Modul 9 Java – Overriding

Tugas Pendahuluan

1. Jelaskan apa yang dimaksud dengan konsep "*overriding*" dalam pemrograman berorientasi objek.
2. Apa perbedaan antara *method* di superclass dan method yang di-override di subclass?
3. Mengapa penggunaan overriding penting dalam pembuatan hierarki kelas?
4. Bagaimana hubungan antara *overriding* dan konsep "*polimorfisme*" dalam pemrograman berorientasi objek?
5. Apa syarat yang harus dipenuhi agar sebuah method bisa di-override dalam subclass?

Materi :

Overriding adalah konsep dalam pemrograman berorientasi objek di mana sebuah subclass menyediakan implementasi yang berbeda untuk sebuah method yang telah didefinisikan dalam superclass. Ini memungkinkan subclass untuk mengganti atau "meng-override" implementasi dari method superclass sesuai dengan kebutuhannya.

Dalam overriding, method dalam superclass dan subclass memiliki nama, tipe pengembalian, dan parameter yang sama. Namun, subclass dapat memberikan implementasi yang berbeda untuk method tersebut. Konsep ini memungkinkan kita untuk memanfaatkan polimorfisme, di mana meskipun kita memiliki objek dari tipe superclass, method yang dijalankan adalah method dari subclass jika overriding dilakukan.

Beberapa poin penting tentang overriding:

1. Nama, Parameter, dan Tipe Pengembalian Sama: Method yang di-override di subclass harus memiliki nama, parameter, dan tipe pengembalian yang sama dengan method di superclass.
2. Akses Modifier: Method di subclass tidak boleh memiliki akses modifier yang lebih ketat daripada method di superclass. Misalnya, jika method di superclass bersifat protected, maka method di subclass juga bisa bersifat protected atau public, tapi tidak bisa bersifat private.
3. Exception: Subclass tidak boleh melempar exception yang tidak sesuai dengan exception yang dilempar oleh method di superclass. Subclass bisa melempar subclass dari exception yang dilempar oleh method superclass.

Contoh penggunaan overriding:

```
1. class Animal {
2.     void makeSound() {
3.         System.out.println("Animal makes a sound");
```

```
    }
}

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

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

```

Dalam contoh di atas, method makeSound() di subclass Dog telah meng-override method yang sama di superclass Animal. Ketika kita memanggil makeSound() pada objek Dog, implementasi dari method dalam subclass yang dijalankan.

Super Keyword

Kata kunci super dalam bahasa Java digunakan untuk merujuk ke anggota (atribut dan metode) dari superclass dari dalam subclass. Ini sering digunakan saat subclass melakukan overriding pada sebuah method atau ingin mengakses atribut dari superclass. Dengan menggunakan super, Anda dapat menghindari konflik penamaan antara anggota subclass dan anggota superclass.

Berikut adalah beberapa cara utama penggunaan kata kunci super:

1) Mengakses Anggota Superclass: Anda dapat menggunakan super untuk mengakses method atau atribut dari superclass. Misalnya, super.namaMethod() memanggil method di superclass, dan super.namaAtribut mengakses atribut di superclass.
2) Memanggil Konstruktor Superclass: Ketika constructor sebuah subclass dipanggil, Anda dapat menggunakan super() untuk memanggil constructor dari superclass. Ini berguna untuk menginisialisasi atribut-atribut di superclass.

Berikut adalah contoh yang menggambarkan kedua penggunaan tersebut:

```
class Hewan {
    String jenis = "Hewan";

    void bunyi() {
        System.out.println("Hewan mengeluarkan suara");
    }
}

class Anjing extends Hewan {
    String jenis = "Anjing";

    void bunyi() {
```

```
13.         System.out.println("Anjing menggonggong");
14.     }
15.
16.     void cetakJenis() {
17.         System.out.println("Jenis subclass: " + jenis);
18.         System.out.println("Jenis superclass: " + super.jenis);
19.     }
20. }
21.
22. public class Utama {
23.     public static void main(String[] args) {
24.         Anjing anjing = new Anjing();
25.         anjing.bunyi(); // Output: Anjing menggonggong
26.         anjing.cetakJenis();
27.     }
28. }
29.
```

Pada contoh di atas, anjing.cetakJenis() menggunakan baik jenis dari subclass (Anjing) maupun superclass (Hewan) dengan menggunakan kata kunci super untuk membedakan keduanya.

```
 1. class Animal {
 2.     void makeSound() {
 3.         System.out.println("Animal makes a sound");
 4.     }
 5. }
 6.
 7. class Dog extends Animal {
 8.     void makeSound() {
 9.         super.makeSound(); // Memanggil method makeSound() dari superclass
10.         System.out.println("Dog barks");
11.     }
12. }
13.
14. public class Main {
15.     public static void main(String[] args) {
16.         Dog dog = new Dog();
17.         dog.makeSound();
18.     }
19. }
20.
```

Dalam contoh di atas, kelas Dog meng-override method makeSound() dari superclass Animal. Namun, menggunakan super.makeSound() dalam method overridden memungkinkan kita untuk tetap memanggil implementasi dari method makeSound() yang ada di superclass. Dengan demikian, kita bisa menambahkan perilaku khusus dari subclass setelah memanggil method superclass. Output dari program ini akan menjadi:

Output :

```
Animal makes a sound
Dog barks
```

Penggunaan super dapat membantu menjaga kejelasan kode Anda, terutama dalam situasi di mana anggota subclass dan superclass memiliki nama yang sama.

Kapan Overriding digunakan

Overriding digunakan ketika Anda ingin mengubah implementasi dari sebuah method yang sudah didefinisikan dalam superclass di dalam subclass. Ini umumnya terjadi ketika Anda ingin menyediakan perilaku yang berbeda untuk method yang sudah ada dalam superclass. Dengan menggunakan overriding, Anda dapat memanfaatkan hierarki kelas untuk memberikan implementasi yang lebih spesifik dalam subclass, sambil tetap mempertahankan tipe data dan struktur yang sama dari method tersebut.

Berikut adalah beberapa scenario umum di mana overriding digunakan:

1. Polymorphism: Overriding memungkinkan Anda menggunakan polimorfisme, di mana Anda dapat merujuk ke objek dari kelas yang berbeda melalui referensi yang sama dan memanggil method yang menghasilkan perilaku yang sesuai dengan objek yang sebenarnya.
2. Memodifikasi Perilaku: Misalnya, dalam sebuah game, Anda mungkin memiliki kelas Character dengan method move(). Subclass seperti PlayerCharacter dan EnemyCharacter dapat meng-override method move() untuk memberikan perilaku yang sesuai dengan karakter yang bersangkutan.
3. Spesifikasi Khusus Subclass: Anda mungkin ingin memberikan implementasi yang lebih spesifik untuk method tertentu dalam subclass. Contohnya, Anda memiliki kelas Shape dengan method calculateArea(), dan subclass seperti Circle dan Rectangle meng-override method ini dengan formula yang sesuai.
4. Penyesuaian Fungsionalitas: Pada framework atau library, overriding digunakan untuk memungkinkan pengguna untuk mengubah atau memperluas fungsionalitas kelas yang sudah ada tanpa perlu merubah kode sumber asli.
5. Implementasi Interface: Ketika sebuah kelas mengimplementasikan sebuah interface, kelas tersebut harus meng-override semua method yang didefinisikan oleh interface tersebut.

Dalam semua situasi ini, overriding memungkinkan Anda memanfaatkan hierarki kelas untuk memisahkan perilaku yang berbeda sesuai dengan jenis atau karakteristik objek yang bersangkutan. beberapa contoh penggunaan overriding dalam berbagai skenario:

1. Polymorphism:

```
1. class Animal {
2.     void makeSound() {
3.         System.out.println("Animal makes a sound");
4.     }
5. }
6.
7. class Dog extends Animal {
8.     void makeSound() {
9.         System.out.println("Dog barks");
10.    }
11. }
12.
13. class Cat extends Animal {
14.    void makeSound() {
15.        System.out.println("Cat meows");
16.    }
```

```
}

public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();

        animal1.makeSound(); // Output: Dog barks
        animal2.makeSound(); // Output: Cat meows
    }
}

```

2. Memodifikasi Perilaku:

```
class Shape {
    double calculateArea() {
        return 0;
    }
}

class Circle extends Shape {
    double radius;

    double calculateArea() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    double width;
    double height;

    double calculateArea() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.radius = 5;

        Rectangle rectangle = new Rectangle();
        rectangle.width = 4;
        rectangle.height = 6;

         System.out.println("Circle Area: " + circle.calculateArea());        // Output: Circle Area: 78.53981633974483
         System.out.println("Rectangle Area: " + rectangle.calculateArea()); // Output: Rectangle Area: 24.0
    }
}

```

3. Spesifikasi Khusus Subclass:

```
class Character {
    void attack() {
        System.out.println("Character attacks");
    }
}

class PlayerCharacter extends Character {
    void attack() {
        System.out.println("PlayerCharacter attacks with a sword");
    }
}

class EnemyCharacter extends Character {
    void attack() {
        System.out.println("EnemyCharacter attacks with claws");
    }
}

public class Main {
    public static void main(String[] args) {
        Character player = new PlayerCharacter();
        Character enemy = new EnemyCharacter();

        player.attack(); // Output: PlayerCharacter attacks with a sword
        enemy.attack();  // Output: EnemyCharacter attacks with claws
    }
}

```

4. Penyesuaian Fungsionalitas:

```
class Logger {
    void log(String message) {
        System.out.println("Log: " + message);
    }
}

class FileLogger extends Logger {
    void log(String message) {
        System.out.println("Writing to file: " + message);
    }
}

public class Main {
    public static void main(String[] args) {
        Logger logger = new FileLogger();
        logger.log("Error: Critical issue detected"); // Output: Writing to file: Error:
Critical issue detected
    }
}
```

```
19.
```

Dalam semua contoh ini, subclass meng-override method yang didefinisikan dalam superclass untuk memberikan perilaku yang sesuai dengan karakteristiknya. Ini memungkinkan Anda untuk memanfaatkan hierarki kelas dan polimorfisme untuk mendapatkan perilaku yang sesuai dengan jenis objek yang digunakan.

Latihan :

Latihan 1 : Memperbaiki Program Overriding

Diberikan program di bawah ini, identifikasi kesalahan dalam konsep overriding dan perbaiki program tersebut agar menghasilkan output yang sesuai:

```
1. class Animal {
2.     void makeSound() {
3.         System.out.println("Animal makes a sound");
4.     }
5. }
6.
7. class Dog extends Animal {
8.     void bark() {
9.         System.out.println("Dog barks");
10.    }
11. }
12.
13. class Cat extends Animal {
14.    void makeSound() {
15.        System.out.println("Cat meows");
16.    }
17. }
18.
19. public class Main {
20.    public static void main(String[] args) {
21.        Animal animal1 = new Dog();
22.        Animal animal2 = new Cat();
23.
24.        animal1.makeSound(); // Output: ???
25.        animal2.makeSound(); // Output: ???
26.    }
27. }
28.
```

Petunjuk :

1) Perhatikan hubungan antara kelas Animal, Dog, dan Cat.
2) Identifikasi kesalahan dalam implementasi method makeSound() pada kelas Cat.
3) Perbaiki implementasi method makeSound() pada kelas Cat agar menghasilkan output yang diharapkan.

Latihan 2 : Memperbaiki Program Overriding

```
1. class Shape {
2.     void display() {
3.         System.out.println("This is a shape");
4.     }
5. }
6.
7. class Circle extends Shape {
8.     void display() {
9.         System.out.println("This is a circle");
10.    }
11. }
12.
13. public class Main {
14.    public static void main(String[] args) {
15.        Shape shape = new Circle();
16.        shape.display();
17.    }
18. }
19.
```

Petunjuk :

1) Perhatikan konsep overriding dan hubungan antara kelas Shape dan Circle.
2) Perbaiki method display() dalam kelas Circle agar menghasilkan output yang sesuai.
3) Pastikan program tidak mengandung error dan menghasilkan output yang benar.

Tugas Rumah :

1. Polymorphism dan Overriding
   Buatlah tiga kelas: Vehicle, Car, dan Bike. Vehicle memiliki method displayInfo() yang mencetak "This is a vehicle." Car dan Bike masing-masing meng-override method ini dan mencetak "This is a car." dan "This is a bike." Buat objek dari masing-masing kelas dan panggil method displayInfo() pada setiap objek.
2. Hewan dan Suara
   Buatlah kelas Animal dengan method makeSound() yang mencetak "Animal makes a sound." Buat dua subclass: Dog dan Cat. Dog meng-override method makeSound() dan mencetak "Dog barks." Sedangkan Cat meng-override method makeSound() dan mencetak "Cat meows." Buat objek dari masing-masing kelas dan panggil method makeSound() pada setiap objek.
3. Perhitungan Luas Bangun
   Buat kelas Shape dengan method calculateArea() yang mengembalikan nilai 0. Buat dua subclass: Rectangle dan Circle. Rectangle memiliki atribut width dan height, serta meng-override method calculateArea() untuk menghitung luas persegi panjang. Circle memiliki atribut radius dan meng-override method calculateArea() untuk menghitung luas lingkaran. Buat objek dari masing-masing kelas dan panggil method calculateArea() pada setiap objek.
4. Penyesuaian Fungsionalitas

Buat kelas Logger dengan method log(message) yang mencetak "Log: [message]". Buat subclass FileLogger yang meng-override method log(message) untuk mencetak "Writing to file: [message]". Buat objek dari kelas FileLogger dan panggil method log().