Polymorphism Abstract Class Interface

Pemrograman Berorientasi Object

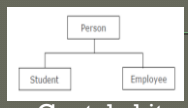
Polymorphism

- 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

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, polimorphisme 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)

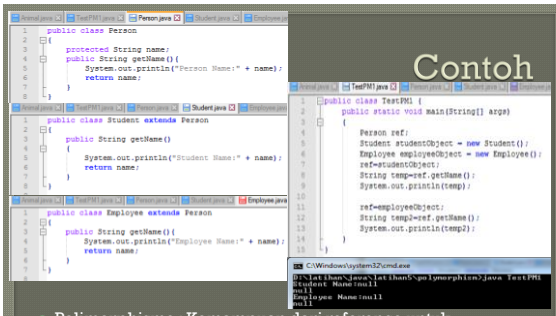
Polymorphisme



- 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
}
```

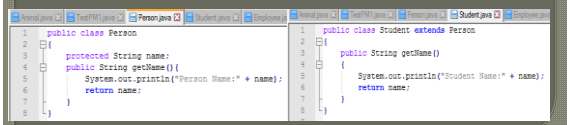

Contoh

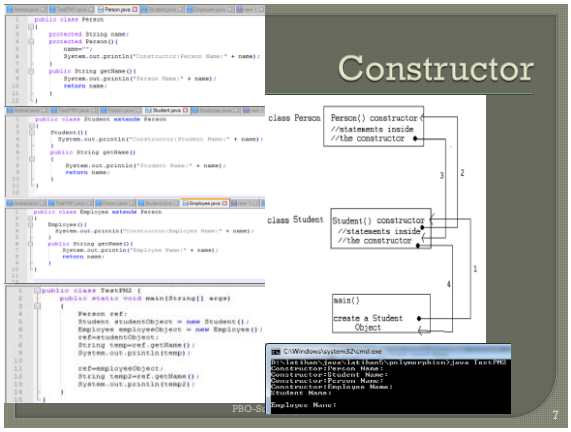


- Polimorphisme : Kemampuan dari reference untuk mengubah sifat menurut object apa yang dijadikan acuan.
- Polimorphisme menyediakan multiobject dari subclasses yang berbeda untuk diperlakukan sebagai object dari superclass

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.





Polymorphisme

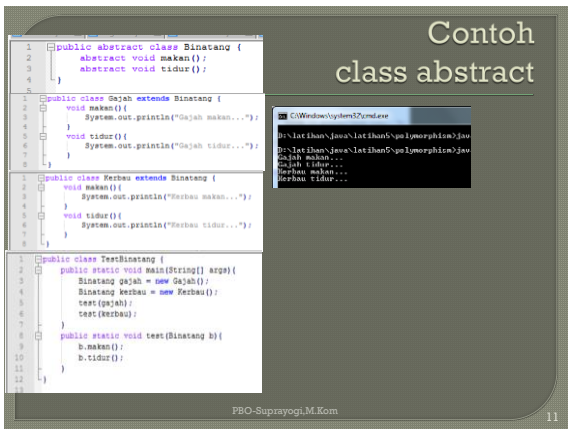
- Cara menghasilkan efek polimorphisme
  - Extends dari class biasa
  - Extend dari abstract class
  - Implement sebuah interface

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.
  - Misl:

    Person someOne = new Student();

  - Jika kita membuat class Person menjadi absctract, kita masih dapat menggunakan sintaks tsb, tetapi kita tidak dapat membuat objek Person.
  - Ini berarti semua referensi tipe Person akan merujuk objek subclass Student.



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 dicreate 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 absctract.
- Jika sub-class dari super-class absctract 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

```
public abstract class Binatang {
    abstract void makan();
    abstract void tidur();
    void berlari(){
        System.out.println("Binatang berlari...");
    }
}
```


Class Abstract

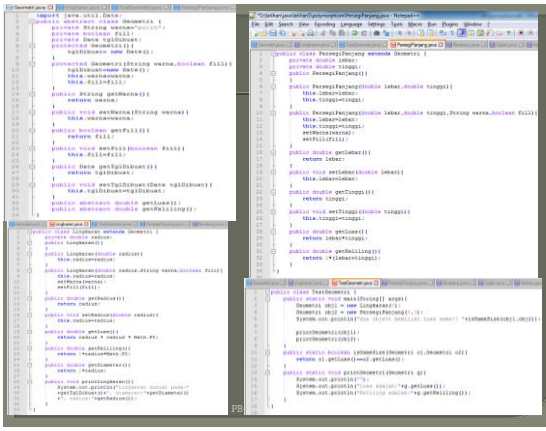
```
public abstract class Binatang {
    abstract void makan();
    abstract void tidur();
}
```

```
public abstract class Kerbau extends Binatang {

    void tidur(){
        System.out.println("Kerbau tidur...");
    }
}
```

```
public class TestBinatang {
    public static void main(String[] args){
        Binatang gajah = new Gajah();
        //Binatang kerbau = new Kerbau();
        test(gajah);
        //test(kerbau);
    }
    public static void test(Binatang b){
        b.makan();
        b.tidur();
    }
}
```

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



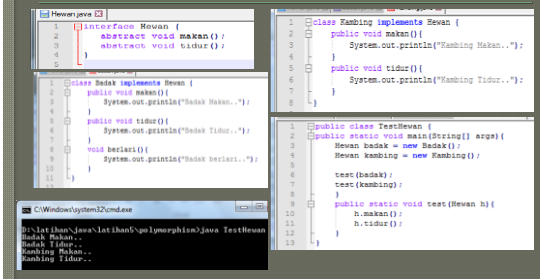
Manfaat Abstract Class

- 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.

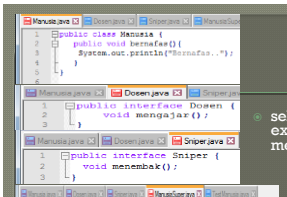
Interface



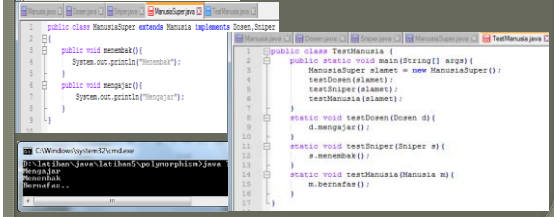
Interface

- 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.

Extends + implements

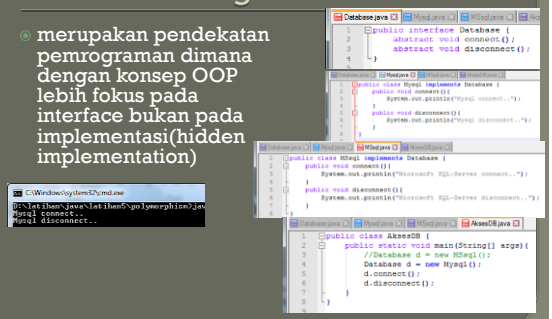


- sebuah class hanya bisa meng-extends satu class saja, namun bisa meng-implement banyak interface.



Program to Interface

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



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.

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 *method*nya.