Pemrograman Berorientasi Obyek

Abstract class, interface, dan inner class

Oleh Politeknik Elektronika Negeri Surabaya

2020

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Kasus 1

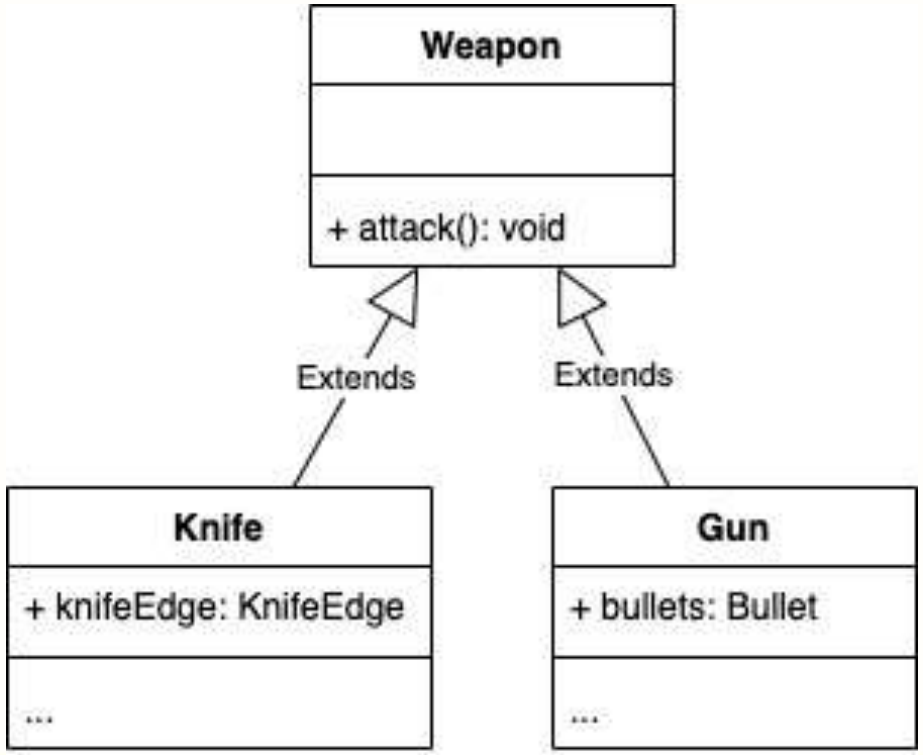
- Misal dalam aplikasi kita terdapat dua buah objek yaitu :
  - Objek Knife
  - Objek Gun
- Kedua objek tersebut dapat digunakan oleh player untuk menyerang.
- Tapi cara menyerang kedua objek tersebut berbeda.
  - Objek Knife : menusuk dan menyayat lawan
  - Objek Gun : menembakkan peluru
- Maka bagaimana kita mengimplementasikan kasus tersebut dalam aplikasi?

Kasus 1

- Langkah pertama adalah mendeskripsikan setiap objek dengan data dan sifat yang dimiliki.
  - Knife : attack()
  - Gun : attack()
- Jika diimplementasikan dalam class diagram menjadi:

Kasus 1

- Jika terdapat data (attribute) atau sifat (method) yang sama dalam beberapa objek berbeda, maka kita dapat melakukan **generalisasi** atau **abstraksi**

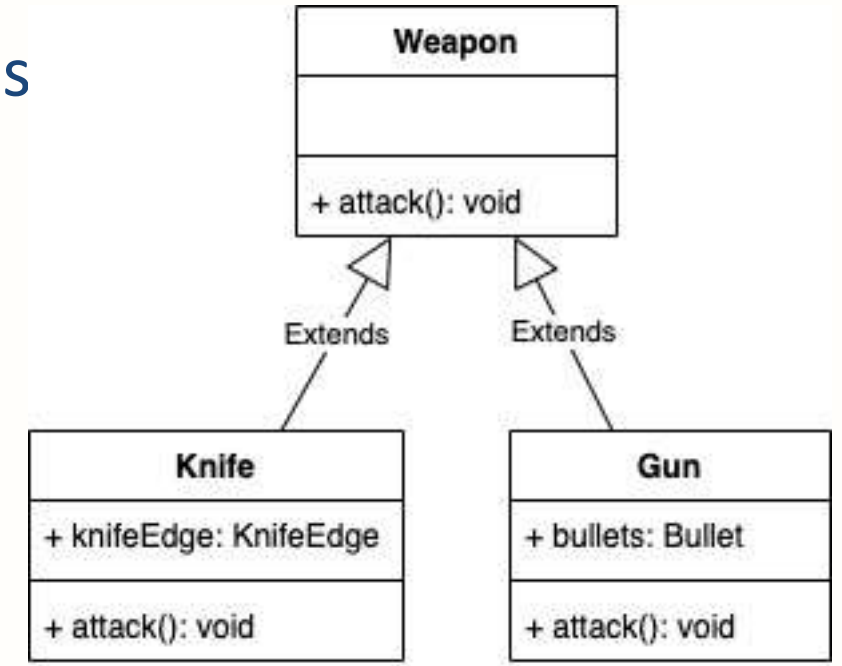


Kasus 1

- Tapi masalah belum terselesaikan
- Isi dari method attack pada Knife dan Gun berbeda.
  - Objek Knife : menusuk dan menyayat lawan
  - Objek Gun : menembakkan peluru
- Jika method attack ditulis pada class Weapon, maka proses apa yang akan ditulis? Apakah **menusuk** atau **menembak**?

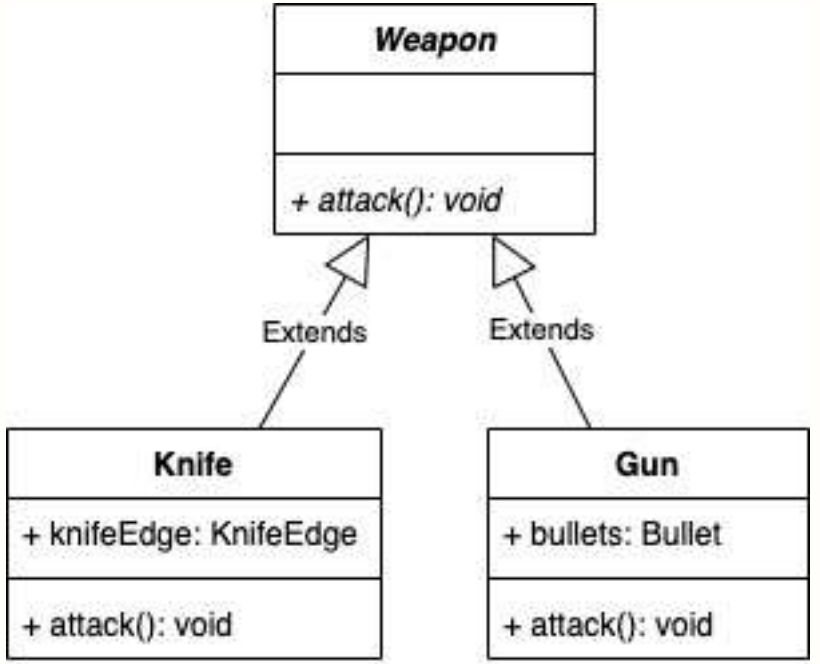
Kasus 1

- Dalam kondisi seperti ini kita tidak bisa menggunakan overriding. Karena dengan overriding seharusnya method milik parent tidak boleh ditolak/dihapus oleh subclass.
- Apapun yang ditulis pada parent/super class harus diwarisi oleh subclass.
- Jika parent menyerang dengan menusuk maka Gun juga harus bisa menusuk.
- Jika parent menyerang dengan menembak maka Knife juga harus bisa menembak.



Kasus 1

- Dalam kondisi seperti ini kita harus menggunakan abstract class dan abstract method.
- Abstract method adalah method kosong atau method yang tidak memiliki body.
- Abstract class adalah class yang memiliki minimal 1 abstract method.
- Penulisan abstract dalam class diagram ditandai dengan penulisan **italic/miring**



Abstract Class dan Abstract Method

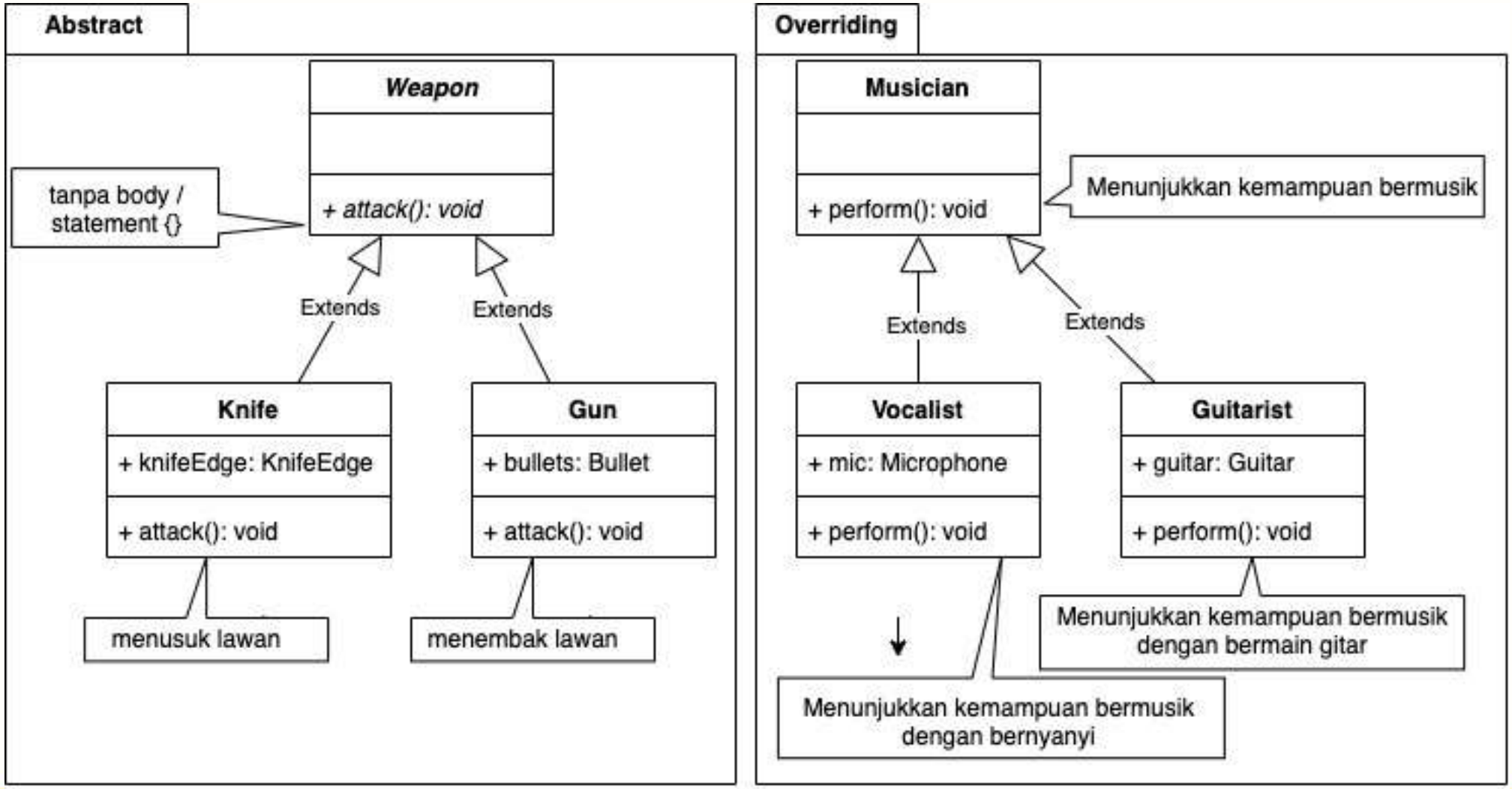
- Deklarasi abstract method menggunakan kata kunci **abstract** sebelum penulisan kata kunci **class**
  - public **abstract class** Weapon**{…}**
- Deklarasi abstract method menggunakan kata kunci **abstract** sebelum return type sebuah method.
  - public **abstract** void attack();
- Method abstract tidak memiliki blok statement yang ditandai dengan tanda kurung kurawal “**{}**”

```
//deklarasi abstract class
public abstract class Weapon{

    //deklarasi abstract method
    public abstract void attack();

}
```

Abstract Method vs Overriding Method



Abstract Method vs Overriding Method

- Kita menggunakan overriding method jika isi method super class dan subclass ada kemiripan. Atau kita masih bisa mendeskripsikan isi dari method milik super class.
- Kita menggunakan abstract method saat kita tidak bisa mendeskripsikan isi dari method yang ada pada parent. Sehingga method pada parent kosong dan lebih baik menjadi abstract method yang tidak memiliki body.

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.

```
public interface Performer{
    //deklarasi konstanta
    public Boolean CONFIDENCE = true;

    //deklarasi method
    public void perform();
    public void attract();
}
```

Interface

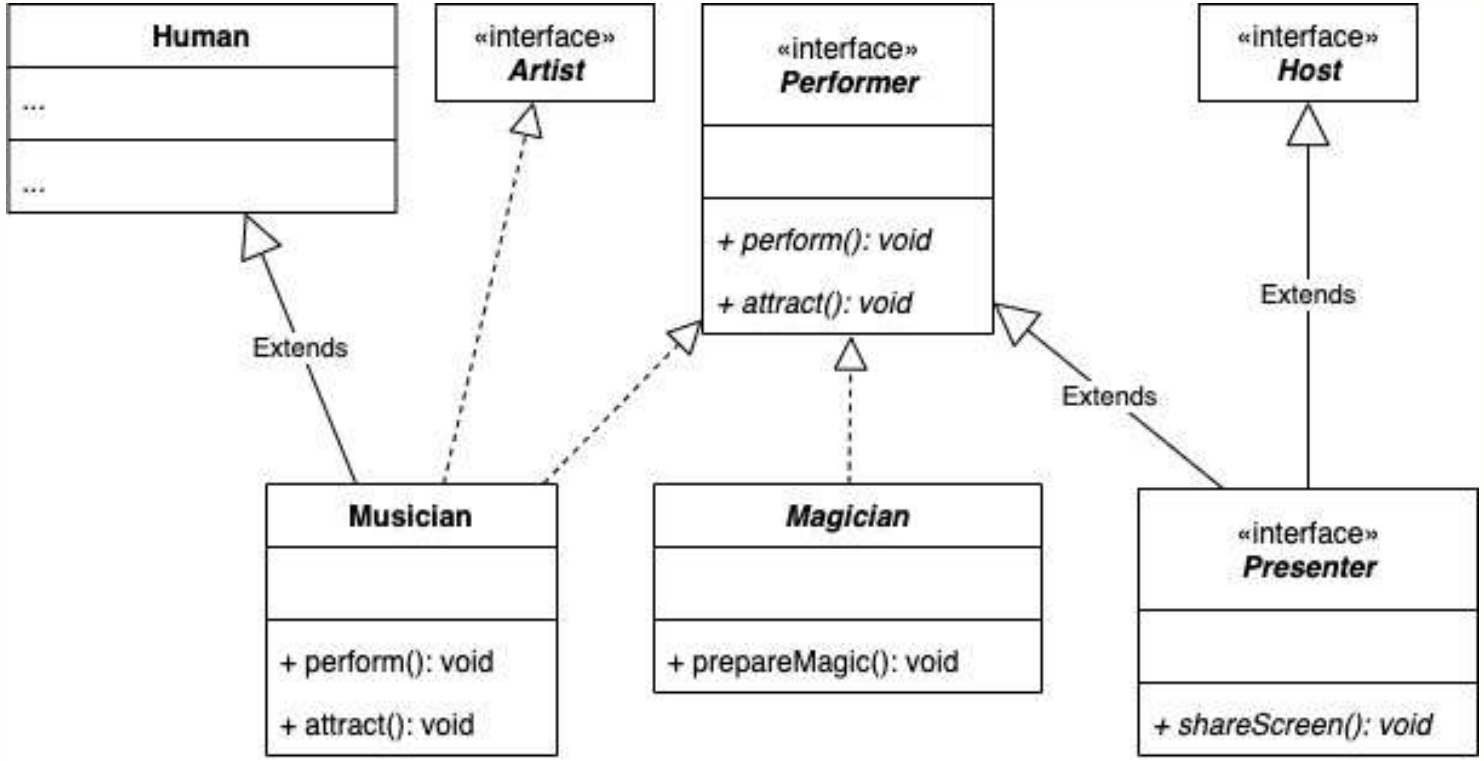
- Bila sebuah class mengimplementasikan suatu interface, maka semua konstanta dan method interface akan dimiliki oleh class ini.
- Method pada interface harus diimplementasikan pada class yang mengimplementasikan interface ini.
- Bila class yang mengimplementasikan interface tidak menginplemetasikan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

```
public class Musician implements Performer{
    public void perform(){
        …
    }
    public void attract(){
        …
    }
}
```

```
public abstract class Magician implements Performer{
    public void prepareMagic(){
        …
    }
}
```

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.
- Bila suatu class akan menjadi subclass dan akan mengimplementasikan interface, maka kata extends harus lebih dulu dari implements.



	Method	Deklarasi
Concrete Class	All concrete	<modifier> **class** <identifier> {…}
Abstract Class	Minimum 1 abstract method	<modifier> **abstract class** <identifier>{…}
Interface	All abstract	<modifier> **interface** <identifier>{…}

Inner (Nested) Class

```
class OuterClass {
    int x = 10;
    class InnerClass {
        int y = 5;
    }
}
```

```
public class MyMainClass {
    public static void main(String[] args) {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
        System.out.println(myInner.y + myOuter.x);
    }
}
```

Anonymous Inner Class

```
1. public void aMethod() {
2.  TheButton.addActionListener(
3.   new ActionListener() {
4.    public void actionPerformed(ActionEvent e) {
5.     System.out.println("The action has occurred");
6.    }
7.   }
8.  );
9. }
```

bridge to the future

http://www.eepis-its.edu

Materi Lama

Konten

- Abstract Class
- Interface
- Inner Class
  - Non-static Inner class
    - Mendeklarasikan class di dalam class
  - Local Inner Class
    - Mendeklarasikan class di dalam method
  - Anonymous inner class
  - Static inner class

Abstract

- Abstract class adalah class yang mempunyai setidaknya satu abstract method.
- Abstract method adalah method yang tidak memiliki body (hanya deklarasi method).
- Implementasi dari isi abstract method tersebut biasanya dilakukan di subclass.

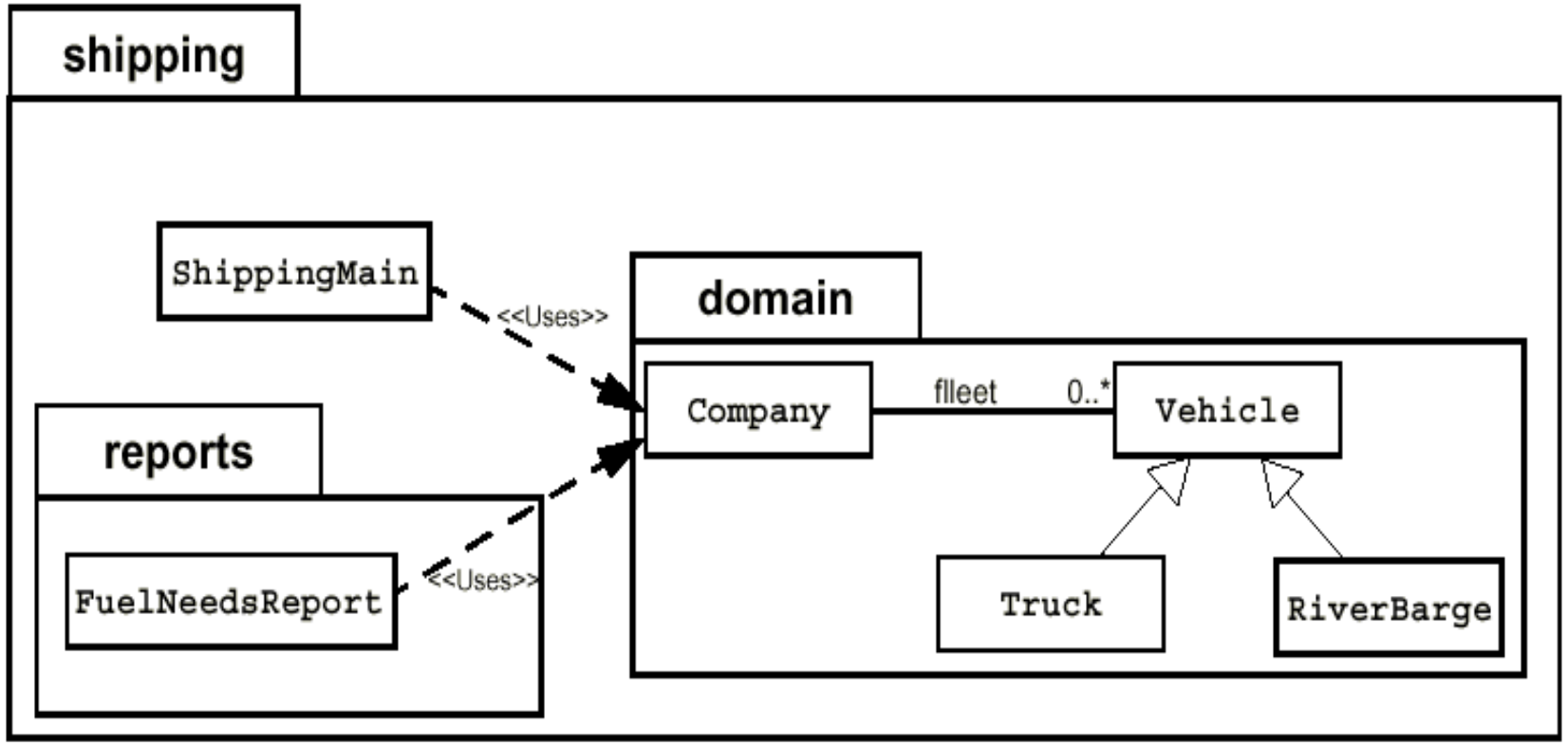
Abstract

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

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.

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 }
```

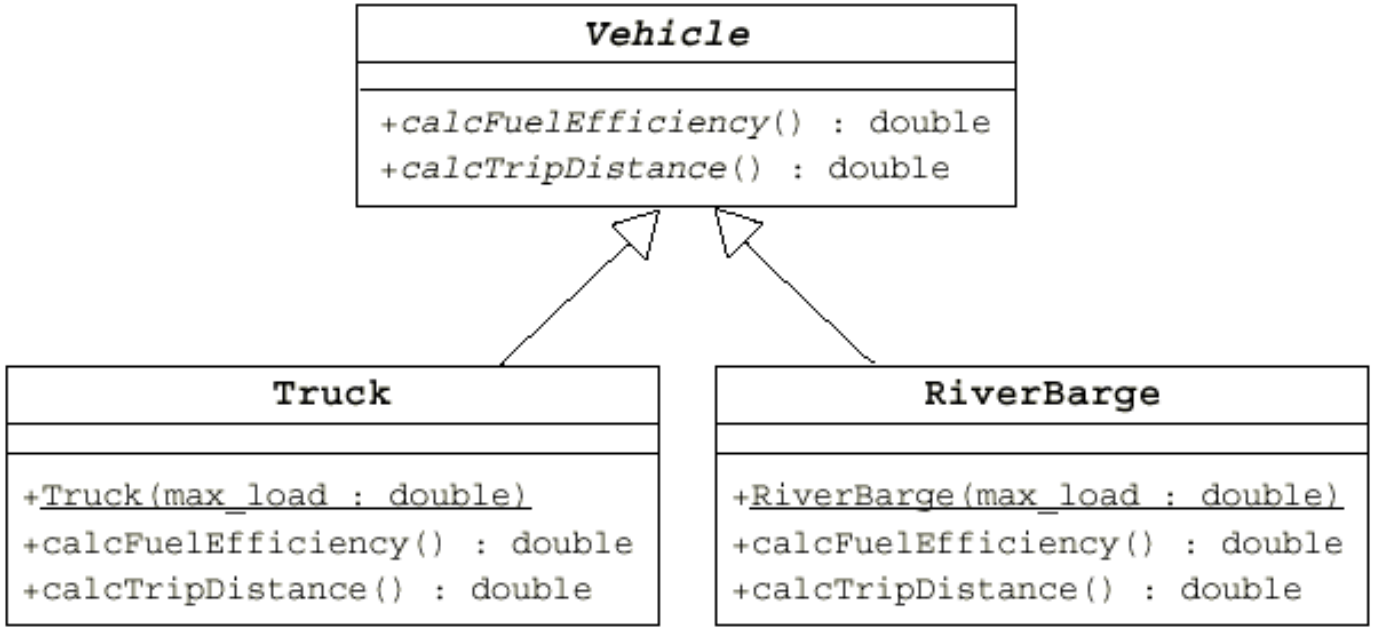
FuelNeedsReport code:

```
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

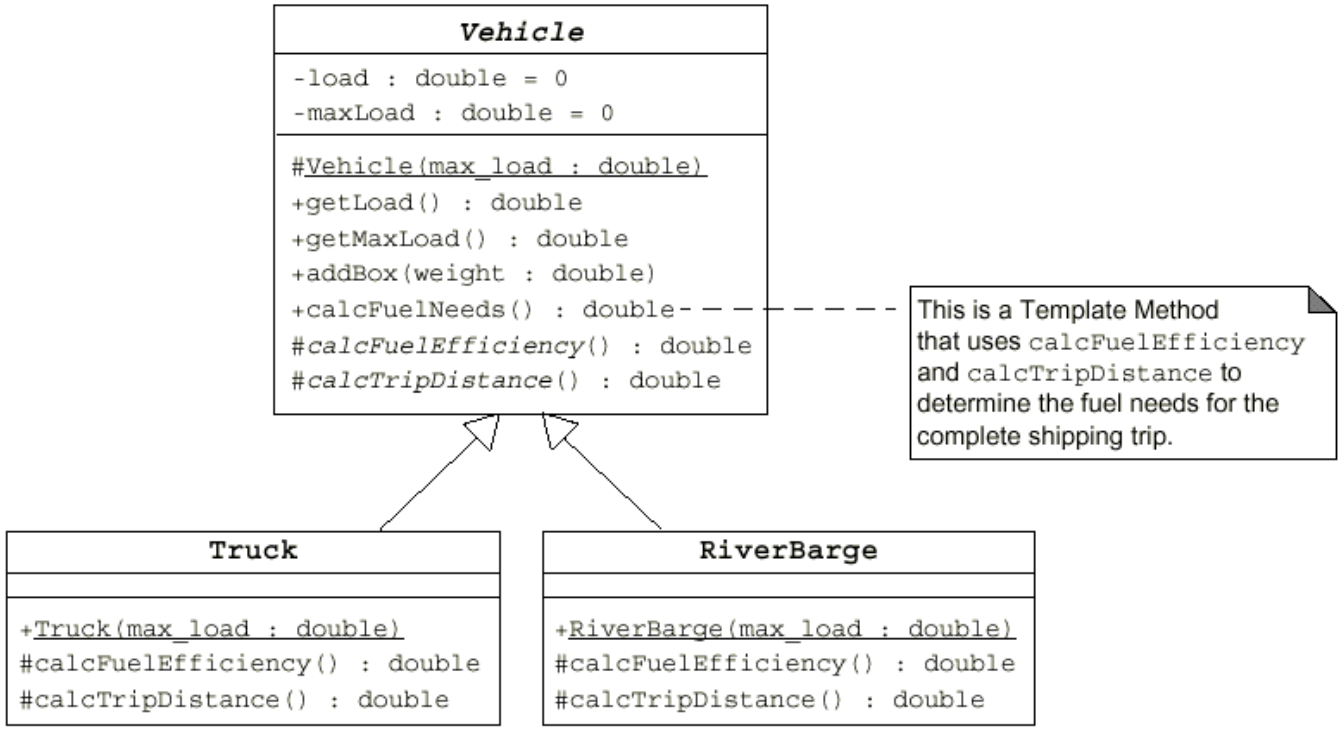
- Perhatikan kembali class FuelNeedsReport.
- Pada class FuelNeedsReport terdapat perhitungan kebutuhan bahan bakar.
- Padahal tidak seharusnya pada class FuelNeedsReport terdapat perhitungan kebutuhan Fuel.
- FuelNeedsReport hanya bertugas membuat report saja.

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.

```
public class FuelNeedsReport{
  public void generateText(PrintStream output){

        // calculate the fuel needed for this trip
        fuel = v.calcFuelNeeds();
  }
}
```

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 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

- 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 menginplemetasikan semua method dalam interface, maka class tersebut harus dideklarasikan abstract.

Interface

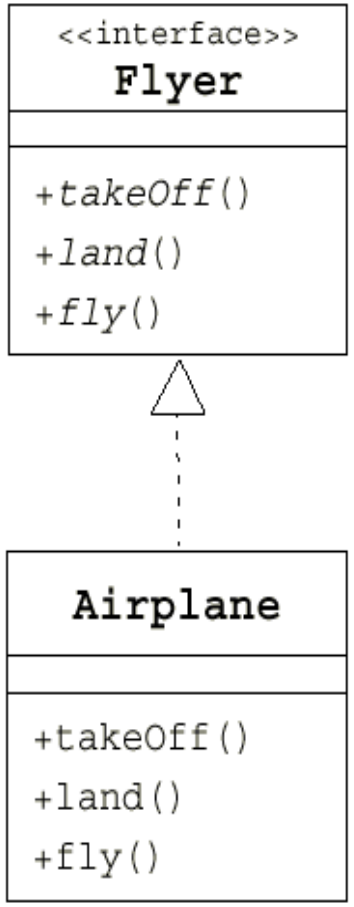
- Kita bisa membuat subinterface dengan menggunakan kata extends.
- Satu class boleh mengimplementasikan lebih dari satu interface.
- Suatu interface boleh mengextends lebih dari satu interface.
- Bila suatu class akan dijadikan subclass dan akan mengimplementasikan interface, maka kata extends harus lebih dulu dari implements.

Interface

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


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

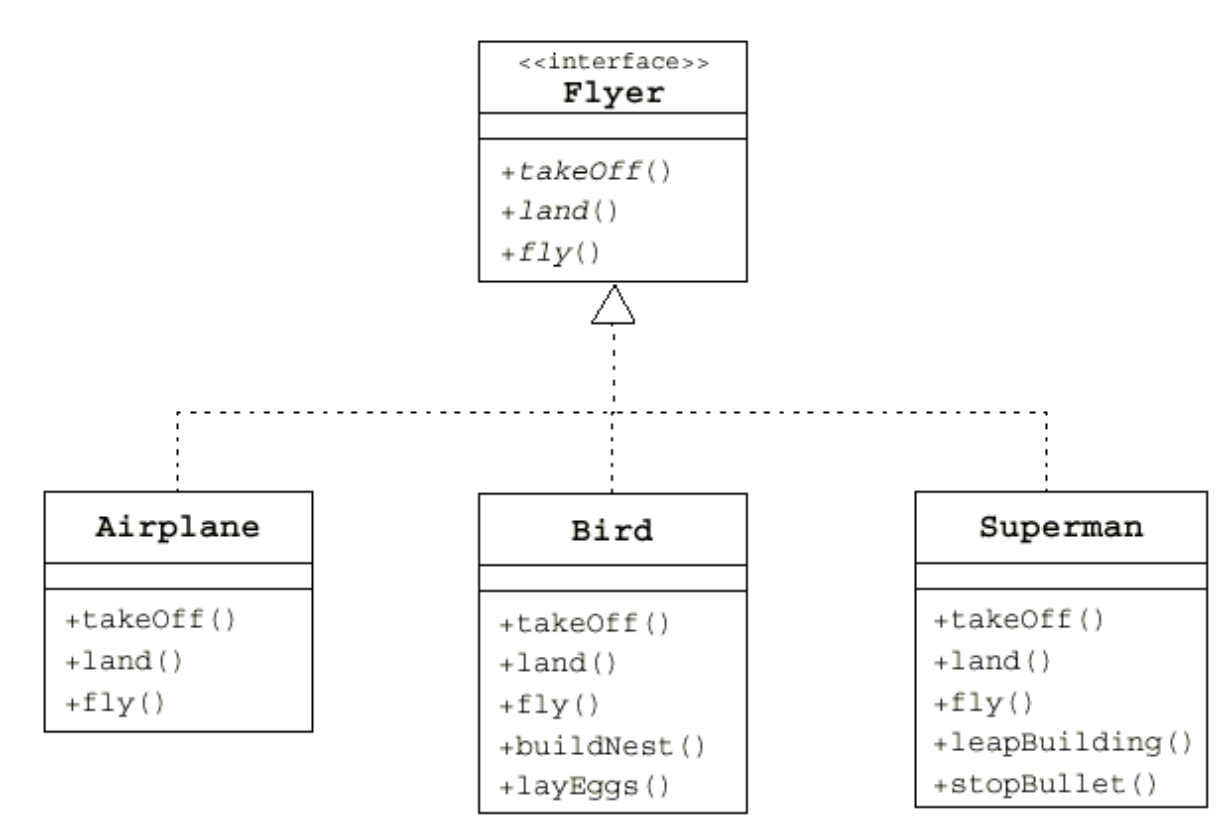
Interface Flyer dan Airplane Implementation



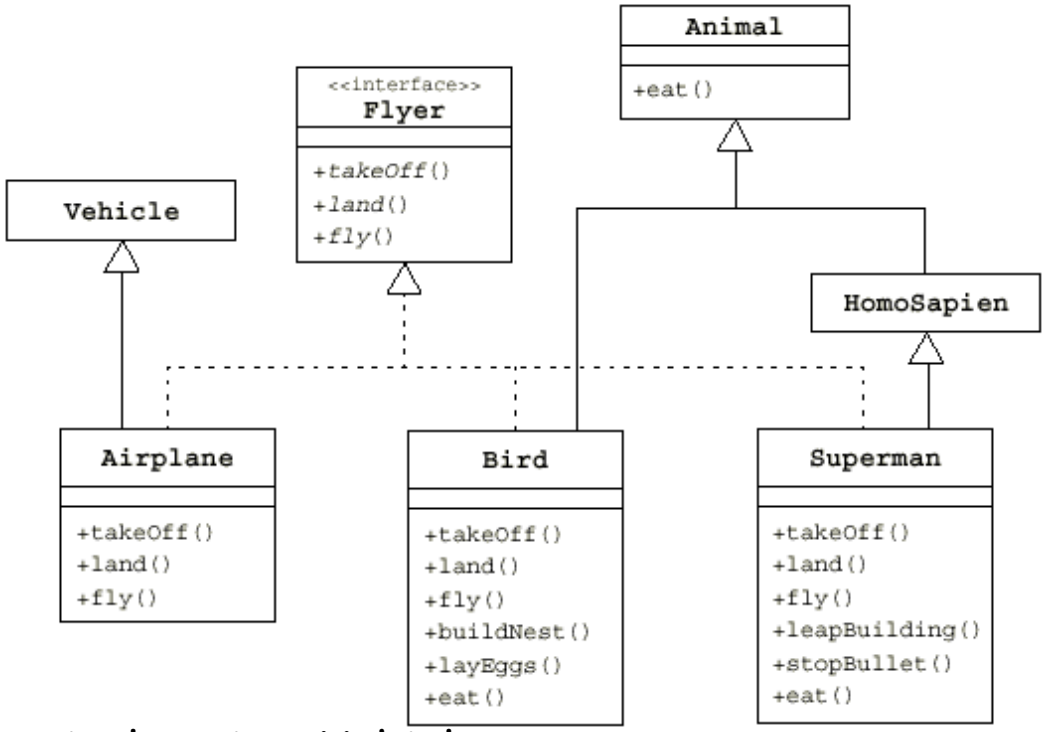

```
public interface Flyer {
  public void takeOff();
  public void land();
  public void fly();
}

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



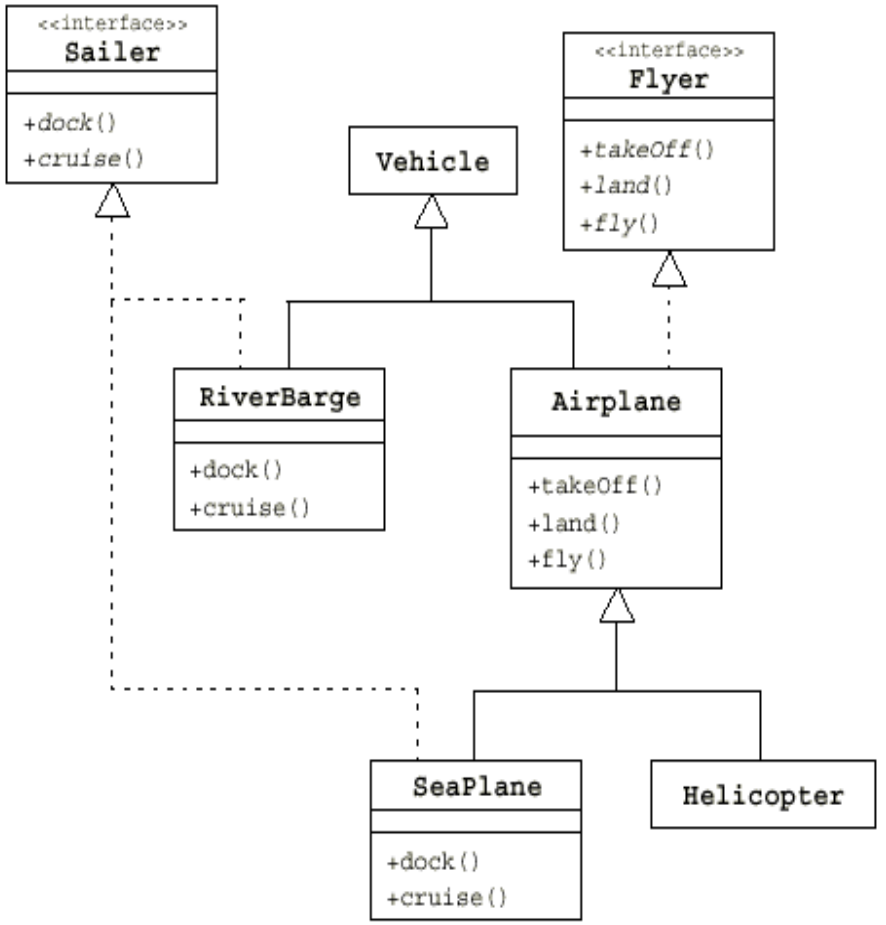
Airplane is a Vehicle

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



Inner Class

- Mulai ada sejak JDK 1.1
- Disebut juga nested class.
- Membuat class di dalam class.
- Nama inner class harus berbeda dengan nama outer class.

Kegunaan Inner Class

- Mengelompokkan class-class.
- Melakukan kontrol terhadap class lain. Mengimplementasikan detail statement yang seharusnya tidak dishare dengan class lain.
- Mudah dibaca dan dimaintain.

Why Use Nested Classes?

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

1. **It is a way of logically grouping classes that are only used in one place**.

   If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

2. **It increases encapsulation**

   Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

3. **It can lead to more readable and maintainable code**

   Nesting small classes within top-level classes places the code closer to where it is used.

Jenis Inner Class

- Non static inner class (inner class)
- Static inner class (static nested class)
- Local Inner Class (Inner Class di dalam Method)
- Anonymous Inner Class

Non-Static Inner Class
Mendeklarasikan Class di dalam Class

- Inner class dapat didefinisikan abstract.
- Inner class (kecuali inner class yang didefinisikan di method) bisa mempunyai mode akses.
- Inner classes boleh dideklarasikan **public, protected, private, abstract, static** or **final.**
- Aturan mode akses sama dengan class biasa.
  - Private inner class hanya bisa diakses oleh outer class nya sendiri.
  - Protected inner class hanya bisa diakses oleh subclass.
  - Inner class bisa berupa interface yang diimplementasikan oleh inner class lain
- Ingat !! Local inner class dan anonymous inner class tidak boleh punya mode akses.
- Interface tidak boleh dibuat di dalam inner class.

Non-Static Inner Class
Mendeklarasikan Class di dalam Class

- Inner classes tidak boleh mendeklarasikan static initializers atau static members, kecuali static final variabel, contoh: static final var = value;
- Inner class yang dideklarasikan di dalam outer class bisa mengakses member dari outer class.
- Untuk merefer ke method atau variabel outer class dari inner class lakukan dengan cara Outer.this.fieldname.

Non-Static Inner Class
Mendeklarasikan Class di dalam Class

```
class Outer {
   public class PublicInner{}
   protected class ProtectedInner {}
   private class PrivateInner{}
   abstract class AbstractInner {}
   final class FinalInner {}
   class StaticInner {}
 }
```

Membuat Obyek Non-Static Inner Class

```
class Outer {
    class Inner {}    // class definition within the
                      // the body of class Outer
}
```

- Hasil kompilasi adalah Outer.class dan Outer$Inner.class.
- Membuat obyek class Outer:

```
Outer o1 = new Outer();
```

- Membuat obyek class Inner :

```
Outer.Inner i1 = o1.new Inner();
```

atau langsung dari inner:

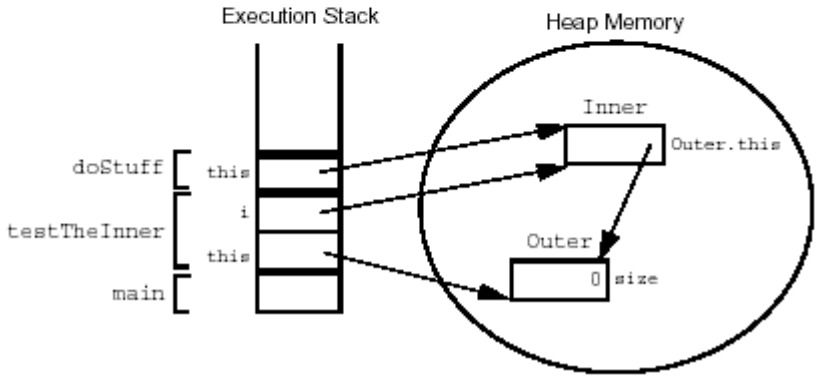
```
Outer.Inner i2 = new Outer().new Inner();
```

atau melalui konstruktor Outer:

```
Outer {
    Outer() {
        new Inner();
    }
}
```

Mendeklarasikan Non-Static Inner Class di dalam Class

```
1   public class Outer1 {
2     private int size;
3
4     /* Declare an inner class called "Inner" */
5     public class Inner {
6       public void doStuff() {
7         // The inner class has access to 'size' from Outer
8         size++;
9       }
10    }
11
12    public void testTheInner() {
13      Inner i = new Inner();
14      i.doStuff();
15    }
16  }
```



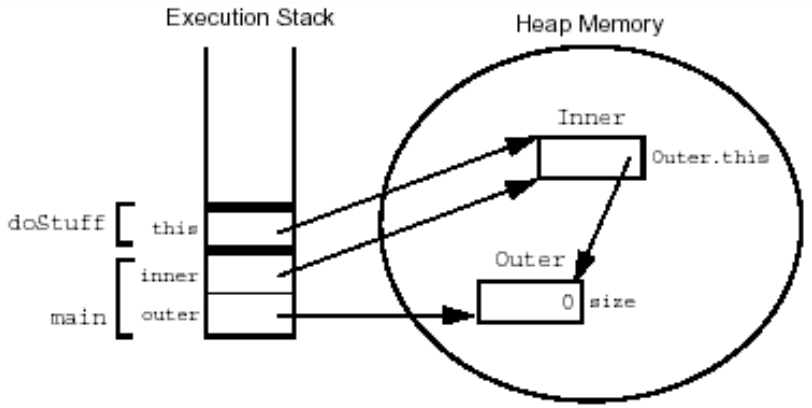
- Class `Outer1` mendeklarasikan variabel bernama `size`, inner class bernama `Inner`, dan method bernama `testTheInner`.
- Class `Inner` mendeklarasikan method bernama `doStruff`, method ini bisa mengakses member dari class `Outer1`. Sehingga variabel `size` pada deklarasi method ini merefer pada instance variabel class `Outer1`.
- Gambar menunjukkan memory representation dari Inner class.

Non-Static Inner Class
Mendeklarasikan Class di dalam Class

- Hasil kompilasi inner class yang dideklarasikan di class adalah ClassOuterName$ClassInnerName.class
- Hasil kompilasi program diatas adalah Outer1.class dan Outer1$Inner.class

Membuat Obyek dari Non-Static Inner Class

```
1   public class Outer2 {
2     private int size;
3
4     public class Inner {
5       public void doStuff() {
6         size++;
7       }
8     }
9   }
```

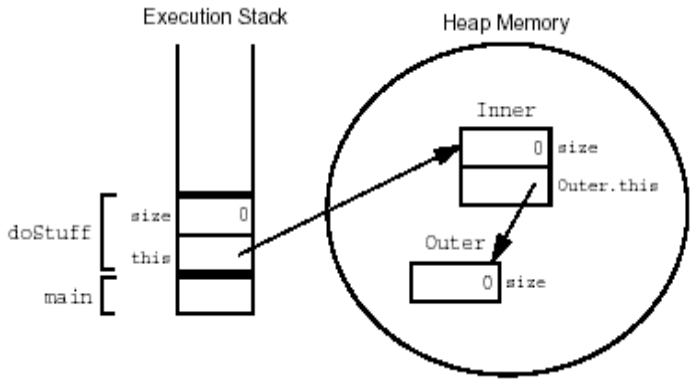



```
1   public class TestInner {
2     public static void main(String[] args) {
3       Outer2 outer = new Outer2();
4
5       // Must create an Inner object relative to an Outer
6       Outer2.Inner inner = outer.new Inner();
7       inner.doStuff();
8     }
9   }
```

- Contoh ini menunjukkan bagaimana membuat obyek dari inner class di class yang lain (`TestInner`).
- Gambar menunjukkan akses dari class lain ke Inner class.

Pemberian nama variabel pada non-static inner class yang sama dengan instance variable.

```
1   public class Outer3 {
2     private int size;
3
4     public class Inner {
5       private int size;
6
7       public void doStuff(int size) {
8         size++;              // the local parameter
9         this.size++;         // the Inner object attribute
10        Outer3.this.size++;  // the Outer3 object attribute
11      }
12    }
13  }
```



- Variabel size digunakan dalam tiga konteks, instance variable bagi class Outer3, instance variable bagi class Inner, dan local variabel bagi method doStuff.
- Hal ini diperbolehkan.

Static Inner Class

- Inner class yang dideklarasikan static.
- Inner class ini akan menjadi top-level class, bisa langsung dibuat tanpa melewati object dari outer class.
- Bila ingin mendeklarasikan member maka member ini harus dideklarasikan static.
- Di kelas lain, obyek inner class bisa dibuat dengan cara: Outer.Inner obj = new Outer.Inner();
- Variabel static milik static inner class juga bisa langsung diakses melalui nama outer class:

```
Outer.Inner.variabel;
```

Static Inner Class

Untuk membuat obyek dari static inner class, tidak diperlukan pembuatan obyek outer class terlebih dahulu.

```
class Outer {
    static class Inner {
        static int value = 100;
    }
}

class TestOuter{
    public static void main(String[] args) {
        Outer.Inner obj = new Outer.Inner();
        int x = Outer.Inner.value;
        System.out.println(x);
    }
}
```

Static Inner Class

```
class Outer {
  public static void main(String[] args) {
        int     x = Inner.value;
  }

  static class Inner {
        static int value = 100;
  }
}
```

Local Inner Class
Mendefinisikan Inner Class di dalam Method

- Disebut dengan local inner class atau local nested class.
- Inner class yang dideklarasikan di method hanya bisa mengakses member variabel outer, final local variabel dan final formal parameter.
- Local inner class hanya bisa diinstansiasi saat method dipanggil saja.
- Local inner class bukan merupakan member dari outer class.
- Karena bukan class members maka tidak dapat diinstansiasi di luar blok method dengan cara new Outer.new Local(); →cara ini tidak bisa !!!.

Local Inner Class
Mendefinisikan Inner Class di dalam Method

- Obyek yang dibuat dari inner class yang dideklarasikan dalam method bisa mengakses variable yang dideklarasikan didalam method.
- Tidak bisa menggunakan access modifier: tidak boleh dideklarasikan **private, public, protected,** or **static**. Boleh dideklarasikan **final.**
- Boleh mengakses static dan non-static member kepunyakan inner class itu sendiri.

Local Inner Class

```
class Outer {
  void display() {
        class Local {
              // body of Local class
        }
  }
}
```

- Hasil kompilasi program diatas adalah **Outer.class** dan **Outer1Local.class**

Inner Class di dalam Method

```
1   public class Outer4 {
2     private int size = 5;
3
4     public Object makeTheInner(int localVar) {
5       final int finalLocalVar = 6;
6
7       // Declare a class within a method!?!
8       class Inner {
9         public String toString() {
10          return ("#<Inner size=" + size +
11                  // " localVar=" + localVar + // ERROR: ILLEGAL
12                  "finalLocalVar=" + finalLocalVar + ">");
13        }
14      }
15
16      return new Inner();
17    }
18
19    public static void main(String[] args) {
20      Outer4 outer = new Outer4();
21      Object obj = outer.makeTheInner(47);
22      System.out.println("The object is " + obj);
23    }
24  }
```

Anonymous Inner Class

- Inner class yang tidak mempunyai nama.
- Didefinisikan di dalam method.
- Tidak boleh menggunakan modifier.
- Boleh melakukan extends atau implements.
- Tidak boleh melakukan overloading method dan menambahkan method baru.
- Tidak mempunyai konstruktor.
- Bisa mengembalikan obyek baru sesuai dengan definisi class dalam method.
- Biasanya digunakan untuk mendapatkan obyek yang mengimplementasikan interface tertentu.
- Paling banyak digunakan untuk mengimplementasikan event listener.
- Biasanya berisi statement singkat.

Anonymous Inner Class

```
1. public void aMethod() {
2.  TheButton.addActionListener(
3.   new ActionListener() {
4.    public void actionPerformed(ActionEvent e) {
5.     System.out.println("The action has occurred");
6.    }
7.   }
8.  );
9. }
```

```
public class SomeGUI extends JFrame implements ActionListener
{
   protected JButton button1;
   protected JButton button2;
   ...
   protected JButton buttonN;
   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==button1)
      {
         // do something
         }
      else if(e.getSource()==button2)
      {
         ... you get the picture
```

```
public class SomeGUI extends JFrame
{
   ... button member declarations ...
   protected void buildGUI()
   {
      button1 = new JButton();
      button2 = new JButton();
      ...
      button1.addActionListener(
         new java.awt.event.ActionListener()
         {
            public void
actionPerformed(java.awt.event.ActionEvent e)
            {
               // do something
            }
         }
      );
      .. repeat for each button
```

```
public class SomeGUI extends JFrame
{
   ... button member declarations
   // inner class definitions
   class Button1Handler implements ActionListener
   {
      public void actionPerformed(ActionEvent e)
   {
         // do something
      }
   }
   ... define an inner member class for each button
   protected void buildGUI()
   {
      // initialize the buttons
      button1 = new JButton();
      button2 = new JButton();
      ...
      // register an inner class action listener instance
      // for each button
      button1.addActionListener(new Button1Handler());
      .. repeat for each button
```

Tugas

1. Buatlah ringkasan mengenai Abstract class, Interface, dan Inner Class!

1. Oracle Java Documentation, The Java™ Tutorials, https://docs.oracle.com/javase/tutorial/, Copyright © 1995, Oracle 2015.
2. Tita Karlita, Yuliana Setrowati, Rizky Yuniar Hakkun, Pemrograman Berorientasi Obyek, PENS-2012
3. Sun Java Programming, Sun Educational Services, Student Guide, Sun Microsystems, 2001.
4. John R. Hubbard, Programming With Java, McGraw-Hill, ISBN: 0-07-142040-1, 2004.
5. Patrick Niemeyer, Jonathan Knudsen, Learning Java, O'reilly, CA, ISBN: 1565927184, 2000.
6. Philip Heller, Simon Roberts, Complete Java 2 Certification Study Guide, Third Edition, Sybex, San Francisco, London, ISBN: 0-7821-4419-5, 2002.
7. Herbert Schildt, The Complete Reference, Java™ Seventh Edition, Mc Graw Hill, Osborne, ISBN: 978-0-07-163177-8, 2007