6️⃣

Modul 6 - Inheritance dan Polimorfisme

Durasi Modul = 2-5 menit

Capaian pembelajaran:

Setelah mempelajari modul ini, mahasiswa diharapkan mampu:

- Memahami konsep dasar inheritance (pewarisan) dalam pemrograman berorientasi objek
- Menjelaskan hubungan antara superclass dan subclass
- Mengimplementasikan inheritance dalam kode Java
- Memahami konsep polimorfisme dan perbedaannya dengan inheritance
- Menjelaskan dan mengimplementasikan runtime polimorfisme (method overriding)
- Menjelaskan dan mengimplementasikan compile-time polimorfisme (method overloading)
- Menganalisis penggunaan inheritance dan polimorfisme dalam desain program

Deskripsi Pembelajaran

Modul ini akan memperkenalkan dua konsep penting dalam pemrograman berorientasi objek: inheritance (pewarisan) dan polimorfisme. Mahasiswa akan mempelajari:

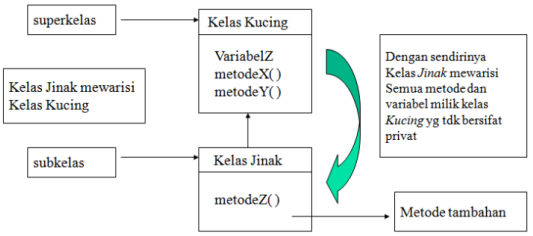
- Konsep dasar inheritance, termasuk hubungan antara superclass dan subclass
- Implementasi inheritance dalam bahasa pemrograman Java
- Konsep polimorfisme dan perbedaannya dengan inheritance

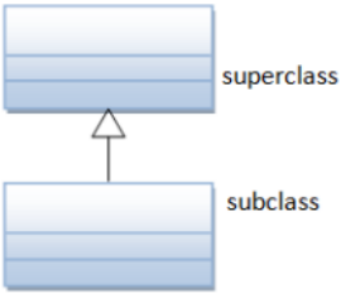
- Runtime polimorfisme (method overriding) dan compile-time polimorfisme (method overloading)
- Penggunaan inheritance dan polimorfisme dalam desain program

Melalui penjelasan konsep, contoh kode, dan latihan praktis, mahasiswa akan memahami bagaimana inheritance dan polimorfisme dapat digunakan untuk membuat kode yang lebih efisien, fleksibel, dan mudah dipelihara dalam pengembangan perangkat lunak.

6.1. *Inheritence* (Pewarisan)

Modifier	class yang sama	package yang sama	subclass package lain	class manapun
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√





📌

1. ***Inheritenc****e a*dalah **pewarisan sifat sebuah class kepada class yang lain**. Artinya sebuah class dapat mewariskan atribut dan methodnya kepada class lain.
2. **Hubungan yang terbangun** antar class dalam **konsep inheritance** adalah “IS-A”.
3. ***Term* dan *Keyword*** yang harus diingat dalam inheritance:
   a.
   Superclass/class parent: Class yang mewariskan atribut dan method
   b.
   ***Subclass/Class anak*:** Class yang dapat menggunakan atribut dan method dari

   superclass-nya
   c.
   “extends”: Jika ingin membuat sebuah subclass
   d.
   “final”: Jika ingin membuat sebuah class “mandul” ==tidak dapat memiliki subclass
   e.
   “@override”: jika **subclass** ingin **membuat method** dengan **nama yang sama** dengan

   method yang ada pada **superclass**
   f.
   “super”: jika **subclass** masih **ingin menjalankan method** yang ada pada **superclass**

   setelah memakai **keyword “@override”**
   g.
   Modifier “protected”: **Untuk mempertegas level akses** bahwa **class** tersebut **hanya**
   dapat diakses oleh class
   pada **package yang sama dan subclass**

6.1.1. Kata Kunci Super

Kata kunci super dipakai untuk merujuk pada ***parent class atau superclass*** dari class yang menggunakannya. Kata kunci super dapat digunakan untuk memanggil *constructor* dan *method*
milik
superclass. Format penulisannya adalah sebagai berikut:

> ***super.data_member***

> ***super.function_member()***

> ***super()***

a. **Pemanggilan *Constructor Superclass***
Constructor digunakan untuk membuat instance dari sebuah class. Berbeda dari data field (variable) dan method, constructor superclass tidak diwariskan ke subclass. Sehingga untuk dapat memanggil constructor superclass dari constructor subclass, maka digunakan kata kunci (keyword) super. Perintah yang digunakan untuk memanggil constructorsuperclass:

> ***super () atau super (parameter)***

Perintah ***super()*** digunakan untuk memanggil ***constructor default superclass*****,** sedangkan pernyataan ***super(parameter)*** digunakan untuk memanggil ***constructor berparameter*** milik superclass yang sesuai urutan parameternya. Kedua pernyataan tersebut harus muncul dibaris pertama ***constructor subclass***. **Jika keyword super tidak dituliskan secara eksplisit, maka constructor default milik *superclass* otomatis akan dipanggil secara implisit.**

Contoh :

```
public class ClassA  {
   ClassA()
    {
     System.out.println ("Ini Constructor Class
A);
     }
}
```

```
public class ClassB extends ClassA {
   ClassB()
    {
     super();
     System.out.println ("Ini Constructor Class
B);
     }
}
```

b. Pemanggilan *Method Superclass*

Selain untuk memanggil ***constructor superclass,*** kata kunci *(keyword)* ***super*** juga digunakan untuk memanggil **method milik *superclass*.** Perintah yang digunakan untuk memanggil *method superclass* seperti berikut:

> ***super.method(parameters);***

Contoh:
Dalam
Method getSalary () pada *class Manager,* terdapat kata kunci super untuk memanggil *method getSalary* yang terdapat pada *class Employee.*

```
public double getSalary() {
        double baseSalary = super.getSalary();
        return baseSalary + bonus;
    }
```

📌 Suatu **Subclass** juga dapat memanggil **constructor** yang terdapat di **superclass** menggunakan **kata kunci "super". Kata kunci super** juga dapat digunakan **untuk merujuk pada anggota dari superclass**, hal ini sama dengan penggunaan **referensi "this"**

Class → SuperClass

Class → SubClass

```java
class Animal {
   protected Animal() {
     System.out.println("I am an Animal");
   }

   protected void eat() {
     System.out.println("I can eat");
   }
}
```

```java
class Dog extends Animal {
  protected Dog(){
     super();
     System.out.println("I am a dog");
  }

  @Override
  protected void eat() {
     super.eat();
     System.out.println("I eat dog food");
  }

   protected void bark() {
     System.out.println("I can bark");
   }
}
```

```java
class Main {
   public static void main(String[] args) {
      Dog dog1 = new Dog();

      dog1.eat();
      dog1.bark();
   }
}
```

6.2. *Polimorfisme*

> ***Polimorfisme*** artinya **"banyak bentuk"**. Ada **2 jenis polimorfisme**, yaitu *run-time polimorfisme* (polimorfisme yang dicapai melalui ***method overriding***) dan *compile-time polimorfisme* (polimorfisme yang melalui ***method overloading***)

📌

1. ***Polimorfisme*** bisa diartikan satu bentuk banyak aksi, sekilas mirip dengan ***inheritance*** tetapi **dalam polimorfisme** kita akan memerintah sebuah objek untuk melakukan **tindakan yang secara prinsip sama** tapi **secara proses** dan **outputnya berbeda.**
2. ***Polimorfisme*** mengizinkan **kelas induk** untuk **mendefinisikan sebuah method general** (bersifat umum) untuk **semua kelas turunannya**, dan selanjutnya kelas-kelas turunan dapat memperbaharui implementasi dari method tersebut secara lebih spesifik sesuai dengan karakteristiknya masing-masing.

> Contoh 1 ***(RUNTIME POLIMORFISME) / Overriding:***

Overriding method adalah **suatu teknik** dimana didalam suatu class terdapat **nama yang sama** dan **parameter yang sama** (antara subclass dan superclass) namun **menjalankan perintah dan fungsi yang berbeda. *Overriding*** juga sering dinyatakan suatu keadaan dimana **method pada subclass menolak method pada parent class (superclass)-nya.**

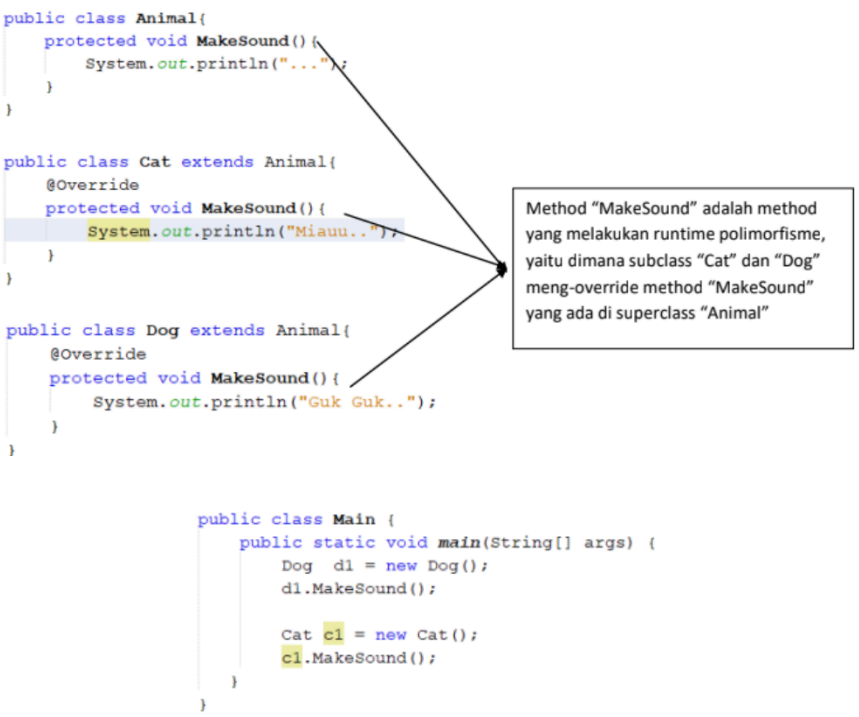
Dengan kata lain subclass yang berusaha memodifikasi tingkah laku yang diwarisi dari superclass. Overriding dilakukan **bertujuan** agar subclass memiliki tingkah laku yang spesifik. *Overridding* dilakukan dengan cara mendeklarasikan kembali di *subclass method* yang dimiliki oleh superclass. Method pada superclass (parent class) disebut dengan overridden method, sedangkan method pada subclass disebut overriding method.

💡 **Ciri-ciri dari overriding** adalah sebagai berikut:

- **Nama method pada subclass harus sama dengan nama method pada superclass.**
- **Daftar parameter yang dimiliki oleh subclass harus sama dengan daftar parameter pada superclass.**
- **Return type pada subclass harus sama dengan return type pada superclass.**

📎 **Aturan yang harus diperhatikan** ketika melakukan overriding seperti berikut.

- Mode akses overriding method harus sama atau lebih luas dari pada overriden method.
- *Subclass* hanya boleh meng-*override method superclass* satu kali saja.
- *Overriding method* tidak boleh *throw checked exceptions* yang tidak dideklarasikan oleh *overridden method* (dijelaskan pada materi *Exception).*




```
public class Main {
    public static void main(String[] args) {
        Dog  d1 = new Dog();
        d1.MakeSound();

        Cat c1 = new Cat();
        c1.MakeSound();
    }
}
```

Contoh 2 ***(COMPILE-TIME POLIMORFISME) / Overloading :***

Overloading method adalah **suatu teknik** dimana didalam suatu class terdapat **method** dengan **nama yang sama namun** prilaku yang berbeda.

Overloading adalah suatu keadaan dimana beberapa method memiliki nama yang sama tetapi fungsionalitasnya berbeda. Tujuan dilakukannya overloading adalah untuk memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip. Overloading dapat terjadi antara parent class dengan subclass-nya jika memenuhi ketiga syarat/ ciri overload

📌 Ciri-ciri dari overloadingadalah sebagai berikut:

- Nama method pada subclass harus sama dengannnama method pada superclass.
- Daftar parameter yang dimiliki oleh subclass harus berbeda dengan daftar parameter pada superclass.
- Return type pada subclass boleh sama, boleh berbeda dengan return type pada superclass.

> Seperti halnya method, costructor juga dapat dioverloading. Aturan/ciri-ciri ***overloading constructor,*** sama dengan ***Overloading method.***

📌 **Aturan penulisannya :**

1. Nama method harus sama
2. Parameter harus berbeda
3. Tipe Kembalian boleh sama ataupun berbeda

```
class Demo {
 public void displayPattern(){
   for(int i = 0; i < 10; i++) {
     System.out.print("*");
   }
 }

 public void displayPattern(char symbol) {
   for(int i = 0; i < 10; i++) {
     System.out.print(symbol);
   }
 }
}
```

Method “DisplayPattern” adalah method yang melakukan compile-time polimorfisme, yaitu dimana dalam 1 class yang yang sama terdapat 2 method “DisplayPattern”, yang membedakan hanya jumlah dan type data parameternya saja.

```
class Main {
 public static void main(String[] args) {
   Demo d1 = new Demo();
   d1.displayPattern();
   System.out.println("\n");
   d1.displayPattern('#');
 }
}
```

📌 Secara singkat, perbedaan keduanya adalah:

1. ***Overload*** memiliki **nama method yang sama** namun **berbeda parameter** (perintah yang dijalankan bisa jadi sama maupun tidak antara satu dengan yang lainnya). Sedangkan ***override*** memiliki **nama yang sama** dan **parameter yang sama** (antara subclass dan superclass) namun **menjalankan perintah dan fungsi yang berbeda.**
2. ***Overload*** terjadi di **class yang sama**, namun ***override*** terjadi pada **kasus inheritance dimana terdiri dari dua kelas (atau lebih).**

📚REFERENSI

Aziefah, & Nurmasari, J. (2013). Pemrograman Berorientasi Objek. Pekanbaru: Politeknik Caltex Riau.

Barclay, K., & Savage, J. (2004). Object Oriented Design with UML and Java. Elsevier.

Deitel, P., & Deitel, H. (2012). Java: How to Program (9 ed.). US: Prentice Hall.