Modul 10 – Polymorphism

Tugas Pendahuluan :

1. Apa yang dimaksud dengan polymorphism dalam pemrograman berorientasi objek?
2. Bagaimana konsep polymorphism terlihat dalam studi kasus di atas?
3. Apa yang terjadi ketika metode suara() dipanggil pada objek dari kelas Anjing atau Kucing?
4. Apakah Anda melihat bagaimana pemahaman tentang polymorphism dapat meningkatkan fleksibilitas dan modularitas kode?

Materi:

Polymorphism adalah salah satu konsep utama dalam pemrograman berorientasi objek (OOP) yang mengacu pada kemampuan objek untuk mengambil bentuk yang berbeda atau berperilaku dengan cara yang berbeda dalam konteks yang berbeda. Konsep ini memungkinkan Kita untuk menangani objek dari berbagai kelas dengan cara yang seragam, meningkatkan fleksibilitas dan modularitas dalam kode Kita. Ada dua tipe polimorfisme dalam OOP:

1. Compile-time Polymorphism (Polimorfisme Waktu Kompilasi): Ini juga dikenal sebagai "polimorfisme statis". Ini terjadi saat Kita melakukan pemanggilan method yang akan dieksekusi selama waktu kompilasi. Dalam bahasa pemrograman seperti Java, ini umumnya terkait dengan konsep "overloading", di mana beberapa method memiliki nama yang sama tetapi jumlah atau tipe parameter yang berbeda.

   Contoh:

```
1. class Calculator {
2.     int add(int a, int b) {
3.         return a + b;
4.     }
5.
6.     double add(double a, double b) {
7.         return a + b;
8.     }
9. }
10.
```

2. Runtime Polymorphism (Polimorfisme Waktu Eksekusi): Ini juga dikenal sebagai "polimorfisme dinamis". Ini terjadi saat Kita melakukan pemanggilan method yang akan dieksekusi selama waktu eksekusi. Ini terkait dengan konsep "overriding", di mana subclass dapat mengganti (override) implementasi method yang sudah ada di superclass.

   Contoh:

```
1. lass 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.
```

Penggunaan polimorfisme memungkinkan kita untuk membuat kode yang lebih fleksibel dan mudah diubah. Kita dapat mengakses objek dari berbagai kelas yang berbeda dengan menggunakan referensi yang sama dan memanggil method yang sesuai dengan jenis objek yang sebenarnya (runtime polymorphism). Ini juga meningkatkan reusable dan extensibility kode Kita, memungkinkan Kita menambahkan kelas baru tanpa memodifikasi kode yang sudah ada (compile-time polymorphism).

Dalam Java, salah satu contoh polimorfisme yang sering digunakan adalah penggunaan "interface" dan "inheritance" untuk menciptakan hierarki kelas yang beraneka ragam dan kemudian menggunakan polimorfisme untuk mengakses objek-objek ini dengan cara seragam.

Kapan dan kenapa kita harus menggunakan Polymorphism.?

Polymorphism digunakan dalam pemrograman berorientasi objek (OOP) ketika Kita ingin mencapai fleksibilitas, modularitas, dan reusabilitas dalam kode Kita. Berikut adalah beberapa situasi di mana dan mengapa Kita harus menggunakan polymorphism:

1. Meningkatkan Fleksibilitas: Polymorphism memungkinkan Kita untuk menangani objek dari berbagai kelas dengan cara yang seragam. Ini berarti Kita dapat mengganti objek dengan objek lain yang

memiliki jenis yang berbeda, tetapi perilaku yang serupa, tanpa perlu mengubah banyak bagian dari kode Kita. Ini meningkatkan fleksibilitas dalam menangani objek-objek yang beragam.

2. Meningkatkan Reusabilitas: Dengan menggunakan polymorphism, Kita dapat membuat kelas-kelas yang dapat digunakan ulang dengan cara yang lebih luas. Ini karena Kita dapat membuat kode yang mengkitalkan antarmuka atau superclass yang lebih abstrak daripada kelas yang spesifik. Ini memungkinkan Kita menambahkan kelas baru ke dalam hierarki kelas tanpa harus memodifikasi banyak bagian dari kode yang sudah ada.
3. Mendukung Polimorfisme Dinamis: Polymorphism memungkinkan Kita untuk menciptakan code yang berjalan secara dinamis. Kita dapat mengganti implementasi di runtime, yang sangat berguna ketika Kita memiliki hierarki kelas yang besar dan ingin mengganti perilaku berdasarkan objek yang sebenarnya saat program berjalan.
4. Meningkatkan Ekstensibilitas: Polymorphism memungkinkan Kita untuk menambahkan fitur-fitur baru ke dalam kode Kita dengan mudah. Kita dapat membuat subclass baru yang menggantikan metode yang ada di superclass tanpa mengubah perilaku di kelas yang sudah ada.
5. Meningkatkan Keterbacaan: Polymorphism membuat kode Kita lebih mudah dibaca dan dimengerti. Saat Kita melihat sebuah objek dipanggil menggunakan metode yang sama, Kita tahu bahwa objek itu akan berperilaku sesuai dengan jenisnya, bahkan jika Kita tidak tahu jenisnya secara pasti.

Contoh nyata penggunaan polymorphism bisa ditemukan dalam kode yang berurusan dengan koleksi objek yang berbeda, seperti dalam sistem manajemen konten, game, atau framework. Polymorphism juga digunakan dalam pengembangan aplikasi yang berfokus pada antarmuka pengguna (UI) untuk mengatur interaksi dengan berbagai elemen UI. Jadi, polymorphism berguna ketika kita ingin menciptakan kode yang lebih fleksibel, dapat diubah, dan dapat digunakan ulang dalam konteks yang beragam.

Contoh 1: Polymorphism pada Collections

```
1. import java.util.ArrayList;
2. import java.util.List;
3. 
4. class Animal {
5.     void makeSound() {
6.         System.out.println("Animal makes a sound");
7.     }
8. }
9. 
```

```
10. class Dog extends Animal {
11.     void makeSound() {
12.         System.out.println("Dog barks");
13.     }
14. }
15.
16. class Cat extends Animal {
17.     void makeSound() {
18.         System.out.println("Cat meows");
19.     }
20. }
21.
22. public class Main {
23.     public static void main(String[] args) {
24.         List<Animal> animals = new ArrayList<>();
25.         animals.add(new Dog());
26.         animals.add(new Cat());
27.
28.         for (Animal animal : animals) {
29.             animal.makeSound(); // Output: Dog barks, Cat meows
30.         }
31.     }
32. }
33.
```

Dalam contoh di atas, kita memiliki koleksi objek dari kelas Animal, tetapi kita dapat menambahkan objek dari subclass seperti Dog dan Cat. Saat kita mengakses method makeSound() pada setiap objek dalam loop, polymorphism memastikan bahwa metode yang benar dipanggil untuk setiap objek.

Contoh 2: Polymorphism dalam Abstract Class

```
 1. abstract class Shape {
 2.     abstract void draw();
 3. }
 4.
 5. class Circle extends Shape {
 6.     void draw() {
 7.         System.out.println("Drawing a circle");
 8.     }
 9. }
10.
11. class Rectangle extends Shape {
```

```
12.     void draw() {
13.         System.out.println("Drawing a rectangle");
14.     }
15. }
16.
17. public class Main {
18.     public static void main(String[] args) {
19.         Shape shape1 = new Circle();
20.         Shape shape2 = new Rectangle();
21.
22.         shape1.draw(); // Output: Drawing a circle
23.         shape2.draw(); // Output: Drawing a rectangle
24.     }
25. }
26.
```

Dalam contoh ini, kita memiliki kelas abstrak Shape dengan metode abstrak draw(). Subclass seperti Circle dan Rectangle mengimplementasikan metode ini sesuai dengan jenis bentuk mereka. Dengan menggunakan polymorphism, kita dapat membuat objek dari kelas abstrak dan memanggil metode yang sesuai dengan jenis objek yang sebenarnya.

Contoh 3: Polymorphism pada Interfaces

```
1. interface Sound {
2.     void makeSound();
3. }
4.
5. class Dog implements Sound {
6.     public void makeSound() {
7.         System.out.println("Dog barks");
8.     }
9. }
10.
11. class Cat implements Sound {
12.     public void makeSound() {
13.         System.out.println("Cat meows");
14.     }
15. }
16.
17. public class Main {
18.     public static void main(String[] args) {
19.         Sound dog = new Dog();
20.         Sound cat = new Cat();
```

```
21.
22.         dog.makeSound(); // Output: Dog barks
23.         cat.makeSound(); // Output: Cat meows
24.     }
25. }
26.
```

Dalam contoh ini, kita memiliki antarmuka Sound yang diterapkan oleh kelas Dog dan Cat. Kita dapat membuat objek dari antarmuka ini dan menggunakan polymorphism untuk memanggil metode makeSound() yang sesuai dengan jenis objek yang sebenarnya.

Latihan :

Latihan 1: Perbaiki Polymorphism dalam Hierarki Kelas

Program di bawah ini mencoba mengimplementasikan hierarki kelas sederhana untuk hewan. Namun, ada beberapa masalah dalam program ini yang perlu diperbaiki. Identifikasi masalah-masalahnya dan perbaiki program ini sehingga mencapai hasil yang diharapkan.

```
 1. class Animal {
 2.     void sound() {
 3.         System.out.println("Animal makes a sound")
 4.     }
 5. }
 6.
 7. class Dog extends Animal {
 8.     void sound() {
 9.         System.out.println("Dog barks");
10.     }
11. }
12.
13. class Cat extends Animal {
14.     void sound() {
15.         System.out.println("Cat meows");
16.     }
17. }
18.
19. public class Main {
20.     public static void main(String[] args) {
21.         Animal[] animals = new Animal(3);
22.         animals[0] = new Animal();
23.         animals[1] = new Dog();
24.         animals[2] = new cat();
25.
```

```
        for (Animal animal : Animals) {
            animal.sound();
        }
    }
}

```

Soal 2: Memperbaiki Polymorphism dalam Interface

Program berikut mencoba menggunakan interface untuk mengimplementasikan polimorfisme. Namun, ada kesalahan dalam penulisan dan logika yang perlu diperbaiki. Temukan dan perbaiki kesalahan-kesalahan tersebut.

```
interface Shape {
    void draw();
}

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

class Rectangle implements Shape {
    void draw() {
        System.out.println("Drawing a rectangle");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape[] shapes = new Shape(2)
        shapes[0] = new Circle();
        shapes[1] = new Rectangle();

        for (Shape shape : shapes) {
            shape.draw();
        }

}

```

Soal 3: Polymorphism dalam Perpustakaan Buku

Program berikut mencoba menggunakan polimorfisme dalam konteks perpustakaan buku. Namun, ada beberapa masalah dalam program ini. Temukan masalah-masalah tersebut dan perbaiki program ini.

```
1. class Book {
2.     void display() {
3.         System.out.println("This is a generic book.");
4.     }
5. }
6.
7. class Fiction extends Book {
8.     void display() {
9.         System.out.println("This is a fiction book.")
10.    }
11. }
12.
13. class NonFiction extends Book {
14.    void display() {
15.        System.out.println("This is a non-fiction book.");
16.    }
17. }
18.
19. public class Library {
20.    public static void main(String[] args) {
21.        Book[] books = new Book[3];
22.        books[0] = new Fiction();
23.        books[1] = new NonFiction();
24.        books[2] = new Book();
25.
26.        for (Book book : books) {
27.            book.display();
28.        }
29.    }
30.
31.
```

Tugas :

Soal 1: Polymorphism dengan Kelas Hewan

Anda diberikan hierarki kelas berikut: Hewan, Anjing, Kucing, Sapi. Setiap kelas ini memiliki metode suara() yang mencetak suara karakteristik hewan ke layar. Implementasikan kelas-kelas ini dan buatlah objek-objeknya.

1. Buat objek dari setiap kelas: Anjing, Kucing, dan Sapi.
2. Panggil metode suara() pada setiap objek.
3. Amati hasilnya dan catat suara yang dihasilkan oleh masing-masing hewan.

Soal 2: Polymorphism dengan Interface Bentuk

Anda diberikan sebuah interface Bentuk yang mendefinisikan metode hitungLuas(). Implementasikan tiga kelas yang mengimplementasikan interface ini: Lingkaran, Persegi, dan Segitiga. Setiap kelas harus memiliki implementasi yang sesuai untuk menghitung luas bentuknya.

1. Buat objek dari setiap kelas: Lingkaran, Persegi, dan Segitiga.
2. Panggil metode hitungLuas() pada setiap objek.
3. Amati hasilnya dan catat luas masing-masing bentuk.

Soal 3: Polymorphism dalam Array

Anda memiliki sebuah array objek dengan elemen-elemen dari kelas yang berbeda, seperti Kucing, Anjing, dan Sapi. Anda ingin mengeluarkan suara dari setiap hewan dalam array tersebut.

1. Buat array objek dengan elemen-elemen dari kelas yang berbeda.
2. Gunakan loop untuk memanggil metode suara() pada setiap objek dalam array.
3. Amati bagaimana polymorphism memungkinkan Anda untuk mengakses metode yang sesuai bahkan jika objeknya berbeda kelas.

Soal 4: Polymorphism dalam Perpustakaan Buku

Anda memiliki hierarki kelas buku yang mencakup kelas Buku, Fiksi, dan NonFiksi. Setiap kelas memiliki metode deskripsi() yang mencetak deskripsi buku.

1. Buat objek dari setiap kelas buku: Fiksi dan NonFiksi.

2. Panggil metode deskripsi() pada setiap objek.
3. Amati bagaimana polymorphism memungkinkan Anda untuk menggunakan metode yang sesuai untuk setiap objek dalam hierarki kelas buku.