Lab 27

Classroom Activity: Explain Polymorphism in Java, Method overloading and method overriding in Java

Polymorphism, method overloading, and method overriding are fundamental concepts in object-oriented programming and play a crucial role in Java. Let's explain each of these concepts:

1. Polymorphism:

Polymorphism is a concept that allows objects of different classes to be treated as objects of a common superclass. It provides a way to perform a single action in different ways. In Java, polymorphism is achieved through method overriding and interfaces.

Method Overloading:

Method overloading is a form of compile-time (static) polymorphism in which multiple methods with the same name are defined within the same class or in subclasses. The key difference is that they have different parameter lists (number or type of parameters). Java resolves which method to call at compile time based on the method's signature.

Here's an example of method overloading:

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

public class MethodOverloadingDemo {

public static void main(String[] args) {

Calculator calc = new Calculator();

int sum1 = calc.add(5, 3); // Calls the int version of add()

double sum2 = calc.add(2.5, 3.7); // Calls the double version of add()

System.out.println("Sum1: " + sum1);

System.out.println("Sum2: " + sum2);

}

}

In this example, the `Calculator` class has two `add` methods with different parameter types (int and double). Java determines which version of the method to call based on the argument types during compile time.

3. Method Overriding:

Method overriding is a form of runtime (dynamic) polymorphism in which a subclass provides a specific implementation for a method that is already defined in its superclass. This allows you to change the behavior of a method in the subclass while maintaining a common method signature.

Here's an example of method overriding:

class Animal {

void sound() {

System.out.println("Animal makes a sound.");

}

}

class Dog extends Animal {

@Override

void sound() {

System.out.println("Dog barks.");

}

}

public class MethodOverridingDemo {

public static void main(String[] args) {

Animal myAnimal = new Dog();

myAnimal.sound(); // Calls the overridden sound() method in Dog

}

}

In this example, the `Dog` class overrides the `sound` method inherited from the `Animal` class. When we create an instance of `Dog` and call `sound()`, it calls the overridden method in the `Dog` class, demonstrating runtime polymorphism.