Assessment Solution of Pokhara University Central College

Notes:

*** Write the answer according to the marks**

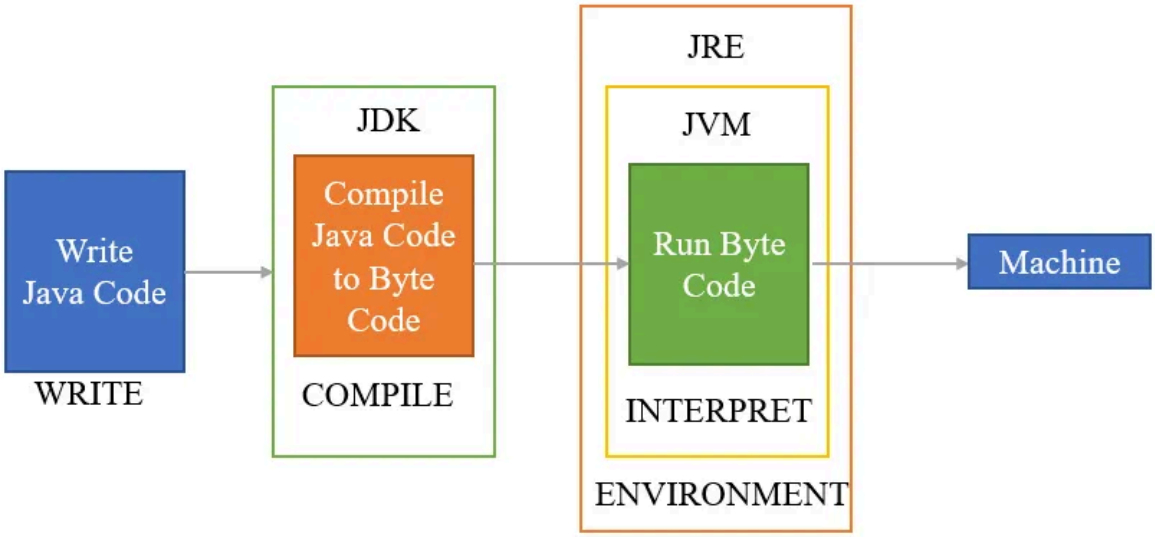
*** RED text means it is not the part of the answer**

*** BLUE text represents comments**

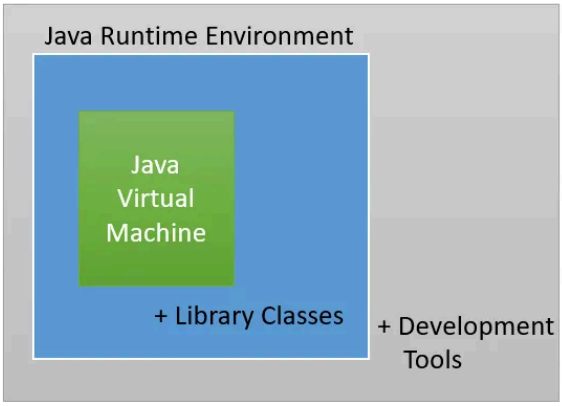
Explain the Java architecture and the significance of class path.

The JVM, JRE, and JDK are the three main components of Java Architecture. It combines the interpretation and compilation processes. It enumerates all of the steps involved in writing a Java program. The Java Architecture describes each step in the compilation and execution of a program.

There are two processes in Java — Compilation and Interpretation.



- The Java source code goes to the compiler.
- The Java Compiler converts it into byte codes
- The bytes codes are then converted into machine by the JVM
- The Machine code is executed directly by the machine(Operating System)



JDK = JRE + Development Tools
JRE = JVM + Library Classes

Java Runtime Environment(JRE)
Java Runtime Environment provides a platform where all the applications like JVM and other runtime libraries linked together to run your Java Program. It builds a runtime environment where you can execute the Java program. The JRE also initiates the JVM for its execution. JRE has the required software and libraries to run the programs.

Java Development Kit (JDK)
Java Development Kit is the set of libraries, compiler, interpreter and other set of programs that will help you to build the Java program. Once JDK is installed on the machine then start developing, compile and run the Java program. You cannot compile a Java program without a JDK installed on the machine. Once you compile the code with JDK tools, you can get an Byte Code File.

JVM
The main feature of Java is WORA. WORA stands for Write Once Run Anywhere. The feature states that we can write our code once and use it anywhere or on any operating system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.

Classpath
Classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable. It would be impractical to have the VM look through every folder on your machine, so you have to provide the VM a list of places to look. This is done by putting folder and jar files on your classpath.

2. Explain the concept of exception handling in Java. Write an example program

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

The Exception Handling in Java is a mechanism to handle the runtime errors so that the normal flow of the application can be maintained. Java provides five keywords that are used to handle the exception.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signatures.

Sample Program (**You can write any program that you like)

```
public class Example{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){
       System.out.println(e);}
    } finally{
      System.out.println("This block will always execute");
           }
}
```

3. Describe the purpose of interfaces in Java and how they differ from abstract classes.

Interfaces are used in Java to achieve abstraction. By using the implements keyword, a java class can implement an interface. An interface specifies the behavior of a class by providing an abstract type. Key features of OOP such as abstraction, polymorphism and multiple inheritance can be achieved by using interfaces.
Interface in Java can be viewed as a container that stores the signatures of the methods to be implemented. Interface also represents IS-A relationship.
We can create an interface in Java using the 'interface' keyword.
Syntax:

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

Some key features of interface are as follows:

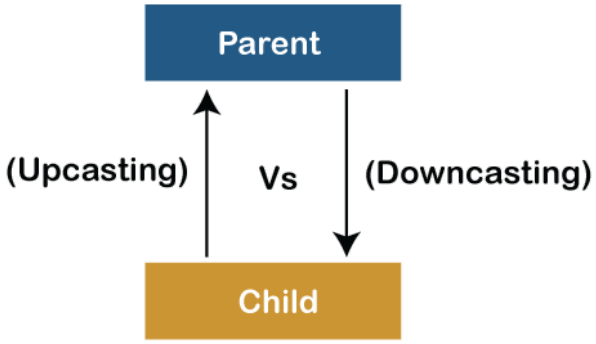
- Supports multiple inheritance
- All methods are by default abstract i.e; methods do not have codes.
- It has only static and final variables
- Since Java 8 , it can have default and static methods.

Difference between Abstract class and Interface

Abstract Class	**Interface**
Abstract classes can have abstract and non-abstract methods.	Interfaces can have only abstract methods. Since Java 8, it can have default and static methods also.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
Abstract class can provide the implementation of interface.	Interfaces can't provide the implementation of abstract classes.

The abstract keyword is used to declare abstract class.	The interface keyword is used to declare the interface.
An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
An abstract class can be extended using the keyword "extends".	An interface can be implemented using the keyword "implements".
A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.
It can have constructors	It can not have a constructor.
Abstract class provides abstraction from 0 to 100%	It provides abstraction of 100%

4. Differentiate between upcasting and downcasting in Java. Provide examples to illustrate each.
A process of converting one data type to another is known as Typecasting and Upcasting and Downcasting is the type of object typecasting. In Java, the object can also be type-casted like the datatypes. Parent and Child objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., Parent to Child (Downcasting) and Child to Parent (Upcasting)
In other words, converting an object of Parent class to Child class is called Downcasting and converting an object of a Child class into an object Parent class is called Upcasting.



Typecasting is used to ensure whether variables are correctly processed by a function or not.We can perform Upcasting implicitly or explicitly, but downcasting cannot be implicitly possible.

Upcasting
Upcasting is a type of object typecasting in which a child object is typecasted to a parent class object. By using Upcasting, we can easily access the variables and methods of the parent class to the child class.Upcasting is also known as Generalization and Widening.
Syntax of Upcasting
Parent p = new Child();

Upcasting will be done internally and due to upcasting the object is allowed to access only parent class members and child class specified members (overridden methods, etc.) but not all members. It is rarely used in Java.
Sample code of Upcasting

```
class  Parent{
  void PrintData() {
    System.out.println("method of parent class");
  }
}

class Child extends Parent {
  void PrintData() {
    System.out.println("method of child class");
  }
}
class UpcastingExample{
  public static void main(String args[]) {

    Parent obj1 = (Parent) new Child();
    Parent obj2 = (Parent) new Child();
    obj1.PrintData();
    obj2.PrintData();
  }
}
```

Downcasting
Downcasting means the typecasting of a parent object to a child object. Downcasting cannot be implicit.
Syntax
Child c = (Child)p;
Downcasting has to be done externally and due to downcasting a child object can acquire the properties of the parent object. It is used when we need to develop a code that accesses behaviors of the child class.

Sample Program

```
class Parent {
   String name;
```

```
    // A method which prints the data of the parent class
    void showMessage()
    {
        System.out.println("Parent method is called");
    }
}

// Child class
class Child extends Parent {
    int age;

    // Performing overriding
    @Override
    void showMessage()
    {
        System.out.println("Child method is called");
    }
}

public class Downcasting{

    public static void main(String[] args)
    {
        Parent p = new Child();
        p.name = "Alexa";

        // Performing Downcasting Explicitly
        Child c = (Child)p;

        c.age = 18;
        System.out.println(c.name);
        System.out.println(c.age);
        c.showMessage();
    }
}
```

Output

Alexa
18

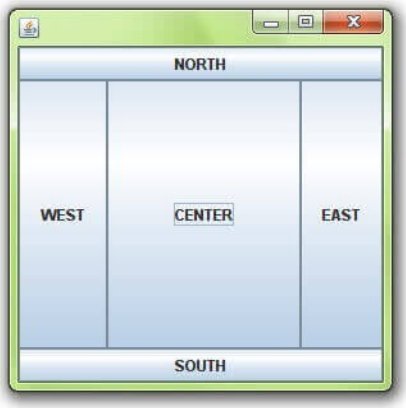
Child method is called.

5. Explain layout management in Java Swing. Write a program to demonstrate a login form with username and password field with a sign in button.

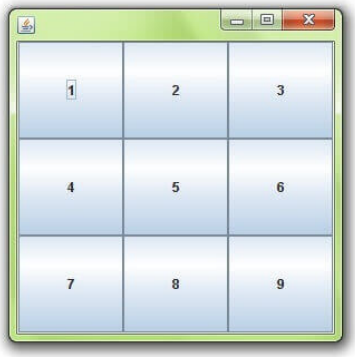
Layout refers to the arrangement of components within the container.The layout manager automatically positions all the components within the container. It facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers.

Some commonly used layouts are

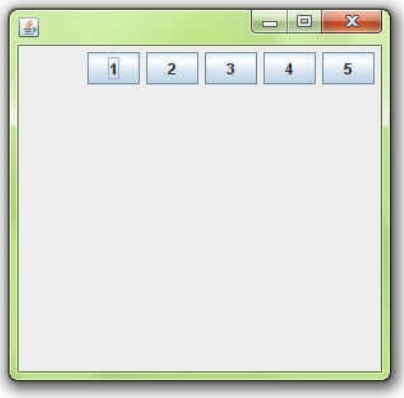
- BorderLayout
  It is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window.



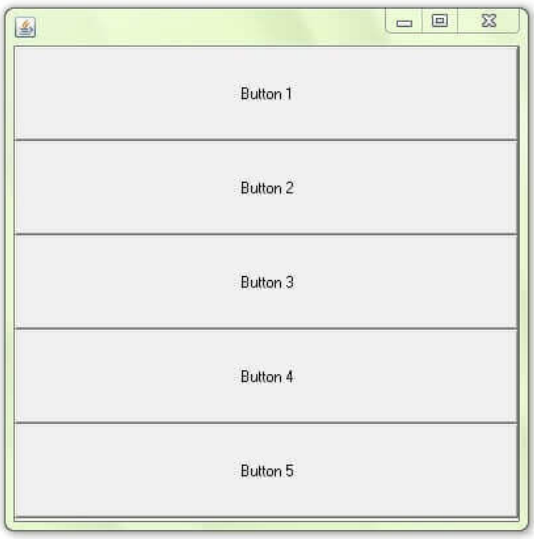
- GridLayout
  The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.



- FlowLayout
  The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.



- BoxLayout
  It is used to arrange the components either vertically or horizontally.



(** If the question is asked for 8 marks you need to write sample code as well **)

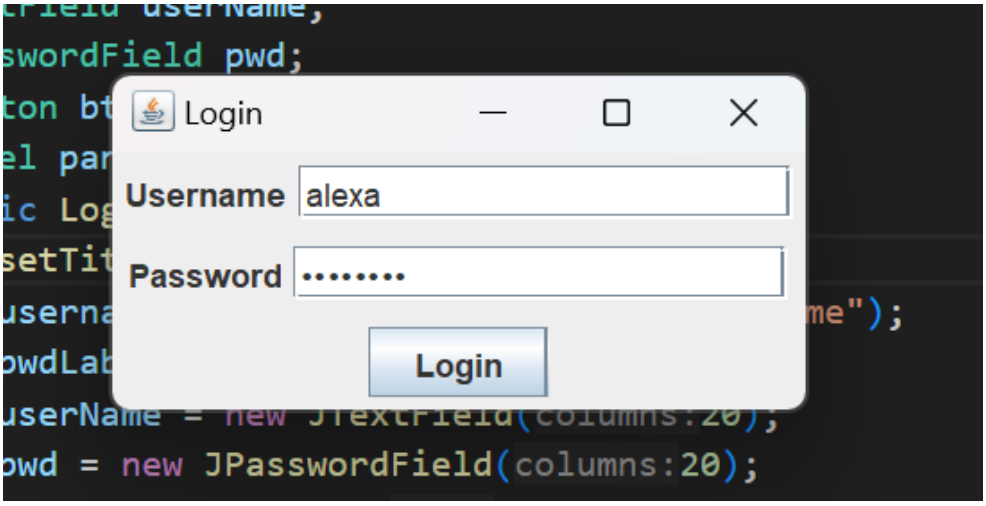
Code for Login Page

** You can create this on your own so the code below is just a reference. If you want to do it in more simpler ways you can remove the labels and also use GridLayout instead but it will look ugly. Also we don't have to handle events since it is not specified in the question.**

```
import javax.swing.*;

class Login extends JFrame{
   JLabel usernameLabel, pwdLabel;
   JTextField userName;
   JPasswordField pwd;
   JButton btn;
   JPanel panelOne, panelTwo, panelThree;
   public Login(){
      setTitle("Login");
      usernameLabel = new JLabel("Username");
      pwdLabel = new JLabel("Password");
      userName = new JTextField(20); // number 20 means the max length; if you don't give this the field will be too small
      pwd = new JPasswordField(20);
      btn =  new JButton("Login");

      // lets add the label and textfield for username in panelOne
      panelOne = new JPanel();
      panelOne.add(usernameLabel);
      panelOne.add(userName);

      // add pwd label and field in panel two
      panelTwo = new JPanel();
      panelTwo.add(pwdLabel);
      panelTwo.add(pwd);

      // add the login button on panelThree
      panelThree = new JPanel();
```

```
        panelThree.add(btn);


        // add all three panels in the frame using BoxLayout so that they appear vertically
        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS));

        add(panelOne);
        add(panelTwo);
        add(panelThree);

        // use BoxLayout for our main Frame

        setVisible(true);
        pack(); // you can use setSize instead

    }
    public static void main(String [] args){
        new Login();
    }
}
```

Output



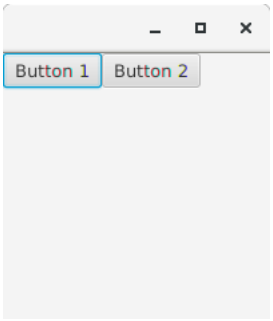
6. Explain VBox and HBox in Java. Write a program to demonstrate the user registration form with a submit button.

Layouts are the top level container classes that define the UI styles for scene graph objects. Layout can be seen as the parent node to all the other nodes. In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene-graph nodes.

HBox

HBox layout pane arranges the nodes in a single row. It is represented by javafx.scene.layout.HBox class. The HBox class contains two constructors that are given below.

- *new HBox()* : create HBox layout with 0 spacing
- *new Hbox(Double spacing)* : create HBox layout with a spacing value

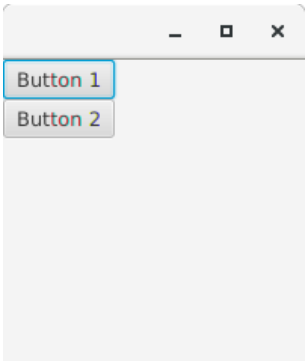


VBox

Instead of arranging the nodes in horizontal rows, Vbox Layout Pane arranges the nodes in a single vertical column. It is represented by javafx.scene.layout.VBox class which provides all the methods to deal with the styling and the distance among the nodes.

The VBox class contains 4 constructors that are given below:

- *VBox()* : creates layout with 0 spacing
- *Vbox(Double spacing)* : creates layout with a spacing value of double type
- *Vbox(Double spacing, Node? children)* : creates a layout with the specified spacing among the specified child nodes
- *Vbox(Node? children)* : creates a layout with the specified nodes having 0 spacing among them



User Registration Page in JavaFX

*** For registration let's use first name , last name, email and password. You can add more or use less if you want. Also we won't write code for event handling.*

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;


public class Login extends Application{
        @Override
        public void start(Stage primaryStage) throws Exception{
                primaryStage.setTitle("Login");

                Label first_name=new Label("First Name");
      Label last_name=new Label("Last Name");
      Label email = new Label("Email");
      Label password = new Label("Password");

      TextField tf1=new TextField();
      TextField tf2=new TextField();
      TextField tf3 = new TextField();
      PasswordField tf4 = new PasswordField();

      Button Submit=new Button ("Submit");
      GridPane root=new GridPane();
      Scene scene = new Scene(root,400,200);
      root.addRow(0, first_name,tf1);
      root.addRow(1, last_name,tf2);
      root.addRow(2, tf3);
      root.addRow(4, tf4);

      root.addRow(5, Submit);
      primaryStage.setScene(scene);
      primaryStage.show();

        }
```

```
    public static void main(String [] args) {
        launch(args);
    }
}
```

7. Compare JavaFX and Swing. Discuss the strengths and weaknesses.

JavaFX	**Java Swing**
JavaFX supports MVC	Swing supports MVC but is not consistent with components.
It has some UI components but is lesser than Swing.	Swing has a more sophisticated set of GUI items.
JavaFX is platform support for GUI Application	Swing is a standard toolkit for GUI
It has UI Components but is still evolving. It also has several components built over the Swing.	Swing is the legacy library for GUI.
It can help to create rich internet applications with modern UI.	Swing can help to create UI components with a decent look.
JavaFX scripts the UI development with screen builder.	Swing APIs are used to write the UI components.

Strengths of Swing

- Java Swing is compatible with every platform and is supported by many plugins.
- Java Swing is one of the most popular GUI toolkits and best for mobile app development.

- With Java Swing, it is easy to migrate legacy apps, which means you can reuse your code anytime.

Weaknesses of Swing

- Java Swing is directly connected to JVM, which means desktop to mobile cross-platform Java application is impossible.
- Java Swing is not compatible with Java 7 and the previous version.

Strengths of JavaFX

- JavaFX is a standard compatible class library included in IDEs like Eclipse.
- JavaFX enables developers to preview and test UI for different applications.
- JavaFX supports compositing controls, inputs, and a wide range of graphics.

Weaknesses of JavaFX

- Limitation in Technical Capability requires proficient technical knowledge to use.
- Not suitable for applications with high-level complexity.
- The Community of JavaFX is limited, which is a barrier when you are stuck at some problem in JavaFX.

8. Explain the concepts of TCP, UDP, IP Address, and Ports in the context of network programming.

TCP

- Transmission Control Protocol
- TCP enables data to be transferred between applications and devices on a network
- It is designed to break down a message, such as an email, into packets of data to ensure the message reaches its destination successfully and as quickly as possible.
- It is a connection oriented protocol. Before data transmission, client and server has to be connected.
- It guarantees the transmission of data.
- It uses error detection and correction mechanisms.
- If the data is lost or corrupted during transmission, it has the provision of re-sending the data.
- It implements three-ways handshaking (acknowledgment)
- It is slower compare to UDP
- It is used for data sensitive applications.

- Its header size is 20 bytes.
- It is used in HTTP, HTTPS, SMTP etc

UDP

- User Datagram Protocol
- It is a connection-less protocol which means the transmitter and receiver do not have to be connected.
- It doesn't guarantee the successful transmission of data.
- Its header size is 8 bytes.
- It doesn't implement acknowledgment
- Error correction and detection mechanisms are not implemented
- It is faster
- If the data is lost during transmission, they are not resent
- It is used in DNS, VoIP, live streaming

IP Address

- An Internet Protocol address (IP address) is a numerical label that is assigned to a device connected to a computer network that uses the Internet Protocol for communication.
- IP addresses serve two main functions: network interface identification, and location addressing.
- Internet Protocol version 4 (IPv4) defines an IP address as a 32-bit number.
- IP (IPv6) uses 128 bits for the IP address
- The IP address space is managed globally by the Internet Assigned Numbers Authority (IANA)
- IP addresses are classified into several classes of operational characteristics: unicast, multicast, anycast and broadcast addressing.

Ports

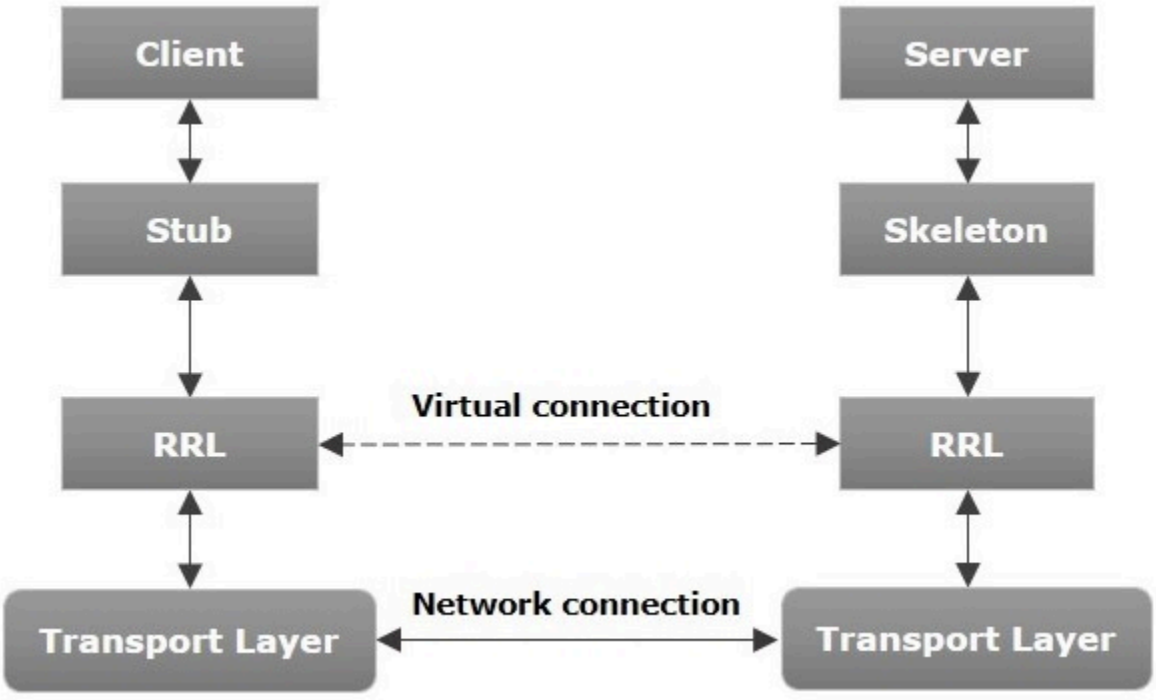
- A port or port number is a number assigned to uniquely identify a connection endpoint and to direct data to a specific service.
- At the software level, within an operating system, a port is a logical construct that identifies a specific process or a type of network service
- A port at the software level is identified for each transport protocol and address combination by the port number assigned to it.
- The most common transport protocols that use port numbers are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP)
- For TCP and UDP, a port number is a 16-bit unsigned integer, thus ranging from 0 to 65535.
- The port numbers are divided into three ranges: the well-known ports, the registered ports, and the dynamic or private ports.

- The well-known ports (also known as system ports) are those numbered from 0 through 1023. For example HTTP (80), HTTPS(443) etc
- The registered ports are those from 1024 through 49151.
- The dynamic or private ports are those from 49152 through 65535.

9. Explain the architecture of RMI

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications 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.



In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

Components of RMI

- Transport Layer − This layer connects the client and the server. It manages the existing connection and also sets up new connections.

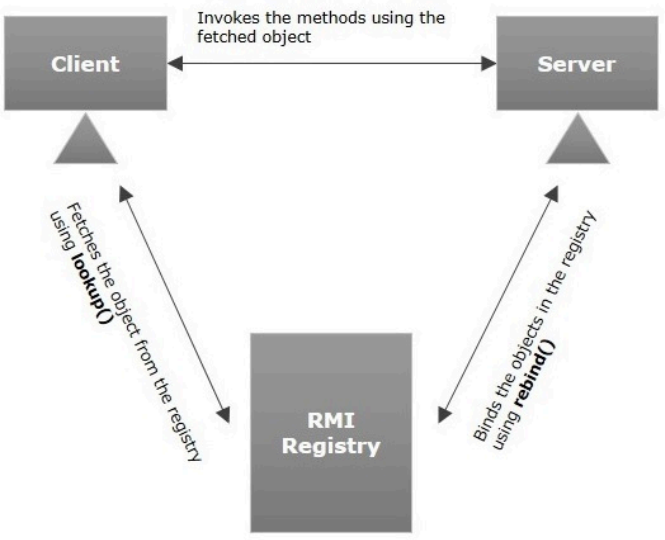
- Stub − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton − This is the object which resides on the server side. stub communicates with this skeleton to pass requests to the remote object.
- RRL(Remote Reference Layer) − It is the layer which manages the references made by the client to the remote object.

Workings of RMI

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as marshaling.
At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as unmarshalling.

RMI Registry



RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using bind() or reBind() methods). These are registered using a unique name known as bind name.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using lookup() method).

10. Explain the concepts of IDL and provide a simple CORBA program example

Java IDL is a technology for distributed objects--that is, objects interacting on different platforms across a network. Java IDL is similar to RMI (Remote Method Invocation), which supports distributed objects written entirely in the Java programming language. However, Java IDL enables objects to interact regardless of whether they're written in the Java programming language or another language such as C, C++, COBOL, or others.

IDL is a key feature of CORBA. It is a language-neutral Interface Definition Language. Each language that supports CORBA has its own IDL mapping--and as its name implies, Java IDL supports the mapping for Java. CORBA and the IDL mappings are the work of an industry consortium known as the OMG, or Object Management Group. Sun is a founding member of the OMG, and the Java IDL team has played an active role in defining the IDL-to-Java mapping.To support interaction between objects in separate programs, Java IDL provides an Object Request Broker, or ORB. The ORB is a class library that enables low-level communication between Java IDL applications and other CORBA-compliant applications.

Simple CORBA Program

The code below is a sample CORBA example. The application has following steps

1. The client (applet or application) invokes the sayHello operation of the HelloServer.
2. The ORB transfers that invocation to the servant object registered for that IDL interface.
3. The servant's sayHello method runs, returning a Java String.
4. The ORB transfers that String back to the client.
5. The client prints the value of the String.

Filename: Hello.idl

```
module HelloApp {
   interface Hello
   {
      string sayHello();
   };
};
```

Filename: HelloClient.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  static Hello helloImpl;

  public static void main(String args[])
    {
      try{
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);

        // get the root naming context
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

        } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
          e.printStackTrace(System.out);
          }
    }

}
```

Filename: HelloSever.java

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class HelloImpl extends HelloPOA {
 private ORB orb;

 public void setORB(ORB orb_val) {
  orb = orb_val;
 }

 // implement sayHello() method
 public String sayHello() {
  return "\nHello world !!\n";
 }

 // implement shutdown() method
 public void shutdown() {
  orb.shutdown(false);
 }
}


public class HelloServer {

 public static void main(String args[]) {
  try{
   // create and initialize the ORB
   ORB orb = ORB.init(args, null);

   // get reference to rootpoa & activate the POAManager
   POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
   rootpoa.the_POAManager().activate();

   // create servant and register it with the ORB
```

```
    HelloImpl helloImpl = new HelloImpl();
    helloImpl.setORB(orb);

    // get object reference from the servant
    org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
    Hello href = HelloHelper.narrow(ref);

    // get the root naming context
    // NameService invokes the name service
    org.omg.CORBA.Object objRef =
       orb.resolve_initial_references("NameService");
    // Use NamingContextExt which is part of the Interoperable
    // Naming Service (INS) specification.
    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

    // bind the Object Reference in Naming
    String name = "Hello";
    NameComponent path[] = ncRef.to_name( name );
    ncRef.rebind(path, href);

    System.out.println("HelloServer ready and waiting ...");

    // wait for invocations from clients
    orb.run();
  }

    catch (Exception e) {
      System.err.println("ERROR: " + e);
      e.printStackTrace(System.out);
    }

    System.out.println("HelloServer Exiting ...");

  }
}
```

11. Differentiate between JDBC driver types and discuss the configurations.

** Study this on your own **

12. Discuss HTTP methods and responses in web applica tions.

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

GET
The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

HEAD
The HEAD method asks for a response identical to a GET request, but without the response body.

POST
The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

PUT
The PUT method replaces all current representations of the target resource with the request payload.

DELETE
The DELETE method deletes the specified resource.

PATCH
The PATCH method applies partial modifications to a resource.

HTTP Responses
HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

- Informational responses (100 – 199)
- Successful responses (200 – 299)
- Redirection messages (300 – 399)
- Client error responses (400 – 499)
- Server error responses (500 – 599)

Information responses
100 Continue
This interim response indicates that the client should continue the request or ignore the response if the request is already finished.

Successful responses

200 OK

The request succeeded. The result meaning of "success" depends on the HTTP method:

201 Created

The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

Redirection messages

300 Multiple Choices

The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

301 Moved Permanently

The URL of the requested resource has been changed permanently. The new URL is given in the response.

Client error responses

400 Bad Request

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

401 Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

Server error responses

500 Internal Server Error

The server has encountered a situation it does not know how to handle.

504 Gateway Timeout

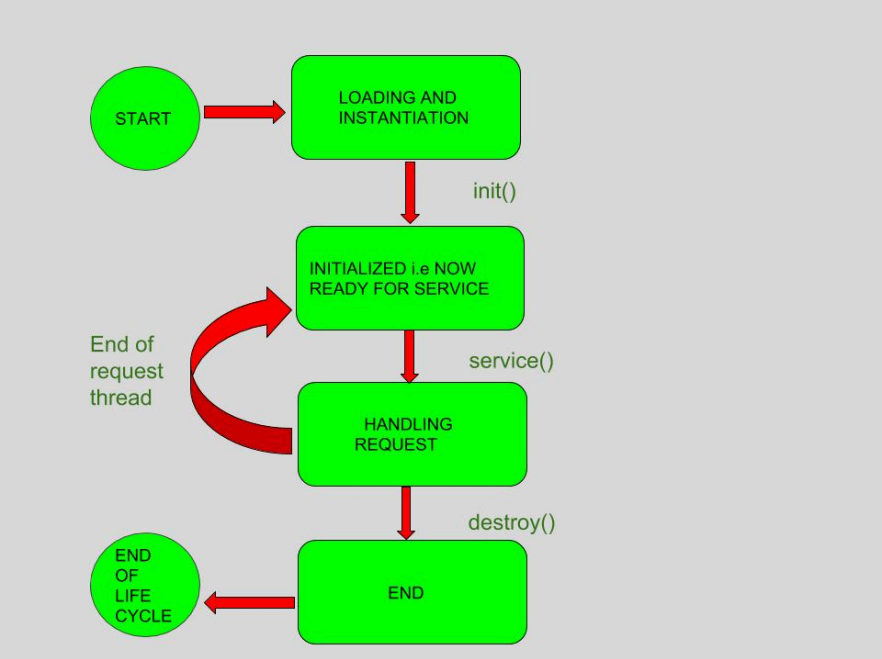
This error response is given when the server is acting as a gateway and cannot get a response in time.

12. Explain the life cycle of web servlets.

The entire life cycle of a Servlet is managed by the Servlet container which uses the javax.servlet.Servlet interface to understand the Servlet object and manage it. So, before creating a Servlet object, let’s first understand the life cycle of the Servlet object which is actually

understanding how the Servlet container manages the Servlet object. Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

1. Loading a Servlet.
2. Initializing the Servlet.
3. Request handling.
4. Destroying the Servlet.



\

1. **Loading a Servlet:** The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :
   Initializing the context, on configuring the Servlet with a zero or positive integer value.
   If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.
   The Servlet container performs two operations in this stage :
   Loading : Loads the Servlet class.
   Instantiation : Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.
2. **Initializing a Servlet:** After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter. The Servlet container invokes the Servlet.init(ServletConfig) method only once, immediately after the Servlet.init(ServletConfig) object is instantiated successfully.

3. **Handling request:** After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request :
   It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively. After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.
4. **Destroying a Servlet:** When a Servlet container decides to destroy the Servlet, it performs the following operations,
   - It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
   - After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.
   - After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.