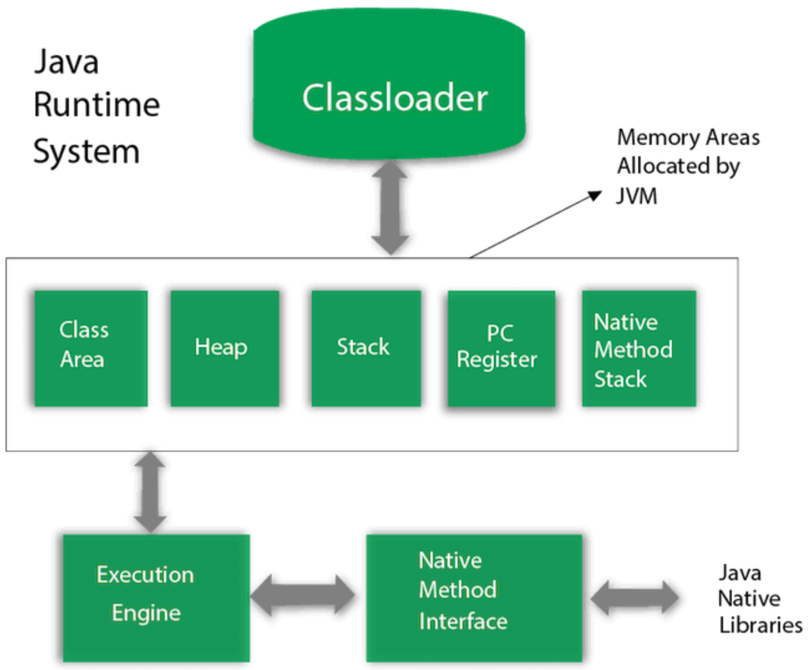
Years - 2016 Group "A"

Discuss JVM. Mention features of java. Write a program in java to demonstrate multilevel inheritance.

JVM:



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

- **Bootstrap ClassLoader:** This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
- **Extension ClassLoader:** This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA_HOME/jre/lib/ext directory.
- **System/Application ClassLoader:** This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

- A virtual processor
- Interpreter: Read bytecode stream then execute the instructions.
- Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Following are the notable features of Java:

1. **Object Oriented**

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

2. **Platform Independent**

Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

3. **Simple**

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

4. **Secure**

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

5. **Architecture-neutral**

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

6. **Portable**

Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

7. **Robust**

Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

8. **Multithreaded**

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

9. **Interpreted**

Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

10. **High Performance**

With the use of Just-In-Time compilers, Java enables high performance.

11. **Distributed**

Java is designed for the distributed environment of the internet.

12. **Dynamic**

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

Multilevel inheritance - A class inherits properties from a class which again has inherits properties.

```
class Shape {
  public void display() {
    System.out.println("Inside display");
  }
}
class Rectangle extends Shape {
  public void area() {
    System.out.println("Inside area");
  }
}
class Cube extends Rectangle {
  public void volume() {
    System.out.println("Inside volume");
  }
}
public class Tester {
  public static void main(String[] arguments) {
    Cube cube = new Cube();
    cube.display();
    cube.area();
    cube.volume();
  }
}
```

Output

```
Inside display
Inside area
Inside volume
```

Write a connection oriented socket program which can exchange the message between server and client once the connection is established.

Creating Server:

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
Socket s=new Socket("localhost",6666);
```

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

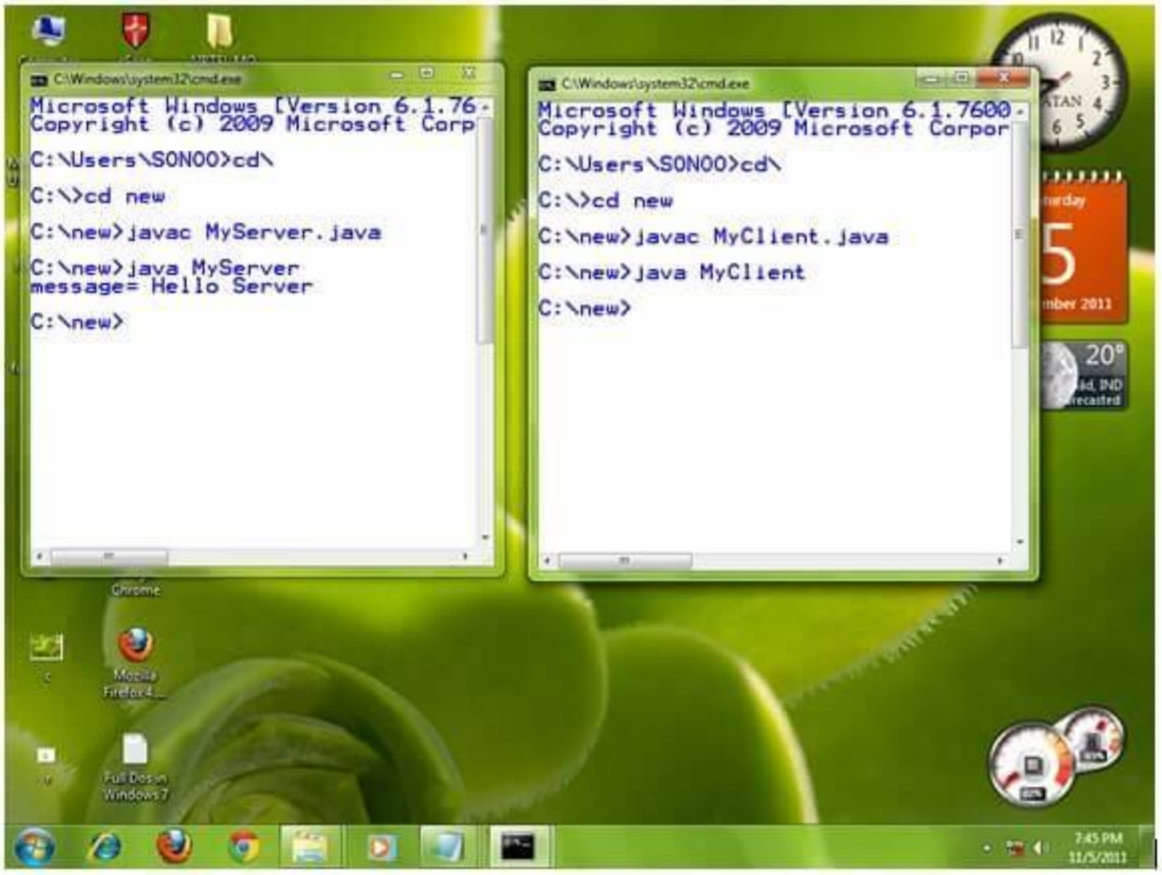
```
import java.io.*;
import java.net.*;
public class MyServer {
        public static void main(String[] args){
        try{
                ServerSocket ss=new ServerSocket(6666);
                Socket s=ss.accept();//establishes connection
                DataInputStream dis=new DataInputStream(s.getInputStream());
                String  str=(String)dis.readUTF();
                System.out.println("message= "+str);
                ss.close();
        }catch(Exception e){System.out.println(e);}
        }
}
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
        public static void main(String[] args) {
        try{
                Socket s=new Socket("localhost",6666);
                DataOutputStream dout=new DataOutputStream(s.getOutputStream());
                dout.writeUTF("Hello Server");
                dout.flush();
                dout.close();
```

```
                        s.close();
                }catch(Exception e){System.out.println(e);}
                }
        }
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.
After running the client application, a message will be displayed on the server console.



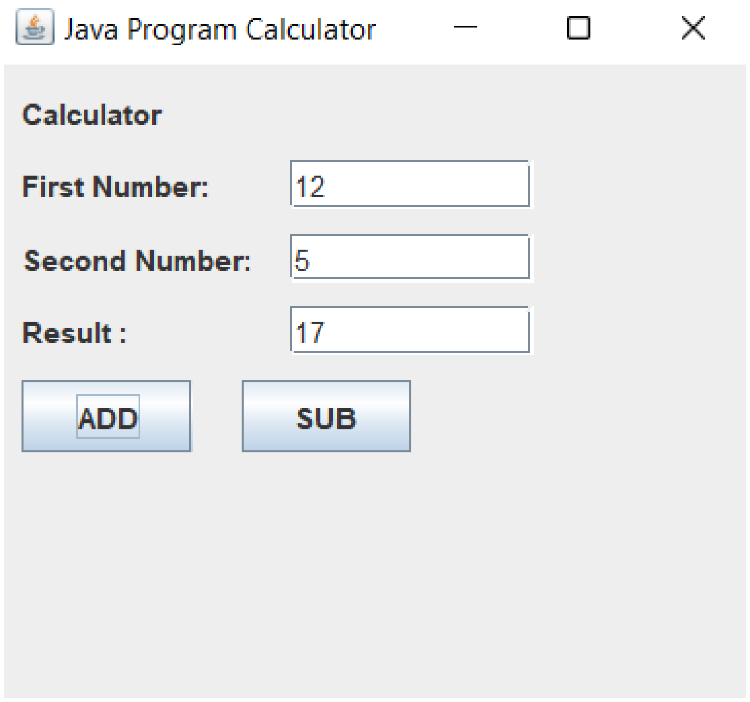
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

Write a GUI program to calculate the sum and difference of two numbers input by users as shown as below window.

```
package com.company.BoardQuestion;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyFrame extends JFrame implements ActionListener{
   private Container c;
   private JLabel label, label1, label2 ;
   private JTextField t1,t2,t3;
   private JButton add, sub;
   private JLabel result;
```

```
MyFrame(){
  // for title
  setTitle("Java Program Calculator");
  setSize(300,300);
  setLocation(100,100);

  c=getContentPane();
  c.setLayout(null);

  label = new JLabel("Calculator");
  label.setBounds(10,10,100,20);
  c.add(label);

  // text and input
  label1 = new JLabel("First Number: ");
  label1.setBounds(10,40,100,20);
  c.add(label1);
  t1 = new JTextField();
  t1.setBounds(120,40,100,20);
  c.add(t1);

  label2 = new JLabel("Second Number: ");
  label2.setBounds(10,70,100,20);
  c.add(label2);
  t2 = new JTextField();
  t2.setBounds(120,70,100,20);
  c.add(t2);

  // show result
  result = new JLabel("Result : ");
  result.setBounds(10,100,100,20);
  c.add(result);
  t3 = new JTextField();
  t3.setBounds(120,100,100,20);
  c.add(t3);

  // for buttion add and sub
  add = new JButton("ADD");
  add.setBounds(10,130,70,30);
  c.add(add);

  sub = new JButton("SUB");
```



```
        sub.setBounds(100,130,70,30);
        c.add(sub);

        // for action to button
        add.addActionListener(this);
        sub.addActionListener(this);

        // for show and exit
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == add){
            int a =  Integer.parseInt(t1.getText());
            int b =  Integer.parseInt(t2.getText());
            int c = a + b;
            //result.setText("Result : " + c);
            t3.setText(String.valueOf(c));
        }
        if (e.getSource() == sub){
            int a =  Integer.parseInt(t1.getText());
            int b =  Integer.parseInt(t2.getText());
            int c = a - b;
            t3.setText(String.valueOf(c));
        }
    }
}
public class Q_2016_3 {
    public static void main(String[] args) {
        MyFrame frame = new MyFrame();
    }
}
```

Group : "B"

Differentiate String and String Buffer class. Explain exception handling mechanism with an example.

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.

3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Example of String

```
public class Main {
  public static void main(String args[]) {
    String s1 = "Hello Tutorials Point";
    String upperCase = s1.toUpperCase();
    System.out.println(upperCase);
  }
}
```

Example of StringBuffer

```
public class StringBufferExample{
  public static void main(String[] args){
    StringBuffer buffer=new StringBuffer("Hi");
    buffer.append("Java 8");
    System.out.println("StringBufferExample" +buffer);
  }
}
```

Exception Handling in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

What is an Exception?

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception such as the name and description of the exception and the state of the program when the exception occurred.

An exception can occur for many reasons. Some of them are:

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

How to handle exceptions in Java

Let's start with the basics of exception handling in Java before we move to more advanced topics. The try-catch is the simplest method of handling exceptions. Put the code you want to run in the try block, and any Java exceptions that the code throws are caught by one or more catch blocks.

This method will catch any type of Java exceptions that get thrown. This is the simplest mechanism for handling exceptions.

```
try {
  // block of code that can throw exceptions
} catch (Exception ex) {
  // Exception handler
}
```

Note: You can’t use a try block alone. The try block should be immediately followed either by a catch or finally block.

Example:

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

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

Explain the multithreaded programming in java with example.

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

- Extending the Thread class
- Implementing the Runnable Interface

Advantages of multithread:

- The users are not blocked because threads are independent, and we can perform multiple operations at times
- As such the threads are independent, the other threads won’t get affected if one thread meets an exception.

Example of Multi thread:

```
package demotest;
```

```
public class GuruThread1 implements Runnable
{
      public static void main(String[] args) {
       Thread guruThread1 = new Thread("Guru1");
       Thread guruThread2 = new Thread("Guru2");
       guruThread1.start();
       guruThread2.start();
       System.out.println("Thread names are following:");
       System.out.println(guruThread1.getName());
       System.out.println(guruThread2.getName());
    }
    @Override
    public void run() {
    }
}
```

What is JDBC? Explain different JDBC drivers.

What is JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver

1) 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.

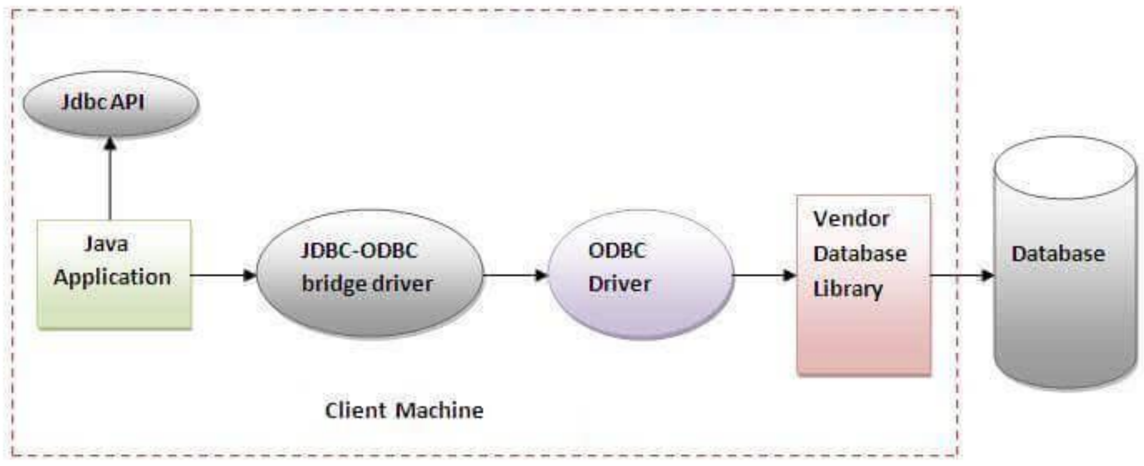


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- Easy to use.
- Can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API 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.

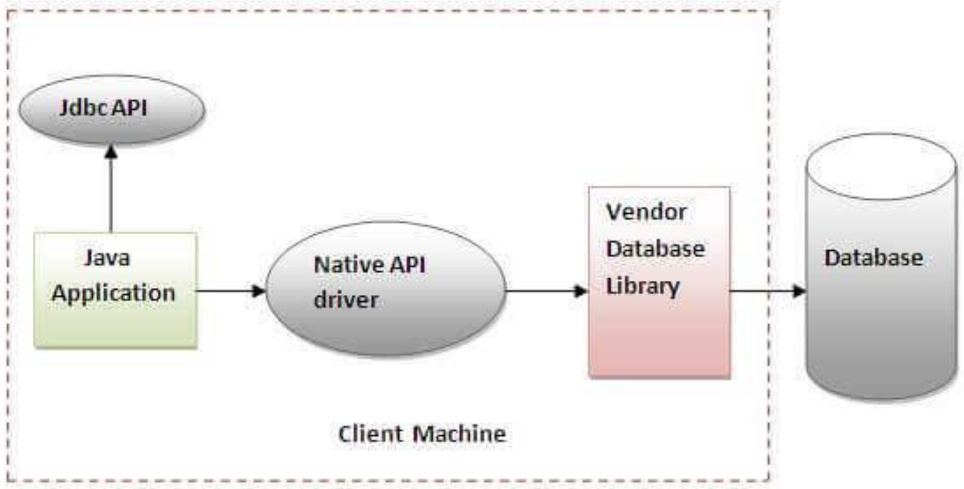


Figure- Native API Driver

Advantage:

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol 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.

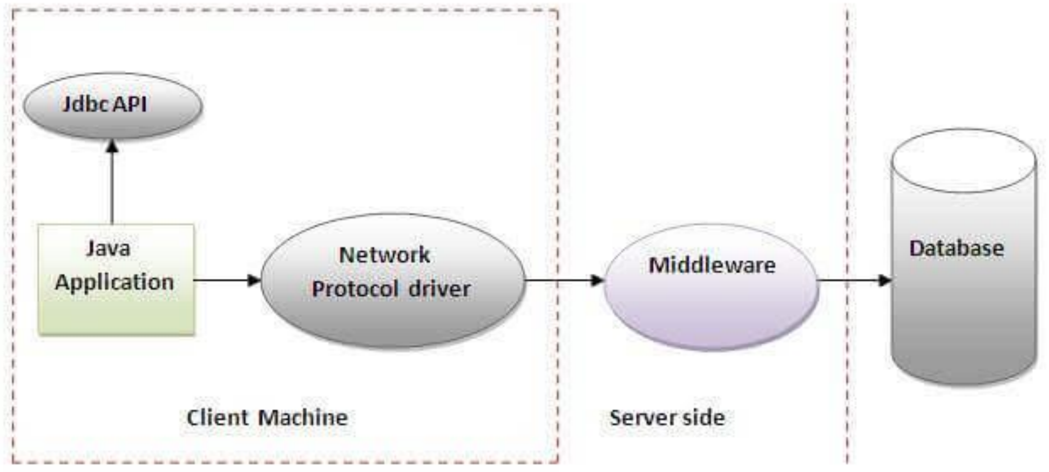


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin 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.

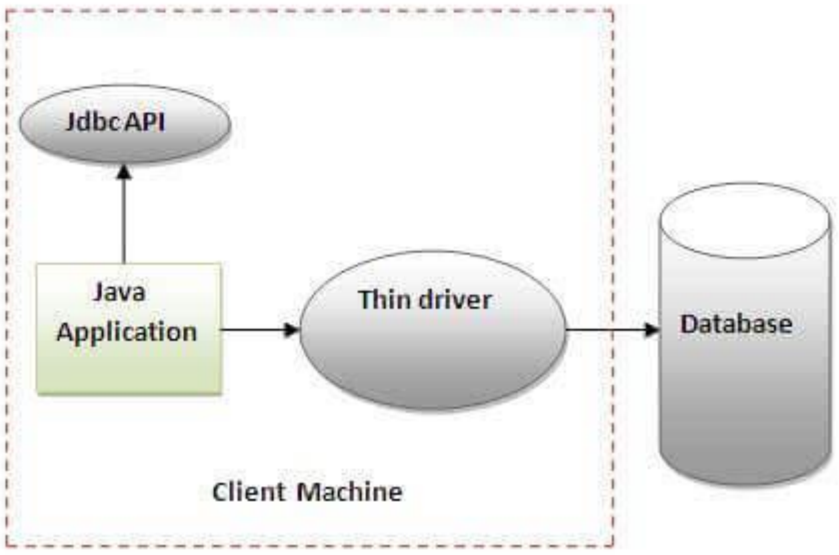


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

What is the importance of Stub and Skeleton? Explain RMI architecture with block diagram.

The Stub/Skeleton hides the communication details away from the developer. The Stub is the class that implements the remote interface. It serves as a client-side placeholder for the remote object. The stub communicates with the server-side skeleton. The skeleton is the stub's counterpart on server-side. Both communicate via the network. The skeleton actually knows the real remote objects delegates the stub's request to it and returns the response to the stub. You require both as they are the essential building blocks for RMI.

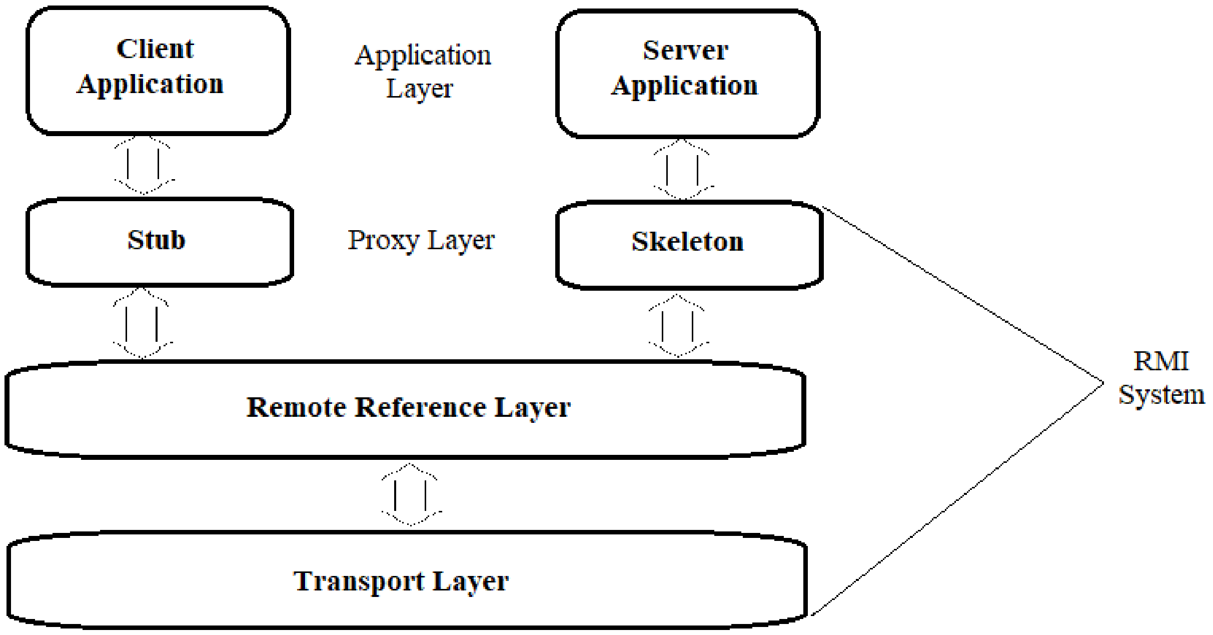
Remote Method Invocation (RMI)

- The RMI stands for Remote Method Invocation is an API mechanism.
- This mechanism allows an object residing in one system (JVM) to access an object running on another JVM.
- This mechanism generally creates distributed applications in java.
- The RMI provides remote communication between the java applications using two objects called stub and skeleton.

Architecture of Remote Method Invocation (RMI)

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

- Java RMI application contains two types of programs such as server program and client program.
- In the server-side program, a remote object is created, and a reference of that remote object is made available for the client-side using registry.
- The client-side program requests the remote objects on the server and tries to invoke its methods.



Stub

- 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.
- When the caller invokes a method on the stub object, it 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.

Skeleton

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

Transport Layer

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

- It is responsible for the transportation of data from one machine to another.
- The default connection is set up only in the TCP/IP protocol.

RRL(Remote Reference Layer)

- It is the layer that manages the references made by the client to the remote object.
- This layer gets a stream-oriented connection from the transport layer.
- It is responsible for dealing with the semantics of remote invocations.
- It is also responsible for handling duplicated objects and for performing implementation-specific tasks with remote objects.

Describe the function of file class? Create a DataInputStream for a file name "purbanchal.dat" and store "I am students of BIT VI semester" in that file.

Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion, etc.
The File object represents the actual file/directory on the disk. Following is the list of constructors to create a File object.

Sr.No.	Method & Description
1	File(File parent, String child) This constructor creates a new File instance from a parent abstract pathname and a child pathname string.
2	File(String pathname) This constructor creates a new File instance by converting the given pathname string into an abstract pathname.
3	File(String parent, String child) This constructor creates a new File instance from a parent pathname string and a child pathname string.
4	File(URI uri) This constructor creates a new File instance by converting the given file: URI into an abstract pathname.

Example:

```
package com.company.BoardQuestion;
import java.io.*;
import java.io.IOException;
import java.util.Scanner;

public class Q_2016_8 {
   public static void main(String[] args) throws IOException {
      // create a new file
      File myFile = new File("purbanchal.dat");
      myFile.createNewFile();
      System.out.println("File is Created : " + myFile);

      // write text in file
      FileWriter fileWriter = new FileWriter("purbanchal.dat");
```

```
            Scanner sc = new Scanner(System.in);
            System.out.println("Write text here....");
            String input = sc.nextLine();
            fileWriter.write(input);
            fileWriter.close();
        }
    }
```

Output:

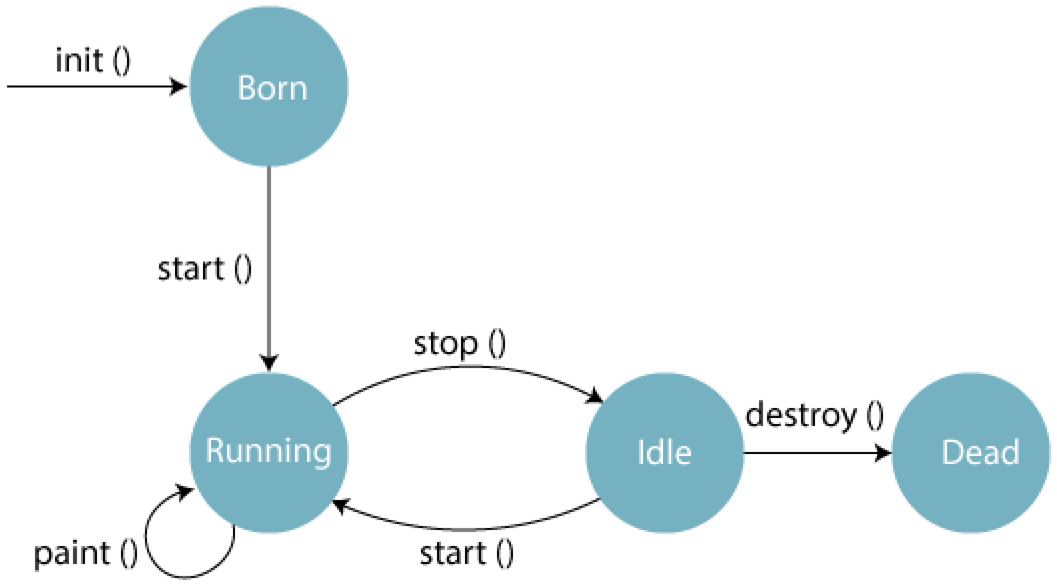
```
    File is Created : purbanchal.dat
    Write text here....
    I am students of BIT VI semester
```

Explain the life cycle of applet with a block diagram.

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy().These methods are invoked by the browser to execute.

Along with the browser, the applet also works on the client side, thus having less processing time.

Methods of Applet Life Cycle



There are five methods of an applet life cycle, and they are:

init(): The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.

start(): The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser. It is in an inactive state until the init() method is invoked.

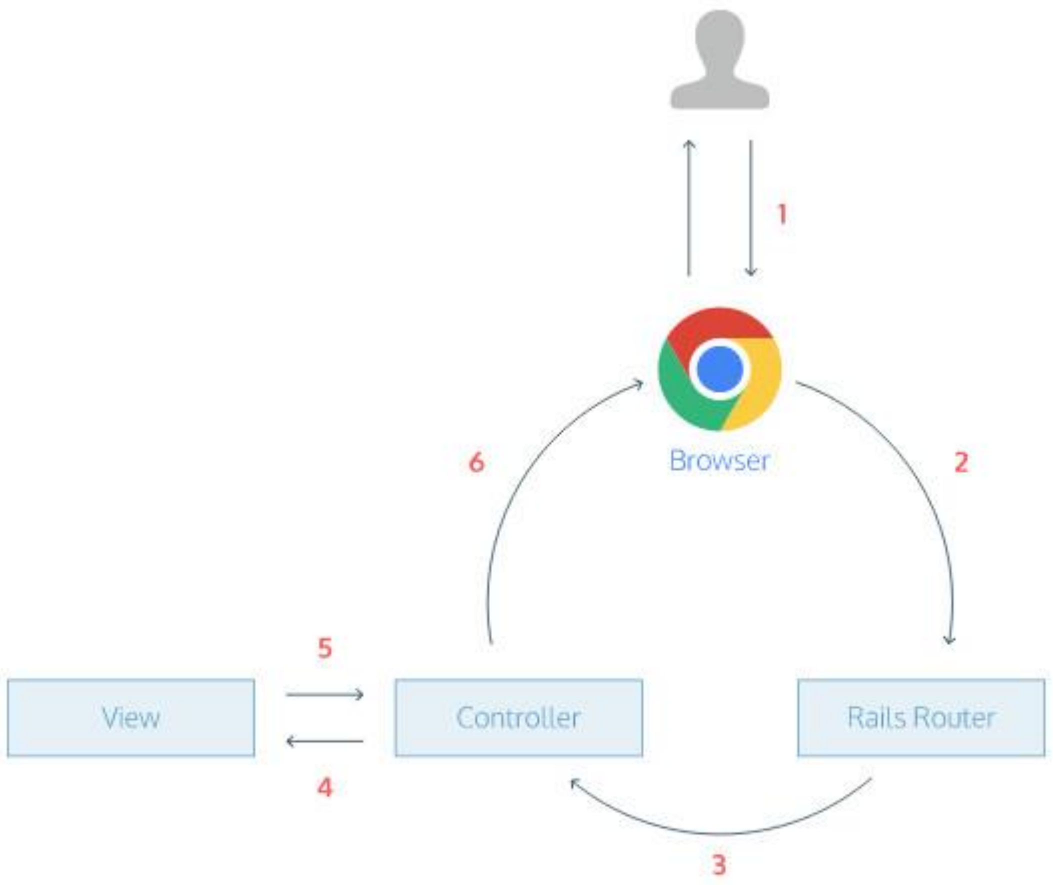
stop(): The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.
destroy(): The destroy() method destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed. It removes the applet object from memory and is executed only once. We cannot start the applet once it is destroyed.
paint(): The paint() method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the start() method and when the browser or applet windows are resized.

What is request/response circle? What are the advantage of using Servlets?

Request/response circle

When developing a Rails app, the request/response cycle is a useful guide to see how all the app's files and folders fit together. The request/response cycle traces how a user's request flows through the app. Understanding the request/response cycle is helpful to figure out which files to edit when developing an app (and where to look when things aren't working).

How it Works



- A user opens their browser, types in a URL, and presses Enter.
- When a user presses Enter, the browser makes a request for that URL.

- The request hits the Rails router (config/routes.rb). The router maps the URL to the correct controller and action to handle the request.
- The action receives the request and passes it on to the view.
- The view renders the page as HTML.
- The controller sends the HTML back to the browser. The page loads and the user sees it.

In this way, the request/response cycle is a useful way to see what a Rails app's files and folders are for and how they fit together.

Advantages of using Servlets

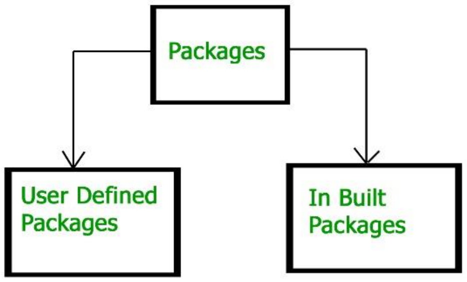
- Servlets load only one copy into the Java Virtual Machine. This makes their memory efficient and faster.
- The response time is significantly less, as it saves time to respond to the first request.
- Servlets are easily accessible, as they use standard API that is used by a large number of web servers.
- It is easy for development and is platform-independent.
- Servlet's usage doesn't constrain the web servers.
- Servlets help developers access a large number of APIs, which are available for Java.
- It is very easy to maintain multiple Servlets for a single web application.
- Servlet containers provide developers with the facility of support to several other features like resource management, sessions, security, persistent, etc.
- If servlets have multiple requests, the web containers provide threads to handle more than one request.

Write short notes on any two

a. **Packages**

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).



Types of packages:

1. **Built-in Packages**

These packages consist of a large number of classes which are a part of Java API.Some of the commonly used built-in packages are:

1) java.lang: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) java.io: Contains classed for supporting input / output operations.
3) java.util: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) java.applet: Contains classes for creating Applets.
5) java.awt: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
6) java.net: Contain classes for supporting networking operations.

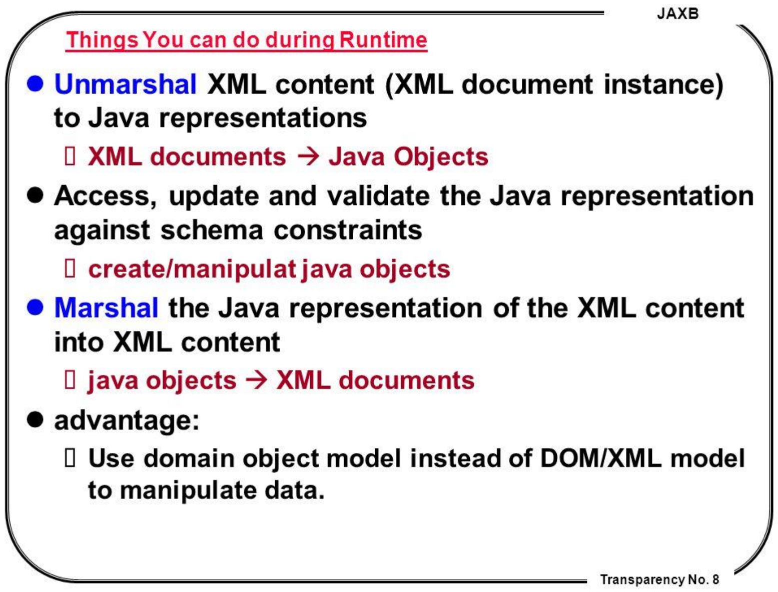
2. User-defined packages

These are the packages that are defined by the user. First we create a directory myPackage (name should be same as the name of the package). Then create the MyClass inside the directory with the first statement being the package names.

b. AWT vs Swing

AWT	**Swing**
Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
AWT provides comparatively less powerful components.	Swing provides more powerful components.

c. Marshals and unmarshals of parameters



JAXB

Things You can do during Runtime

- Unmarshal XML content (XML document instance) to Java representations
  - XML documents → Java Objects
- Access, update and validate the Java representation against schema constraints
  - create/manipulat java objects
- Marshal the Java representation of the XML content into XML content
  - java objects → XML documents
- advantage:
  - Use domain object model instead of DOM/XML model to manipulate data.

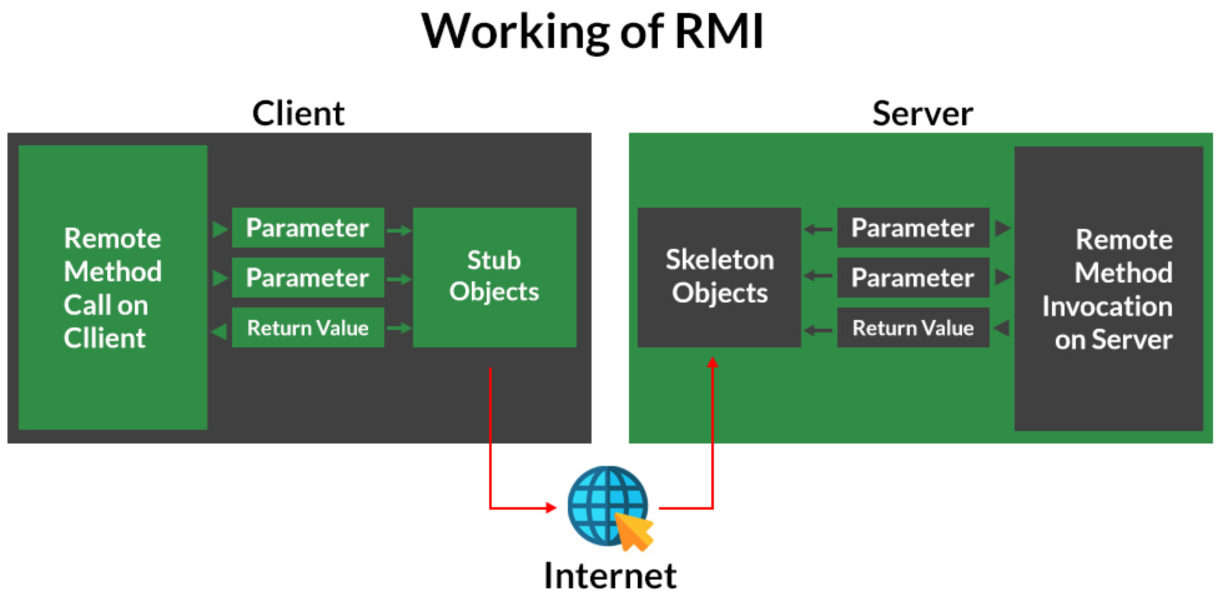
Transparency No. 8

Years : 2017   Group " A"

What is RMI? How does distributed application created using RMI? Explain with an example.

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.
RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package java.rmi.

Creating Distributed Application Using RMI
The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:



These are the steps to be followed sequentially to implement Interface as defined below as follows:

1. Define a remote interface.
2. Implementing remote interface.
3. create and start remote application
4. create and start client application

Define a remote interface
A remote interface specifies the methods that can be invoked remotely by a client. Clients program communicate to remote interfaces, not to classes implementing it. To be a remote interface, a interface must extend the Remote interface of java.rmi package.

```
import java.rmi.*;
public interface AddServerInterface extends Remote
{
public int sum(int a,int b);
}
```

Implementation of remote interface
For implementation of remote interface, a class must either extend UnicastRemoteObject or use exportObject() method of UnicastRemoteObject class.

```
import java.rmi.*;
import java.rmi.server.*;
public class Adder extends UnicastRemoteObject implements AddServerInterface
{
	Adder()throws RemoteException{
	super();
}
public int sum(int a,int b)
{
	return a+b;
}
}
```

Create AddServer and host rmi service

You need to create a server application and host rmi service Adder in it. This is done using rebind() method of java.rmi.Naming class. rebind() method take two arguments, first represent the name of the object reference and second argument is reference to instance of Adder

```
import java.rmi.*;
import java.rmi.registry.*;
public class AddServer {
	public static void main(String args[]) {
		try {
			AddServerInterface addService=new Adder();
			Naming.rebind("AddService",addService);
		      //addService  object is hosted with name AddService

		}
		catch(Exception e) {
			System.out.println(e);
		}
	}
}
```

Create client application

Client application contains a java program that invokes the lookup() method of the Naming class. This method accepts one argument, the rmi URL and returns a reference to an object of type AddServerInterface. All remote method invocation is done on this object.

```
import java.rmi.*;
public class Client {
	public static void main(String args[]) {
		try{
			AddServerInterface st =
(AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddService");
```

```
                    System.out.println(st.sum(25,8));
                }
                catch(Exception e) {
                    System.out.println(e);
                }
            }
        }
```

What is socket programming? Create chat program using TCP.

What is Socket Programming in Java?
Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other in order to form a connection.

Create chat program using TCP
Creating Server:
To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

```
    ServerSocket ss=new ServerSocket(6666);
    Socket s=ss.accept();//establishes connection and waits for the client
```

Creating Client:
To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

```
    Socket s=new Socket("localhost",6666);
```

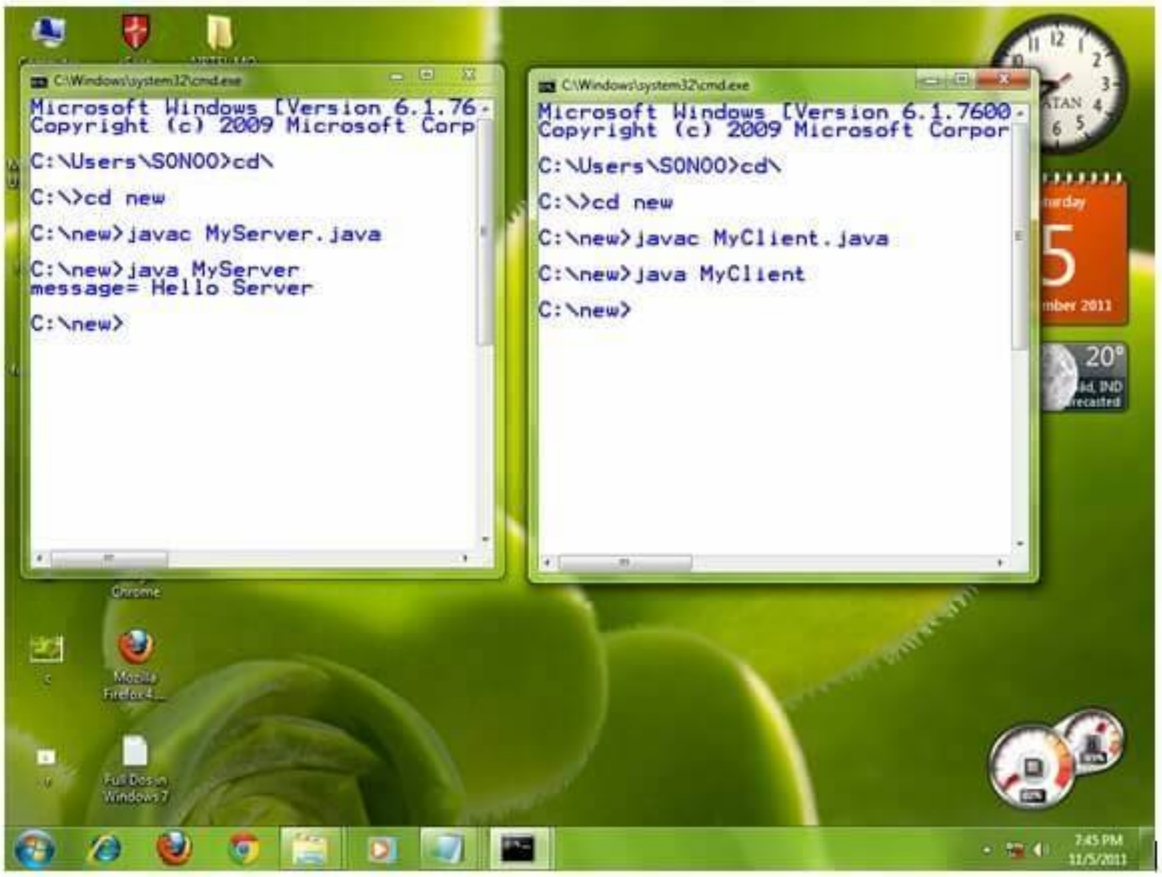
Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

File: MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```



```
Microsoft Windows [Version 6.1.76
Copyright (c) 2009 Microsoft Corp

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyServer.java

C:\new>java MyServer
message= Hello Server

C:\new>
```

```
Microsoft Windows [Version 6.1.7600
Copyright (c) 2009 Microsoft Corpor

C:\Users\SONOO>cd\

C:\>cd new

C:\new>javac MyClient.java

C:\new>java MyClient

C:\new>
```

What is swing? List out the features of swing. Create a swing application whose snapshot is as figure below.

Swing is a set of program component s for Java programmers that provide the ability to create graphical user interface (GUI) components, such as buttons and scroll bars, that are independent of the windowing system for specific operating system . Swing components are used with the Java Foundation Classes (JFC).

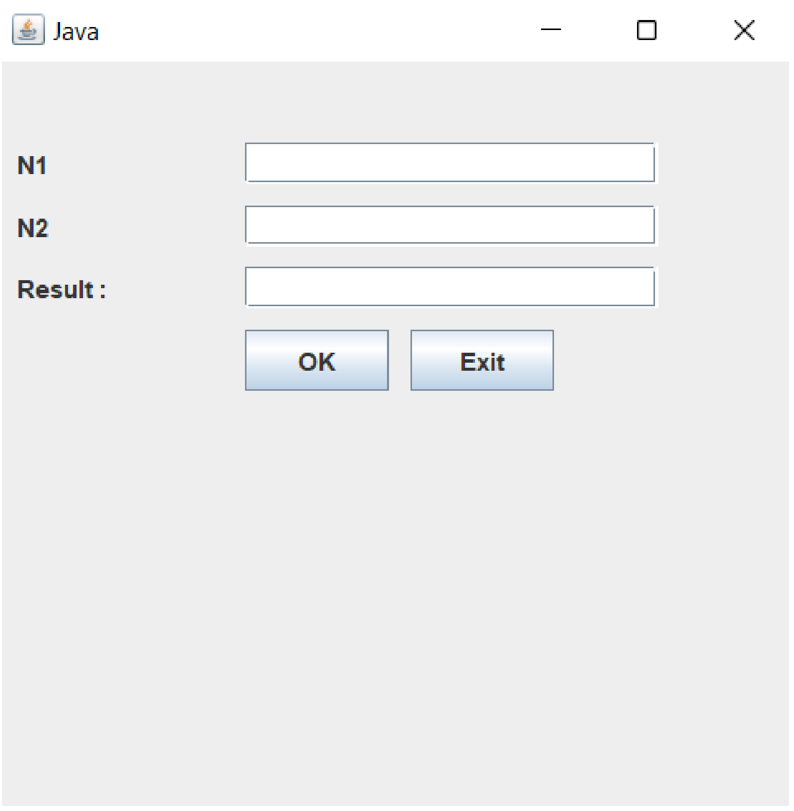
Swing Features

1. Light Weight − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
2. Rich Controls − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
3. Highly Customizable − Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.
4. Pluggable look-and-feel − SWING based GUI Application look and feel can be changed at run-time, based on available values.

Example:

```
package com.company.BoardQuestion;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class MyFrame1 extends JFrame implements ActionListener {
  private Container c;
  private JLabel  label1, label2 ;
  private JTextField t1,t2,t3;
  private JButton add, sub;
  private JLabel result;

  MyFrame1(){
    // for title
    setTitle("Java");
    setSize(400,400);
    setLocation(100,100);

    c=getContentPane();
    c.setLayout(null);

    // text and input
    label1 = new JLabel("N1 ");
    label1.setBounds(10,40,100,20);
    c.add(label1);
    t1 = new JTextField();
    t1.setBounds(120,40,200,20);
    c.add(t1);
```

```
        label2 = new JLabel("N2 ");
        label2.setBounds(10,70,100,20);
        c.add(label2);
        t2 = new JTextField();
        t2.setBounds(120,70,200,20);
        c.add(t2);

        // show result
        result = new JLabel("Result : ");
        result.setBounds(10,100,100,20);
        c.add(result);
        t3 = new JTextField();
        t3.setBounds(120,100,200,20);
        c.add(t3);

        // for buttion add and sub
        add = new JButton("OK");
        add.setBounds(120,130,70,30);
        c.add(add);

        sub = new JButton("Exit");
        sub.setBounds(200,130,70,30);
        c.add(sub);

        // for action to button
        add.addActionListener(this);
        sub.addActionListener(this);

        // for show and exit
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == add){
            int a =  Integer.parseInt(t1.getText());
            int b =  Integer.parseInt(t2.getText());
            int c = a + b;
            t3.setText(String.valueOf(c));
        }
    }
}
```



```
public class Q_2017_3 {
   public static void main(String[] args) {
      MyFrame1 frame = new MyFrame1();
   }
}
```

Group "B"

What is byte code? Explain JVM and JRE.

What is Java Bytecode?

Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

What is JRE?

JRE is a piece of a software which is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs. Though, all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.

Why use JRE?

Here are the important reasons of using JRE:

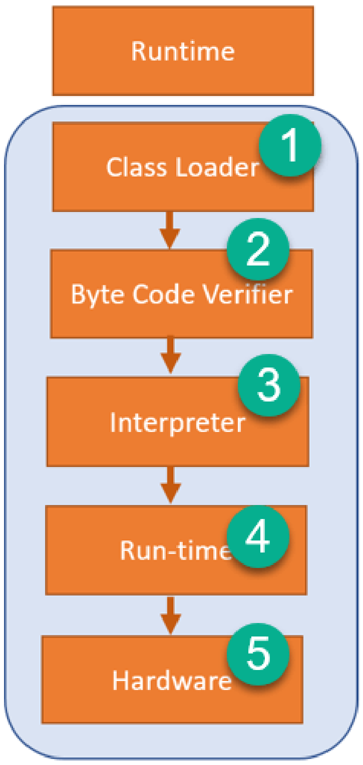
- JRE contains class libraries, JVM, and other supporting files. It does not contain any tool for Java development like a debugger, compiler, etc.
- It uses important package classes like math, swingetc, util, lang, awt, and runtime libraries.
- If you have to run Java applets, then JRE must be installed in your system.

Features of JRE

Here are the important features of JRE:

- Java Runtime Environment is a set of tools using which the JVM actually runs.
- JRE contains deployment technology, including Java Web Start and Java Plug-in.
- Developers can easily run the source code in JRE, but he/she cannot write and compile the Java program.
- It includes integration libraries like Java Database Connectivity (JDBC), Remote Method Invocation (RMI), Java Naming and Directory Interface (JNDI), and more.
- JRE has JVM and Java HotSpot virtual machine client.

How JRE Functions?



JRE Functionality

JRE has an instance of JVM with it, library classes, and development tools. Once you write and compile Java code, the compiler generates a class file having byte code.

Here are the important components of JRE:

- **Class loaders:** The class loader loads various classes that are necessary for running a Java program. JVM uses three class loaders called the bootstrap class loader, extensions class loader, and system class loader.
- **Byte code verifier:** Byte code verifier verifies the bytecode so that the code doesn't disturb the interpreter.
- **Interpreter:** Once the classes get loaded, and the code is verified, the interpreter reads the code line by line.
- **Run-time:** Run-time is a system used mainly in programming to describe time period during which a particular program is running.
- **Hardware:** Once you compile Java native code, it runs on a specific hardware platform.

What is JVM?

JVM is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of Java Run Environment (JRE). It cannot be separately downloaded and installed. To install JVM, you need to install JRE. The full form of JVM is Java Virtual Machine.

In many other programming languages, the compiler produces machine code for a specific system. However, Java compiler produces code for a virtual machine which is called as JVM.

Why JVM?

Here are the important reasons of using JVM:

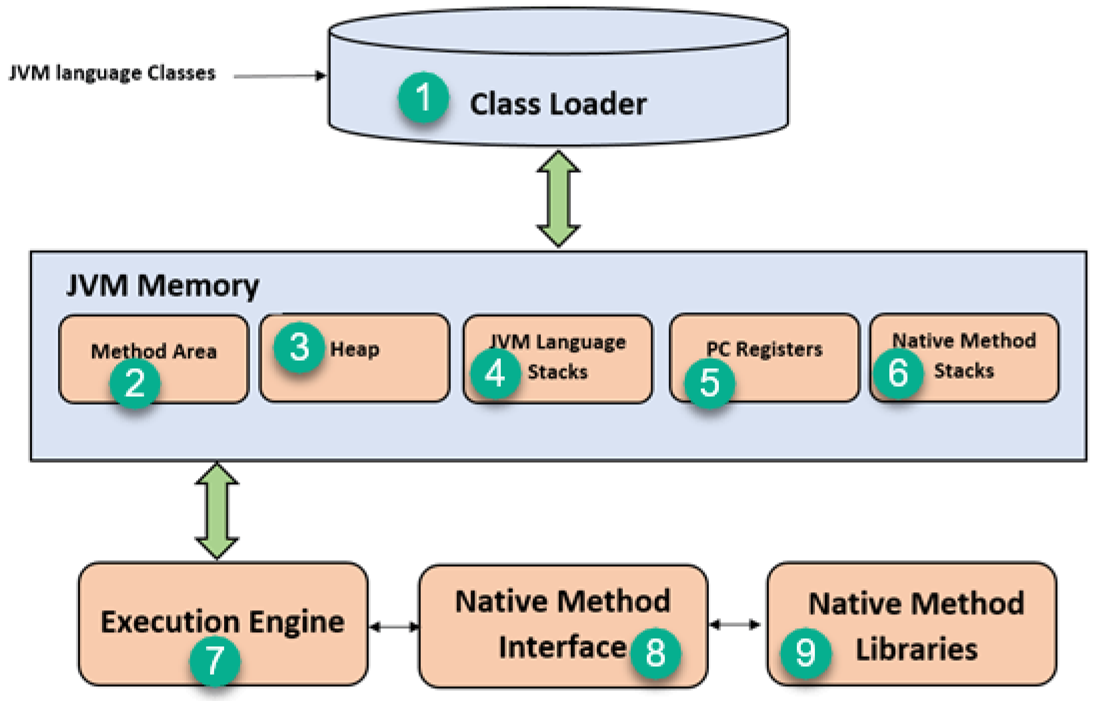
- JVM provides a platform-independent way of executing Java source code.
- It has numerous libraries, tools, and frameworks.
- Once you run Java program, you can run on any platform and save lots of time.
- JVM comes with JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs more faster as a regular application.

Features of JVM

Here are the important features of JVM:

- It enables you to run applications in a cloud environment or in your device.
- Java Virtual Machine converts byte code to the machine-specific code.
- It provides basic java functions like memory management, security, garbage collection, and more.
- JVM runs the program by using libraries and files given by Java Runtime Environment.
- JDK and JRE both contain Java Virtual Machine.
- It can execute the java program line by line hence it is also called as interpreter.
- JVM is easily customizable for example, you can allocate minimum and maximum memory to it.
- It is independent from hardware and the operating system. So, you can write a java program once and run anywhere.

How JVM Functions?



JVM functionality

Here are the important components of JVM:

1) Class Loader

The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.

2) Method Area

JVM Method Area stores structure of class like metadata, the code for Java methods, and the constant runtime pool.

3) Heap

All the Objects, arrays, and instance variables are stored in a heap. This memory is shared across multiple threads.

4) JVM language Stacks

Java language Stacks store local variables, and its partial results. Each and every thread has its own JVM language stack, created concurrently as the thread is created. A new frame is created when method is invoked, and it is removed when method invocation process is complete.

5) PC Registers

PC registers store the address of the Java virtual machine instruction, which is currently executing. In Java, each thread has its separate PC register.

6) Native Method Stacks

Native method stacks hold the instruction of native code depends on the native library. It allocates memory on native heaps or uses any type of stack.

7) Execution Engine

It is a type of software that is used to test software, hardware, or complete systems. The test execution engine never carries any information about the tested product.

8) Native Method interface

The Native Method Interface is a programming framework. It allows Java code, which is running in a JVM to call by libraries and native applications.

9) Native Method Libraries

Native Libraries is a collection of the Native Libraries (C, C++), which are needed by the Execution Engine.

What is thread? What are the merits of multithreaded programming? Explain two ways of creating thread with example.

What is a Thread in Java?

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program’s execution. At this point, when the main thread is provided, the main() method is invoked by the main thread.

Advantages of multithreading:

- Enhanced performance by decreased development time
- Simplified and streamlined program coding
- Improvised GUI responsiveness
- Simultaneous and parallelized occurrence of tasks
- Better use of cache storage by utilization of resources
- Decreased cost of maintenance
- Better use of CPU resource

Disadvantages of multithreading:

- Complex debugging and testing processes
- Overhead switching of context
- Increased potential for deadlock occurrence

- Increased difficulty level in writing a program
- Unpredictable results

How to create a thread in Java

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

Thread class:

This is another way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run() method which is the entry point of new thread.

Example:

FileName: Multi.java

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

Output:

```
thread is running...
```

Runnable interface:

It introduces a concurrent thread into your program. This thread will end when run() method terminates. You must specify the code that your thread will execute inside run() method.

run() method can call other methods, can use other classes and declare variables just like any other normal method.

Example:

FileName: Multi3.java

```
class Multi3 implements Runnable{
        public void run(){
                System.out.println("thread is running...");
        }
        public static void main(String args[]){
                Multi3 m1=new Multi3();
                Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
                t1.start();
         }
}
```

Output:

```
thread is running...
```

Explain differences between servlet and JSP.

Servlet	JSP
Servlets are faster as compared to JSP, as they have a short response time.	JSP is slower than Servlets, as the first step in the JSP lifecycle is the conversion of JSP to Java code and then the compilation of the code.
Servlets are Java-based codes.	JSP are HTML-based codes.
Servlets are harder to code, as here, the HTML codes are written in Java.	JSPs are easier to code, as here Java is coded in HTML.
In an MVC architecture, Servlets act as the controllers.	In MVC architectures, the JSPs act as a view to present the output to the users.
The service() function can be overridden in Servlets.	The service() function cannot be overridden in JSPs.
The Servlets are capable of accepting all types of protocol requests.	The JSPs are confined to accept only the HTTP requests.
Modification in Servlets is a time-consuming and challenging task, as here, one will have to reload, recompile, and then restart the servers.	Modification is easy and faster in JSPs as we just need to refresh the pages.
Servlets require the users to enable the default sessions management explicitly, as Servlets do not provide default session management.	JSPs provide session management by default.
Servlets require us to implement the business logic and presentation logic in the same servlet file.	JSPs give us the flexibility to separate the business logic from the presentation logic using javaBeans.
Servlets can handle extensive data processing.	JSPs cannot handle data processing functions efficiently.
Servlets do not provide the facility of writing custom tags.	JSPs can provide the facility of building the JSP tags easily, which can directly call javaBeans.
In Servlets, we do not have implicit objects.	In JSPs, we have support for implicit objects.
Servlets are hosted and executed on Web Servers.	JSP is compiled in Java Servlets before their execution. After that, it has a similar lifecycle as Servlets.
We need to import all the packages at the top of the Servlets.	In JSPs, we can import packages anywhere in the file.

What is an applet? Explain its life cycle with example.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

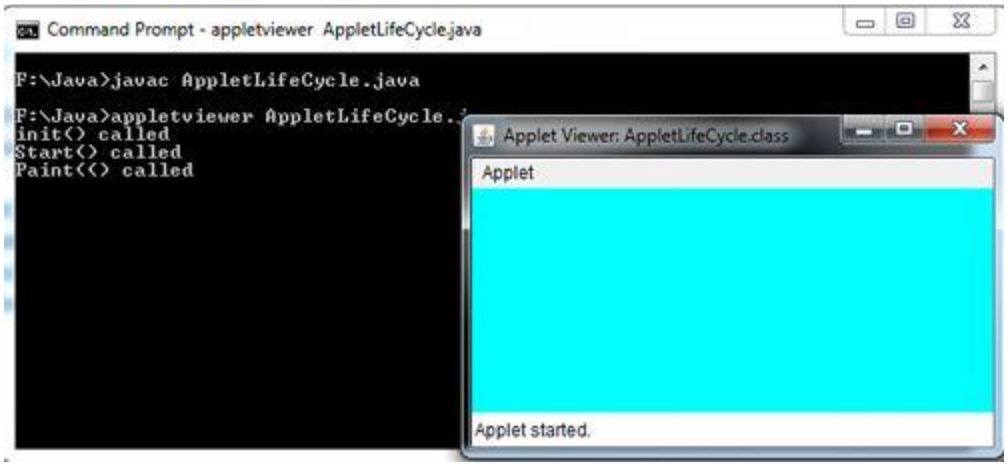
Drawback of Applet

- Plugin is required at client browser to execute applet.

Example:

```
import java.applet.Applet;
 import java.awt.Graphics;
 import java.awt.*;
 /*<applet code="AppletLifeCycle.class" width="350" height="150"> </applet>*/
 public class AppletLifeCycle extends Applet
 {
     public void init()
     {
       setBackground(Color.CYAN);
       System.out.println("init() called");
     }
     public void start(){ System.out.println("Start() called"); }
     public void paint(Graphics g){ System.out.println("Paint(() called"); }
     public void stop() { System.out.println("Stop() Called"); }
     public void destroy()   { System.out.println("Destroy)() Called"); }
 }
```

Output:


Command Prompt - appletviewer AppletLifeCycle.java
Applet Viewer: AppletLifeCycle.class
Applet

```
F:\Java>javac AppletLifeCycle.java

F:\Java>appletviewer AppletLifeCycle.j
init() called
Start() called
Paint(() called
```

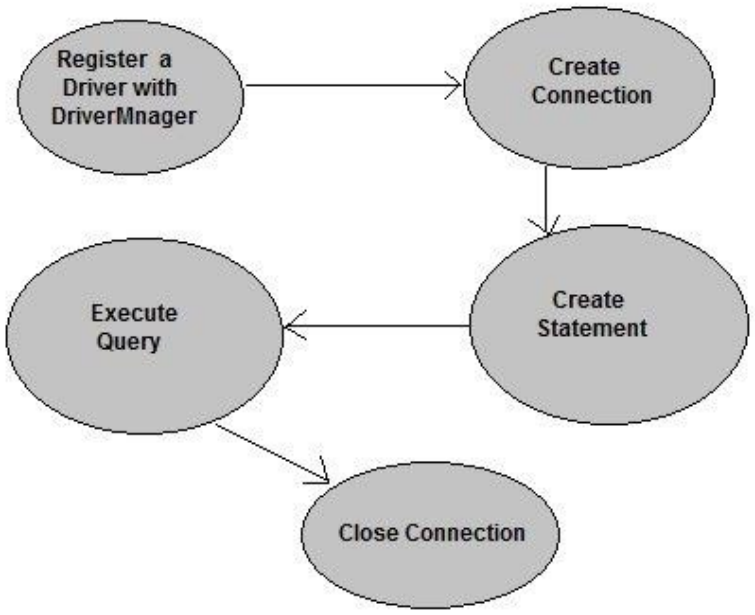
```
Applet started.
```

What is JDBC? Explain the steps for connecting with any database using example.

Setting in a database

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection



1. **Register the driver class**

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

2. **Create a Connection**

After registering and loading the driver in step1, now we will create a connection using getConnection() method of DriverManager class. This method has several overloaded methods that can be used based on the requirement. Basically it require the database name, username and password to establish connection. Syntax of this method is given below.

3. **Create the Statement object**

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

4. **Execute the query**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

5. **Close the connection object**

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Example:
```
import java.sql.*;
class FirstJDBC {
   public static void main(String[] args) {
        try{
           // Load the driver:
           //Class.forName("com.mysql.jdbc.Driver");

           //creating a connection
           String url="jdbc:mysql://localhost:3306/java_db";
           String username = "root";
           String password = "";
           Connection con = DriverManager.getConnection(url,username,password);

           if (con.isClosed()){
              System.out.println("Connection is Closed");
           }else {
              System.out.println("Connection is Created...");
           }
        }catch (Exception e){
           e.printStackTrace();
        }
     }
  }
```

Write short note

a. Vector

Vector is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the List interface, so we can use all the methods of List interface here.

It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.
The Iterators returned by the Vector class are fail-fast. In case of concurrent modification, it fails and throws the ConcurrentModificationException.
It is similar to the ArrayList, but with two differences-

- Vector is synchronized.
- Java Vector contains many legacy methods that are not the part of a collections framework.

Java Vector class Declaration

```
public class Vector<E>
extends Object<E>
implements List<E>, Cloneable, Serializable
```

b. JAR

A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.

In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We can imagine a .jar file as a zipped file(.zip) that is created by using WinZip software. Even, WinZip software can be used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

Let us see how to create a .jar file and related commands which help us to work with .jar files

1. **Create a JAR file**

In order to create a .jar file, we can use jar cf command in the following ways as discussed below:

***Syntax**:*

```
jar cf jarfilename inputfiles
```

2. **View a JAR file**

Now, pack.jar file is created. In order to view a JAR file '.jar' files, we can use the command as:

Syntax:

```
jar tf jarfilename
```

3. **Extracting a JAR file**

In order to extract the files from a .jar file, we can use the commands below listed:

Syntax:

```
jar xf jarfilename
```

4. **Updating a JAR File**

The Jar tool provides a 'u' option that you can use to update the contents of an existing JAR file by modifying its manifest or by adding files. The basic command for adding files has this format as shown below:

Syntax:

```
jar uf jar-file input-file(s)
```

5. **Running a JAR file**

In order to run an application packaged as a JAR file (requires the Main-class manifest header), the following command can be used as listed:

Syntax:

```
C:\>java -jar pack.jar
```

c. Super and final keyword

Super Keyword

The super keyword in Java is a reference variable which is used to refer immediate parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Example:

```
class Vehicle
{
	int maxSpeed = 120;
}
/* sub class Car extending vehicle */
class Car extends Vehicle
{
	int maxSpeed = 180;

	void display()
	{
		/* print maxSpeed of base class (vehicle) */
		System.out.println("Maximum Speed: " + super.maxSpeed);
	}
}
/* Driver program to test */
class Test
{
	public static void main(String[] args)
	{
		Car small = new Car();
		small.display();
	}
}
```

Output:

```
Maximum Speed: 120
```

Final keyword

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike10();
 }
}
```
Output: 70

d. J2EE

J2EE stands for Java 2 Platform, Enterprise Edition. J2EE is the standard platform for developing applications in the enterprise and is designed for enterprise applications that run on servers. J2EE provides APIs that let developers create workflows and make use of resources such as databases or web services. J2EE consists of a set of APIs. Developers can use these APIs to build applications for business computing.
A J2EE application server is software that runs applications built with J2EE APIs and lets you run multiple applications on a single computer. Developers can use different J2EE application servers to run applications built with J2EE APIs

Explain interface in java. How does interface support polymorphism?

An interface in Java is a blueprint of a class. It has static constants and abstract methods.
The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
Java Interface also represents the IS-A relationship.
It cannot be instantiated just like the abstract class.
Since Java 8, we can have default and static methods in an interface.
Since Java 9, we can have private methods in an interface.

Syntax:
```
interface <interface_name>{
    // declare constant fields
   // declare methods that abstract
   // by default.
}
```
How does interface support polymorphism?

Java interfaces are a way to achieve polymorphism. Polymorphism is a concept that takes some practice and thought to master. Basically, polymorphism means that an instance of an class (an object) can be used as if it were of different types. Here, a type means either a class or an interface.
Interfaces formalize polymorphism. Interfaces allow us to define polymorphism in a declarative way, unrelated to implementation. Two elements are polymorphic with respect to a set of behaviors if they realize the same interfaces. You always heard that polymorphism was this big benefit of object orientation, but without interfaces there was no way to enforce it, verify it, or even express it, except in

informal ways, or language-specific ways. Formalization of interfaces strips away the mystery, and gives us a good way to describe, in precise terms, what polymorphism was trying to do all along. Interfaces are testable, verifiable, and precise.

Interfaces are the key to the "plug-and-play" ability of an architecture. Classes that realize the same interface may be substituted for one another in the system, thereby supporting the changing of implementations without affecting clients.

```
interface Stack
{
  public void push ( char item );  // inserts an item at the top
  public char pop ();              // removes an item from the top
  public char peek ();             // returns an item from the top
                                   // without removing
  public boolean isEmpty ();       // determines if the Stack is empty
  public boolean isFull ();        // determines if the Stack is full
  public String toString ();       // returns a String representation of
                                   // the Stack
}

class StackArray implements Stack
{
  private char stackArray[];      // array that implements the Stack
  private int top;                // index of the top element in the Stack
  // Constructor
  public StackArray ( int n )
  {
    stackArray = new char [ n ];
    top = -1;
  }

  // Implementation of the methods in the interface
  public void push ( char item )
  {  stackArray [ ++top ] = item;  }

  public char pop ()
  {  return stackArray [ top-- ];  }

  public char peek ()
  {  return stackArray [ top ];  }

  public boolean isEmpty ()
  {  return ( top < 0 );  }

  public boolean isFull ()
```

```
 {  return ( top == stackArray.length - 1 );  }

 public String toString ()
 { StringBuffer aBuffer = new StringBuffer();
   for ( int i = top; i >= 0; i-- )
   {
     aBuffer.append ( stackArray [ i ] + "  " );
   }
   return aBuffer.toString();
 }
}

public class TestStack
{
 public static void main ( String args [] )
 {
   Stack theStack = new StackArray ( 10 );
   char ch = ' ';

   if ( !theStack.isFull() )
     theStack.push ( 'a' );

   if ( !theStack.isFull() )
     theStack.push ( 'b' );

   if ( !theStack.isFull() )
     theStack.push ( 'c' );

   if ( !theStack.isEmpty() )
     ch = theStack.pop ();
   System.out.println ( "The item on top of the stack is " + ch );

   if ( !theStack.isEmpty() )
     ch = theStack.peek ();
   System.out.println ( "The item on top of the stack is " + ch );

   System.out.println ( theStack.toString() );
 }
}
```

Real-life Example - The real-world example of interfaces is that we have multiple classes for different levels of employees working in a particular company and the necessary property of the class is the salary of the employees and this. We must be implemented in every class and. Also, it is different for every employee here. The concept of the interface is used. We simply create an interface containing an abstract salary method and implement it in all the classes and we can easily define different salaries of the employees.

```
interface Salary{
   void insertSalary(int salary);
}

// implementing the salary in the class
class SDE1 implements Salary{
   int salary;
   public void insertSalary(int salary){
      this.salary = salary;
   }
   void printSalary(){
      System.out.println(this.salary);
   }
}

class SDE2 implements Salary{
   int salary;
   public void insertSalary(int salary){
      this.salary = salary;
   }
   void printSalary(){
      System.out.println(this.salary);
   }
}

public class Demo{
   public static void main(String args[]){
      SDE1 ob1 = new SDE1();
      ob1.insertSalary(10000);
      ob1.printSalary();

      SDE2 ob2 = new SDE2();
      ob2.insertSalary(20000);
      ob2.printSalary();
   }
}
```

Years : 2018

Differentiated swing and AWT

S.NO	**AWT**	**Swing**
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.

S.No.	AWT	SWING
1.	The full form of AWT is Abstract Window Toolkit.	It has no full version.
2.	It is an API used to develop window-based applications in Java.	Swing is a graphical user interface (GUI) and a part of Oracle’s Java Foundation Classes that are used to design different applications.
3.	Its components are heavy weighted.	Its components are light weighted.
4.	In Java AWT, the components are platform dependent.	In Java swing, the components are independent.
5.	The functionality of JAVA AWT is less as compared to the Java swing.	The functionality of the JAVA swing is higher than AWT.
6.	It requires more time for execution.	It requires less time for execution.
7.	It has less powerful components compared to the Java swing.	It has more powerful components than Java AWT.
8.	It does not support the MVC pattern.	It supports the MVC pattern.

Discuss any five classes to handle files in java.

1. File
2. fileReader
3. FileWriter
4. FileInputStream
5. FileOutputStream

Years : 2019   Group " A "

What makes RMI different with socket programming? Write program using RMI technology to send two numbers to the server and the server returning the greatest numbers between them.

Socket programming - you have to handle exactly which sockets are being used, you specify TCP or UDP, you handle all the formatting of messages travelling between client and server. However, if you have an existing program that talks over sockets that you want to interface to, it doesn't matter what language it's written in, as long as message formats match.

RMI - hides much of the network specific code, you don't have to worry about specific ports used (but you can if you want), RMI handles the formatting of messages between client and server. However, this option is really only for communication between Java programs. (You could interface Java RMI programs with programs written in other languages, but there are probably easier ways to go about it...)

What are JDBC and ODBC? Write a java program using JDBC to extract name of those students who live in Morang district, assuming that students table has four attributes (ID, name, district, and age).

ODBC	JDBC
ODBC Stands for Open Database Connectivity.	JDBC Stands for java database connectivity.
Introduced by Microsoft in 1992.	Introduced by SUN Micro Systems in 1997.
We can use ODBC for any language like C,C++,Java etc.	We can use JDBC only for Java languages.
We can choose ODBC only windows platform.	We can Use JDBC in any platform.
Mostly ODBC Driver developed in native languages like C,C++.	JDBC Stands for java database connectivity.
For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent.	For Java application it is highly recommended to use JDBC because there are no performance & platform dependent problem.
ODBC is procedural.	JDBC is object oriented.

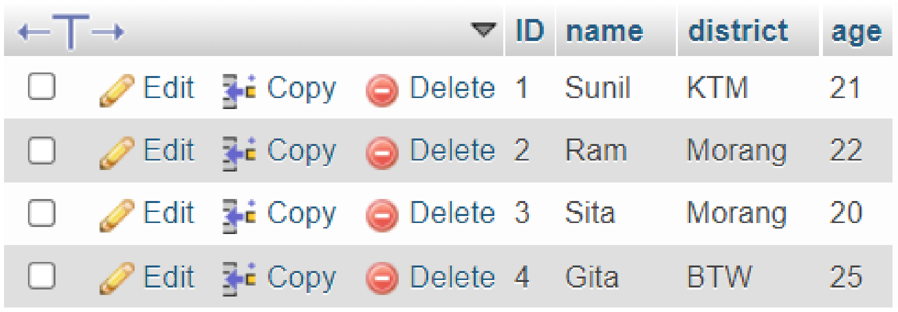
Program:

```
import java.sql.*;
public class Q_2019_2 {
  public static void main(String[] args) throws SQLException {
    String url = "jdbc:mysql://localhost:3306/java_db";

    String username="root";
    String password = "";

    Connection conn = DriverManager.getConnection(url,username,password);
    System.out.println("Successfully Connected");

    /*-----------------Create a table---------------*/
```

```
        String query = "create table students(ID int (20) PRIMARY key auto_increment, name varchar(200)
not null, district varchar(200) not null, age int (20))";
        Statement table = conn.createStatement();
        table.executeUpdate(query);
        System.out.println("Table Created...");

        /*--------------------Insert data--------------*/
        Statement stmt = conn.createStatement();
        int row1 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(1,'Sunil','KTM',21)");
        int row2 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(2,'Ram','Morang',22)");
        int row3 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(3,'Sita','Morang',20)");
        int row4 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(4,'Gita','BTW',25)");

        System.out.println("Data inserted in Table = " + row1);
        System.out.println("Data inserted in Table = " + row2);
        System.out.println("Data inserted in Table = " + row3);
        System.out.println("Data inserted in Table = " + row4);

        /*--------------------Extract name live in morang--------------*/
        String sql = "select * from students where district = 'Morang'";
        ResultSet rs = stmt.executeQuery(sql);
        while(rs.next()) {
          System.out.printf("%s \n",
                rs.getString("name")
          );
        }
    }
}
```

Output:

```
        Ram
        Sita
```



				ID	name	district	age
☐	Edit	Copy	Delete	1	Sunil	KTM	21
☐	Edit	Copy	Delete	2	Ram	Morang	22
☐	Edit	Copy	Delete	3	Sita	Morang	20
☐	Edit	Copy	Delete	4	Gita	BTW	25

What is the task of socket class? Write program using socket program to send the string “hello” to the client.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

Example:

```
// A Java program for a Client
import java.net.*;
import java.io.*;

public class Client
{
	// initialize socket and input output streams
	private Socket socket		= null;
	private DataInputStream input = null;
	private DataOutputStream out	= null;

	// constructor to put ip address and port
	public Client(String address, int port)
	{
		// establish a connection
		try
		{
			socket = new Socket(address, port);
			System.out.println("Connected");

			// takes input from terminal
			input = new DataInputStream(System.in);

			// sends output to the socket
			out = new DataOutputStream(socket.getOutputStream());
		}
		catch(UnknownHostException u)
		{
			System.out.println(u);
		}
		catch(IOException i)
		{
			System.out.println(i);
		}

		// string to read message from input
		String line = "";

		// keep reading until "Over" is input
		while (!line.equals("Over"))
		{
			try
			{
				line = input.readLine();
				out.writeUTF(line);
			}
			catch(IOException i)
			{
				System.out.println(i);
			}
		}

		// close the connection
		try
		{
			input.close();
			out.close();
			socket.close();
		}
		catch(IOException i)
		{
			System.out.println(i);
		}
	}

	public static void main(String args[])
	{
		Client client = new Client("127.0.0.1", 5000);
	}
}
```

Group “ B ”

Write a program in java that reads line of text from keyboard and write to file. Also read the content of the same file and display on monitor.

```
package com.company.BoardQuestion;
import java.io.*;
import java.io.IOException;
import java.util.Scanner;

public class Q_2019_4 {
   public static void main(String[] args) throws IOException {
      // create a new file
      File myFile = new File("file.txt");
      myFile.createNewFile();
      System.out.println("File is Created : " + myFile);

      // write text in file
      FileWriter fileWriter = new FileWriter("file.txt");
      Scanner sc = new Scanner(System.in);
      System.out.println("Write text here....");
      String input = sc.nextLine();
      fileWriter.write(input);
      fileWriter.close();

      // reading text form file
      File myFile = new File("file.txt");
      System.out.println("Reading file");
      Scanner sc = new Scanner(myFile);
      while (sc.hasNextLine()){
         String line = sc.nextLine();
         System.out.println(line);
      }
      sc.close();
   }
}
```

Write an object oriented program in java to find area of circle.

```
package com.company.BoardQuestion;
import java.util.Scanner;
class AreaOfCircle{
   int Radius;
   public int getRadius(){
       return Radius;
   }
```

```
    public  void setRadius(int Radius){
        this.Radius = Radius;
    }
    public  double area(){
        return Math.PI * Radius * Radius;
    }
}
public class Q_2019_5 {
    public static void main(String[] args) {
        AreaOfCircle r = new AreaOfCircle();
        r.setRadius(9);
        System.out.println("Area of circle is : " + r.area());
    }
}
```

What is thread? How it is created? Make a thread using runnable interface to display number from 1 to 20; each number should be display in the interval of 2 seconds.

```
class Demo implements Runnable{
    public void run(){
        System.out.println("Thread is running ...");
        for(int i = 1; i < 20; i++){
            try{
                Thread.sleep(2000);
            } catch(InterruptedException e){
                System.out.println(e);
            }
            System.out.println(i);
        }
    }

    public static void main(String args[]){
        Demo demo = new Demo();
        Thread thread = new Thread(demo);
        thread.start();
    }
}
```

What is the difference between error and an exception? How to create custom exceptions? Give an example.

Key	Error	Exception
Type	Classified as an unchecked type	Classified as checked and unchecked
Package	It belongs to java.lang.error	It belongs to java.lang.Exception

Recoverable/ Irrecoverable	It is irrecoverable	It is recoverable
	It can't be occur at compile time	It can occur at run time compile time both
Example	OutOfMemoryError ,IOError	NullPointerException , SqlException

Custom Exception

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Consider the example 1 in which InvalidAgeException class extends the Exception class.

Using the custom exception, we can have your own exception and message. Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

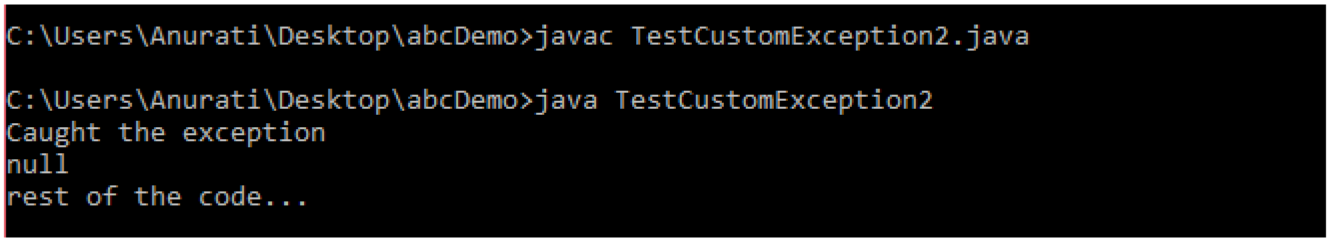
In this section, we will learn how custom exceptions are implemented and used in Java programs.

Example:

TestCustomException2.java

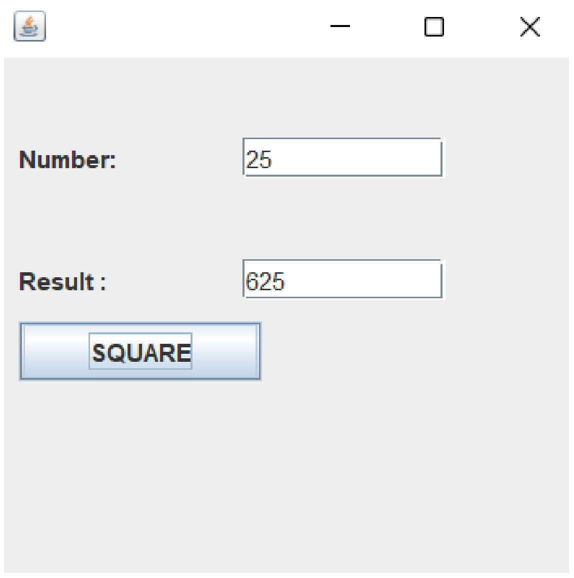
```
// class representing custom exception
class MyCustomException extends Exception
{

}

// class that uses custom exception MyCustomException
public class TestCustomException2
{
   // main method
   public static void main(String args[])
   {
      try
      {
         // throw an object of user defined exception
         throw new MyCustomException();
      }
      catch (MyCustomException ex)
      {
         System.out.println("Caught the exception");
         System.out.println(ex.getMessage());
      }

      System.out.println("rest of the code...");
   }
}
```

Output:



```
C:\Users\Anurati\Desktop\abcDemo>javac TestCustomException2.java

C:\Users\Anurati\Desktop\abcDemo>java TestCustomException2
Caught the exception
null
rest of the code...
```

What are the swing components? Created swing application that receive a number through a JTextField and display the square of number in a JTextField when the SQUARE button is pressed.



Number: 25

Result : 625

```
package com.company.BoardQuestion;
 import javax.swing.*;
 import java.awt.*;
 import java.awt.event.*;

class MyFrame2 extends JFrame implements ActionListener {
   private Container c;
   private JLabel  label1 ;
   private JTextField t1,t2;
   private JButton square;
   private JLabel result;

   MyFrame2(){
      setSize(300,300);
      c=getContentPane();
      c.setLayout(null);

      // text and input
      label1 = new JLabel("Number: ");
      label1.setBounds(10,40,100,20);
      c.add(label1);
      t1 = new JTextField();
      t1.setBounds(120,40,100,20);
      c.add(t1);

      // show result
      result = new JLabel("Result : ");
      result.setBounds(10,100,100,20);
      c.add(result);
      t2 = new JTextField();
      t2.setBounds(120,100,100,20);
      c.add(t2);

      // for SQUARE
```

```
        square = new JButton("SQUARE");
        square.setBounds(10,130,120,30);
        c.add(square);

        // for action to button
        square.addActionListener(this);

        // for show and exit
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == square){
            int a =  Integer.parseInt(t1.getText());
            int b = a * a;
            t2.setText(String.valueOf(b));
        }
    }
}
public class Q_2019_9 {
    public static void main(String[] args) {
        MyFrame2 frame = new MyFrame2();
    }
}
```

What is an applet? Explain life cycle of an applet. How an image can be loaded in applet? Give example.

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet {
    Image picture;
    public void init() {
        picture = getImage(getDocumentBase(),"sonoo.jpg");
    }
    public void paint(Graphics g) {
        g.drawImage(picture, 30,30, this);
    }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

What is interface? Develop a java code to implement the interface concept for finding the sum and average of give N number.

What is Interface in Java?

An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. A Java interface contains static constants and abstract methods. A class can implement multiple interfaces. In Java, interfaces are declared using the interface keyword. All methods in the interface are implicitly public and abstract.

Now, we will learn how to use interface in Java.

Syntax for Declaring Interface

To use an interface in your class, append the keyword "implements" after your class name followed by the interface name.

```
        interface {
                //methods
        }
```

Finding the sum and average of give N number

```
import java.util.Scanner;
interface N{
  void findSum();
  void findAverage();
}
public class Demo implements N{
  float sum = 0;
  float average = 0;
  float nums[];
  Demo(float[] nums){
    this.nums = nums;
  }
  public void findSum(){
    for(float num: nums){
      sum += num;
    }
    System.out.println("The sum of entered nums are: " + sum);
  }

  public void findAverage(){
    average = sum / nums.length;
```

```
        System.out.println("The average of entered nums are: " + average);
    }

    public static void main(String args[]){
        int userInput = 0;
        float inputs[];
        Scanner obj = new Scanner(System.in);
        System.out.println("How many numbers do you want to enter?");
        userInput = obj.nextInt();
        inputs = new float[userInput];
        System.out.printf("Enter %d numbers: ", userInput);
        for(int i = 0; i < inputs.length; i++){
            inputs[i] = obj.nextFloat();
        }

        Demo d = new Demo(inputs);
        d.findSum();
        d.findAverage();
    }
}
```

Years: 2021

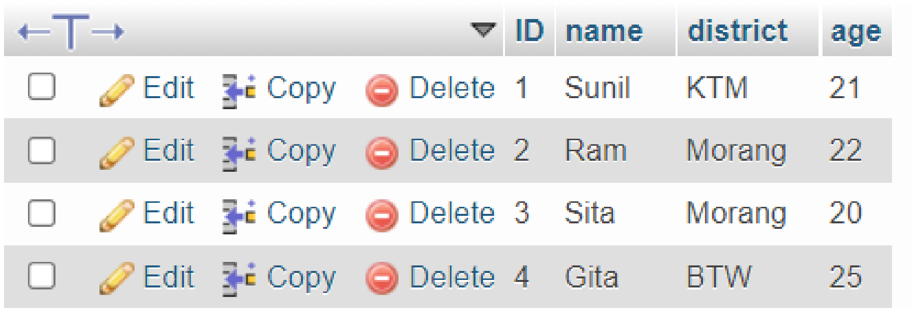
Write a JDBC program to display students details from students table.

```
import java.sql.*;
public class Q_2019_2 {
   public static void main(String[] args) throws SQLException {
      String url = "jdbc:mysql://localhost:3306/java_db";

      String username="root";
      String password = "";

      Connection conn = DriverManager.getConnection(url,username,password);
      System.out.println("Successfully Connected");

      /*-----------------Create a table---------------*/
      String query = "create table students(ID int (20) PRIMARY key auto_increment, name varchar(200)
not null, district varchar(200) not null, age int (20))";
      Statement table = conn.createStatement();
      table.executeUpdate(query);
      System.out.println("Table Created...");

      /*--------------------Insert data--------------*/
      Statement stmt = conn.createStatement();
      int row1 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(1,'Sunil','KTM',21)");
      int row2 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(2,'Ram','Morang',22)");
      int row3 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(3,'Sita','Morang',20)");
      int row4 = stmt.executeUpdate("insert into students(ID,name,district,age)
values(4,'Gita','BTW',25)");

      System.out.println("Data inserted in Table = " + row1);
      System.out.println("Data inserted in Table = " + row2);
      System.out.println("Data inserted in Table = " + row3);
      System.out.println("Data inserted in Table = " + row4);

      /*--------------------Extract name live in morang--------------*/
      String sql = "select * from students";
      ResultSet rs = stmt.executeQuery(sql);
      while(rs.next()) {
         System.out.printf("%s \n",
               rs.getString("name")
         );
```

```
        }
    }
}
```

Output :

```
            1 Sunil KTM 21
            2 Ram Morang 22
            3 Sita Morang 20
            4 Gita BTW 25
```



				ID	name	district	age
☐	Edit	Copy	Delete	1	Sunil	KTM	21
☐	Edit	Copy	Delete	2	Ram	Morang	22
☐	Edit	Copy	Delete	3	Sita	Morang	20
☐	Edit	Copy	Delete	4	Gita	BTW	25

How thread is created in java? Explain the different states of a thread. WAP which will display your name in one thread and your address in another thread in every 500 milliseconds. There should be 10000 iteration.

```
class Demo extends Thread{
    String text;
    Demo(String s){
        this.text = s;
    }

    public void run(){
        for(int i = 1; i <= 10; i++){
            try{
                Thread.sleep(1000);
            } catch(InterruptedException e){
                System.out.println(e);
            }
            System.out.println(text);
        }
    }
    public static void main(String args[]){
        Demo thread1 = new Demo("Saroj");
        Demo thread2 = new Demo("Kathmandu");
        thread1.start();
        thread2.start();
    }
}
```

Differential between overloading and overriding methods with example.

S.NO	Method Overloading	Method Overriding
1.	Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
2.	It helps to increase the readability of the program.	It is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
3.	It occurs within the class.	It is performed in two classes with inheritance relationships.
4.	Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
5.	In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
6.	In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.

Java Method Overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

What is a package? What are the benefits of using packages? Write down the steps in creating a package and using it in a java program with an example.

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Benefits

- Make easy searching or locating of classes and interfaces.
- Avoid naming conflicts. For example, there can be two classes with the name Student in two packages, university.csdept.Student and college.itdept.Student
- Implement data encapsulation (or data-hiding).

- Provide controlled access: The access specifiers protected and default have access control on package level. A member declared as protected is accessible by classes within the same package and its subclasses. A member without any access specifier that is default specifier is accessible only by classes in the same package.
- Reuse the classes contained in the packages of other programs.
- Uniquely compare the classes in other packages.

How to Create a package?

Creating a package is a simple task as follows

- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

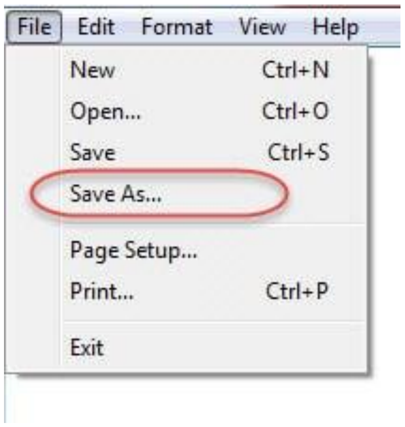
Step 1) Consider the following package program in Java:

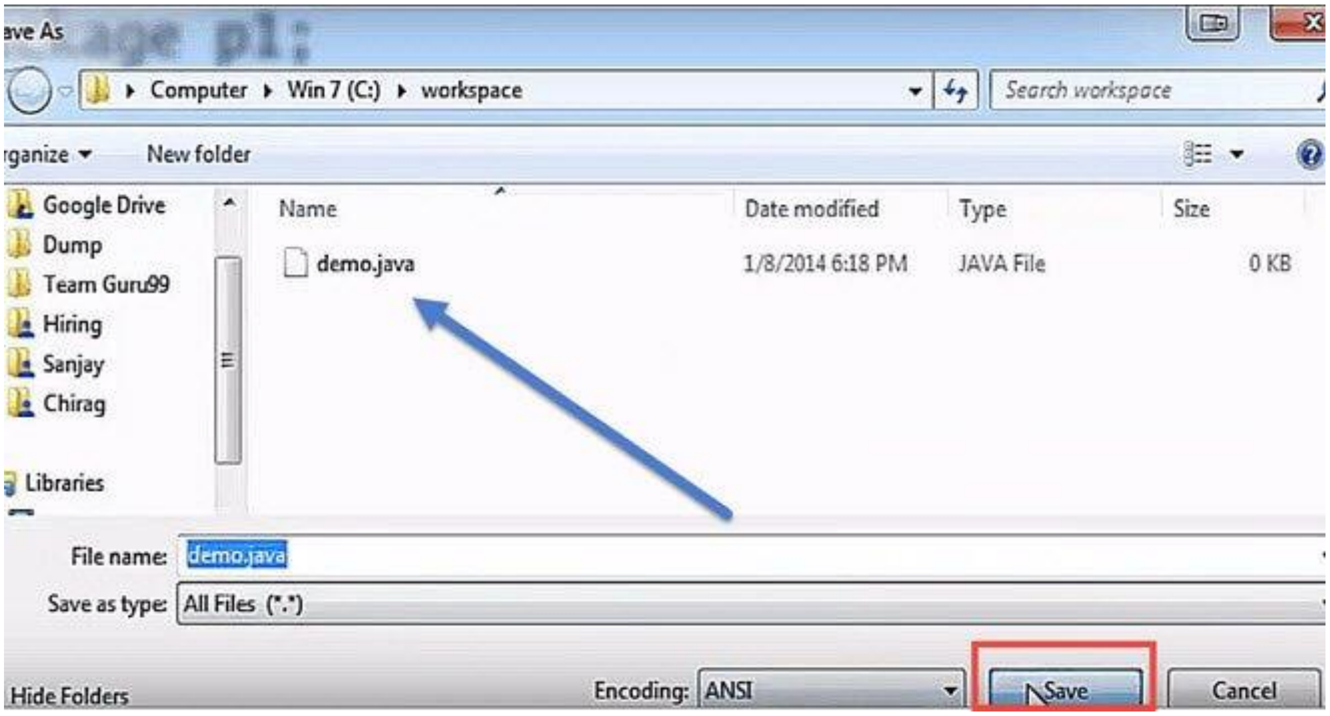
```
package p1;
class c1(){
        public void m1(){
        System.out.println("m1 of c1");
}
public static void main(string args[]){
        c1 obj = new c1();
        obj.m1();
        }
}
```

Here,

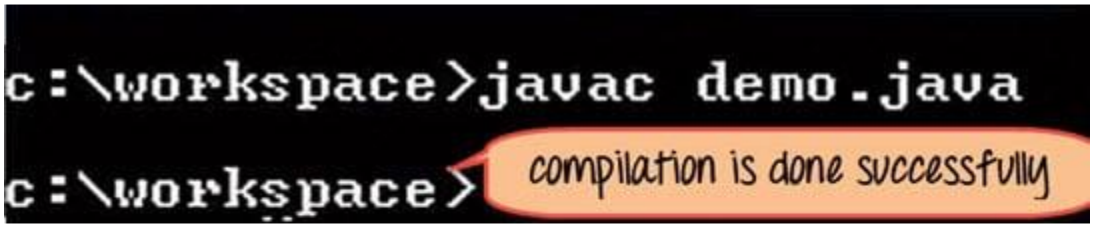
- To put a class into a package, at the first line of code define package p1
- Create a class c1
- Defining a method m1 which prints a line.
- Defining the main method
- Creating an object of class c1
- Calling method m1

Step 2) In next step, save this file as demo.java



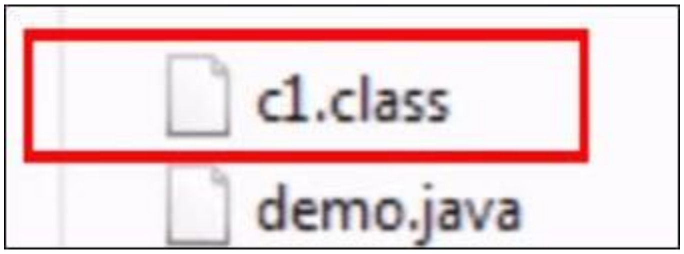


Step 3) In this step, we compile the file.

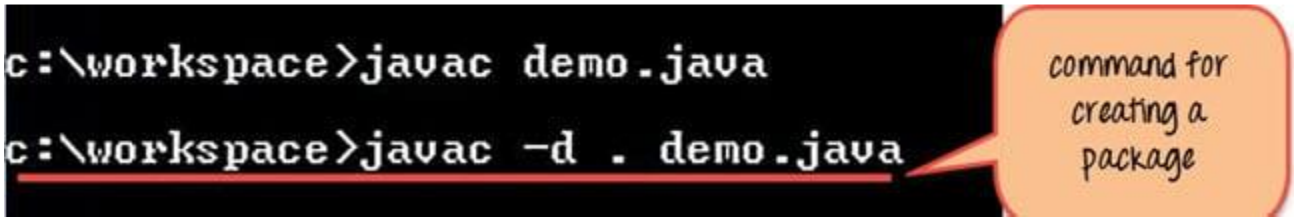


```
c:\workspace>javac demo.java

c:\workspace>
```

The compilation is completed. A class file c1 is created. However, no package is created? Next step has the solution

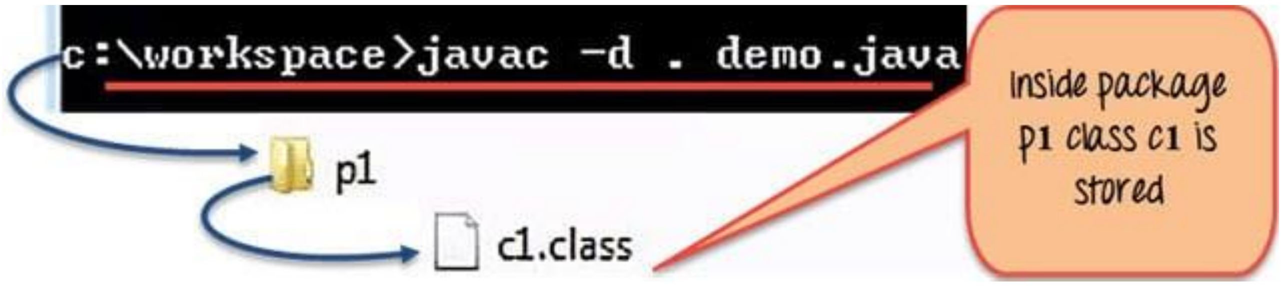


Step 4) Now we have to create a package, use the command
javac –d . demo.java
This command forces the compiler to create a package.
The **"."** operator represents the current working directory.



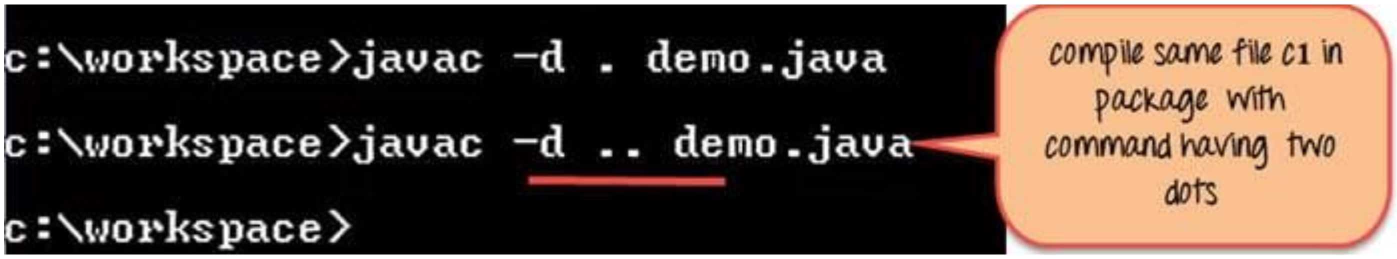
```
c:\workspace>javac demo.java

c:\workspace>javac -d . demo.java
```

Step 5) When you execute the code, it creates a package p1. When you open the java package p1 inside you will see the c1.class file.



```
c:\workspace>javac -d . demo.java
```

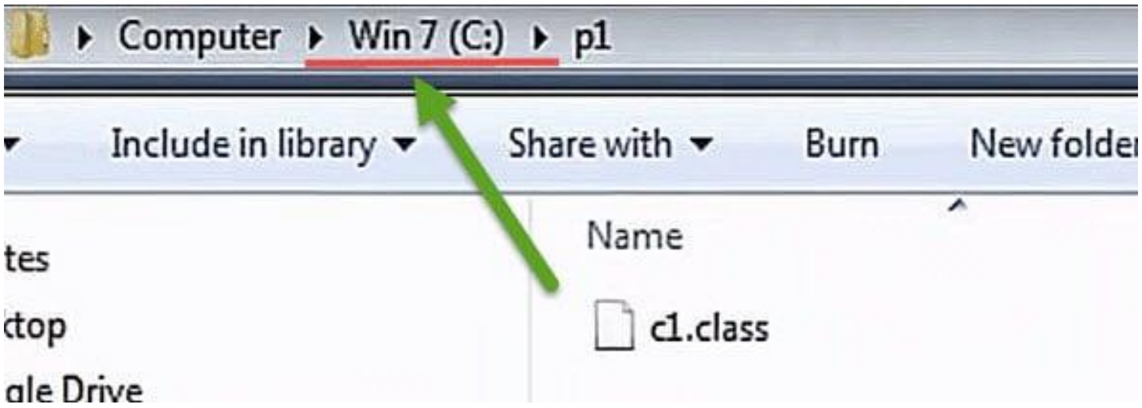
Step 6) Compile the same file using the following code
javac –d .. demo.java
Here “..” indicates the parent directory. In our case file will be saved in parent directory which is C Drive



```
c:\workspace>javac -d . demo.java

c:\workspace>javac -d .. demo.java

c:\workspace>
```

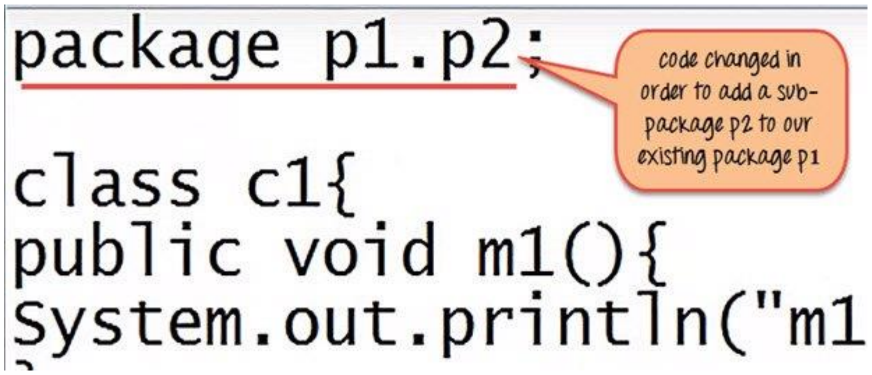
compile same file c1 in package with command having two dots

File saved in parent directory when above code is executed.



Step 7) Now let’s say you want to create a sub package p2 within our existing java package p1. Then we will modify our code as

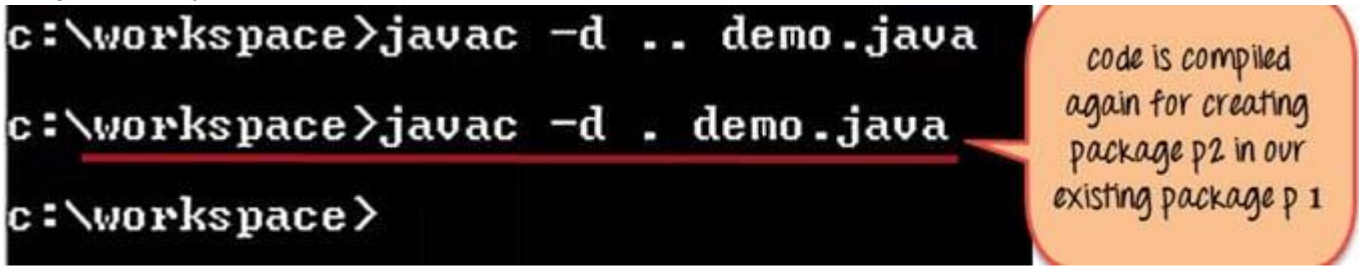
```
        package p1.p2;
        class c1{
                public void m1() {
                System.out.println("m1 of c1");
                }
        }
```



```
package p1.p2;

class c1{
public void m1(){
System.out.println("m1
```

code changed in order to add a sub-package p2 to our existing package p1

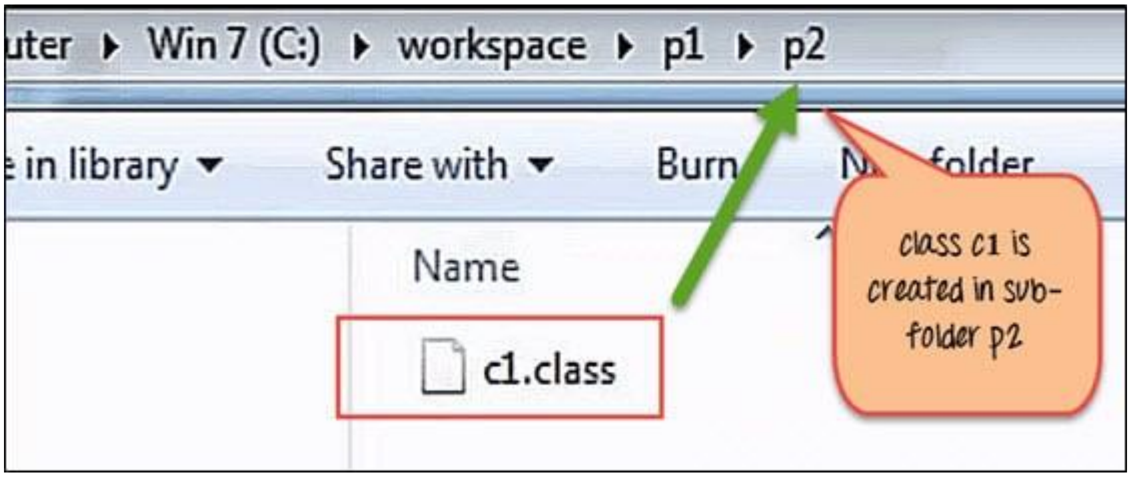
Step 8) Compile the file



```
c:\workspace>javac -d .. demo.java

c:\workspace>javac -d . demo.java

c:\workspace>
```

code is compiled again for creating package p2 in our existing package p 1

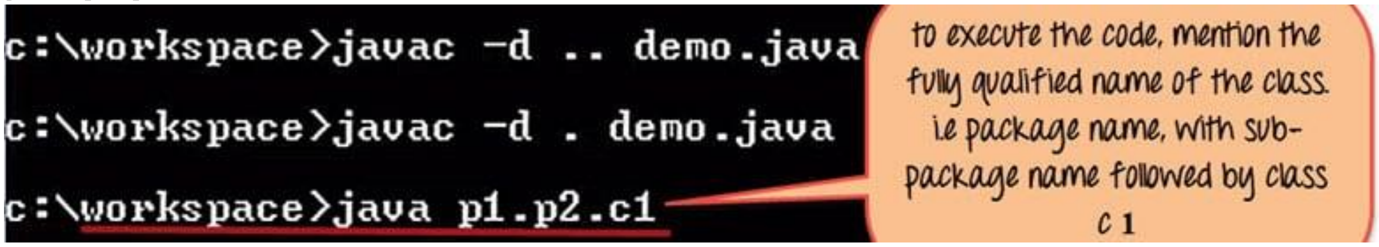
As seen in below screenshot, it creates a sub-package p2 having class c1 inside the package.



uter ▸ Win 7 (C:) ▸ workspace ▸ p1 ▸ p2
in library ▾
Share with ▾
Burn
New folder
Name
c1.class
class c1 is created in sub-folder p2

Step 9) To execute the code mention the fully qualified name of the class i.e. the package name followed by the sub-package name followed by the class name –

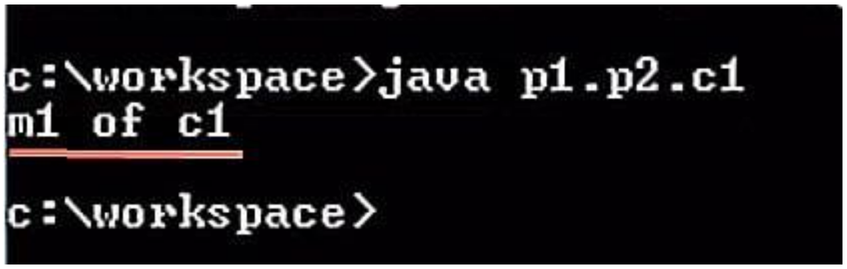
```
java p1.p2.c1
```



```
c:\workspace>javac -d .. demo.java

c:\workspace>javac -d . demo.java

c:\workspace>java p1.p2.c1
```

to execute the code, mention the fully qualified name of the class. ie package name, with sub-package name followed by class c 1

This is how the package is executed and gives the output as “m1 of c1” from the code file.



```
c:\workspace>java p1.p2.c1
m1 of c1

c:\workspace>
```

Example:

Add.java

```
package p1;
import java.util.*;
public class Add
{
        int s;
```

```
	public void sum()
	{
		System.out.print("Enter the first number: ");
		Scanner scan=new Scanner(System.in);
		int x=scan.nextInt();
		System.out.print("Enter the second number: ");
		Scanner scan1=new Scanner(System.in);
		int y=scan1.nextInt();
		s=x+y;
		System.out.println("sum="+s);
	}
}
```