Konsep Polymorphism dengan Overriding dan Overloading

Definisi Polymorphism:

- Polymorphism adalah kemampuan sebuah objek untuk mengambil banyak bentuk. Dalam OOP, ini memungkinkan objek dari berbagai class untuk diakses melalui interface yang sama. Polymorphism dapat dicapai melalui method overloading dan method overriding.

Method Overloading:

- Method overloading terjadi ketika dua atau lebih method dalam satu class memiliki nama yang sama tetapi dengan parameter yang berbeda (jumlah atau tipe parameter).

Syarat-syarat Method Overloading:

- Nama method harus sama.
- Parameter harus berbeda (jumlah atau tipe parameter).
- Return type boleh sama atau berbeda.

Contoh Kode Java yang Menunjukkan Method Overloading:

```java
public class Calculator {
    // Overloaded method with two int parameters
    public int add(int a, int b) {
        return a + b;
    }

    // Overloaded method with two double parameters
    public double add(double a, double b) {
        return a + b;
    }

    // Overloaded method with three int parameters
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String[] args) {
        Calculator math = new Calculator();
        System.out.println(math.add(2, 3));          // Output: 5
        System.out.println(math.add(2.5, 3.5));      // Output: 6.0
        System.out.println(math.add(1, 2, 3));       // Output: 6
```

```
    }
}
```

Method Overriding:

- Method overriding terjadi ketika subclass menyediakan implementasi spesifik untuk method yang sudah ada di superclass.

Syarat-syarat Method Overriding:

- Nama method, parameter, dan return type harus sama dengan method di superclass.
- Access modifier tidak boleh lebih ketat dari method di superclass.
- Method di superclass harus tidak berstatus final atau static.

Contoh Kode Java yang Menunjukkan Method Overriding:

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

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

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Dog();
        myAnimal.sound(); // Output: Dog barks
    }
}
```

4.2 Penggunaan Interface dan Abstract Class

Interface:

- Interface adalah blueprint dari class yang hanya memiliki method abstract (tanpa implementasi). Interface digunakan untuk mendefinisikan kontrak yang harus diikuti oleh class yang mengimplementasikannya.
- Deklarasi Interface:

```
public interface Drawable {
    void draw();
}
```

- Implementasi Interface:

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

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

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

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

Abstract Class:

- Abstract class adalah class yang tidak dapat diinstansiasi dan dapat memiliki method abstract (tanpa implementasi) dan method non-abstract (dengan implementasi).
- Deklarasi Abstract Class:

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

```
    public void description() {
        System.out.println("This is a shape.");
    }
}
```

- Implementasi Abstract Class:

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

public class Rectangle extends Shape {
    @Override
    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.
        circle.description(); // Output: This is a shape.

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

Kesimpulan Polimorfisme

Pada bab ini, kita telah mempelajari konsep polymorphism dalam OOP dengan Java. Polymorphism memungkinkan objek untuk mengambil banyak bentuk, yang dicapai melalui method overloading dan method overriding. Kita juga telah melihat bagaimana menggunakan interface dan abstract class untuk mencapai polymorphism. Pemahaman

konsep polymorphism sangat penting untuk membuat aplikasi yang fleksibel dan mudah di-maintain.