Pemrograman Berorientasi Obyek

Pertemuan 8: Abstract Class dan Interface pada Java

Tambahan Materi Java

- Class Abstract
- Interface

Class Abstract

- Sebuah class yang didefinisikan `abstract` berarti class tersebut tidak dapat dibuat instansiasi objeknya
- Class abstract dibuat dengan tujuan menjadi *base class* dari class turunannya
- Cara deklarasi class abstract:

```
public abstract class MyAbstractClass {

}
```

- Kita tidak dapat membuat objek dari `MyAbstractClass`

```
MyAbstractClass myClassInstance =
    new MyAbstractClass();  //not valid
```

Method Abstract

- Sebuah class abstract dapat memiliki method abstract
- Tambahkan kata kunci `abstract` di depan deklarasi method untuk membuat sebuah method abstract

```
public abstract class MyAbstractClass {

    public abstract void abstractMethod();
}
```

- Method yang dideklarasikan abstract tidak memiliki implementasi (hanya method kosong)
- Class yang memiliki method abstract, harus dideklarasikan abstract. Tetapi class yang dideklarasikan abstract, tidak harus memiliki method abstract (dengan kata lain, boleh memiliki method yang tidak abstract)
- Subclass dari class abstract harus mengimplementasikan (meng-override) semua method abstract dari superclass
- Method non-abstract dari superclass diwarisi apa adanya, namun dapat pula diimplementasikan ulang jika diperlukan

Contoh

- Abstract class:

```
public abstract class MyAbstractClass {

    public abstract void abstractMethod();
}
```

- Subclass dari abstract class:

```
public class MySubClass extends MyAbstractClass {

    public void abstractMethod() {
        System.out.println("My method implementation");
    }
}
```

Mengapa harus Abstract Class?

- Abstract class berfungsi sebagai base class yang dapat diturunkan kepada subclass yang akan mengimplementasikan base class
- Misal sebuah proses memerlukan 3 tahap:
  1. Tahap pre-proses
  2. Tahap proses utama
  3. Tahap post-proses
- Jika tahap 1 dan 3 selalu sama untuk semua objek, maka ini dapat diimplementasikan dengan menggunakan class abstract sebagai berikut:

Contoh Abstract Class Sederhana

```
public abstract class MyAbstractProcess {

    public void process() {
        stepBefore();
        action();
        stepAfter();
    }

    public void stepBefore() {
        //implementation directly in abstract superclass
    }

    public abstract void action(); // implemented by subclasses

    public void stepAfter() {
        //implementation directly in abstract superclass
    }
}
```

Subclass dari `MyAbstractProcess` hanya perlu mengimplementasikan method `action()`. Method `stepBefore()` dan `stepAfter()` tinggal mewarisi dari superclass

Sekali lagi, mengapa harus Abstract Class?

- Contoh sebelumnya tetap dapat diimplementasikan dengan class biasa. Mengapa harus dibuat abstract?
- Dengan mendefinisikan sebuah class sebagai abstract, kita memberitahu pengguna code kita bahwa class tersebut ***tidak seharusnya digunakan apa adanya***
- Class tersebut hanya dipergunakan sebagai base class dari sebuah subclass, dan method abstract-nya harus diimplementasikan di dalam subclass
- Sebuah class dapat dideklarasikan abstract walaupun ia tidak memiliki method abstract sama sekali

Contoh Konkrit Class Abstract

- Contoh berikut adalah base class yang membuka sebuah URL, memprosesnya, dan menutup koneksi URL setelah selesai

```
public abstract class URLProcessorBase {

    public void process(URL url) throws IOException {
        URLConnection urlConnection = url.openConnection();
        InputStream input = urlConnection.getInputStream();

        try{
            processURLData(input);
        } finally {
            input.close();
        }
    }

    protected abstract void processURLData(InputStream input)
        throws IOException;

}
```

Contoh Konkrit Class Abstract

- Berikut contoh subclass dari class `URLProcessorBase`:

```
public class URLProcessorImpl extends URLProcessorBase {

    @Override
    protected void processURLData(InputStream input) throws IOException {
        int data = input.read();
        while(data != -1){
            System.out.println((char) data);
            data = input.read();
        }
    }
}
```

- Subclass hanya mengimplementasikan method `processURLData`
- Code lainnya langsung mewarisi dari superclass
- Berikut contoh penggunaan subclass `URLProcessorImpl`:

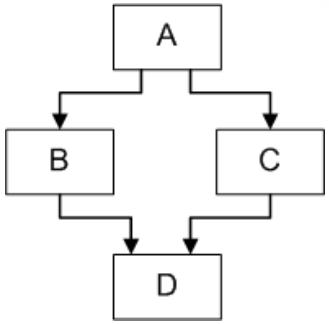
```
URLProcessorImpl urlProcessor = new URLProcessorImpl();

urlProcessor.process(new URL("http://jenkov.com"));
```

Contoh Abstract Class pada Java

- Lihat JavaDoc

JAVA INTERFACE

Review Hybrid Inheritance



- Jika bahasa pemrograman tidak mendukung *multiple inheritance*, maka dia juga tidak mendukung *hybrid inheritance*
- Hanya C++ saja yang mendukung *hybrid inheritance*. Ruby dan C# tidak
- Java dapat mendukung *hybrid inheritance* tidak dengan menggunakan *class*, tetapi *interface*

Interface

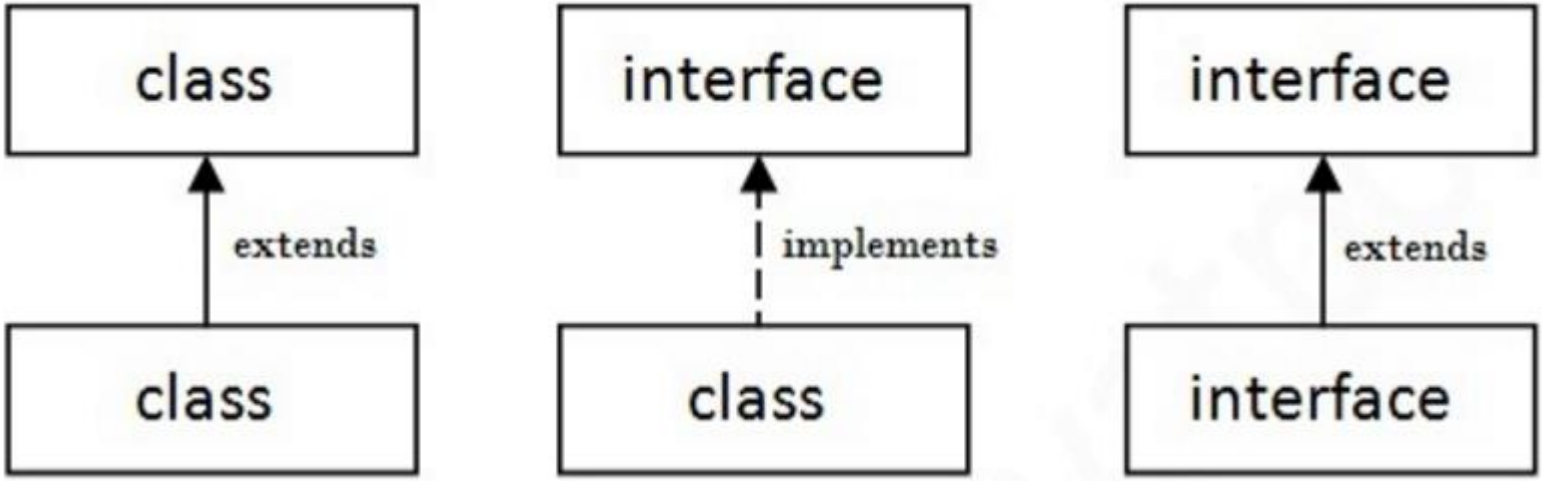
- Java Interface mirip dengan class, tetapi interface hanya memiliki method *signature* dan atribut
- Method *signature* adalah method tanpa implementasi yang hanya berisi nama, parameter, dan *exception*
- Java interface dapat digunakan sebagai sarana untuk melakukan *polymorphism*
- Contoh interface sederhana:

```
public interface MyInterface {

    public String hello = "Hello";

    public void sayHello();
}
```

- Interface `MyInterface` memiliki 1 variabel atribut (`hello`) dan 1 method *signature* (`sayHello()`)
- Kita dapat mengakses variabel atribut dengan memanggil langsung nama interface-nya, diikuti dengan nama variabelnya. Contoh:

```
System.out.println(MyInterface.hello);
```

Interface vs Class



Multiple Inheritance dari Interface

Multiple Inheritance in Java

Contoh Interface Bawaan Java

- Interface `Comparable`, memiliki method `compareTo`, yang digunakan untuk membandingkan dua buah objek
- Interface `Serializable`, interface marker tanpa method dan atribut apapun, yang digunakan untuk mengindikasikan bahwa sebuah class dapat diserialisasi

Serialisasi: mengkonversi objek menjadi stream biner yang kemudian dapat disimpan dalam database dengan tipe data blob (*binary large object*)

Mengimplementasikan Interface

- Sebelum interface digunakan, ia harus diimplementasikan dulu oleh sebuah class. Kita tidak dapat langsung membuat instance dari interface. Contoh:

```
public class MyInterfaceImpl implements MyInterface {

    public void sayHello() {
        System.out.println(MyInterface.hello);
    }
}
```

- Class yang mengimplementasi sebuah interface harus mengimplementasikan semua method yang dideklarasikan dalam interface tersebut, dengan nama dan parameter yang sama
- Setelah itu, kita dapat menggunakan instance dari class sebagai instance dari interface

```
MyInterface myInterface = new MyInterfaceImpl();

myInterface.sayHello();
```

Contoh: Implementasi Interface `Comparable`

```
package samples;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class SongComparable implements Comparable<SongComparable>{
    String name;
    String artist;
    int duration;
    public SongComparable(String n, String a, int d){
       this.name=n;
       this.artist=a;
       this.duration=d;
    }
    public SongComparable(){
       this.name=null;
       this.artist=null;
       this.duration=0;
    }
    public void to_s(){
       System.out.println("Song: "+this.name+"--"+this.artist+"("+this.duration+" seconds)");
    }
```

```
@Override
public int compareTo(SongComparable arg0) {
   // TODO Auto-generated method stub
   if (this.duration==((SongComparable)arg0).duration)
     return 0;
   else if (this.duration>((SongComparable)arg0).duration)
     return 1;
   else return -1;
}

public static void main (String[] args){
   List songList=new ArrayList();
   songList.add(new SongComparable("Hello","Adele",350));
   songList.add(new SongComparable("Hello","Lionel Richie",200));
   songList.add(new SongComparable("Roar","Katy Perry",362));

   Collections.sort(songList);
   Iterator itr=songList.iterator();
   while (itr.hasNext()){
     SongComparable obj=(SongComparable)itr.next();
     obj.to_s();
   }
}
}
```

Output:

```
Song: Hello--Lionel Richie(200 seconds)
Song: Hello--Adele(350 seconds)
Song: Roar--Katy Perry(362 seconds)
```

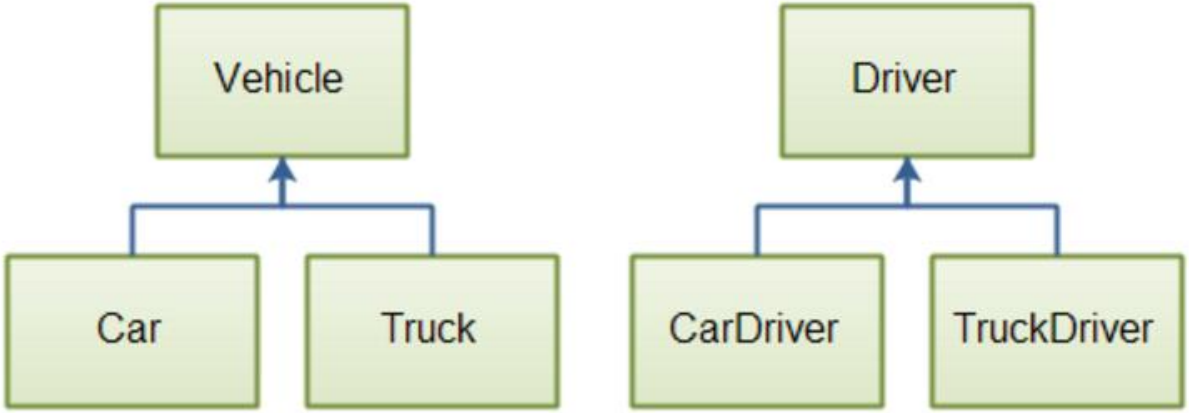
Mengimplementasikan Banyak Interface

- Sebuah class dapat mengimplementasikan lebih dari satu interface
- Dalam hal ini, class tersebut harus mengimplementasikan semua method yang ada di semua interface yang digunakan

```
public class MyInterfaceImpl
    implements MyInterface, MyOtherInterface {

    public void sayHello() {
        System.out.println("Hello");
    }

    public void sayGoodbye() {
        System.out.println("Goodbye");
    }
}
```

Interface dan Polymorphism

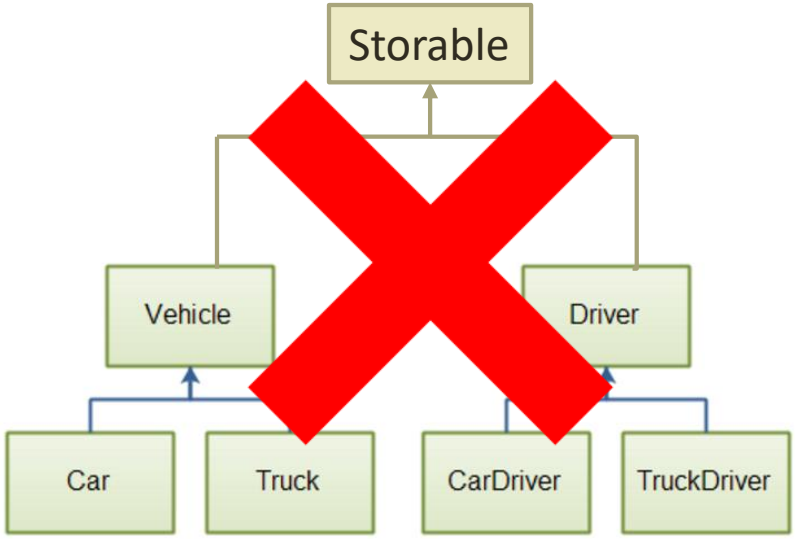
- Perhatikan class diagram berikut:



- Untuk menyimpan objek dari class-class tersebut ke database (atau ke file system, atau akan dikirim melalui jaringan), objek harus diserialisasi terlebih dahulu
- Sebaliknya, untuk membaca objek yang disimpan di database ke dalam program, maka objek harus dideserialisasi

Interface dan Polymorphism

- Misal proses serialisasi-deserialisasi kita masukkan ke dalam sebuah method `storeToDatabase()` dan diletakkan pada class `Storable`
- Agar dapat diakses oleh semua objek dari class `Vehicle` dan `Driver`, kita buat `Storable` menjadi superclass di atas `Vehicle` dan `Driver` sehingga method `storeToDatabase()` dapat digunakan oleh class di bawahnya
- Hal ini akan mengakibatkan hirarki class menjadi kacau secara konseptual karena diagram tidak lagi memodelkan class vehicle dan driver saja, tetapi juga terikat pada mekanisme penyimpanan dan proses serialisasi



Interface dan Polymorphism

- Salah satu solusinya adalah dengan membuat interface yang memiliki deklarasi method `storeToDatabase()`
- Class yang perlu untuk menyimpan objek ke database dapat mengimplementasikan interface tersebut dan mengimplementasikan method `storeToDatabase()`. Contoh:

```
public interface Storable {
    public void storeToDatabase();
}
```

- Lalu untuk menggunakannya, kita dapat melakukan seperti berikut:

```
public class Vehicle implements Storable{
    public void storeToDatabase(){
        //sejumlah langkah untuk penyimpanan ke
        //database
    public static void main()(String[] args){
        Vehicle v = new Vehicle();
        v.storeToDatabase();
    }
}
```

Summary

- Dengan menggunakan interface, penulisan program yang membutuhkan fungsionalitas tambahan menjadi lebih bersih dan rapi ketimbang menggunakan konsep pewarisan/inheritance