Name:- Himanshu Chand / Section-A / 19183751 / Class Roll no-41

B.Tech (C.S.E) IV^th sem.

Chand

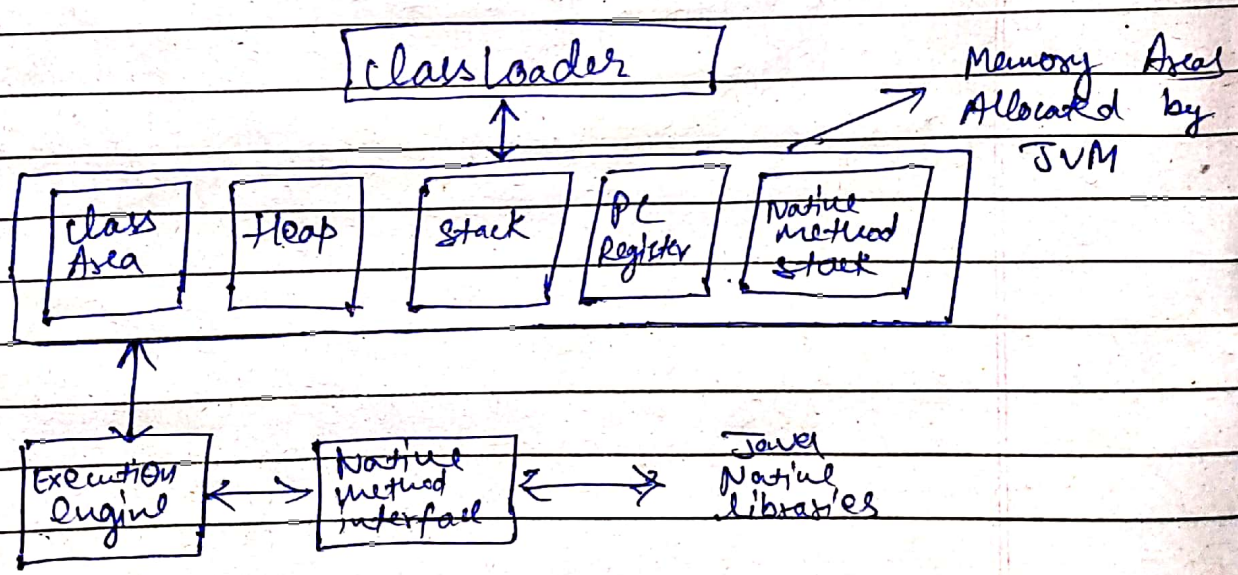
Date
Page

Answers

1) Java virtual machine (JVM) is the heart of entire Java program execution. At first .java program is converted into (.class) file consisting of byte code instructions by the java compiler at the time of compilation. Remember, this java compiler is outside the JVM. This .class file is given to the JVM. JVM mainly performs following operations.

- Allocating sufficient memory space for the class properties.
- Provides runtime environment in which java bytecode can be executed
- Converting ~~java~~ byte code instructions into machine level instruction. JVM is separately available for every operating system while installing java software so that JVM is platform dependent.

JVM architecture



i) Class loader → 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 class loader.

ii) Class Area → It stores per-class structure such as the runtime constant pool, field and method data.

iii) Heap → It is the runtime data area in which objects are allocated.

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

v) Program Counter Register → PC register contains the address of the Java virtual machine instruction currently being executed.

vi) Native method stack → It contains all the native methods used in the application.

2- i) Polymorphism in Java is concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. ~~The word~~ " There are 2 type of polymorphism in Java.

i) Compile time polymorphism
ii) Run time polymorphism

Compile time polymorphism	Run time polymorphism
i) The call is resolved by Computer	i) The call is not resolved by computer
ii) It is also known as static binding, Early binding and overloading as well.	ii) It is also known as dynamic binding, late binding and overriding as well.
iii) Method overloading is the compile-time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	iii) Method overriding is the runtime polymorphism having same method with same parameters or signature but associated in different classes

iii) Java final keyword :-

The java final keyword is used to denote constants. It can be used with variables, methods and classes.

- The final variable cannot be reinitialized with another value
- The final method cannot be overridden
- The final class cannot be extended

Ex:-
```
class Main {
    public static void main (String [] Args) {
        final int AGE = 32;
        AGE = 45;
        System.out.println ("Age: " + AGE);
    }
}
```

Output of the above program will be a compilation error. As we are trying to change value of the final variable.

Java this keyword

In Java, this keyword is used to refer to the current object inside a method or a constructor.

for example :-

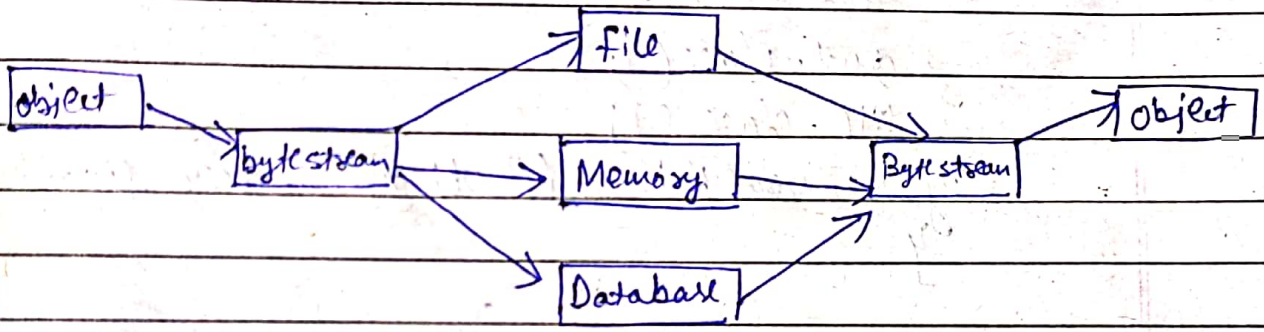
```
class Main {
        int instVar;
    Main ( int instVar){
        this. instVar = instVar;
    System.out.println ("This reference =" + this);
    }

    public static void main( String [] args) {
        Main obj = new Main(8);
    System.out.println (" object reference = "+ obj);
    }
}
```

output :- This reference = Main @23fc 625e
object reference = Main @ 23fc 625e

Here we created an object of the class Main. Then we point the reference to the object obj and this keyword of the class.

3- Serialization and Desorialization in Java

Serialization ~~and Deserialization~~ is a mechanism of converting the state of an object into byte stream. Desorialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object



Example of Serialization & Desorialization

```
import java.io.*;

class Demo implements java.io.serializable
{
    public int a;
    public string b;

    public Demo(int a, string b)
    {
        this.a = a;
        this.b = b;
    }
}
```

Date
Page

```
class Test
{   public static void main (String[] args) {
    Demo object = new Demo ( 1, "Himanshu");
    String filename = "file.ser";
```

Serialization

```
    try
    {
    FileOutputStream file = new FileOutputStream(filename);
    ObjectOutputStream out = new ObjectOutputStream(file);

    out.writeObject (object);
    out.close();
    file.close ();
  System.out.println (" object has been serialized");
    }

Catch (IOException ex)
    {
        System.out.println ("IOException is caught");
    }
Demo object1 = null;
```

Deserialization

```
try {  FileInputStream file= new FileInputStream (filename);
       ObjectInputStream in= new ObjectInputStream (file);

  Object1 = (Demo)in.readObject ();
    in close ();
  file.close ();
```

```
System.out.println("object has been deserialized");
System.out.println("a = " + object.a);
System.out.println("b = " + object.b);
}

catch (IOException ex)
{ system.out.println("IOException is caught");
}

catch (classNotFoundException ex)
{ system.out.println("Class not found is caught");
}
}
}
```

Output :-

object has been serialized
object has been deserialized
a = 7
b = Himanshu

Date
Page

4 Java Layout Managers :-

The Layout managers are used to arrange components in a particular manner. Layout manager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

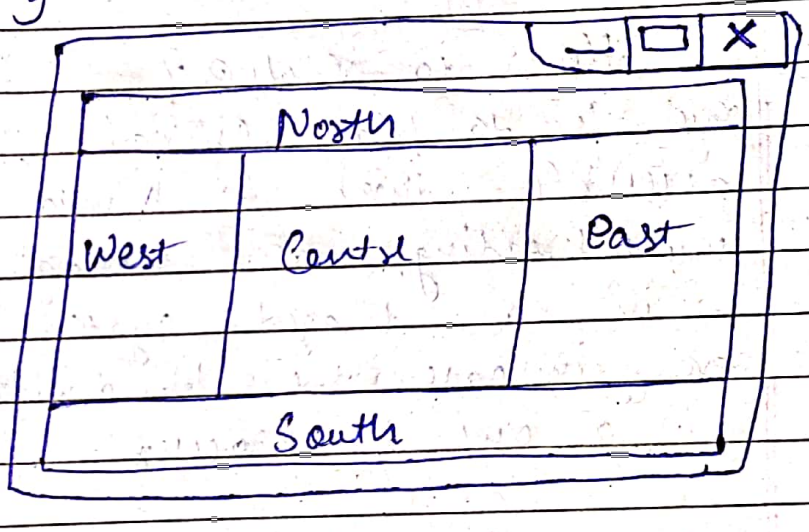
i) java.awt.BorderLayout
ii) java.awt.FlowLayout
iii) java.awt.GridLayout
iv) java.awt.CardLayout
v) java.swing.BoxLayout etc.

Example of Border Layout Class :-

```
import java.awt.*;
import javax.swing.*;

public class Border {
    JFrame f;
    Border() { f = new JFrame();

        JButton b1 = new JButton("North");
        JButton b2 = new JButton("South");
        JButton b3 = new JButton("East");
        JButton b4 = new JButton("West");
        JButton b5 = new JButton("Centre");
```

```
f.add(b1, BorderLayout.North);
f.add(b2, BorderLayout.South);
f.add(b3, BorderLayout.East);
f.add(b4, BorderLayout.West);
f.add(b5, BorderLayout.Centre);

    f.setSize(300,300);
    f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

Output :-



5

JDBC Driver

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

i) JDBC-ODBC bridge driver

This driver uses ODBC driver to connect to the databases. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC functions calls. This is now discouraged because of this driver.

Pros:—
- Easy to use
- Can be easily connected to any database.

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

ii) 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.

Pros :- • Performance upgraded than JDBC-ODBC.

Cons :- • The native driver needs to be installed on the each client machine.
• The vendor client library needs to be installed on client machine.

iii) 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.

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

Cons :- • Network support is required on client machine.
• Requires database-specific coding to be done in the middle tier.

iv) 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.

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

Cons :- • Driver depend on the database.