

Polymorphism Abstract Class Interface

Pemrograman Berorientasi
Object

PBO-Suprayogi,M.Kom

1

Polymorphism

- Polymorphism: Suatu object dapat memiliki berbagai bentuk, sebagai object dari classnya sendiri atau object dari superclassnya.
 - Overloading: Penggunaan satu nama untuk beberapa method yang berbeda (beda parameter)
 - Overriding: Terjadi ketika deklarasi method subclass dengan nama dan parameter yang sama dengan method dari superclassnya

PBO-Suprayogi,M.Kom

2

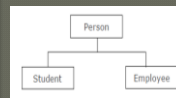
Polymorphism

- Ketika sebuah variabel atau referensi dapat merujuk ke berbagai jenis objek maka disebut polimorfisme.
- Polimorfisme adalah istilah yang berarti "banyak bentuk"
- Dalam kasus pemrograman, polimorfisme memungkinkan variabel untuk merujuk ke berbagai jenis objek, yang berarti mereka dapat memiliki beberapa bentuk.
- Misalnya, Student "is-a" Person, ada dua kemungkinan referensi untuk menentukan jenis objeknya (Person atau Student)

PBO-Suprayogi,M.Kom

3

Polymorfisme



- Contoh kita dapat membuat referensi yang merupakan tipe dari superclass ke sebuah object dari subclass tersebut.

```

public static main( String[] args )
{
    Person ref;
    Student studentObject = new Student();
    Employee employeeObject = new Employee();
    ref = studentObject; //Person menunjuk kepada
                        // object Student
}
  
```

PBO-Suprayogi,M.Kom

4

Contoh

```

1 public class Person
2 {
3     protected String name;
4     public String getName(){
5         System.out.println("Person Name:" + name);
6         return name;
7     }
8 }
9
10 public class Student extends Person
11 {
12     public String getName()
13     {
14         System.out.println("Student Name:" + name);
15         return name;
16     }
17 }
18
19 public class Employee extends Person
20 {
21     public String getName(){
22         System.out.println("Employee Name:" + name);
23         return name;
24     }
25 }
26
27 public class TestPM1 {
28     public static void main(String[] args)
29     {
30         Person ref;
31         Student studentObject = new Student();
32         Employee employeeObject = new Employee();
33         ref=studentObject;
34         String temp=ref.getName();
35         System.out.println(temp);
36
37         ref=employeeObject;
38         String temp=ref.getName();
39         System.out.println(temp);
40     }
41 }
  
```

- Polimorfisme : Kemampuan dari reference untuk mengubah sifat menurut object apa yang dijadikan acuan.
- Polimorfisme menyediakan multioject dari subclasses yang berbeda untuk diperlakukan sebagai object dari superclass

PBO-Suprayogi,M.Kom

5

Overriding Methods

- Overriding Method** adalah cara mendefinisikan ulang *method* dengan jenis dan parameter pengembalian yang sama dengan menambahkan, atau memodifikasi logika yang ada, dalam metode subClass.
- Overriding** pada dasarnya menyembunyikan method - parent , dan membolehkan dipanggil di dalam object sub-class dengan menggunakan keyword super.

```

1 public class Person
2 {
3     protected String name;
4     public String getName(){
5         System.out.println("Person Name:" + name);
6         return name;
7     }
8 }
9
10 public class Student extends Person
11 {
12     public String getName()
13     {
14         System.out.println("Student Name:" + name);
15         return name;
16     }
17 }
  
```

PBO-Suprayogi,M.Kom

6

Constructor

```

public class Person {
    protected String name;
    protected Person() {
        name = "";
        System.out.println("Constructor(Person Name" + name);
    }
    public String getName() {
        System.out.println("Person Name" + name);
        return name;
    }
}

public class Student extends Person {
    Student() {
        System.out.println("Constructor(Student Name" + name);
    }
    public String getName() {
        System.out.println("Student Name" + name);
        return name;
    }
}

public class Employee extends Person {
    Employee() {
        System.out.println("Constructor(Employee Name" + name);
    }
    public String getName() {
        System.out.println("Employee Name" + name);
        return name;
    }
}

public class TestMain {
    public static void main(String[] args) {
        Person p = new Person();
        Student s = new Student();
        Employee e = new Employee();
        System.out.println(p.getName());
        System.out.println(s.getName());
        System.out.println(e.getName());
    }
}
                
```

Polymorphisme

```

public class Person {
    protected String name;
    protected Person(String name) {
        this.name = name;
    }
    public String getName() {
        System.out.println("Person Name" + name);
        return name;
    }
}

public class Student extends Person {
    Student(String name) {
        super(name);
    }
    public String getName() {
        System.out.println("Student Name" + name);
        return name;
    }
}

public class Employee extends Person {
    Employee(String name) {
        super(name);
    }
    public String getName() {
        System.out.println("Employee Name" + name);
        return name;
    }
}

public class TestMain {
    public static void main(String[] args) {
        Student studentObject = new Student("Student");
        Employee employeeObject = new Employee("Employee");
        printInformation(studentObject);
        printInformation(employeeObject);
        printInformation(new Person(""));
    }
}
                
```

polymorphism adalah kemampuan untuk menghasilkan sesuatu yang berbeda dengan cara yang sama. Pemberian obyek dari subclass ke obyek dari superclass dapat dilakukan tanpa perlu melakukan konversi.

Polymorphisme

- Cara menghasilkan efek polimorphisme
 - Extends dari class biasa
 - Extend dari abstract class
 - Implement sebuah interface

Class abstract

- Class abstract adalah class yang tidak dapat di-instansiasi:
 - Artinya tidak dapat membuat objek dari class ini.
 - Tapi bisa membuat variabel, atau referensi dari class ini.
 - Misal:


```
Person someOne = new Student();
```
- Jika kita membuat class Person menjadi abstract, kita masih dapat menggunakan sintaks tsb, tetapi kita tidak dapat membuat objek Person.
- Ini berarti semua referensi tipe Person akan merujuk objek subclass Student.

Contoh class abstract

```

1 public abstract class Binatang {
2     abstract void makan();
3     abstract void tidur();
4 }
5
6 public class Gajah extends Binatang {
7     void makan() {
8         System.out.println("Gajah makan...");
9     }
10    void tidur() {
11        System.out.println("Gajah tidur...");
12    }
13 }
14
15 public class Kerbau extends Binatang {
16     void makan() {
17         System.out.println("Kerbau makan...");
18     }
19     void tidur() {
20         System.out.println("Kerbau tidur...");
21     }
22 }
23
24 public class TestBinatang {
25     public static void main(String[] args) {
26         Binatang gajah = new Gajah();
27         Binatang kerbau = new Kerbau();
28         test(gajah);
29         test(kerbau);
30     }
31     public static void test(Binatang b) {
32         b.makan();
33         b.tidur();
34     }
35 }
                
```

Penggunaan Abstract Class

- Abstract class merupakan class yang mendeklarasikan methodnya, tanpa implementasi isi.
- Abstract boleh tidak memiliki satupun method abstract (class ini digunakan sbg class basis untuk mendefinisikan sub-class baru).
- Baik class maupun method harus ditambahkan keyword "abstract"
- Konsekuensi abstract class, tidak dapat create objectnya, tapi yang mengimplementasikannya bisa.
- Keuntungannya, lebih simpel dan hemat memori (tidak dibuatkan objectnya oleh jvm).
- Penggunaannya ketika hanya mengetahui sebagian dari implementasi method

Aturan Pembuatan Abstract Class

- Method Abstract tidak bisa dimuat dalam class non abstract.
- Jika sub-class dari super-class abstract tidak mengimplementasikan semua method abstract maka sub-class tersebut harus didefinisikan sebagai class abstract.
- Method abstract harus bersifat non-static.
- Class abstract tidak bisa diinstansiasi menggunakan operator new, tapi constructor dari class abstract tsb. Dapat dipanggil dari sub-class.
- Suatu class yang memuat method abstract harus didefinisikan sbg class abstract.
- Sub-class dapat menjadi abstract meskipun super-classnya konkrit.
- Suatu class abstract dapat digunakan sebagai tipe data.
Misl: objGeometri[] obj = new objGeometri[10];
obj[0]= new Lingkaran();

Class Abstract

- Boleh mengandung method non-abstract

```

1 public abstract class Binatang {
2     abstract void makan();
3     abstract void tidur();
4     void berlari() {
5         System.out.println("Binatang berlari...");
6     }
7 }
    
```

Class Abstract

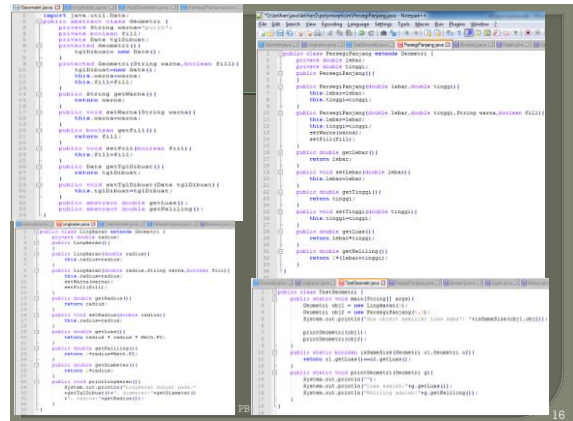
```

1 public abstract class Binatang {
2     abstract void makan();
3     abstract void tidur();
4 }
5
6 public abstract class Kerbau extends Binatang {
7     void tidur() {
8         System.out.println("Kerbau tidur...");
9     }
10 }
    
```

```

1 public class TestBinatang {
2     public static void main(String[] args) {
3         Binatang gajah = new Gajah();
4         //Binatang kerbau = new Kerbau();
5         test(gajah);
6     }
7
8     public static void test(Binatang b) {
9         b.makan();
10        b.tidur();
11    }
12 }
    
```

Class kerbau tdk mengimplementasikan method makan maka class kerbau wajib dibuat abstract, krn abstract maka tidak dapat dibuat object-nya



Manfaat Abstract Class

- Abstract Class digunakan sebagai basis bagi penurunan class lainnya.
- digunakan ketika diperlukan menerapkan fungsi dasar dan atribut dasar.

Interface

- Merupakan konstruksi class yang hanya memuat konstanta-konstanta dan method-method abstract.
- Tujuan interface adalah untuk menentukan watak umum dari object dengan menetapkan sifat-sifat dari class.

Interface

```

1 interface Hewan {
2     abstract void makan();
3     abstract void tidur();
4 }
5
6 class Badak implements Hewan {
7     public void makan() {
8         System.out.println("Badak Makan..");
9     }
10    public void tidur() {
11        System.out.println("Badak Tidur..");
12    }
13    void berlati() {
14        System.out.println("Badak berlati..");
15    }
16 }
17
18 class Faming implements Hewan {
19     public void makan() {
20         System.out.println("Faming Makan..");
21     }
22     public void tidur() {
23         System.out.println("Faming Tidur..");
24     }
25 }
26
27 public class TestHewan {
28     public static void main(String[] args) {
29         Hewan badak = new Badak();
30         Hewan faming = new Faming();
31         test(badak);
32         test(faming);
33     }
34     public static void test(Hewan h) {
35         h.makan();
36         h.tidur();
37     }
38 }
  
```

PBO-Suprayogi,M.Kom

19

Interface

- Mirip seperti abstract class, tapi semua method harus abstract
- Class yang mengimplement interface harus menggunakan keyword "implement" (bukan "extends").
- Class yang mengimplement interface harus mengimplement semua method yang dideklarasikan dalam interface dan harus diberi label "public".
- Sebuah class bisa mengimplement lebih dari satu interface.
- Semua implementasi dalam class harus public.

PBO-Suprayogi,M.Kom

20

Extends + implements

- sebuah class hanya bisa meng-extends satu class saja, namun bisa meng-implement banyak interface.

```

1 public class Manusia {
2     public void bernafas() {
3         System.out.println("Manusia bernafas..");
4     }
5 }
6
7 public class Dosen extends Manusia {
8     void mengajar() {
9     }
10 }
11
12 public interface Sniper {
13     void menembak();
14 }
15
16 public class ManusiaSuper extends Manusia implements Dosen, Sniper {
17     public void menembak() {
18         System.out.println("ManusiaSuper menembak");
19     }
20     public void mengajar() {
21         System.out.println("ManusiaSuper mengajar");
22     }
23 }
24
25 public class TestManusia {
26     public static void main(String[] args) {
27         ManusiaSuper almet = new ManusiaSuper();
28         testDosen(almet);
29         testSniper(almet);
30         testManusia(almet);
31     }
32     static void testDosen(Dosen d) {
33         d.mengajar();
34     }
35     static void testSniper(Sniper s) {
36         s.temembak();
37     }
38     static void testManusia(Manusia m) {
39         m.bernafas();
40     }
41 }
  
```

PBO-Suprayogi,M.Kom

21

Program to Interface

- merupakan pendekatan pemrograman dimana dengan konsep OOP lebih fokus pada interface bukan pada implementasi (hidden implementation)

```

1 public interface Database {
2     abstract void connect();
3     abstract void disconnect();
4 }
5
6 public class MyDB implements Database {
7     public void connect() {
8         System.out.println("MyDB connect..");
9     }
10    public void disconnect() {
11        System.out.println("MyDB disconnect..");
12    }
13 }
14
15 public class AksesDB {
16     public static void main(String[] args) {
17         Database d = new MyDB();
18         d.connect();
19         d.disconnect();
20     }
21 }
  
```

PBO-Suprayogi,M.Kom

22

Interface

- Interface digunakan apabila kita ingin menentukan apa yang harus dilakukan oleh suatu class tapi tidak menentukan bagaimana cara untuk melakukannya.
- Gunakan Interface ketika diperlukan menekankan fungsi-fungsi tertentu yang perlu didefinisikan.

PBO-Suprayogi,M.Kom

23

Final

- ada kondisi dimana *method* tidak ingin dioverride atau *method* tidak boleh dimiliki oleh sub-class.
- Java menyediakan keyword final untuk mencegah programmer dari meng override method atau melarang membuat sub-class.

```
public final class String {}
```

- class immutable, artinya tidak boleh ada yang meng-extends String dan memodifikasi /mengganti *methodnya*.

PBO-Suprayogi,M.Kom

24