TUGAS 4 (TERSTRUKTUR)

Nama Anggota :

1. Farras Lathief (12250111328)
2. Muh. Zaki Erbai Syas (12250111134)
3. Rahma Laksita (12250124357)

Mata Kuliah : Pemrograman Lanjut
Dosen Pengampu : Liza Afriyanti, M.Kom.
Semester / Kelas : 2 / A

I. ABSTRACT

A. Aturan Class Abstract

- Class abstract tidak dapat diinstansiasi secara langsung, artinya kita tidak dapat membuat objek langsung dari class abstract.
- Class abstract digunakan sebagai template atau kerangka dasar untuk subclass yang akan diwarisi darinya.
- Class abstract dapat memiliki method abstrak (tanpa implementasi) dan method non-abstrak (dengan implementasi).
- Class abstract dapat memiliki variabel anggota.
- Digunakan untuk menggambarkan konsep yang lebih umum dan tidak spesifik.

B. Mendeklarasikan Class Abstract

- Untuk mendeklarasikan sebuah class sebagai abstract, kita perlu menggunakan kata kunci "abstract" sebelum kata kunci "class".
- Contoh:
  public abstract class Animal { ... }

C. Class Diagram Abstract Class

- Class diagram merupakan salah satu jenis diagram pada UML (Unified Modeling Language) yang digunakan untuk merepresentasikan struktur dan relasi antar

objek dalam suatu sistem. Abstract class pada class diagram merupakan salah satu jenis kelas yang tidak bisa di-instansiasi menjadi objek, namun dapat diwarisi oleh kelas-kelas lainnya..

- Dalam class diagram, abstract class ditandai dengan tanda kurung kurawal ({}) pada nama kelas dan garis miring (/) pada method yang bersifat abstract.
- Hubungan antara class abstract dengan subclass ditunjukkan dengan garis panah yang menghubungkan subclass dengan class abstract.

D. Keuntungan Method Abstrak

- Method abstrak dideklarasikan di dalam class abstract tanpa implementasi. Implementasi dari method ini akan dilakukan oleh subclass yang mewarisi class abstract tersebut.
- Keuntungan menggunakan method abstrak adalah memastikan bahwa setiap subclass yang mewarisi class abstract tersebut akan mengimplementasikan method tersebut.
- Method abstrak memungkinkan pengembang untuk memaksa adanya perilaku yang sama pada setiap subclass, sementara bagian implementasinya dapat bervariasi.

Contoh Abstract:

```
// Abstract class
abstract class Animal {
        public abstract void animalSound();
        public void sleep() {
                System.out.println("Zzz");
        }
}

class Pig extends Animal {
        public void animalSound() {
                System.out.println("The pig says: wee wee");
        }
}
```

```
class Main {
        public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
        }
}
```

II. POLYMORPHISM

A. Apa itu Polymorphism?

- Polymorphism adalah konsep dalam pemrograman berorientasi objek di mana objek dari kelas yang berbeda dapat dianggap sebagai objek dari superclass yang lebih umum.
- Polymorphism memungkinkan objek untuk merespons dengan cara yang berbeda tergantung pada kelas atau tipe yang memanggilnya.
- Polymorphism membantu dalam mencapai fleksibilitas dan modularitas dalam desain perangkat lunak.

B. Bagaimana Membuat Polymorphism?

- Polymorphism dicapai melalui konsep pewarisan dan overriding dalam pemrograman berorientasi objek.
- Subclass dapat menggantikan atau mengubah perilaku metode yang diwarisi dari superclass dengan membuat metode dengan nama yang sama dalam subclass.
- Saat metode dipanggil pada objek, metode yang sesuai akan dieksekusi berdasarkan tipe objek yang sebenarnya, bukan tipe variabel referensi.

C. Polymorphisme Dinamis

- Juga dikenal sebagai late binding atau overriding dinamis.
- Polymorphisme dinamis terjadi saat metode yang akan dieksekusi ditentukan saat runtime berdasarkan tipe objek yang sebenarnya.
- Polymorphisme dinamis memungkinkan objek untuk mengambil banyak bentuk dan merespons dengan cara yang berbeda sesuai dengan implementasi yang sesuai dengan kelasnya.

D. Polymorphisme Static

- Juga dikenal sebagai early binding atau method overloading.
- Polymorphisme static terjadi saat metode yang akan dieksekusi ditentukan saat kompilasi berdasarkan tipe referensi variabel.
- Polymorphisme static terjadi saat terdapat beberapa metode dengan nama yang sama namun memiliki parameter yang berbeda dalam satu class.
- Pemilihan metode yang tepat terjadi berdasarkan parameter atau tipe argumen yang diberikan saat pemanggilan metode.

E. Polymorphism dalam Pewarisan

- Polymorphism memainkan peran penting dalam pewarisan antara superclass dan subclass.
- Subclass dapat memperluas atau mengubah perilaku superclass dengan mengoverride metode yang diwarisi.
- Ketika metode yang diwarisi dari superclass dipanggil melalui objek subclass, metode yang diimplementasikan di subclass akan dieksekusi.

F. Variable superclass dapat memegang referensi dari objek subclass

- Dalam pemrograman berorientasi objek, variabel dengan tipe superclass dapat digunakan untuk memegang referensi dari objek subclass.
- Hal ini memungkinkan fleksibilitas dalam penggunaan objek, di mana objek subclass dapat ditangani melalui variabel superclass.
- Keuntungan dari ini adalah memungkinkan penggunaan polimorfisme, di mana objek dari kelas yang berbeda dapat diproses secara seragam menggunakan referensi superclass.

Contoh Polymorphism:

```
class Animal {
        public void animalSound() {
                System.out.println("The animal makes a sound");
        }
}

class Pig extends Animal {
```

```
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
    Animal myAnimal = new Animal();
    Animal myPig = new Pig();
    Animal myDog = new Dog();
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
    }
}
```

Pemahaman yang mendalam tentang abstract class dan polymorphism penting dalam pengembangan perangkat lunak berorientasi objek. Abstract class membantu dalam mengorganisir hierarki kelas, sedangkan polymorphism memungkinkan fleksibilitas dan modularitas dalam desain perangkat lunak. Dengan menerapkan konsep ini dengan baik, pengembang dapat menciptakan solusi yang lebih bersih, terstruktur, dan mudah dimengerti.