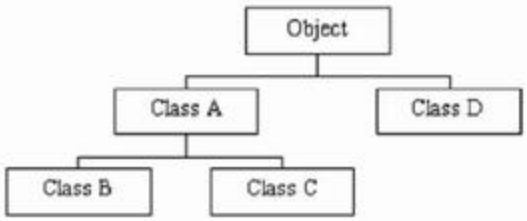
PENDEKATAN BERORIENTASI OBYEK

8.1. Pewarisan

Dalam Java, semua class, termasuk class yang membangun Java API, adalah subclasses dari superclass Object. Contoh hirarki class diperlihatkan pada gambar 5.1 dibawah ini. Beberapa class di atas class utama dalam hirarki class dikenal sebagai superclass. Sementara beberapa class di bawah class pokok dalam hirarki class dikenal sebagai subclass dari class tersebut.



Gambar 8.1. Hirarki Kelas dalam Java

Pewarisan adalah keuntungan besar dalam pemrograman berbasis obyek karena suatu sifat atau metode didefinisikan dalam superclass, sifat ini secara otomatis diwariskan dari semua subclasses. Jadi kita dapat menuliskan kode metode hanya sekali dan mereka dapat digunakan oleh semua subclass. Subclass hanya perlu mengimplementasikan perbedaannya sendiri dengan induknya.

8.1.1. Mendefinisikan Superclass dan Subclass

Untuk memperoleh suatu class, kita menggunakan kata kunci extend. Untuk mengilustrasikan ini, kita akan membuat contoh class induk. Dimisalkan kita mempunyai class induk yang dinamakan Person.

```
public class Person{
  protected String name;
  protected String address;

  /**
  * Default constructor
  */
  public Person(){
    System.out.println("Inside Person:Constructor");
    name = "";
    address = "";
  }

  /**
  * Constructor dengan dua parameter
  */
  public Person( String name, String address ){
    this.name = name;
    this.address = address;
  }
```

```
  /**
  * Method accessor
  */
  public String getName(){
    return name;
  }

  public String getAddress(){
    return address;
  }

  public void setName( String name ){
    this.name = name;
  }

  public void setAddress( String add ){
    this.address = add;
  }
}
```

Perhatikan bahwa atribut name dan address dideklarasikan sebagai protected. Alasannya kita melakukan ini yaitu, kita inginkan atribut-atribut ini untuk bisa diakses oleh subclasses dari superclassess. Jika kita mendeklarasikannya sebagai private, subclasses tidak dapat menggunakannya. (Catatan bahwa semua properti dari superclass yang dideklarasikan sebagai public, protected dan default dapat diakses oleh subclasses-nya).

Sekarang, kita ingin membuat class lain bernama Student. Karena Student juga sebagai Person, kita putuskan hanya meng-extend class Person, sehingga kita dapat mewariskan semua properti dan metode dari setiap class Person yang ada. Untuk melakukan ini, kita tulis kode program berikut ini :

```
public class Student extends Person{
  public Student(){
    System.out.println("Inside Student:Constructor");
    //beberapa kode di sini
  }

  // beberapa kode di sini
}
```

Ketika obyek Student di-instantiate, default constructor dari superclass secara mutlak meminta untuk melakukan inisialisasi yang seharusnya. Setelah itu, pernyataan di dalam subclass dieksekusi. Untuk mengilustrasikannya, ketik kode program dibawah ini didalam class Student.

```
public static void main( String[] args ){
  Student anna = new Student();
}
```

Dalam kode ini, kita membuat sebuah obyek dari class Student. Keluaran dari program adalah :

```
Inside Person:Constructor
Inside Student:Constructor
```

8.1.2. Kata Kunci Super

Subclass juga dapat memanggil constructor secara eksplisit dari superclass terdekat. Hal ini dilakukan dengan pemanggil constructor super. Pemanggil constructor super dalam constructor dari subclass akan menghasilkan eksekusi dari superclass constructor yang bersangkutan, berdasar dari argumen sebelumnya.

Sebagai contoh, pada contoh class sebelumnya. Person dan Student, kita tunjukkan contoh dari pemanggil constructor super. Diberikan kode berikut untuk Student :

```
public Student(){
  super( "SomeName", "SomeAddress" );
  System.out.println("Inside Student:Constructor");
}
```

Kode ini memanggil constructor kedua dari superclass terdekat (yaitu Person) dan mengeksekusinya. Contoh kode lain ditunjukkan sebagai berikut :

```
public Student(){
  super();
  System.out.println("Inside Student:Constructor");
}
```

Kode ini memanggil default constructor dari superclass terdekat (yaitu Person) dan mengeksekusinya.

Ada beberapa hal yang harus diingat ketika menggunakan pemanggil constuktor super:

a. Pemanggil super() harus dijadikan pernyataan pertama dalam constructor.
b. Pemanggil super() hanya dapat digunakan dalam definisi constructor.
c. Termasuk constructor this() dan pemanggil super() tidak boleh terjadi dalam constructor yang sama.

Pemakaian lain dari super adalah untuk menunjuk anggota dari superclass (seperti reference this). Sebagai contoh :

```
public Student(){
  super.name = "somename";
  super.address = "some address";
}
```

8.1.3. Overriding Method

Untuk beberapa pertimbangan, terkadang class asal perlu mempunyai implementasi berbeda dari metode yang khusus dari superclass tersebut. Oleh karena itulah, metode overriding digunakan. Subclass dapat mengesampingkan metode yang didefinisikan dalam superclass dengan menyediakan implementasi baru dari metode tersebut.

Misalnya kita mempunyai implementasi berikut untuk metode getName dalam superclass Person :

```
public class Person{
  :
  :
  public String getName(){
    System.out.println("Parent: getName");
    return name;
  }
  :
}
```

Untuk override, metode getName dalam subclass Student, kita tulis :

```
public class Student extends Person{
  :
  :
  public String getName(){
    System.out.println("Student: getName");
    return name;
  }
  :
}
```

Jadi, ketika kita meminta metode getName dari obyek class Student, metode overridde akan dipanggil, keluarannya akan menjadi :

```
Student: getName
```

8.1.4. Method final dan class final

Dalam Java, juga memungkinkan untuk mendeklarasikan class-class yang tidak lama menjadi subclass. Class ini dinamakan class final. Untuk mendeklarasikan class untuk menjadi final kita hanya menambahkan kata kunci final dalam deklarasi class. Sebagai contoh, jika kita ingin class Person untuk dideklarasikan final, kita tulis :

```
public final class Person {
  //area kode
}
```

Beberapa class dalam Java API dideklarasikan secara final untuk memastikan sifatnya tidak dapat di-override. Contoh-contoh dari class ini adalah Integer, Double, dan String.

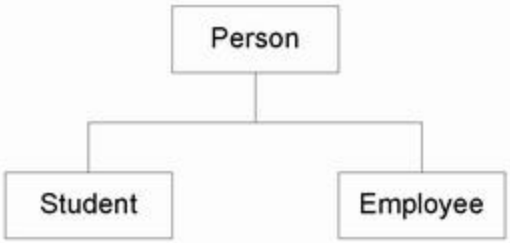
Ini memungkinkan dalam Java membuat metode yang tidak dapat di-override. Metode ini dapat kita panggil method final. Untuk mendeklarasikan metode untuk menjadi final, kita tambahkan kata kunci final ke dalam deklarasi metode. Contohnya, jika kita ingin metode getName dalam class Person untuk dideklarasikan final, kita tulis :

```
public final String getName(){
  return name;
}
```

Metode static juga secara otomatis final. Ini artinya Anda tidak dapat membuatnya override.

8.2. Polimorfisme

Sekarang, class induk Person dan subclass Student dari contoh sebelumnya, kita tambahkan subclass lain dari Person yaitu Employee. Di bawah ini adalah hierarkinya :



Gambar 8.3. Hirarki dari class induk Person

Dalam Java, kita dapat membuat referensi yang merupakan tipe dari superclass ke sebuah obyek dari subclass tersebut. Sebagai contohnya :

```
public static main( String[] args ){
  Person ref;

  Student studentObject = new Student();
  Employee employeeObject = new Employee();
  ref = studentObject; //Person menunjuk kepada object Student

  //beberapa kode di sini
}
```

Sekarang dimisalkan kita punya metode getName dalam superclass Person kita, dan kita override metode ini dalam kedua subclasses Student dan Employee:

```
public class Person{
  public String getName(){
    System.out.println("Person Name: " + name);
    return name;
  }
}
```

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

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

Kembali ke metode utama kita, ketika kita mencoba memanggil metode getName dari reference Person ref, metode getName dari obyek Student akan dipanggil. Sekarang, jika kita berikan ref ke obyek Employee, metode getName dari Employee akan dipanggil.

```
public static main( String[] args ){
  Person ref;

  Student studentObject = new Student();
  Employee employeeObject = new Employee();

  ref = studentObject; //Person menunjuk kepada object Student
  String temp = ref.getName(); //getName dari Student class dipanggil
  System.out.println( temp );

  ref = employeeObject; //Person menunjuk kepada object Employee
  String temp = ref.getName(); //getName dari Employee class dipanggil
  System.out.println( temp );
}
```

Kemampuan dari reference untuk mengubah sifat menurut obyek apa yang dijadikan acuan dinamakan polimorfisme. Polimorfisme menyediakan multiobject dari subclasses yang berbeda untuk diperlakukan sebagai obyek dari superclass tunggal, secara otomatis menunjuk metode yang tepat untuk menggunakannya ke particular obyek berdasar subclass yang termasuk di dalamnya.

Contoh lain yang menunjukkan properti polimorfisme adalah ketika kita mencoba melalui reference ke metode. Misalkan kita punya metode static printInformation yang mengakibatkan obyek Person sebagai reference, kita dapat me-reference dari tipe Employee dan tipe Student ke metode ini selama itu masih subclass dari class Person.

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

public static printInformation( Person p ){
  //beberapa kode di sini
}
```