener.

Examples in table:

User action	Source Object	Generated Event type
Click a button	Jbutton	ActionEvent
Press return on a text field	JTextField	ActionEvent
Select an item	JList	ListSelectionEvent

c. **What is the advantage of multithreading? Illustrate a program to implement multiple threads.** **8M**

Ans:
Thread is basically a lightweight sub-process, a smallest unit of processing.
Multiprocessing and multithreading, both are used to achieve multitasking
Multithreading in Java:

- Process of executing multiple threads simultaneously
- Does not block the user because running independently
- One can perform many operations together
- Used to achieve multitasking
- Thread based multitasking is multithreading
- Used in games and animation software

Advantages of Multithreading:

- Saves time
- One thread does not effect other
- Shares common memory area
- Context-switching takes less time than processes

```
//Example of Multithreading - 2 threads:
//one displays "computer Science" and another displays "information Science"
public class MultiVtu1 {
   public static void main(String[] args) {
      Runnable r1 = new Runnable1();
      Thread t1 = new Thread(r1);
      Runnable r2 = new Runnable2();
      Thread t2 = new Thread(r2);
      t1.start(); t2.start();
   }
}
class Runnable2 implements Runnable{
   public void run(){ System.out.println("Information Science");}
}
class Runnable1 implements Runnable{
   public void run(){ System.out.println("Computer Science");}
}
```

4.

a. **List the drawbacks of AWT. Explain two key swing features.** **6M**

Ans:
Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in *java*. The **java.awt** package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Drawbacks of AWT:

- AWT components are platform-dependent.
- AWT components are heavyweight.
- AWT doesn't support pluggable look and feel
- AWT provides less components than Swing.
- AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.

Key features of Java Swing:

Two key features:

I. **Swing components are lightweight.**
II. **Swing supports pluggable look and feel.**

Other features of Java swing:

- Java swing components are platform-independent.
- Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
- Swing follows MVC.

b. Create a simple swing applet to contain two buttons "Alpha" and "Beta" and display appropriate message clicked. 6M

Ans:

```
//Alpha & Beta buttons & when they are clicked
   import java.awt.event.*;
   import javax.swing.*;
   public class BetaGama {
   public static void main(String[] args) {
      JFrame f=new JFrame("Button Example");
      final JTextField tf=new JTextField();
      tf.setBounds(100,50, 150,20);
      JButton b=new JButton("Alpha");
      b.setBounds(70,100,95,30);
      JButton c=new JButton("Beta");
      c.setBounds(170,100,95,30);
      b.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent e){ tf.setText("Alpha was pressed."); }
      });
      c.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent e){ tf.setText("Beta was pressed."); }
      });
      f.add(tf);
      f.add(b);
      f.add(c);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   }
```

c. Explain briefly swing Buttons with code snippet for each. 8M

Ans:
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu etc.

For each swing button we will
import javax.swing.*;

JButton
The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Here complete code example of creating frame, panel and the button
```
// JFrame, JPanel & JButton - Java swing
import javax.swing.*;
class frame1 {
  public static void main(String[] args) {
     JFrame f = new JFrame("MyFrame");
     JPanel p = new JPanel( );
     JButton b = new JButton("Press me");
     f.setSize(400,500);
     p.add(b);      // add button to panel
     f.setContentPane(p);    // add panel to frame
     f.setVisible(true); // f.show();
     f.setDefaultCloseOperation(f.EXIT_ON_CLOSE);
  }
}
```

//rest with code snippets
Jlabel
The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.
```
 JLabel l1,l2;
  l1=new JLabel("First Label.");
  l1.setBounds(50,50, 100,30);
  l2=new JLabel("Second Label.");
  l2.setBounds(50,100, 100,30);
```
// add button to panel and add panel to frame like first code

JTextField
The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class
```
 JTextField t1,t2;
  t1=new JTextField("Welcome to CMRIT.");
  t1.setBounds(50,100, 200,30);
  t2=new JTextField("Java Swing Tutorial");
  t2.setBounds(50,150, 200,30);
```
// add button to panel and add panel to frame like first code

JcheckBox
The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

```
    JCheckBox checkBox1 = new JCheckBox("C++");
    checkBox1.setBounds(100,100, 50,50);
    JCheckBox checkBox2 = new JCheckBox("Java", true);
    checkBox2.setBounds(100,150, 50,50);
```

// add button to panel and add panel to frame like first code

JRadioButton
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.
It should be added in ButtonGroup to select one radio button only.

```
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
```

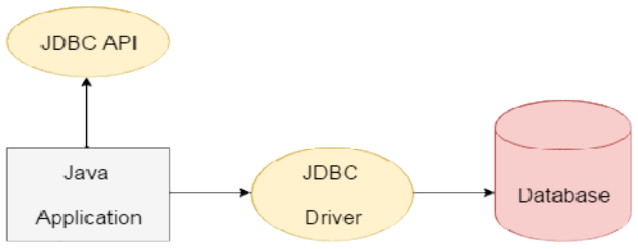
// add button to panel and add panel to frame like first code

5.

a. **Explain the concept of JDBC and JDBC driver types.** **6M**

Ans:
JDBC (Java DataBase Connectivity)
- Java JDBC is a java API to connect and execute query with the database.
- JDBC API uses jdbc drivers to connect with the database.
- ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).

API
- API (Application programming interface) is a document that contains description of all the features of a product or software.
- It represents classes and interfaces that software programs can follow to communicate with each other.
- An API can be created for applications, libraries, operating systems, etc



JDBC Driver is a software component that enables java application to interact with the database.
JDBC Driver types:

There are 4 types of JDBC drivers:

I. JDBC-ODBC bridge driver
II. Native-API driver (partially java driver)
III. Network Protocol driver (fully java driver)
IV. Thin driver (fully java driver)

I. JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.
- Adv: Easy to use & to connect database
- Disadv: Performance low because JDBC method call converted to ODBC function call

II. Native-API driver (partially Java driver)

The Native API driver uses the client-side libraries of the database.

- The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
- Adv: performance upgraded than JDBC-ODBC bridge driver.
- Disadv:
  i. The Native driver needs to be installed on the each client machine.
  ii. The Vendor client library needs to be installed on client machine.

III. Network Protocol driver (fully java driver)

- The Network Protocol driver uses middleware application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
- **Adv:** No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- **Disadv:**
  i. Network support is required on client machine.
  ii. Requires database-specific coding to be done in the middle tier.
  iii. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

IV. Thin Driver (Fully Java Driver)

- The thin driver converts JDBC calls directly into the vendor-specific database protocol.
- That is why it is known as thin driver. It is fully written in Java language.
- Adv:
- Better performance than all other drivers.
- No software is required at client side or server side.
- Disadv:
- Drivers depends on the Database.

b. Explain JDBC process steps with example code. 7M

Ans:

Steps to connect any java application with the database in java using JDBC.

I. **Register the driver class:**
public static void forName(String className)throws ClassNotFoundExceptio
n

II. **Creating connection:**
public static Connection getConnection(String url,String name,String passwor
d) throws SQLException
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhos
t:1521:xe","system","password");

III. **Creating**
statement:public Statement createStatement()throws SQLException

IV. **Executing**
queries:public ResultSet executeQuery(String sql)throws SQLException

V. **Closing connection:**public void close()throws SQLException

```
//Example code:
import java.sql.*;
public class jdbcResultSet {
  public static void main(String[] args) {
    try {
      Class.forName("org.apache.derby.jdbc.ClientDriver");
    } catch(ClassNotFoundException e) {
      System.out.println("Class not found "+ e);
    }
    try {
      Connection con = DriverManager.getConnection(
        "jdbc:derby://localhost:1527/testDb","username", "password");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM employee");
      System.out.println("id  name    job");
      while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        String job = rs.getString("job");
        System.out.println(id+"   "+name+"    "+job);
      }
    } catch(SQLException e) {
      System.out.println("SQL exception occured" + e);
    }
  }
}
```

c. What are three types of statement objects? Explain with code snippets for each.
7M

Ans:

There are 3 types of Statements, as given below:

I. **Statement:** It can be used for general-purpose access to the database. It is useful when you are using static SQL statements at runtime.
II. **PreparedStatement:** It can be used when you plan to use the same SQL statement many times. The PreparedStatement interface accepts input parameters at runtime.
III. **CallableStatement:** CallableStatement can be used when you want to access database stored procedures.

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
public Statement createStatement()throws SQLException

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
String sql="insert into emp values(?,?,?)";

passing parameter (?) for the values
value will be set by calling the setter methods of PreparedStatement.
Performance of the application will be faster if because query is compiled only once.
public PreparedStatement prepareStatement(String query)throws SQLException{}
create table emp(id number(10),name varchar2(50));

```
// completed code
import java.sql.*;
class InsertPrepared{
    public static void main(String args[]){
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localh
                                    ost:1521:xe","system","oracle");
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
             stmt.setInt(1,101);//1 specifies the first parameter in the query
             stmt.setString(2,"Ratan");
             int i=stmt.executeUpdate();
            System.out.println(i+" records inserted");
            con.close();
        }catch(Exception e){ System.out.println(e);}
    }
}
```

CallableStatement interface is used to call the **stored procedures and functions**.

- We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.
- may create a function that receives date as the input and returns age of the employee as the output.

- The prepareCall() method of Connection interface returns the instance of CallableStatement.

public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");

6.

a. Explain briefly the lifecycle of a servlet. Also mention the packages required. 6M

Ans:

- **Servlet** technology is used to create web application (resides at server side and generates dynamic web page).
- **Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language.

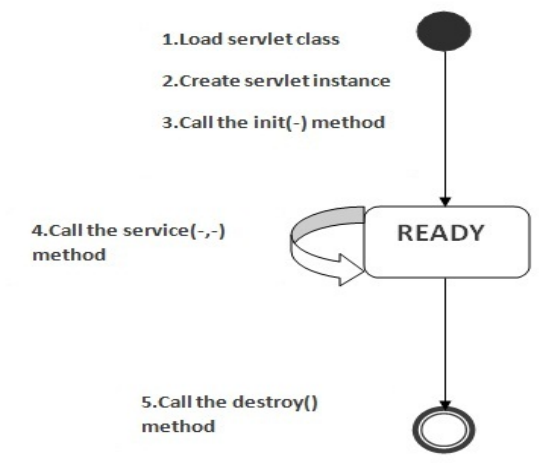
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the
  incoming request. It can respond to any type of requests.

Lifecycle of a servlet: The web container maintains the life cycle of a servlet instance.

I. Servlet class is loaded.
II. Servlet instance is created.
III. init method is invoked.
IV. service method is invoked.
V. destroy method is invoked.

3 states of a servlet: new, ready and end.

I. The servlet is in new state if servlet instance is created.
II. After invoking the init() method, Servlet comes in the ready state. In the ready state,
servlet performs all the tasks.
III. When the web container invokes the destroy() method, it shifts to the end state.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

Packages required:
Servlet packages, while developing an application depends on the any domain programmer has to import required packages these packages will have some functionalities that provide support.

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- **javax.servlet**
- **javax.servlet.http**

All these packages will be available in the servlet API along with the packages The programmer has to import required jar files also, Following are the some packages which are commonly used in servlet applications.

• javax.servlet.Servlet	(interface)
• javax.servlet.GenericServlet	(abstract class)
• javax.servlet.http.HttpServlet	(abstract calss)

Others:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

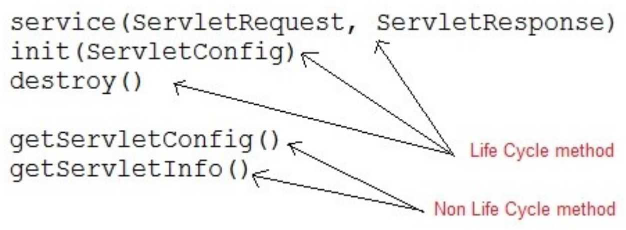
b. List and explain the need of core classes and interfaces of a servlet. 7M

Ans:
Generic Servlet class: GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

Core Classes and Interface of javax.servlet.http

Classes	**Interfaces**
HttpServlet	HttpServeletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.

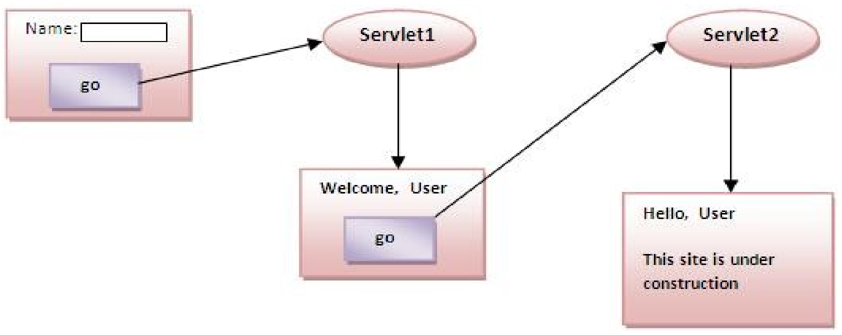

```
service(ServletRequest, ServletResponse)
init(ServletConfig)
destroy()

getServletConfig()
getServletInfo()
```

c. **What is a cookie? Write a program to display all cookies using servlets.** **7M**

Ans:

- Cookies are small bits of textual information that a Web server sends to a browser and that the browser returns unchanged when visiting the same Web site or domain later.
- A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number
domain qualifiers, a maximum age, and a version number. Some Web browsers have bugs in how they handle the optional **attributes**, so those should be used carefully to improve the interoperability of the **servlets**.



```
//Example servlet program to add & display cookies
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response
    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
   //Display cookies
   for(int i=0;i<ck.length;i++){
      out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());
                              //printing name and value of cookie
    out.close();
      }catch(Exception e){System.out.println(e);}
   }
}
```

7.

a. List and Explain JSP tags with example for each. 8M

Ans:

In JSP, java code can be written inside the jsp page using the scriptlet tag. The scripting elements / JSP tags provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1. scriptlet tag
2. expression tag
3. declaration tag

Other tags are
   4. comment & 5. directive tags

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax:
<% java source code %>
Example:

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

JSP expression tag

The code placed within JSP expression tag is *written to the output stream of the response*. So we need not write out.print() to write data. Syntax:
<%= statement %>
Example:

```
<html>
<body>
<%= "welcome to jsp" %>  </body>
</html>
```

JSP Declaration Tag

The JSP declaration tag is used *to declare fields and methods*. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

Syntax :

```
<%!  field or method declaration %>
```

Example - declares field

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example - declares method:

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JAP Directive Tag:

A Directive tag opens with <%@ and closes with %>.

Example:

```
<%@ page import = “java.sql.*” %>
```

JSP Comment Tag:

A comment tag opens with <%-- and closes with --%>,

Example:

```
<%-- jsp comment tag --%>
```

b. Write a note on session. 4M

c. Explain RMI concept and illustrate server side program. 8M

Ans:

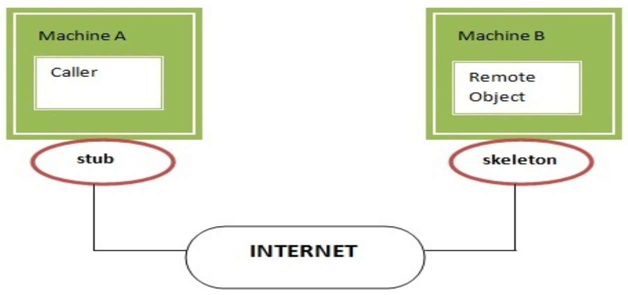
The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton which communicate with remote object.

A **remote object** is an object whose method can be invoked from another JVM.

The **stub** is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.

The **skeleton** is an object, acts as a gateway for the server side object. All the incoming requests are routed through it.

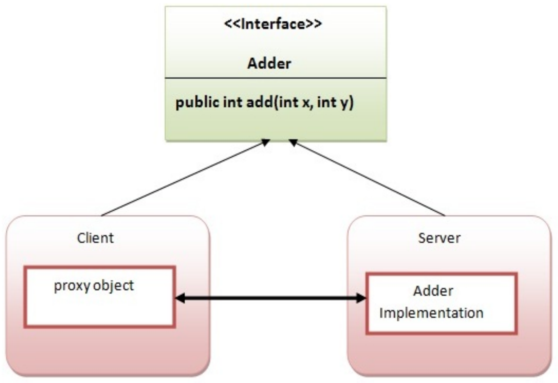


Stub object does the following tasks:

- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (unmarshals) the return value or exception, and
- It finally, returns the value to the caller.

When the skeleton receives the incoming request, it does the following tasks:

- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshals) the result to the caller.



1. create the remote interface
Extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, example of a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
//Adder.java
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2. Provide the implementation of the remote interface
Either extend the UnicastRemoteObject class, or use the exportObject() method of the UnicastRemoteObject class.
In case, we extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException

```
//AdderRemote.java
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
```

```
super();
}
public int add(int x,int y){return x+y;}
}
```

3. create the stub and skeleton objects using the rmic tool on command prompt

```
rmic AdderRemote
```

4. Start the registry service by the rmiregistry tool on command prompt

```
rmiregistry 5000
```

5. Create and run the server application

```
//Myserver.java
import java.rmi.registry.*;
public class MyServer{
  public static void main(String args[]){
    try{ Adder stub=new AdderRemote();
          Naming.rebind("rmi://localhost:5000/sonoo",stub);
        }catch(Exception e){System.out.println(e);}
  }
}
```

6. Create & run client application

```
//MyClient.java
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

8.

a. **Explain the functions of EJB transaction attributes with program to set the attribute. 10M**

Ans:
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.

EJB Container/Servers are transaction servers and handles transactions context propagation and distributed transactions. Transactions can be managed by the container or by custom code handling in bean's code.

- **Container Managed Transactions** − In this type, the container manages the transaction states.
- **Bean Managed Transactions** − In this type, the developer manages the life cycle of transaction states.

EJB 3.0 has specified following attributes of transactions, which EJB containers implement −

- **REQUIRED** − Indicates that business method has to be executed within transaction, otherwise a new transaction will be started for that method.
- **REQUIRES_NEW** − Indicates that a new transaction, is to be started for the business method.
- **SUPPORTS** − Indicates that business method will execute as part of transaction.

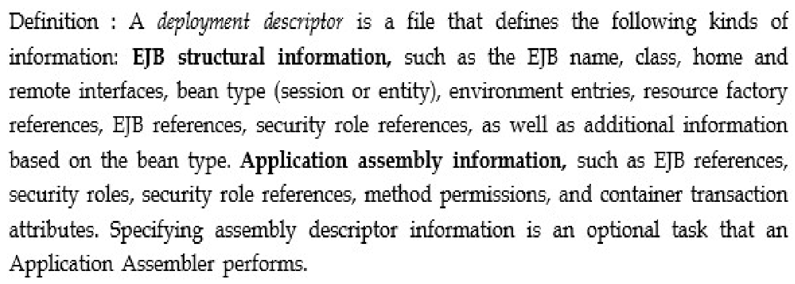
- **NOT_SUPPORTED** – Indicates that business method should not be executed as part of transaction.
- **MANDATORY** – Indicates that business method will execute as part of transaction, otherwise exception will be thrown.
- **NEVER** – Indicates if business method executes as part of transaction, then an exception will be thrown.

//Example program

```
import javax.ejb.Stateful;
import javax.annotation.PostConstruct;
import javax.ejb.Remove;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;
import javax.ejb.TransactionAttribute;
import static javax.ejb.TransactionAttributeType.REQUIRED;
import static javax.ejb.TransactionAttributeType.REQUIRES_NEW;
import com.acme.Cart;

@Stateful
@TransactionManagement(value=TransactionManagementType.CONTAINER)
@TransactionAttribute(value=REQUIRED)
public class CartBean implements Cart {
    private ArrayList items;

    @PostConstruct
    public void initialize() {
        items = new ArrayList();
    }

    @Remove
    @TransactionAttribute(value=REQUIRES_NEW)
    public void finishedShipping() {
        // Release any resources.
    }

    public void addItem(String item) {
        items.add(item);
    }

    public void removeItem(String item) {
        items.remove(item);
    }
}
```

b. Describe the concept of deployment descriptors and explain the need of a JAR file.
10M

Ans:



Definition : A *deployment descriptor* is a file that defines the following kinds of information: **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.

Note: Deployment descriptor of EJB 2.0: ejb-jar.xml & weblogic-ejb-jar.xml

The **jar (Java Archive)** tool of JDK provides the facility to create the executable jar file. An executable jar file calls the main method of the class if we double click it.
To create the executable jar file, we need to create **.mf file**, also known as manifest file.
Creating manifest file: To create manifest file, you need to write Main-Class, then colon, then space, then classname then enter. For example: **myfile.mf**
Main-Class: First
In mf file, new line is must after the class name.
Creating executable jar file using jar tool:
The jar tool provides many switches, some of them are as follows:
-c creates new archive file
-v generates verbose output. It displays the included or extracted resource on the standard output.
-m includes manifest information from the given mf file.
-f specifies the archive file name
-x extracts files from the archive file
We need to write **jar** then **swiches** then **mf_file** then **jar_file** then **.classfile**
jar -cvmf myfile.mf myjar.jar First.class

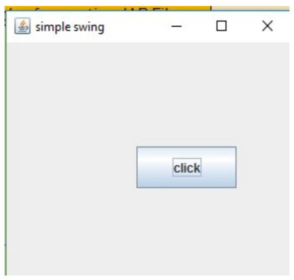
```
//First.java
import javax.swing.*;
public class First{
  First(){
    JFrame f=new JFrame();
    JButton b=new JButton("click");
    b.setBounds(130,100,100, 40);
    f.add(b);
    f.setSize(300,400);
    f.setLayout(null);
    f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args) {
    new First();
}
}
```

//contents of myfile.mf
Main-Class: First

Now executing:

```
D:\JavaProgs\myjar>javac First.java
D:\JavaProgs\myjar>jar -cvmf myfile.mf myjar.jar First.class
added manifest
adding: First.class(in = 736) (out= 506)(deflated 31%)
D:\JavaProgs\myjar>myjar.jar
```

Expected output:



****************Act like a person of thought, think like a person of action. ********************