Class Abstract Interface

Topik

- Abstract Class
- Interface

Abstract

- Abstract class adalah class yang mempunyai setidaknya satu abstract method.
- Abstract method adalah method yang tidak memiliki body (hanya deklarasi method).

Abstract

- Abstract class tidak bisa dibuat obyeknya.
- Obyek hanya bisa dibuat dari non-abstract class (concrete class).
- Konsekuensinya, suatu abstract class haruslah diturunkan dimana pada subclass tersebut berisi implementasi dari abstract method yang ada di super class-nya.

Contoh Class Abstract

```
package Teori;

public abstract class Parent {
    public abstract void mAbstract();

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

Tidak bisa membuat object dari class abstract

Teori.Parent is abstract; cannot be instantiated

Parent p = new Parent();
1 error

Abstract

- Bila subclass yang diturunkan dari abstract class tidak mengimplementasikan isi semua method abstrak parent class, maka subclass tersebut harus tetap dideklarasikan abstract.
- Dan deklarasi method abstract pada subclass tersebut **boleh tidak** dituliskan kembali.

Contoh Class Abstract

```
package Teori;

public abstract class Parent {
    public abstract void mAbstract();
}

class Child extends Parent{
    public void mAbstract(){}
}
```

Class Parent mempunyai suclass yaitu Class Child. Class Child harus mengimplementasikan semua method abstract yang dimiliki class Parent.

Jika tidak mengimplementasikan atau hanya sebagian mengimplementasikan semua method abstract yang dimiliki class Parent maka class Child harus dideklarasikan abstract

```
package Teori;

public abstract class Parent {
    public abstract void mAbstract();
}

abstract class Child extends Parent{

}
```

Kegunaan Class Abstract

- Class Abstract berisi beberapa method non-abstract dan beberapa method abstract. Class Abstract berisi sebagian implementasi (method non-abstract) dan subclass yang melengkapi implementasinya. Dengan kata lain Class Abstract memiliki beberapa kesamaan (Bagian yang diimplementasikan oleh subclass) dan memiliki perbedaan (method yang dimiliki sendiri oleh class abstract)
- Deklarasikan method abstract, jika ada satu atau lebih subclass yang diharapkan mempunyai fungsionalitas yang sama tapi implementasi berbeda.

Kegunaan Class Abstract

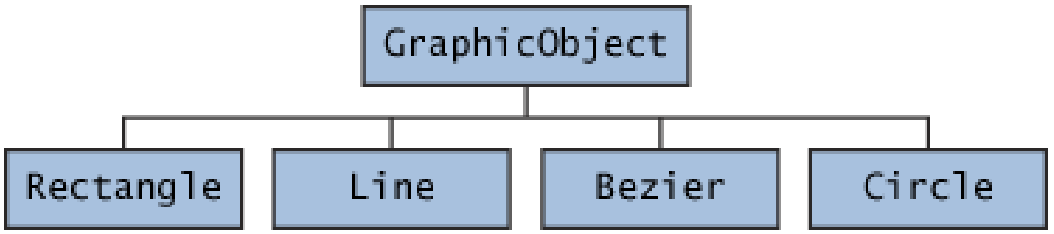
- Gunakan class abstract untuk mendefinisikan behavior secara umum sebagai superclass, sedangkan subclass menyediakan implementasi detail.
- Jika class abstract tersebut semua method merupakan method abstract, sebaiknya class abstract tersebut diganti menjadi Interface (dijelaskan selanjutnya)

Contoh Class Abstract

- Kita dapat menggambar lingkaran, persegipanjang garis, kurva Bezier dan object-object graphic lainnya. Object tersebut mempunyai state tertentu (seperti: position, orientation, line color, fill color) dan behaviour secara umum(contoh: moveTo, rotate, resize, draw). Beberapa state dan behavior ini sama untuk semua object graphic contoh: position, fill color, and moveTo. Implementasi yang berbeda sebagai contoh resize or draw. Semua Object Graphic harus mengetahui bagaimana cara draw dan resize

Contoh Class Abstract 1

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```




```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

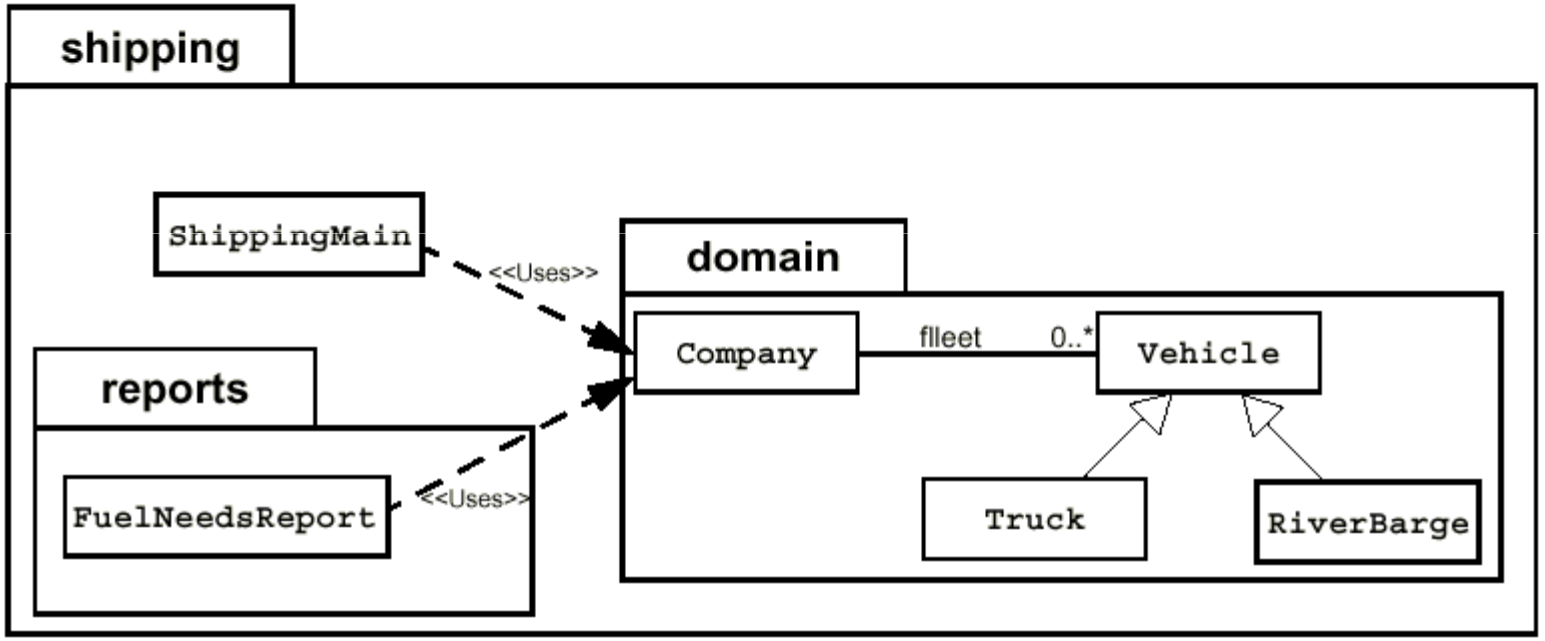
Contoh Class Abstract 2

```
abstract class Shape
{
  public abstract double area(int,int);

} // end Shape

public class Triangle extends Shape
{
      public double area(int b, int h)
      {
        return 0.5 * b * h;
      }

}// end Triangle.

public class Rectangle extends Shape
{
      public double area(int b, int h)
      {
        return  b * h;
      }
}// end Rectangle
```

```
import java.util.Random;

public class PolyAssign
{
    public static void main(String [] args)
    {
        Shape shp = null;

        Random r = new Random();
        int flip = r.nextInt(2);

        if (flip == 0)
              shp = new Triangle();
        else
              shp = new Rectangle();

        System.out.println("Area = " + shp.area(5,10));
    }  // end main
}// end class
```

Area = 25 (area triangle)
or Area = 50 (area rect)

Abstract : Scenario



Shipping

Misal sistem memerlukan report yang melaporkan daftar kendaraan dan kebutuhan bahan bakar untuk melakukan perjalanan . Misal terdapat class ShippingMain yang mengumpulkan daftar kendaraan dan mengenerate Fuel Needs Report

```
1  public class ShippingMain {
2    public static void main(String[] args) {
3      Company c = Company.getCompany();
4
5      // populate the company with a fleet of vehicles
6      c.addVehicle( new Truck(10000.0) );
7      c.addVehicle( new Truck(15000.0) );
8      c.addVehicle( new RiverBarge(500000.0) );
9      c.addVehicle( new Truck(9500.0) );
10     c.addVehicle( new RiverBarge(750000.0) );
11
12     FuelNeedsReport report = new FuelNeedsReport();
13     report.generateText(System.out);
14   }
15 }
```

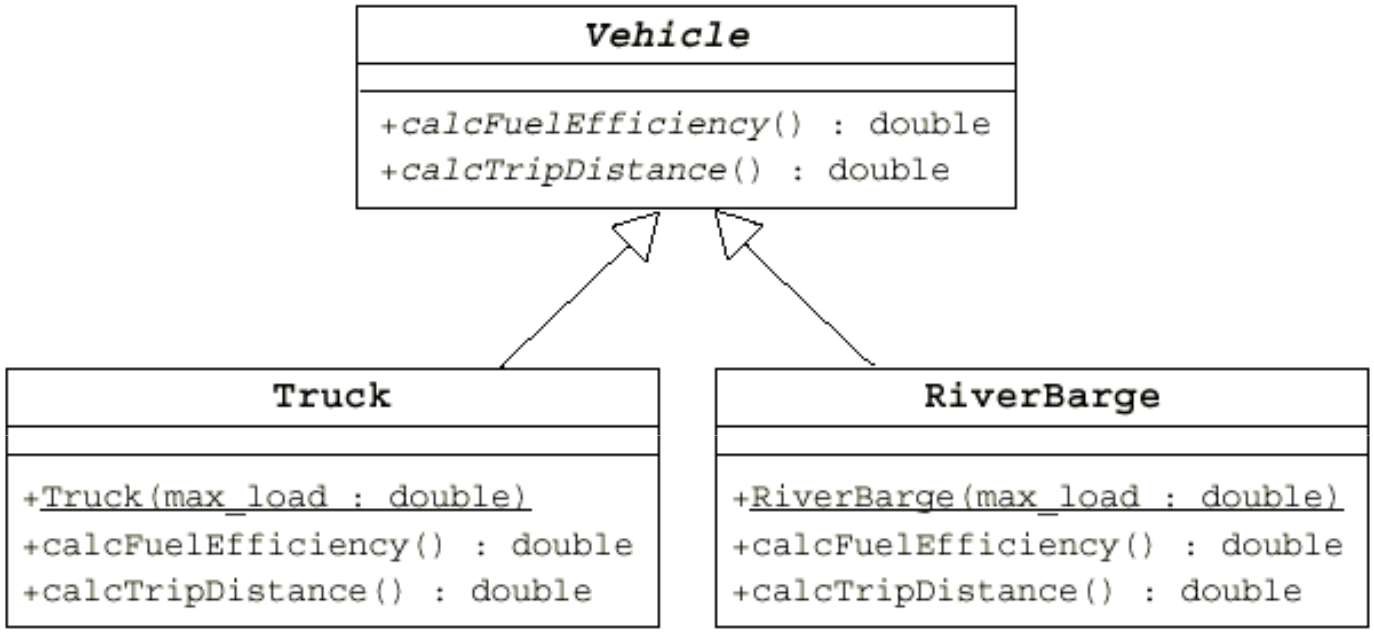
FuelNeedsReport code:

```
1  public class FuelNeedsReport {
2    public void generateText(PrintStream output) {
3      Company c = Company.getCompany();
4      Vehicle v;
5      double fuel;
6      double total_fuel = 0.0;
7
8      for ( int i = 0; i < c.getFleetSize(); i++ ) {
9        v = c.getVehicle(i);
10
11       // Calculate the fuel needed for this trip
12       fuel = v.calcTripDistance() / v.calcFuelEfficency();
13
14       output.println("Vehicle " + v.getName() + " needs "
15                      + fuel + " liters of fuel.");
16       total_fuel += fuel;
17     }
18     output.println("Total fuel needs is " + total_fuel + " liters.");
19   }
20 }
```

Problem 1 : Dimana seharusnya perhitungan jarak dan efisiensi bahan bakar terjadi?

- Perhitungan efisiensi bahan bakar dan jarak antara truck dan river barge sangat berbeda.
- Tidak mungkin perhitungan ini dideklarasikan pada class Vehicle.
- Jadi perhitungan ini harus ada di class Truck dan RiverBarge.
- Di Vehicle cukup ada abstract method dari perhitungan ini, sehingga class vehicle ini merupakan abstract class.

Solusi



- Italic font digunakan untuk menggambarkan element yang bersifat abstract.
- Pada abstract class Vehicle terdapat dua buah method abstract yaitu calcFuelEfficiency() dan calcTripDistance().

Solusi

```
1  public abstract class Vehicle {
2    public abstract double calcFuelEfficiency();
3    public abstract double calcTripDistance();
4  }
```

```
1  public class Truck extends Vehicle {
2    public Truck(double max_load) {...}
3
4    public double calcFuelEfficiency() {
5      /* calculate the fuel consumption of a truck at a given load */
6    }
7    public double calcTripDistrance() {
8      /* calculate the distance of this trip on highway */
9    }
10 }
```

```
1  public class RiverBarge extends Vehicle {
2    public RiverBarge(double max_load) {...}
3
4    public double calcFuelEfficiency() {
5      /* calculate the fuel efficiency of a river barge */
6    }
7    public double calcTripDistrance() {
8      /* calculate the distance of this trip along the river-ways */
9    }
10 }
```

Problem 2

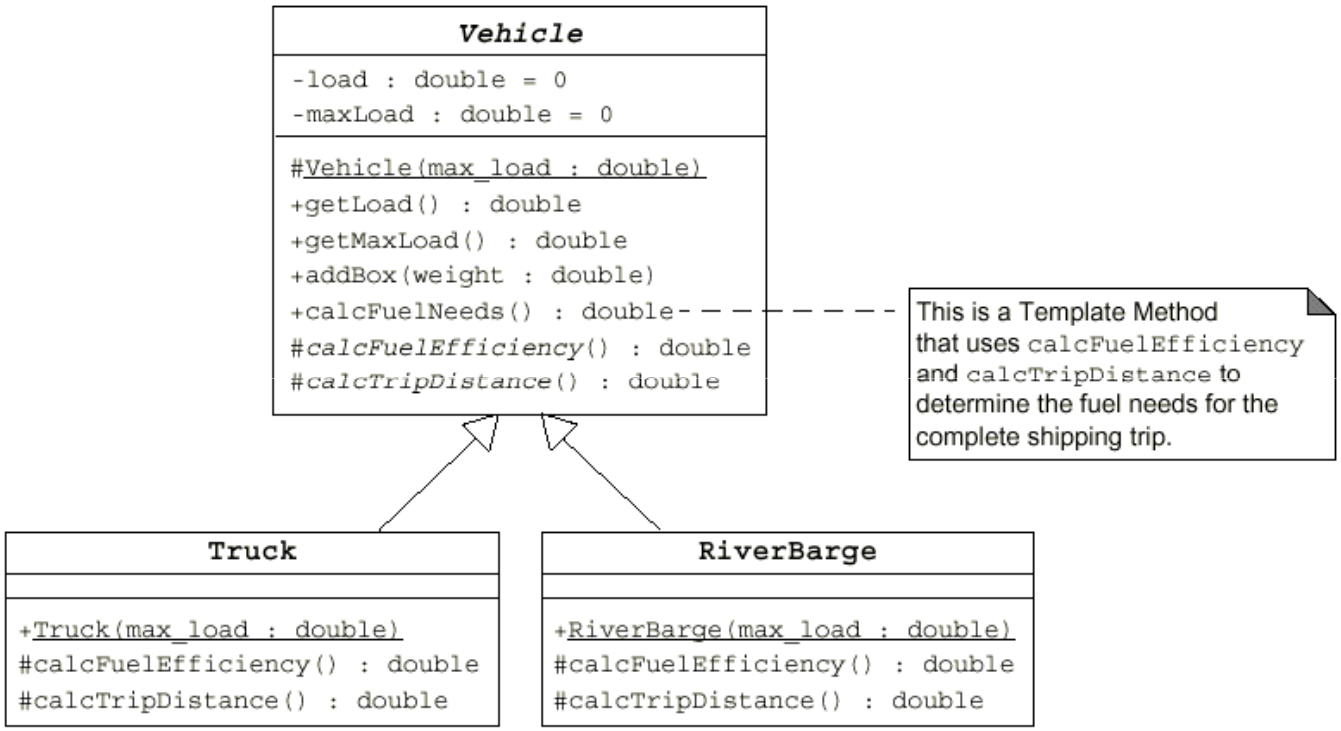
- Perhatikan kembali class FuelNeedsReport.
- Pada class FuelNeedsReport terdapat perhitungan kebutuhan bahan bakar.
- Padahal tidak seharusnya pada class FuelNeedsReport terdapat perhitungan kebutuhan Fuel.
- FuelNeedsReport hanya bertugas membuat report saja.

FuelNeedsReport code:

```
1  public class FuelNeedsReport {
2    public void generateText(PrintStream output) {
3      Company c = Company.getCompany();
4      Vehicle v;
5      double fuel;
6      double total_fuel = 0.0;
7
8      for ( int i = 0; i < c.getFleetSize(); i++ ) {
9        v = c.getVehicle(i);
10
11       // Calculate the fuel needed for this trip
12       fuel = v.calcTripDistance() / v.calcFuelEfficency();
13
14       output.println("Vehicle " + v.getName() + " needs "
15                      + fuel + " liters of fuel.");
16       total_fuel += fuel;
17     }
18     output.println("Total fuel needs is " + total_fuel + " liters.");
19   }
20 }
```

Solusi

Template Method Design Pattern



calcFuelNeeds() disebut Template Method karena calcFuelNeeds() merupakan non-abstract method yang mengakses method abstract yang diimplementasikan di subclassnya.

```
1  public class FuelNeedsReport {
2    public void generateText(PrintStream output) {
3      Company c = Company.getCompany();
4      Vehicle v;
5      double fuel;
6      double total_fuel = 0.0;
7
8      for ( int i = 0; i < c.getFleetSize(); i++ ) {
9        v = c.getVehicle(i);
10
11       // Calculate the fuel needed for this trip
12
13       fuel = v.calcFuelNeeds();
14       output.println("Vehicle " + v.getName() + " needs "
15                      + fuel + " liters of fuel.");
16       total_fuel += fuel;
17     }
18     output.println("Total fuel needs is " + total_fuel + " liters.");
19   }
20 }
```

Abstract: Ingat!!

- Jangan melakukan:

  new Vehicle();

- Bagaimana dengan inisialisai instance atribut class Vehicle? Gunakan constructor untuk menginisialisasi (bisa dengan menggunakan this dan super).

INTERFACE

Interface

- Interface berbeda dengan class.
- Interface berisi method kosong dan konstanta.
- Method dalam interface tidak mempunyai statement.
- Sehingga deklarasi method dalam interface sama dengan deklarasi abstract method pada abstract class.

Interface

- Method yang dideklarasikan didalam interface secara otomatis adalah public dan abstract.
- Variable dalam interface secara otomatis adalah public, static, dan final.

Contoh Interface

```
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Mengimplementasikan Interface

- Bila sebuah class mengimplementasikan suatu interface, maka semua konstanta dan method interface akan dimiliki oleh class ini.
- Method pada interface harus diimplementasikan pada class yang mengimplementasikan interface ini.
- Bila class yang mengimplementasikan interface tidak mengimplemetasikan semua method dalam interface, maka class tersebut harus dideklarasikan abstract.

```
<class_declaration> ::=
  <modifier> class <name> [extends <superclass>]
        [implements <interface> [,<interface>]* ] {
    <declarations>*
  }
```

Mengimplementasikan Interface

```
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Implementasi Interface

```
public class Line implements Relation {
private double x1;
private double x2;
private double y1;
private double y2;
public Line(double x1, double x2, double y1, double y2){
this.x1 = x1;
this.x2 = x2;
this.y1 = y1;
this.y2 = y2;
}
public double getLength(){
double length = Math.sqrt((x2-x1)*(x2-x1) +
(y2-y1)* (y2-y1));
return length;
}
```

Implementasi Interface

```
public boolean isGreater( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen > bLen);
}
public boolean isLess( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen < bLen);
}
public boolean isEqual( Object a, Object b){
double aLen = ((Line)a).getLength();
double bLen = ((Line)b).getLength();
return (aLen == bLen);
}
}
```

Implementasi Interface

- Class Line mengimplementasikan interface Relation, pastikan semua method yang ada di interface diimplementasikan di class Line. Jika tidak maka muncul error.

Line.java:4: Line is not abstract and does not override
abstract method
isGreater(java.lang.Object,java.lang.Object) in Relation
public class Line implements Relation
^
1 error

Inheritance pada Interface

- Kita bisa membuat subinterface dengan menggunakan kata extends.
- Satu class boleh mengimplementasikan lebih dari satu interface.
- Suatu interface boleh mengextends lebih dari satu interface.

Inheritance pada Interface

- Interface bukan bagian dari hirarki class
- Namun interface dapat mempunyai relasi inheritance

```
public interface PersonInterface {
    void doSomething();
}
public interface StudentInterface extends PersonInterface {
    void doExtraSomething();
}
```

Mengimplementasikan Multiple Interface

- Satu class boleh mengimplementasikan lebih dari satu interface.
- Bila suatu class akan dijadikan subclass dan akan mengimplementasikan interface, maka kata extends harus lebih dulu dari implements.

Mengimplementasikan Multiple Interface

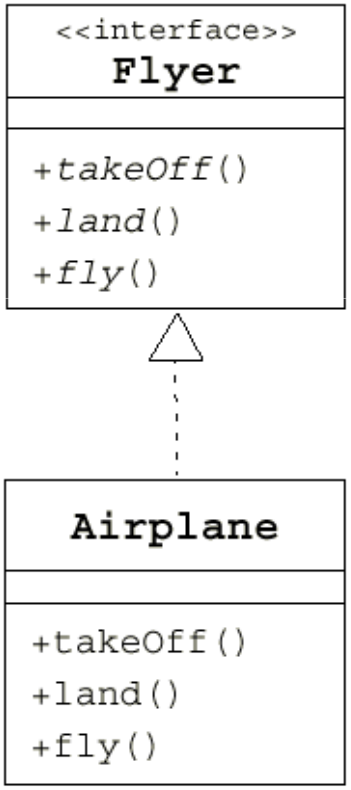
- A concrete class extends one super class but multiple Interfaces:

```
public class ComputerScienceStudent extends Student
implements PersonInterface, AnotherInterface,  Thirdinterface{
// All abstract methods of all interfaces
// need to be implemented.
}
```

Kegunaan Interface

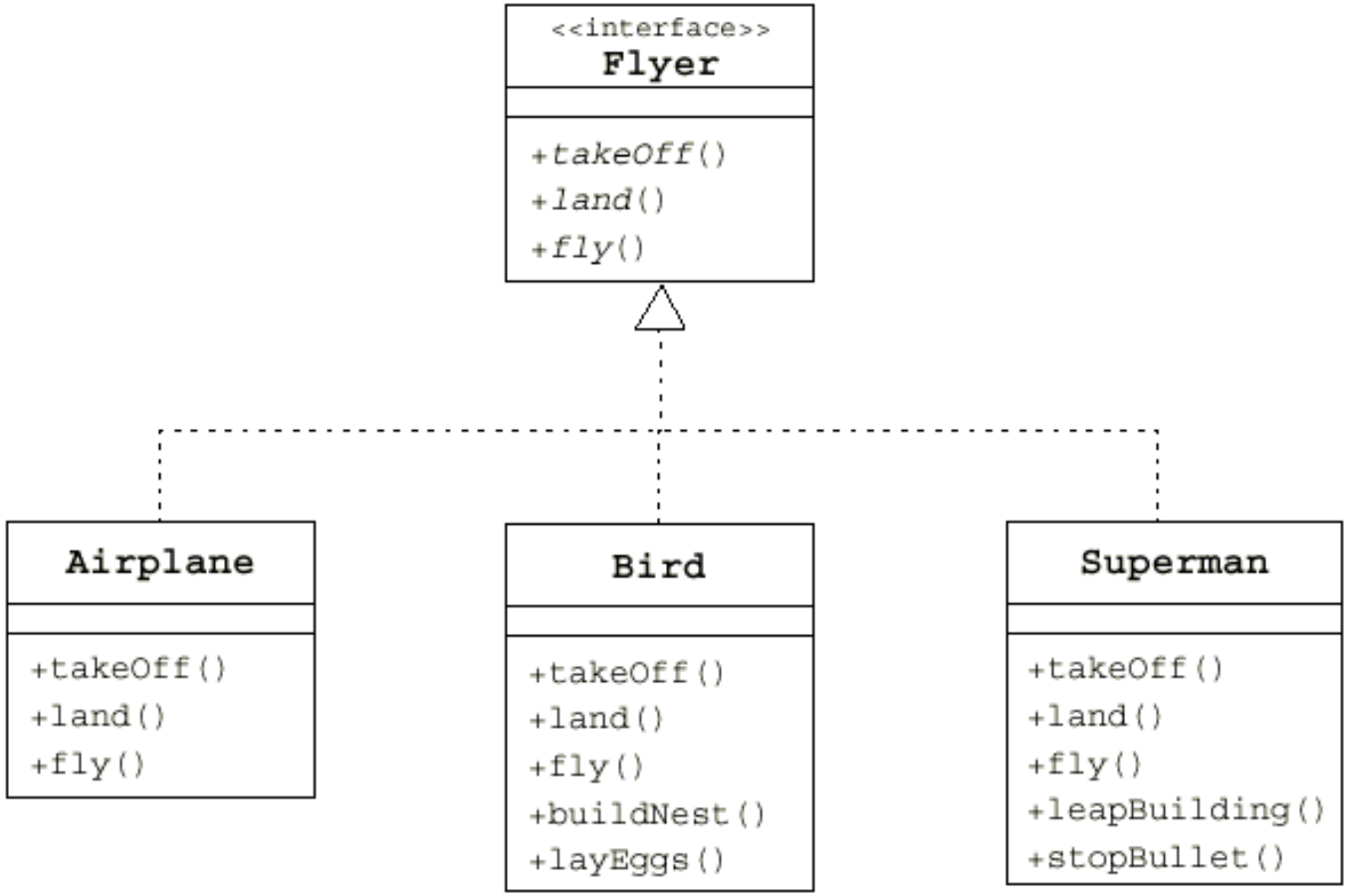
- Semua class yang mengimplementasikan sebuah interface tertentu berarti class-class tersebut mengimplementasikan methods yang sama dengan kata lain class-class tersebut mempunyai fungsionalitas yang sama.

Interface Flyer dan Airplane Implementation

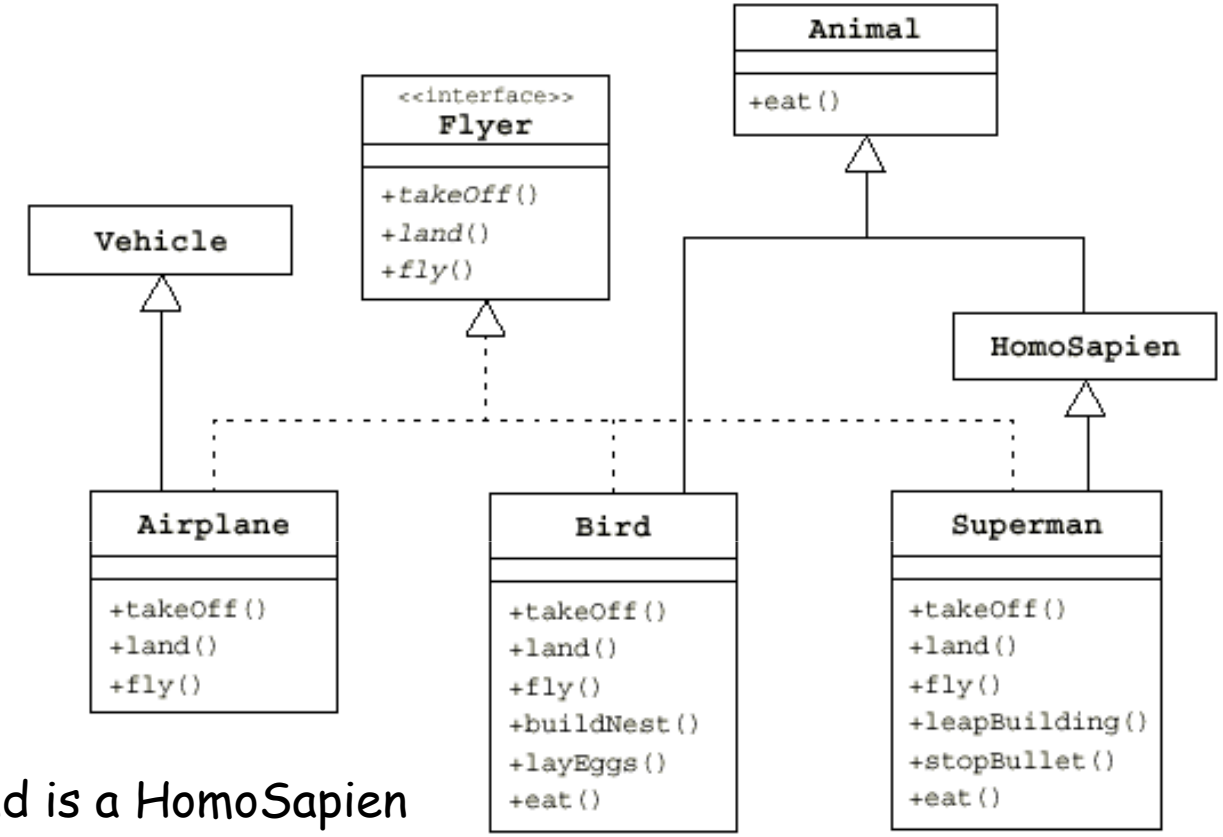



```
public interface Flyer {
  public void takeOff();
  public void land();
  public void fly();
}

public class Airplane implements Flyer {
  public void takeOff() {
    // accelerate until lift-off
    // raise landing gear
  }
  public void land() {
    // lower landing gear
    // deccelerate and lower flaps until touch-down
    // apply breaks
  }
  public void fly() {
    // keep those engines running
  }
}
```

Multiple Implementation of the Flyer Interface



Gabungan Inheritance dan Implementation



Airplane is a Vehicle

Bird is an Animal

Superman is an Animal and is a HomoSapien

Kelemahan multiple inheritance adalah suatu class bisa mewarisi method dari lebih dari satu class dimana method ini tidak diharapkan. Dengan Interface maka hal ini bisa dihindari.

Example: Multiple Interface