# Assignment no. 1

M T W T F S S
Page No.:
Date: 29/9/22
YOUVA

1) Explain concept Java Polymorphism. Write a simple Java program that implements polymorphism.

→ • Polymorphism means 'many forms', and it occurs when we have many classes that are related to each other by inheritance.
• Polymorphism means the ability to take many forms by which we can perform a single task in different ways.
• It is the capability of a method to do different things based on the object that it is acting upon.
• In other words, polymorphism allows you define one interface and have multiple implementations.
• There are two types of polymorphism in Java:
1. Compile time polymorphism.
2. Run time polymorphism.
• Ex.

```
class Bird {
public void ~~anim act bird~~sound(){
System.out.println("Bird makes sound");
}
}
Class sparrow extends Bird {
public void sound(){
System.out.println("sparrow: chew chew");
}
}
```

```
class Crow extends Bird {
 public void sound (){
System.out.println("Crow: kav - kav");
  }
}
 class main {
 public static void main (String args[]){
 Bird obj = new sparrow Bird ();
  Bird obj1 = new sparrow ();
  Bird obj2 = new Crow ();
  obj.sound ();
    obj1.sound ();
     obj2.sound ();
   }
  }
```

Output :- Bird makes sound
Sparrow: Chev Chev
Crow: Kav Kav

Q.2) Define Compile Time Polymorphism with suitable example.

→ • method overloading is related to compile time polymorphism.
• method overloading in Java is feature which makes it possible to use the same method name to perform different tasks.
• overloading allows different methods to have same name, but different signatures where signature differ by number of input parameters, type of input parameters, or order of input parameters.

Ex. public class overload {
void demo (int a)
{
System.out.println ("a: " + a);
}
void demo (int a, int b);
{
System.out.println ("b");
System.out.println ("a & b: " + a + b);
}
}
class add {
public static void main (String args[]){
overload a1 = new overload ();
a1.demo (28);
a1.demo (28+24);
}
}

output:- a: 28
b: 24
a+b: 52

Q.3) Define Run time polymorphism with example.

→ • Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compiletime.
• An overridden method is called through reference variable of superclass.
• A superclass reference variable can refer to subclass object. this is also known as upcasting.

```
Ex. class bike {
 void show (){
 System.out.println ("I am Bike");
   }
 }
 Class Pulsar extends bike{
 void show (){
System.out.println ("Pulsar is bike of Bajaj");
 }
}
 class main{
public static void main (String args[]){
 bike b1 = new bike();
  b1.show();
 bike b2 = new Pulsar();
  b2.show();
 }
}
```

Output :- I am Bike.
Pulsar is bike of Bajaj.

Q.4) Distinguish between Abstract class and Concrete class.

→

| Abstract class | Concrete class |
|---|---|
| 1. An abstract class is declared ~~by~~ using abstract modifier. | 1. A concrete class is not declared using abstract modifier. |
| 2. Abstract class cannot be instantiated using new keyword. | 2. Concrete class can be directly instantiated using new keyword. |

| | |
|---|---|
| 3. Abstract class may or may not contain abstract methods. | 3. Concrete class cannot contain abstract method |
| 4. Abstract class cannot be declared as final. | 4. Concrete class can be declared as final. |
| 5. Interface implementation is not possible. | 5. Interface implementation is possible. |

Q.5) Why we require abstract class in Java. Explain it with simple Java program.

→ • We have a class 'Animal' that has method 'sound()' and subclasses of it like Dog, Lion, etc.

• The animal sound differs from one animal to another. There is no point to implement this method in parent class. This is because every child class must override this method to give its own implementation details like Lion roars.

~~• When we know that all animal child classes will and should override this method,~~

• Thus, making this method abstract would be the good choice as by making this method abstract we force all subclasses to implemen this method.

• Since the 'Animal' class has an abstract method, you must need to declare this class abstract.

Ex. abstract class Animal{
    public abstract void sound();
}

public Dog extends Animal{

```
public void sound(){
System.out.println("Woof");
}
public static void main(String args[]){
  Animal obj = new Dog();
   obj.sound();
  }
}
```

Output :- Woof.

Q.6) Explain final keyword in Java.

→ · Final keyword is used in different contexts.
· Final is non-access modifier applicable only to variable, a method or a class.
· Final variable → To create constant variables.

· Final methods → Prevent method overriding

· Final classes → Prevent inheritance

· When a variable is declared with final keyword, it's value can't be modified, essentially, a constant.
~~· When a class~~ Ex.

```
final int double PI = 3.141592...
```

· When a class is declared with final keyword, it is called a final class. A final class cannot be extended.

```
final class A{
Class B extends A
{
}
```

output: Compile time error.

- When a method is declared with final keyword, it is called a final method.

```
class A {
    final void m1()
    {
System.out.println("This is final method");
    }
}
class B extends A{
    void m1()
    {
System.out.println("Illegal");
    }
}
```

Output:- Error.

Q-7) Explain final classes and final ~~variable~~ methods with a Java program.

→ *Final Classes:-

- When a class is declared with final keyword, it is called a final class.
- A final class cannot be extended.
- There are two uses of final class:-

a) To prevent inheritance, as it cannot be extended.

b) To create an immutable class like predefined String class. You can't make a class immutable without making it final.

- Ex. final class A

```
{
  //methods and feilds
}
```

```
class B extends A
{
//Compile-error! Cannot subclass A
}
```

* Final methods:-

- When a method is declared with final keyword, it is called final method.
- Final method cannot overriden.
- We must declare methods with final keyword for which we required to follow same implementation throughout all derieved classes.
- Ex.

```
class A {
    final void m1()
    {
    System.out.println("this is final method");
    }
    }
class B extends A {
    void m1(){
//Compile error! Can't override
System.out.println("Illegal");
}
}
```

Q.8) Define interface in Java. What is the use of interface in Java?

→ - Interfaces looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are

by default abstract.
- Also, the variables declared in an interface are public, static & final by default.
- Interface are used for full abstraction.
- Since methods in interface do not have body, they have to be implemented by the class before you can access them.
- The class that implements interface must implement all the methods of that interface.
- However, you can implement more than one interface in your class.
- Syntax:-

```
interface myinterface
{
 public void method1();
 public void method2();
}
```

Q.9) Write down a simple Java program defining interface in Java which extends another interface.

→

```
public interface inter1{
public void method1();
public void method2();
}
interface inter2 extends inter1{
 public void method2();
 }
class demo2 implements inter2{
public void method1(){
System.out.println("manisb");
}
```

```
public void method2(){
System.out.println("Jadhav");
  }
 }
public static void main (String args[]){
 inter1 obj1 = new demo2();
   obj.method2();
  }
}
```

Output :- Jadhav.

Q.10) Write down a simple Java program to implement more than one interface.

→

```
interface Printable{
 void print();
  }
 interface Showable{
  void show();
 }
class int1 implements Printable, Showable{
public void print(){
System.out.println("I am Print");
  }
 public void show(){
system. out.println("I am Show");
  }
  public static void main(String args[]){
  int1 obj = new int1();
    obj.print();
```

```
        obj.show();
      }
    }
```

Output:- I am Print.
I am Show.

Q11) Define; why we cannot create the object of abstract class.

→ • Because abstract classes are incomplete, they have abstract methods that have no body.
• There would be no actual implementation of the method to invoke.
• Also because an object is concrete. An abstract class is like a template, so you have to extend it and build on it before you can use it.
• Ex.

Q12)

```
abstract class AbstractDemo{
  public void myMethod(){
  System.out.println("manish");
  }
  abstract public void anotherMethod();
  }
  public class Demo extends AbstractDemo{
   public void anotherMethod(){
   System.out.println("Jadhav");
  }
  public static void main(String args[]){
  AbstractDemo obj=new AbstractDemo();
    obj.anotherMethod();
  }
}
```

Output :- Unresolved compilation problem:
Cannot instantiate the type AbstractDemo.

Q12> What are the rules required to follow in method overriding?

-> Some Rules to follow in method overriding:

- Overriding and Access-Modifiers :- The access modifier for an overriding method can allow more, but not less, access than overridden method.
- Final methods cannot be overridden.
- Static methods cannot be overridden.
- Private methods cannot be overridden.
- The overriding method must have same return type.
- Invoking overridden methods from sub class :- we can call parent class method in overriding method using super keyword.
- We cannot override constructor as parent and child class can never have constructor with same name.
- Abstract methods in an interface or abstract class are meant to be overridden in derieved concrete classes otherwise compile-time error will be thrown.