PERTEMUAN KESEMBILAN

Nested Class, Inner Class, Nested Anonymous Class

Prepared by Adi Wahyu Pribadi

Java Nested Class dan Inner Class

Di Java, kita dapat mendeklarasikan Class di dalam Class, yang biasa dinamakan Nested Class. Terdapat dua macam nested Class:

- Static Nested Class
- Non-static Nested Class

Non-Static Nested Class (Inner Class)

Non-static nested class adalah class di dalam class. Class tersebut memiliki akses ke anggota dari class luarnya. Non-static nested class biasanya disebut sebagai `inner Class`.

```
package app;

class CPU {
    double price;
    // nested class
    class Processor{
        // members of nested class
        double cores;
        String manufacturer;

        double getCache(){
            return 4.3;
        }
    }

    // nested protected class
    protected class RAM{
        // members of protected nested class
        double memory;
```

```
        String manufacturer;

        double getClockSpeed(){
            return 5.5;
        }
    }
}

public class App {
    public static void main(String[] args) {

        // create object of Outer class CPU
        CPU cpu = new CPU();

       // create an object of inner class Processor using outer class
        CPU.Processor processor = cpu.new Processor();

        // create an object of inner class RAM using outer class CPU
        CPU.RAM ram = cpu.new RAM();
        System.out.println("Processor Cache = " + processor.getCache());
        System.out.println("Ram Clock speed = " + ram.getClockSpeed());
    }
}
```

Output:

```
Processor Cache = 4.3
Ram Clock speed = 5.5
```

Pada program di atas, terdapat dua nested Class: Processor dan RAM di dalam outer Class: CPU. Kita dapat mendeklarasikan inner Class sebagai protected. Sehingga kita dapat mendeklarasikan class RAM sebagai protected.

Di dalam App class

- Kita membuat instance dari outer class CPU dinamakan cpu.
- Menggunakan instance dari outer class, kemudian kita membuat objek dari inner Class:

```
CPU.Processor processor = cpu.new Processor;
CPU.RAM ram = cpu.new RAM();
```

Catatan: Kita gunakan operator dot/titik (.) untuk membuat instance dari inner Class menggunakan outer class.

Mengakses Member Outer Class dari Inner Class

Kita dapat mengakses members dari outer Class menggunakan keyword `this`.

```
package app;
class Car {
    String carName;
    String carType;
    // assign values using constructor
    public Car(String name, String type) {
        this.carName = name;
        this.carType = type;
    }
    // private method
    private String getCarName() {
        return this.carName;
    }

    // inner class
    class Engine {
        String engineType;
        void setEngine() {

           // Accessing the carType property of Car
            if(Car.this.carType.equals("4WD")){

                // Invoking method getCarName() of Car
                if(Car.this.getCarName().equals("Crysler")) {
                    this.engineType = "Bigger";
                } else {
                    this.engineType = "Smaller";
                }

            }else{
                this.engineType = "Bigger";
```

```
            }
        }
        String getEngineType(){
            return this.engineType;
        }
    }
}
public class App {
    public static void main(String[] args) {

        // create an object of the outer class Car
        Car car1 = new Car("Mazda", "8WD");

        // create an object of inner class using the outer class
        Car.Engine engine = car1.new Engine();
        engine.setEngine();
        System.out.println("Engine Type for 8WD= " +
engine.getEngineType());

        Car car2 = new Car("Crysler", "4WD");
        Car.Engine c2engine = car2.new Engine();
        c2engine.setEngine();
        System.out.println("Engine Type for 4WD = " +
c2engine.getEngineType());
    }
}
```

Output:

```
Engine Type for 8WD= Bigger
Engine Type for 4WD = Smaller
```

Pada contoh di atas, kita memiliki Class `Engine` sebagai inner Class dari Outer Class `Car`. Sehingga pada baris berikut, digunakan keyword `this` untuk mengakses variable carType dari Outer Class.

```
if (Car.this.carType.equals("4WD") ) { ... }
```

Kita juga dapat mengakses method dari outer Class dari dalam inner Class

```
if (Car.this.getCarName().equals("Crysler") ) { .. }
```

Catatan

- Java memperlakukan inner class sama seperti anggota class biasa.
- Sejak inner class adalah member dari outer class, kita dapat mengakses modifiers seperti private, protected ke dalam inner class dimana tidak mungkin dikerjakan pada class normal.
- Sejak nested class anggota dari outer class, kita gunakan notasi titik/dot (.) untuk mengakses nested class dan anggotanya.
- Menggunakan nested class akan membuat kode lebih mudah dibaca dan lebih baik dalam pembungkusan (paradigma *encapsulation*).
- Non-static nested class (inner class) memiliki akses ke anggota dari outer class, walaupun mereka dideklarasikan sebagai private.

Nested Static Class

Di Java, kita juga dapat mendefinisikan Static class di dalam sebuah Class. Class tersebut dinamakan static nested class. Static nested classes TIDAK DINAMAKAN static inner classes.

Tidak seperti inner Class, static nested class tidak dapat mengakses variable dari outer Class. Karena static nested class tidak membutuhkan instance dari outer class

```
OuterClass.NestedClass obj = new OuterClass.NestedClass();
```

Di sini, kita membuat objek dari static nested class. Jadi outer class tidak dapat direferensikan menggunakan this.

```
package app;
class MotherBoard {
   // static nested class
   static class USB {
       int usb2 = 2;
       int usb3 = 1;
       int getTotalPorts() {
           return usb2 + usb3;
       }
   }
}

public class Main {
   public static void main(String[] args) {
       // create an object of the static nested class
       // using the name of the outer class
       MotherBoard.USB usb = new MotherBoard.USB();
       System.out.println("Total Ports = " + usb.getTotalPorts());
   }
}
```

Output:

```
Total Ports = 3
```

Contoh lain

```
class Animal {

   // inner class
   class Reptile {
      public void displayInfo() {
        System.out.println("I am a reptile.");
      }
   }

   // static class
   static class Mammal {
      public void displayInfo() {
        System.out.println("I am a mammal.");
      }
   }
}

class Main {
   public static void main(String[] args) {
      // object creation of the outer class
      Animal animal = new Animal();

      // object creation of the non-static class
      Animal.Reptile reptile = animal.new Reptile();
      reptile.displayInfo();

      // object creation of the static nested class
      Animal.Mammal mammal = new Animal.Mammal();
      mammal.displayInfo();

   }
}
```

Output:

```
I am a reptile.
I am a mammal.
```

Pada contoh di atas terdapat dua nested class yaitu `class Reptile` dan `static class Mammal`.

Untuk membuat objek dari non-static class Reptile digunakan

```
Animal.Reptile reptile = animal.new Reptile()
```

Untuk membuat objek dari static class Mammal digunakan

```
Animal.Mammal mammal = new Animal.Mammal()
```

Catatan:

Hanya nested class yang dapat dijadikan static. Outer class tidak dapat dijadikan static.

Nested Anonymous Class

Nested class yang tidak memiliki nama dinamakan anonymous class. Sebuah anonymous class harus didefinisikan di dalam sebuah class. Jadi bisa dikatakan sebagai anonymous inner class. Sintaksnya adalah sebagai berikut:

```
Class outerClass {
   // defining anonymous class
   Object1 = new Type(parameterList) {
      // body of the anonymous class
   };
}
```

Anonymous class biasanya extend subclass atau implement interface

Jadi, Type bisa berupa:

- Superclass yang di mana anonymous class extends dari Superclass tersebut
- Interface yang di mana anonymous class implements dari Interface tersebut

Contoh 1: Nested Anonymous Class Extending Class

```
package app;
class Polygon {
   public void display() {
      System.out.println("Inside the Polygon class");
   }
}
class AnonymousDemo {
   public void createClass() {
      // creation of anonymous class extending class Polygon
      Polygon p1 = new Polygon() {
         public void display() {
            System.out.println("Inside an anonymous class.");
         }
      };
      p1.display();
   }
}
```

```
class Main {
   public static void main(String[] args) {
       AnonymousDemo an = new AnonymousDemo();
       an.createClass();

       Polygon pol = new Polygon();
       pol.display();
   }
}
```

Output:

```
Inside an anonymous class.
Inside the Polygon class
```

Pada contoh di atas, kita membuat class `Polygon` yang memiliki method `display()`. Kita kemudian membuat anonymous class yang extends class `Polygon` dan override method `display()`.

Ketika program dijalankan, `p1` adalah objek dari anonymous class. Objek tersebut kemudian memanggil method `display()` dari anonymous class.

Contoh 2: Nested Anonymous Class Implementasi Interface

```
interface PolygonInt {
    public void display();
}

class AnonymousDemoInt {
    public void createClass() {
        // anonymous class implementing interface
        PolygonInt p1 = new PolygonInt() {
          public void display() {
             System.out.println("Inside an anonymous class.");
          }
        };
        p1.display();
    }
```

```
}
class App {
    public static void main(String[] args) {
       AnonymousDemoInt an = new AnonymousDemoInt();
       an.createClass();
    }
}
```

Output:

```
Inside an anonymous class.
```

Java Anonymous Class

Anonymous class adalah class yang tidak memiliki nama. Anonymous class tidak boleh memiliki konstruktor. Biasanya anonymous class dibuat sekali pakai dengan tujuan untuk mengimplementasikan interface dan class abstrak secara langsung.