Pewarisan Sifat Objek

Nur Hasanah, M.Cs

Membuat Kelas Turunan (*Subclass*)

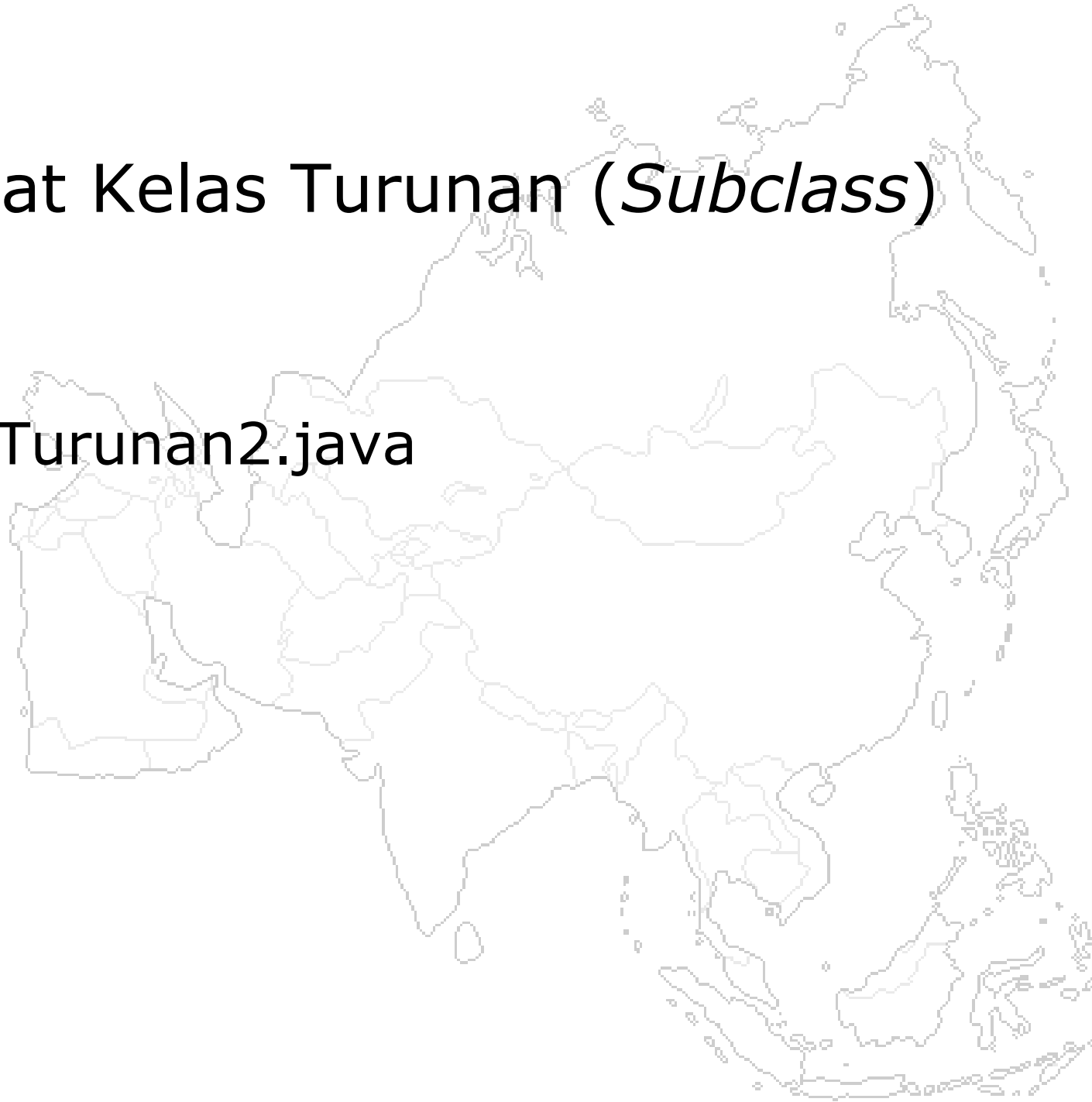
- Java menyediakan kata kunci **`extends`** yang digunakan untuk penurunan terhadap kelas.
- Dalam terminologi Java, kelas induk yang diturunkan disebut *superclass*, sedangkan kelas baru hasil turunan disebut *subclass*.
- Bentuk umumnya:

```
Class nama-subclass extends nama-superclass{
//badan kelas
}
```

Membuat Kelas Turunan (*Subclass*)

Contoh :

DemoKelasTurunan2.java



Tingkat Akses **`Protected`**

Melalui tingkat akses **`protected`**, data dapat diakses oleh semua kelas yang memiliki hubungan turunan, tapi lingkungan luar tetap tidak diberi hak untuk mengakses data tersebut.

Tingkat Akses **`Protected`**

Contoh:

- Demoprotected1.java
- Demoprotected2.java

Menggunakan kata kunci **`super`**

- Constructor yang terdapat pada kelas induk dapat dipanggil dari kelas turunan menggunakan kata kunci **`super`**
- Bentuk umum pemanggilannya:

  `super(daftar-parameter)`
- ***`Daftar-parameter`*** = daftar parameter yang didefinisikan constructor kelas induk
- Dalam constructor kelas turunan, **`super()`** harus ditempatkan pada bagian awal (baris pertama).

Menggunakan `super` untuk Memanggil Constructor Kelas Induk

- Contoh: DemoSuper1.java

```
class Kotak {
  protected double panjang;
  protected double lebar;
  protected double tinggi;

  Kotak() {
    panjang = lebar = tinggi = 0;
  }

  Kotak(int p, int l, int t) {
    panjang = p;
    lebar = l;
    tinggi = t;
  }

  public double hitungVolume() {
    return (panjang * lebar *
     tinggi);
  }
}
```

```
class KotakPejal extends Kotak {
  private double berat;

  KotakPejal(int p, int l, int t, int b) {

    super(p, l, t);
    berat = b;
  }

  public double getBerat() {
    return berat;
  }
}

class DemoSuper1 {
  public static void main(String[] args) {

    KotakPejal k = new KotakPejal(6, 5, 4, 2);
    System.out.println("Volume k : " +
     k.hitungVolume());
    System.out.println("Berat k  : " +
     k.getBerat());
  }
}
```

DemoSuper1.java

Menggunakan `super` untuk Mengakses Anggota Kelas Induk

- Contoh: DemoSuper3.java

```
class A {
  protected int a;
}

class B extends A {
  private int a;                // akan menimpa a yang ada dalam kelas A

 B(int nilai1, int nilai2) {
    super.a = nilai1;                    // a di dalam kelas A
    a = nilai2;                          // a di dalam kelas B
  }

  public void tampilkanNilai() {
    System.out.println("Nilai a di dalam kelas A : " + super.a);
    System.out.println("Nilai a di dalam kelas B : " + a);
  }
}

class DemoSuper3 {
  public static void main(String[] args) {
    B obj = new B(121, 212);
    obj.tampilkanNilai();
  }
}
```

DemoSuper3.java

Melakukan *Override* terhadap Method

- Saat kita mendefinisikan suatu method di kelas turunan yang nama dan daftar parameternya sama persis seperti yang terdapat di kelas induk, maka dikatakan kita telah melakukan override (mengesampingkan) method yang tersimpan di kelas induk.
- Apabila kita memanggil method yang telah di-override melalui objek kelas turunan, maka yang kakan dieksekusi adalah method yang terdapat pada kelas turunan.

Melakukan *Override* terhadap Method

Contoh:

- DemoOverride1.java
- DemoOverride2.java

```
class A {
  private int a;

  public void setA(int nilai) {
    a = nilai;
  }

  public int getA() {
    return a;
  }

  public void tampilkanNilai() {
    System.out.println("Nilai a: " + getA());
  }
}

class B extends A {
  private int b;

  public void setB(int nilai) {
    b = nilai;
  }

  public int getB() {
    return b;
  }

  public void tampilkanNilai() {
    System.out.println("Nilai b: " + getB());
  }
}

class DemoOverride1 {
  public static void main(String[] args) {

    B obj = new B();

    obj.setA(100);
    obj.setB(200);
    obj.tampilkanNilai();
  }
}
```

DemoOverride1.java

```
class A {
  private int a;

  public void setA(int nilai) {
    a = nilai;
  }

  public int getA() {
    return a;
  }

  public void tampilkanNilai() {
    System.out.println("Nilai a: " + getA());
  }
}

class B extends A {
  private int b;

  public void setB(int nilai) {
    b = nilai;
  }

  public int getB() {
    return b;
  }

  public void tampilkanNilai() {
    super.tampilkanNilai();
    System.out.println("Nilai b: " + getB());
  }
}

class DemoOverride2 {
  public static void main(String[] args) {

    B obj = new B();

    obj.setA(100);
    obj.setB(200);
    obj.tampilkanNilai();
  }
}
```

DemoOverride2.java

Perbedaan *Override* dengan *Overload*

- Override : mendefinisikan suatu method dalam kelas turunan yang memiliki nama dan daftar parameter sama persis dengan yang terdapat pada kelas induk.
- Overload : mendefinisikan suatu method di dalam kelas turunan yang namanya sama dengan method yang terdapat pada kelas induknya, **tapi daftar parameternnya berbeda**.
- Contoh : DemoOverloadTurunan.java

```
class Induk {
  public void test() {
    System.out.println("Method di dalam kelas Induk");
  }
}

class Turunan extends Induk {
  public void test(String s) {
    System.out.println("Method di dalam kelas Turunan");
    System.out.println("s : \"" + s + "\"");
  }
}

class DemoOverloadTurunan {
  public static void main(String[] args) {

    Turunan obj = new Turunan();
    obj.test();
    System.out.println();
    obj.test("Contoh overload pada proses pewarisan");
  }
}
```

DemoOverloadTurunan.java

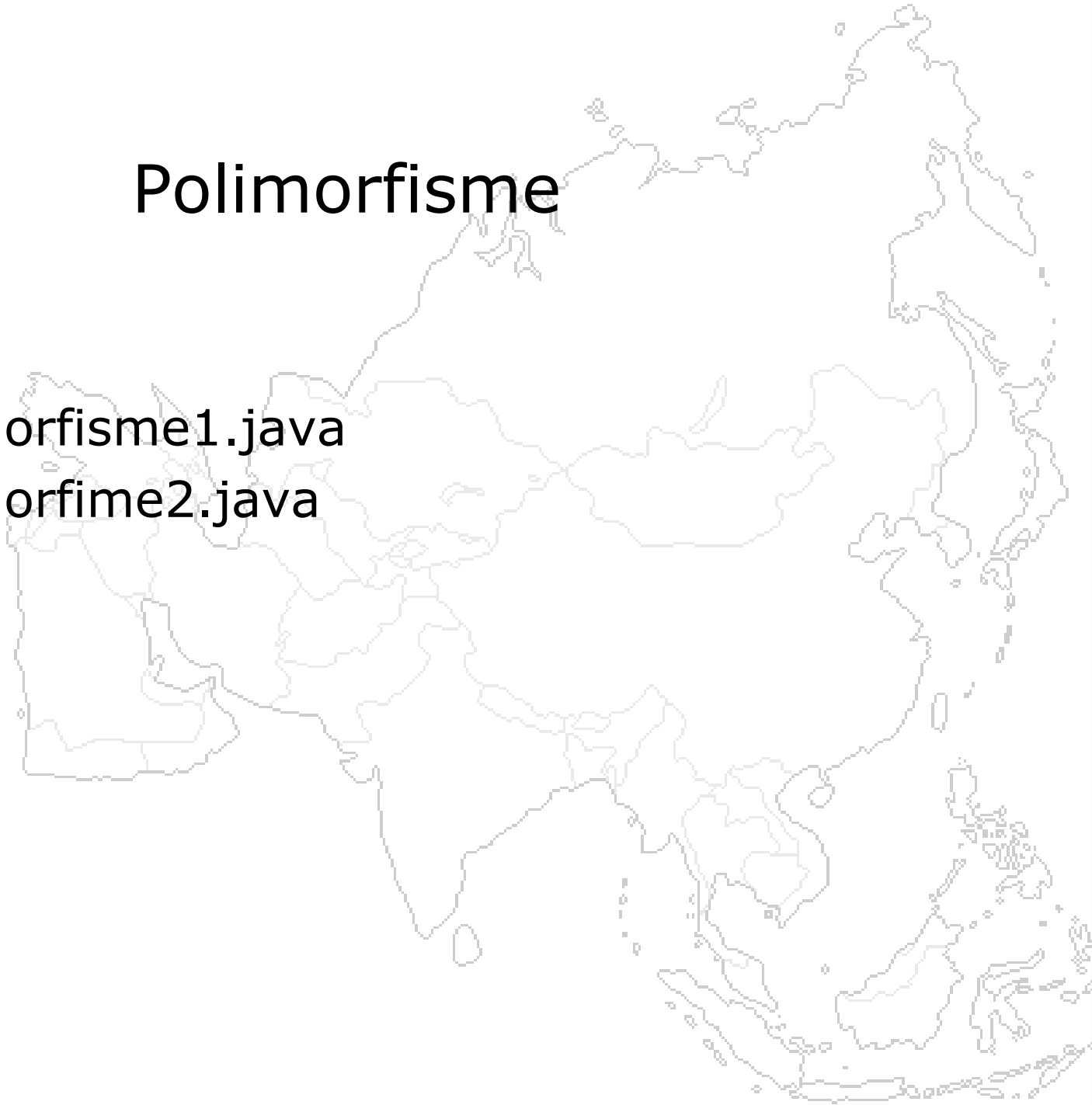
Polimorfisme

- Polimorfisme mengizinkan kelas induk untuk mendefinisikan sebuah method *general* (bersifat umum) untuk semua kelas turunannya, dan selanjutnya kelas-kelas turunan dapat memperbarui implementasi dari method tersebut secara lebih spesifik sesuai dengan karakteristik masing-masing.
- Proses override dibentuk agar Java dapat mendukung konsep polimorfisme.

Polimorfisme

Contoh:

- DemoPolimorfisme1.java
- DemoPolimorfime2.java



Kelas Abstrak

- Pada kasus tertentu, kita ingin mendeklarasikan sebuah kelas induk yang mempunyai method dimana method tersebut tidak memerlukan implementasi sama sekali.
- Method tersebut baru akan diimplementasi oleh kelas-kelas turunannya. Method ini yang disebut **method abstrak**.
- Method abstrak tidak dapat dideklarasikan dengan tingkat *private,* karena harus diimplementasi oleh kelas turunan.
- Bentuk umum penulisan kodenya:

```
abtract tipe namaMethod(daftar-parameter);
```

Kelas Abstrak

- Setiap kelas yang didalamnya terdapat satu atau lebih method abstrak harus dideklarasikan juga sebagai kelas abstrak.
- Bentuk umum penulisan kodenya:

```
abtract class namaKelas{
//badan kelas
}
```

Kelas Abstrak

- Kelas abstrak tidak dapat diinstantiasi (kita tidak diizinkan untuk membentuk objek dari suatu kelas abstrak).
- Meskipun demikian, kita diperbolehkan untuk mendeklarasikan sebuah variabel referensi ke kelas abstrak. Selanjutnya, variabel referensi tersebut dapat digunakan untuk mengacu ke objek-objek dari kelas turunan.
- Contoh : DemoKelasAbstrak1.java

```
abstract class A {

  abstract public void coba();

  public void cobajuga() {
    System.out.println("Method non-abstrak
     " + "di dalam kelas abstrak");
  }
}

class B extends A {

  public void coba() {
    System.out.println("Method di dalam
     kelas B");
  }
}
```

```
class DemoKelasAbstrak1 {

  public static void main(String[] args) {

    A ref;

    B obj = new B();

    ref = obj; // ref menunjuk ke objek dari
     kelas B

    ref.coba();      // memanggil method
     coba() di dalam B
    ref.cobajuga(); // memanggil method
     cobajuga() di dalam A
  }
}
```

DemoKelasAbstrak1.java

Reference

- Budi Rahardjo dkk. (2012). "*Mudah Belajar Java*". Penerbit Informatika Bandung.