Rohan Nyati 500075940 R177219148
Batch - 5 (AI & ML)

Date _____ Page ___

Q.1] JVM is mainly responsible for 3 major activities

1) Loading → The class loader reads the ".class" file, generates the corresponding binary data & save it in the method area.

2) Linking → Performs verification, preparation and resolution

3) Initialization → In this all static variables are assigned with their values defined in the code and static block (if any). This is executed from top to bottom in class and from parent to child in the class hierarchy.

Q.1] Java is based on the concept of Object-oriented programming. As the name suggests, at the centre of it all is an object. Objects contain both data and functionality that operates on that data. This is controlled by following four paradigms.

i) Encapsulation → Encapsulation in JAVA is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

ii) Inheritance → in JAVA is a mechanism in which one object acquires all the properties & behaviours of a parent object. It is important part of OOP's

iii) Information hiding

iv) Polymorphism → refers to a programming language's ability to process objects depending on their class.

It is divided into two parts:-

(i) Compile time Polymorphism
(ii) Runtime Polymorphism

Q.2] Dynamic Method Dispatch (Eg.) Code

```
class A {
    void m1()
    { System.out.println("Inside A's m1 method");
    }
}
class B extends A{
    void m1() {                    //overriding m1()
        S.O.P ("Inside B's m1 method"); }
    }
class C extends A {
        void m1(){                 //overriding m1()
            S.O.P ("Inside C's m1 method"); }
        }
class Dispatch{                    //Driver class
    public static void main(String args[]){
        A a= new A();
        B b= new B();              //Object of type A,B,C
        C c = new C();
        A ref;                     //obtain a ref. of type A
        ref = a;                   // ref refers to an A object
        ref.m1();                  // Calling A's version of m1()
        ref = b;                   // now ref refers to a B object
        ref.m1();                  // Calling B's version of m1()
        ref = c;                   // now ref refers to a C object
        ref.m1();                  // Calling C's version of m1()
    }
}
```

⇒ The above program creates one Superclass (A) & its two subclasses (B & C). These subclasses override m1() method.

Q.3 Polymorphism → refers to a programming language's ability to process objects depending on their class.

It is divided into two parts:-

(i) Compile time Polymorphism
(ii) Runtime Polymorphism

Q.4) Access Modifiers in Java

	Default	Private	Protected	Public
Same Class	✓	✓	✓	✓
Same Package Sub-class	✓	✗	✓	✓
Same Package non sub-class	✗	✗	✓	✓
Diff. Package Subclass	✗	✗	✓	✓
Diff. Package non-sub class	✗	✗	✗	✓

Rohan Nupati 500075940

Date _____ Page _____

Q.5)

Abstract Class	Interface
(1) It can have abstract & non-abstract method	It can only have abstract methods. It can have default & static methods also
(2) Doesn't support multiple inheritance	Supports multiple inheritance
(3) Can have final, non-final, static & non-static variables	Has only static & final variables
(4) Abstract keyword is used to declare abstract class.	Interface keyword is used to declare interface.
(5) Can have class members like private, protected, etc	Members of Java interface are public by default.

(6) Eg.
public abstract class Shape{
public abstract void draw();
}
	Eg.
	public interface Drawable
	void draw();
	}

Q.6] An interface that does not contain methods, fields, constants is known as Marker Interface. In other words, an empty interface is known as marker interface or tag interface. It delivers the run-time type information about an object. It is the reason that the JVM and compiler have additional information about an object.

Various Marker Interfaces are:-

1) Cloneable Interface
2) Serializable Interface
3) Remote Interface

We can also have Custom Marker Interfaces.

Q.9] StringBuilder and StringBuffer classes provide imp. methods that String class does not provide, such as insert(), delete(), and reverse() methods. If we use numerous string manupulations in our code, we should use StringBuilder or StringBuffer because it is much faster and consumes less memory than String.
Eg. If we use String concatenation in loop, its better to use StringBuilder.

Rohan Nyati 500075940

Date Page

Q.7] Methods of Object Class :-

1) public final Class getClass()
2) public int hashCode()
3) public boolean equals(Object obj)
4) public String toString()
5) public final void notify()
6) public final void notifyAll()
7) protected void finalize() throws Throwable

etc.

Q.8] Exception Handling → It is a mechanism to handle runtime error such as ClassNotFoundException, IOException etc. The core advantage of it is to maintain the normal flow of the application.

Java Custom Exception → If we are creating our own exception that is known as custom exception. These are used to customise the exception according to user needs.

Eg.
```
Class InvalidAgeException extends Exception {
    InvalidAgeException(String s) {
        super(s);
    }
}
```

Rohan Nyati 500075940

Q10) Use of Wrapper Classes in Java

1) Change the value in Method

2) Serialization → We need to convert objects into streams to perform serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.

3) Synchronization → It works with objects in Multithreading.

4) java.util package → It provides the utility classes to deal with objects.

5) Collection Framework → It works with objects only.

Primitive Type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double