Assignment No:-25

Name:-Suryawanshi Sangramsingh Sambhaji

Batch: - Delta - DCA (Java) 2024 Date:-11/6/2024

1. Define Polymorphism and Explain Its Types in Java

Ans: Polymorphism in Java refers to the ability of a single interface to represent different underlying forms (data types). There are two main types of polymorphism in Java:

Compile-time Polymorphism (Static Binding): This type of polymorphism is resolved during compile time. Method overloading is an example of compile-time polymorphism.

Runtime Polymorphism (Dynamic Binding): This type of polymorphism is resolved during runtime. Method overriding is an example of runtime polymorphism.

2. How Does Polymorphism Enhance Flexibility and Extensibility in Code?

Ans: Polymorphism allows one interface to be used for a general class of actions. It lets code be more flexible and extensible because:

You can write code that works on the superclass level but will automatically work with any subclass.

New classes can be added with little or no modification to existing code.

It promotes code reuse and can significantly reduce the complexity of code.

3. Explain the Difference Between Compile-time and Runtime Polymorphism

Ans: Compile-time Polymorphism: This is achieved by method overloading. The method to be called is determined at compile time.

Runtime Polymorphism: This is achieved by method overriding. The method to be called is determined at runtime based on the object being referred to by the reference variable.

4. Discuss the Concept of Method Overloading with Examples

Ans: Method overloading allows a class to have more than one method with the same name, but with different parameters.

```
public class MethodOver
{
   public int add(int a, int b)
   {
      return a + b;
   }
   public int add(int a, int b, int c)
   {
      return a + b + c;
   }
}
```

5. How is Method Overriding Different from Method Overloading?

Ans: Method Overloading: Multiple methods with the same name but different parameters within the same class.

Method Overriding: A subclass provides a specific implementation of a method that is already defined in its superclass.

6. Can a Subclass Overload a Method Inherited from Its Superclass?

Ans: Yes, a subclass can overload a method inherited from its superclass. The overloaded methods are treated as independent methods in the subclass.

7. Explain the Concept of Virtual Methods in Java

8. Discuss the Use of the super Keyword in Achieving Polymorphism

The super keyword is used to refer to the immediate parent class object. It is often used to invoke the parent class's method that is overridden in the subclass.

```
class Parent
{
   void display()
   {
      System.out.println("Parent display");
   }
}

class Child extends Parent
{
   void display()
   {
      super.display();
```

```
System.out.println("Child display");
    }
}
```

9. How Does Java Support Polymorphism Through Interfaces?

Java interfaces provide a way to achieve polymorphism. A class can implement multiple interfaces, and an interface reference can point to any object of the implementing class.

```
interface Animal
{
   void sound();
}

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

class Cat implements Animal
{
   public void sound()
   {
      System.out.println("Meow");
   }
}
```

10. What Is the Role of the instanceof Operator in Polymorphism?

11. Provide an Example of Polymorphism Using the toString Method

Polymorphism can be illustrated by overriding the toString method in different classes.

```
class Animal
{
   public String toString()
   {
      return "This is an animal";
   }
}

class Dog extends Animal
{
   public String toString()
   {
      return "This is a dog";
   }
}

class Cat extends Animal
```

```
{
   public String toString()
   {
      return "This is a cat";
   }
}
```

12. Explain How Polymorphism Is Related to Dynamic Method Dispatch

Ans: Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime rather than at compile time. Polymorphism is achieved through this mechanism, where the method that is called is determined by the runtime type of the object.

13. Discuss the Advantages of Using Polymorphism in Software Design

Ans: Code Reusability: Polymorphism promotes reusability and minimizes code duplication.

Maintainability: Code becomes easier to maintain and modify.

Extensibility: New functionalities can be added with minimal changes.

Flexibility: Code can work with objects of different classes that share the same interface.

14. Can a Class Be Both Abstract and Polymorphic in Java? Justify

Ans: Yes, a class can be both abstract and polymorphic. An abstract class can have abstract methods that must be implemented by subclasses. Subclasses can exhibit polymorphic behavior through method overriding.

15. How Does Polymorphism Contribute to Code Readability?

Ans: Polymorphism contributes to code readability by allowing methods to be called on objects without knowing their specific types. This leads to cleaner and more understandable code.

16. Explain the Concept of Covariant Return Types in Polymorphism

17. Discuss the Challenges of Implementing Polymorphism in Java

Ans: Performance Overhead: Dynamic method dispatch can introduce performance overhead.

Complexity: Understanding and debugging polymorphic code can be more complex.

Type Safety: Incorrect type casting can lead to ClassCastException.

18. How Can Polymorphism Be Achieved Using Abstract Classes?

Ans: Abstract classes can define abstract methods that must be implemented by subclasses. Subclasses provide specific implementations, enabling polymorphism.

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

class Circle extends Shape
{
   void draw()
   {
      System.out.println("Drawing Circle");
   }
}

class Rectangle extends Shape
{
   void draw()
   {
      System.out.println("Drawing Rectangle");
   }
}
```

19. What Is the Significance of the final Keyword in Polymorphism?

20. Explain the Concept of Function Overloading in Polymorphism

Function overloading is a form of compile-time polymorphism where multiple methods have the same name but different parameter lists. This allows different implementations based on the method signature.

```
public class Printer
{
   void print(String s)
   {
      System.out.println(s);
   }

   void print(int i)
   {
      System.out.println(i);
   }

   void print(double d)
   {
      System.out.println(d);
   }
}
```