Nested Class

Brigida Arie Minartiningtyas, M.Kom.

Introduction

```
For(int i=0;i<10;i++)
{
    for(int i=0;i<10;i++)
    {
        System.out.print(“*”);
    }
    System.out.println(“”);
}
```

Masih ingat nested loop?

Introduction

```
class OuterClass {
      ...
      class NestedClass {
            ...
      }
}
```

Nested class memiliki arti yang sama yaitu adanya class di dalam class

NESTED CLASS

Static

- Nested class yang dideklarasikan sebagai static disebut sebagai **Static Nested Class**

Non Static

- Nested class yang tidak dideklarasikan sebagai static disebut sebagi **Inner Class**

```
class OuterClass {
    ...

    static class StaticNestedClass{
        ...
    }

    class InnerClass{
    ...
    }
}
```

ATURAN

Inner Class	Kelas dalam (inner class) **diperbolehkan mengakses** atribut/method dari kelas luar meskipun dideklarasikan sebagai private
	Kelas luar **tidak bisa mengakses** atribut/method dari kelas dalamya
Static Nested Class	Kelas dalam (static nested class) dan kelas luar dalam sama-sama **tidak bisa saling mengakses**

Static Nested Class

- berhubungan juga dengan kelas luarnya, tetapi tidak secara langsung berelasi dengan kelas luarnya
- untuk melakukannya nested static class harus mereferense obyek terlebih dahulu

Inner Class

- memiliki hak akses untuk mengakses secara langsung terhadap fungsi-fungsi/atribut yang ada

```
KelasLuar.KelasStaticNested ObjekNested = new KelasLuar.KelasStaticNested();

KelasLuar.KelasDalam ObjekDalam = ObjekLuar.new KelasDalam();
```

Static Nested Class

- Jika kelas Inner bersifat static, maka objek milik kelas Inner dapat dibuat sendiri tanpa melalui kelas Luarnya, (Artinya kelas Inner tidak dapat mengakses attribute ataupun method non static milik kelas Luarnya).

Shadowing

- Jika declarasi atribut memiliki nama yang sama pada kelas inner dan outer maka urutan aksesnya adalah **dimulai dari** deklarasi nama variable yang paling dekat dengan penggunaan variabelnya

```
public class ShadowTest {
    public int x = 0;

    class FirstLevel {
        public int x = 1;

        void methodInFirstLevel(int x) {
        System.out.println("x = " + x);
        System.out.println("this.x = " + this.x);
        System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
}

    public static void main(String[] args){
        ShadowTest st = new ShadowTest();
        ShadowTest.FirstLevel fl = st.new FirstLevel();
        fl.methodInFirstLevel(23);
    }
}
```

Bagaimana Outputnya??

```java
public class ShadowTest {
    public int x = 0;

    class FirstLevel {
        public int x = 1;

        void methodInFirstLevel(int x) {
        System.out.println("x = " + x);
        System.out.println("this.x = " + this.x);
        System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);
        }
}

    public static void main(String[] args){
        ShadowTest st = new ShadowTest();
        ShadowTest.FirstLevel fl = st.new FirstLevel();
        fl.methodInFirstLevel(23);
    }
}
```

ut - JavaApplication1 (run)

```
run:
x = 23
this.x = 1
ShadowTest.this.x = 0
BUILD SUCCESSFUL (total time: 5 seconds)
```

Anonymous Class

- Class anonym sering digunakan pada kelas nested
- Kelas anonimus berarti kelasnya tidak dideklarasikan tersendiri tapi langsung dideklarasikan didalam pembuatan objek

```
Abc a=new abc(){
      private int a;
      ……
}
```

```
class TestAnonymous{
    public static void main(String str[]){
        final int d = 10;
        father f = new father(d);
        father fanon = new father(d){

            // override method parent
            void method (int x){
                System.out.println("Anonymous : " + x);
            }

            // overload method parent
            void method (String str){
                System.out.println("Anonymous : " + str);
            }

            // penambahan method baru di anonymous
            void newMethod(){
                System.out.println("New Method in Anonymous : ");
            }
        };

        //fanon.method("New Number"); // ini akan error
        //fanon.newMethod(); // ini akan error
    }
}
```

Local Class

- Class yang dibuat didalam method
- Kepentingan privatisasi class untuk tujuan method tersebut

```java
public class LocalClassExample {
    static String regularExpression = "[^0-9]";

    public static void validatePhoneNumber(
        String phoneNumber1, String phoneNumber2) {

        final int numberLength = 10;

        // Valid in JDK 8 and later:

        // int numberLength = 10;
    class PhoneNumber {

        String formattedPhoneNumber = null;

        PhoneNumber(String phoneNumber){
            // numberLength = 7;
            String currentNumber = phoneNumber.replaceAll(
              regularExpression, "");
            if (currentNumber.length() == numberLength)
                formattedPhoneNumber = currentNumber;
            else
                formattedPhoneNumber = null;
        }

        public String getNumber() {
            return formattedPhoneNumber;
        }

        // Valid in JDK 8 and later:

          public void printOriginalNumbers() {
              System.out.println("Original numbers are " + phoneNumber1 +
                  " and " + phoneNumber2);
          }
    }

        PhoneNumber myNumber1 = new PhoneNumber(phoneNumber1);
        PhoneNumber myNumber2 = new PhoneNumber(phoneNumber2);

        // Valid in JDK 8 and later:

//          myNumber1.printOriginalNumbers();

        if (myNumber1.getNumber() == null)
            System.out.println("First number is invalid");
        else
            System.out.println("First number is " + myNumber1.getNumber());
        if (myNumber2.getNumber() == null)
            System.out.println("Second number is invalid");
        else
            System.out.println("Second number is " + myNumber2.getNumber());

    }

    public static void main(String... args) {
        validatePhoneNumber("123-456-7890", "456-7890");
    }
}
```

```
public class LocalClassExample {
    static String regularExpression = "[^0-9]";

    public static void validatePhoneNumber(
        String phoneNumber1, String phoneNumber2) {

        final int numberLength = 10;

        // Valid in JDK 8 and later:

        // int numberLength = 10;
    class PhoneNumber {

        String formattedPhoneNumber = null;

        PhoneNumber(String phoneNumber){
            // numberLength = 7;
            String currentNumber = phoneNumber.replaceAll(
              regularExpression, "");
            if (currentNumber.length() == numberLength)
                formattedPhoneNumber = currentNumber;
            else
                formattedPhoneNumber = null;
        }

        public String getNumber() {
            return formattedPhoneNumber;
        }

        // Valid in JDK 8 and later:

          public void printOriginalNumbers() {
              System.out.println("Original numbers are " + phoneNumber1 +
                  " and " + phoneNumber2);
          }
    }

        PhoneNumber myNumber1 = new PhoneNumber(phoneNumber1);
        PhoneNumber myNumber2 = new PhoneNumber(phoneNumber2);

        // Valid in JDK 8 and later:

//          myNumber1.printOriginalNumbers();

        if (myNumber1.getNumber() == null)
            System.out.println("First number is invalid");
        else
            System.out.println("First number is " + myNumber1.getNumber());
        if (myNumber2.getNumber() == null)
            System.out.println("Second number is invalid");
        else
            System.out.println("Second number is " + myNumber2.getNumber());

    }

    public static void main(String... args) {
        validatePhoneNumber("123-456-7890", "456-7890");
    }
}
```

```
First number is 1234567890
Second number is invalid
BUILD SUCCESSFUL (total time: 2 seconds)
```

Kegunaan Nested Class

- Salah satu cara untuk mengelompokan kelas secara logic yang digunakan dalam tempat yang sama, jika suatu kelas hanya digunakan satu kelas ditempat lain maka bisa disatukan dalam nested class
- Meningkatkan enkapsulasi:
  - Class yang ada di dalam dapa dilakukan privatisasi dari kelas luar
- Dapat mudah dibaca dan dimaintenance:
  - Kelas kecil yang disatu tempatkan pada kelas besar, sehingga tidak perlu bolak-balik, membuka slide

Non Static Inner Class yang dideklarasikan di dalam Class

```
public class Luar {
    private String variabelLuar = "Variabel Luar";

    class Dalam {
        String variabelDalam = "Variabel Dalam";


    public void bicara (){
        System.out.println(variabelDalam);
        System.out.println(variabelLuar);
    }
    }
}
```

```
public class TestLuar {
    public static void main (String args[]){
        Luar l = new Luar();
        Luar.Dalam d = l.new Dalam();
        d.bicara();
    }
}
```

```
public class MOuter {
    private int m = (int)(1*100);

    public static void main (String arg[]){
        MOuter that = new MOuter();
        that.go ((int)(2*100), (int)(3*100));
    }

    public void go (int x, final int y){
        int a = x+y;
        final int b = x-y;

        class MInner{
            public void method(){
                System.out.println("Nilai m adalah : "+m);
                System.out.println("Nilai x adalah : "+x);
                System.out.println("Nilai x adalah : "+y);
                System.out.println("Nilai x adalah : "+a);
                System.out.println("Nilai x adalah : "+b);
            }
        }

        MInner that = new MInner();
        that.method();
    }
}
```

Inner Class yang dideklarasikan di dalam method

```
public class TestStaticInnerClass1 {
    static String test = "Outer class static field";
    String instFld = "This is an instance field";

    public static void main (String [] args){
        System.out.println(Inner.value);
        new Inner();
    }

    static class Inner {
    static int value = 100;

    Inner (){
        System.out.println("New static inner class");
        System.out.println(test);
        System.out.println(instFld);
        TestStaticInnerClass1 tsi = new TestStaticInnerClass1();
        System.out.println(tsi.instFld);
    }
}
}
```

Static Inner Class, Error

Local Inner Class

```
public class Outer2 {
    static String classFld = "static class members are accessible";
    String instFld = "instance field of enclosing class are accessible";

    void display (){
        final String str = "only final method variables are accessible";
        class Local{
        Local(){
            System.out.println(str);
            System.out.println(classFld);
            System.out.println(instFld);
        }
    }
       new Local();

    }
}
```

```
public class TestLocalInner {
    public static void main(String [] args){
        Outer2 o = new Outer2();
        o.display();
    }
}
```