Day 21 PW Skills Assignment Solutions OOPS

1. What is inheritance in Java?

Ans. Inheritance is a fundamental concept in object-oriented programming (OOP) that allows classes to inherit properties and behaviours from other classes. In Java, inheritance enables the creation of new classes (derived classes or subclasses) based on existing classes (base classes or super classes). The derived classes inherit the attributes and methods of the base class, allowing for code reuse and the creation of class hierarchies.

2. What is superclass and subclasses?

Ans.

1. Superclass: In object-oriented programming, a superclass, also known as a base class or parent class, is the existing class from which other classes are derived. It is a more general class that provides common attributes and behaviors to its subclasses. Superclasses serve as templates or blueprints for creating more specialized classes. Subclasses inherit the properties and methods of the superclass and can extend or override them as needed.
2. Subclass: A subclass, also referred to as a derived class or child class, is a new class that is created by extending an existing superclass. It inherits the attributes and behaviors of the superclass and can add its own unique attributes, methods, or behaviors. Subclasses specialize or refine the functionality of the superclass by providing more specific implementation details. They establish an "is-a" relationship with the superclass, indicating that the subclass is a specialized type of the superclass.

3. How is inheritance implemented/ achieved in Java?

Ans. The inheritance is implemented in classes and the interfaces.
In classes we use the keyword called **'extends'.**

```
class Superclass {
   // superclass fields and methods
}

class Subclass extends Superclass {
   // subclass fields and methods
}
```

In interfaces we use the keyword called **'implements'.**

```
class Animal {
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    public void sound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.sound(); // Output: Animal makes a sound

        Dog dog = new Dog();
        dog.sound(); // Output: Dog barks
    }
}
```

4. **What is polymorphism?**

Ans. Polymorphism is a fundamental concept in object-oriented programming, including Java. It refers to the ability of an object to take on many forms or have multiple behaviors.
In Java, polymorphism is typically achieved through method overriding and method overloading.

5. **Differentiate between method overloading and overriding.**

Ans.

Method Overloading	**Method Overriding**
Occurs within the same class or different classes in the same inheritance hierarchy.	Occurs between a superclass and its subclass.
Methods have the same name but different parameters (number, type, or order).	Methods have the same name, return type, and parameters.

The return type may or may not be the same.	The return type must be the same or a subtype of the return type in the superclass.
Determined at compile time based on the method signature (number, type, and order of parameters).	Determined at runtime based on the actual object type.
Overloaded methods may or may not have a relationship between each other.	Overridden methods have an inheritance relationship (superclass and subclass).
The intention is to provide different methods with similar functionality but different input parameters.	The intention is to provide a specific implementation of a method in the subclass, modifying or extending the behavior inherited from the superclass.

6. **What is an abstraction explain with an example?**

Ans. Abstraction is a fundamental concept in object-oriented programming that allows you to represent complex real-world entities as simplified models in code. It involves focusing on essential attributes and behaviors while hiding unnecessary details.
In Java, abstraction is achieved through abstract classes and interfaces. An abstract class is a class that cannot be instantiated but can be subclassed, while an interface is a collection of abstract methods. Both abstract classes and interfaces provide a way to define abstract methods that must be implemented by their concrete subclasses.

```
abstract class Shape {
    public abstract void draw();
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a rectangle");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle();
        circle.draw(); // Output: Drawing a circle

        Shape rectangle = new Rectangle();
        rectangle.draw(); // Output: Drawing a rectangle
    }
}
```

7. **What is the difference between an abstract method and final method in Java? Explain with an example**

Ans.

Abstract Method	**Final Method**
Declared within an abstract class or interface.	Declared within a class.
Does not have an implementation in the abstract class or interface.	Has a specific implementation in the class and cannot be overridden.
Serves as a placeholder that must be overridden by concrete subclasses.	Provides a specific implementation that cannot be modified or overridden.
Must be overridden by subclasses to provide their specific implementation.	Cannot be overridden by any subclass.
Declared using the **abstract** keyword.	Declared using the **final** keyword.
Does not contain a method body; ends with a semicolon.	Contains a method body with the implementation details.
Abstract classes containing abstract methods must be marked as abstract.	No special keyword is needed for the class containing a final method.

```
abstract class Shape {
    public abstract void draw(); // Abstract method

    public final void displayArea() { // Final method
        System.out.println("Calculating and displaying area...");
    }
}
```

```
class Circle extends Shape {
   @Override
   public void draw() {
      System.out.println("Drawing a circle...");
   }
}

class Square extends Shape {
   @Override
   public void draw() {
      System.out.println("Drawing a square...");
   }
}

public class Main {
   public static void main(String[] args) {
      Circle circle = new Circle();
      circle.draw(); // Output: Drawing a circle
      circle.displayArea(); // Output: Calculating and displaying area...

      Square square = new Square();
      square.draw(); // Output: Drawing a square
      square.displayArea(); // Output: Calculating and displaying area...
   }
}
```

8. **What is the final class in Java?**

Ans. The **final** keyword can be used to modify classes, methods, and variables. When applied to a class, it indicates that the class cannot be subclassed, meaning that it cannot have any subclasses.

A final class in Java has the following characteristics:

1. Inheritance Restriction: A final class cannot be extended by any other class. It is considered the terminal point in the inheritance hierarchy.
2. Method Overriding Prevention: Methods declared as final within a final class cannot be overridden by subclasses. This ensures that the behaviour of these methods remains consistent and cannot be changed.
3. Constant Behaviour: A final class often contains final fields (constants) whose values cannot be modified after initialization. These fields are typically declared as **static final** and represent constants that remain unchanged throughout the execution of the program.

The use of final classes can be beneficial in certain scenarios, such as when you want to prevent any further modification or extension of a class to ensure its integrity or when you want to establish a fixed implementation for a specific concept or utility class.

9. Differentiate between abstraction and encapsulation.

Ans.

Abstraction	Encapsulation
Focuses on representing complex real-world entities as simplified models.	Focuses on bundling data and methods together and hiding the internal details.
Hides unnecessary details and emphasizes essential attributes and behaviors.	Hides the internal state and implementation details of an object.
Achieved through abstract classes and interfaces.	Achieved by combining data and methods within a class.
Provides a way to define abstract methods that must be implemented by concrete subclasses.	Provides data security and controlled access to class members through access modifiers.
Enables working with objects at a higher level of understanding, without worrying about specific implementation details.	Enforces information hiding and allows for better code organization.
Deals with the external view and behavior of objects.	Deals with the internal organization and state of objects.

10. Difference between Runtime and compile time polymorphism explain with an example.

Ans

Runtime Polymorphism	**Compile-Time Polymorphism**
Occurs at runtime, where the appropriate method implementation is determined based on the actual object type.	Occurs at compile-time, where the appropriate method implementation is determined based on the method signature (number, type, and order of parameters).
Achieved through method overriding.	Achieved through method overloading.
Also known as dynamic polymorphism.	Also known as static polymorphism.

The method implementation is resolved during runtime based on the actual object type.	The method implementation is resolved during compilation based on the method signature.
Requires an inheritance relationship between classes (superclass and subclass).	Does not require an inheritance relationship; methods can be defined within the same class or in different classes.