Inner Class

Inner Class adalah class yang berada (didefinisikan) di dalam sebuah Class

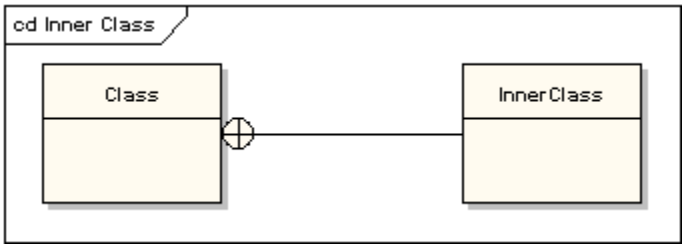
Contoh:

```
class OuterClass {

    class InnerClass{


    }
}
```

Class diagram dalam UML:



> One of the key benefits of an inner class is the "special relationship" an *inner class instance* shares with *an instance of the outer class*. That "special relationship" gives code in the inner class access to members of the enclosing (outer) class, *as if the inner class were part of the outer class*. In fact, hat's exactly what it means: the inner class *is* a part of the outer class. Not just a "part" but a full-fledged, card-carrying *member* of the outer class. Yes, an inner class instance has access to all members of the outer class, *even those marked private*. (Relax, that's the whole point, remember? We want this separate inner class instance to have an intimate relationship with the outer class instance, but we still want to keep everyone *else* out. And besides, if you wrote the outer class, then you also wrote the inner class! So you're not violating encapsulation; you *designed* it this way.)
> *[SCJP Sun® Certified Programmer for Java™ 6 Study Guide (Exam 310-065)]*

Ada 4 jenis inner class, yaitu :
1. “Regular” Inner class
2. Method-local inner class
3. Anonymous inner class
4. Static nested classes

Regular Inner Class

Istilah regular inner class (inner class biasa) dimaksudkan untuk inner class selain jenis 2, 3, dan 4 ☺
Contoh:

```
class OuterClass {

    class InnerClass{


    }
}
```

Jika dicompile dengan perintah :

```
%java Outer.java
```

Maka akan tercipta 2 buah file .class, yaitu :

```
Outer.class
MyOuter$MyInner.class.
```

Beberapa hal tentang inner class:

- Instance inner class memiliki akses ke semua member dari outer class (termasuk member outer class yang ber access modifier private).
- Sebuah inner class tidak dapat memiliki member static
  Misalnya: method public static main() tidak bisa dibuat di dalam innre class.
- Aturan inner class dalam mereferensi dirinya sendiri atau instance dari outer class adalah sebagai berikut :
  - Untuk merujuk pada dirinya sendiri (instance dari inner class) dari dalam inner class, dapat digunakan referensi `this` atau `OuterClass.InnerClass.this`.
  - Untuk merujuk pada instance dari outer classnya dari dalam inner class, dapat digunakan referensi `OuterClass.this`.
- Untuk membuat instance dari inner class, kita harus memiliki instance dari outer class terlebih dahulu. Tidak ada pengecualisan untuk aturan ini.
  Untuk membuat instance dari inner class, terdapat 2 cara, yaitu :
  1. Dari dalam outer class
  2. Dari luar outer class

Contoh:

```
class A {
    class B { //class B ada di dalam class A
      void sebuahMethodPadaB(){
          System.out.println("Hello, saya method di kelas B");
          System.out.println("kelas B berada dalam kelas A");
        }

    }
    void sebuahMethodPadaA() {
        B b = new B(); //object b diinstance  pada sebuah method di class A
        System.out.println("Hello, saya method kelas A");
        System.out.println("Saya membuat object dari inner class B");
        b.sebuahMethodPadaB();
    }
}
```

```
public class C {
    public static void main(String[] args) {
        A.B b = new A().new B(); //kelas B dibuat di luar kelas A
        b.sebuahMethodPadaB();
        A a=new A();
        a.sebuahMethodPadaA();
    }
}
```

Method-Local Inner Class

- Method-local inner class adalah inner class yang dideklarasikan di dalam method.
- Mendeklarasikan method-local inner class bukan berarti kita telah membuat instance dari class tersebut. Jadi, sebelum inner class tersebut digunakan, kita harus membuat instancenya dari suatu tempat di dalam method dan setelah definisi inner class tersebut.
  Contoh :

```
class A {
    void methodDiA() {   // definis method di A
        class B {        // definisi class B didalam method milik A
            int i = 10;
            void methodDiB() { // deklarasi method di B
                System.out.println(" i = " + i);
            }                  // end method B
        }                      // end class B

        B b = new B();
        b.methodDiB();
    }                          //end methodDiA
}                              //end class A
```

- Method-local inner class hanya dapat diinstansiasi dari dalam method yang mendefinisikan method-local inner class tersebut.
- Instance method-local inner class memiliki akses ke semua member dari outer class (termasuk member outer class yang ber access modifier `private`).
  Contoh :

```
class A {
    void methodDiA() {   // definisi method di A
        private int i=10;
        class B {        // definisi class B didalam method milik A
            int i = 10;
            void methodDiB() { // deklarasi method di B
                System.out.println(" i = " + i); //akses member A
            }                  // end method B
        }                      // end class B

        B b = new B();
        b.methodDiB();
    }                          //end methodDiA
}                              //end class A
```

- Instance dari method-local inner class tidak dapat mengakses local variabel (termasuk parameter) dari method dimana method-local inner class tersebut didefinisikan. Kecuali bila variabel tersebut bermodifier `final`.
  Contoh :

```
class A { //1
    void lakukanSesuatu() { //2
        int nonFinalVar = 10;
        final int finalVar = 11;
        class B { //3
```

```
            void aksesLocalVar() { //4
                //ERROR
                System.out.println("nonFinalVar = " + nonFinalVar);
                //TIDAK ERROR
                System.out.println("finalVar = " + finalVar);
            } //4
        } //3

        B b = new B();
        b.aksesLocalVar();
    } //2
} //1
```

- Method-local inner class yang didefinisikan <u>di dalam static method</u> tidak dapat mengakses non-static member dari outer classnya
- Modifier-modifier yang dapat diberikan pada method-local inner class adalah :
    1. `abstract` (tidak dapat digabung dengan `final`)
    2. `final` (tidak dapat digabung dengan `abstract`)

Anonymous Inner Class

- Anonymous inner class adalah suatu inner class yang dideklarasikan tanpa nama kelas.
- Anonymous inner class pasti adalah salah satu dari 2 hal berikut :
  - Subclass dari suatu class yang telah dideklarasikan
  - Class implementasi dari suatu interface
- Suatu anonymous inner class tidak dapat secara bersamaan menjadi subclass dari class yang telah dideklarasikan dan juga sebagai kelas implementasi dari suatu interface.
- Tujuan utama dari anonymous inner class adalah **mengoverride** satu atau lebih method dari super classnya atau **mengimplement** semua method dari suatu interface.
- Anonymous inner class tidak dapat mengimplement lebih dari sebuah interface.
- Anonymous inner class selalu dibuat sebagai bagian dari suatu statement.

Contoh Anonymous inner class sebagai subclass (melakukan override)

```
class A {
    int i = 10;
    void methodDiA() {
        System.out.println("i = " + i);
    }
}

public class BBB {    //1
    public static void main(String[] args) {    //2
        A aa = new A() {   //3      <<== lihat ini
            void methodDiA() { //4 <<== lihat ini
                i++;
                System.out.println("i = " + i);
            }   //4
        };  //3

        aa.methodDiA();
    }   //2
}   //1
```

Variabel referensi `aa` di atas mereferensi ke suatu instance anonymous inner class yang merupakan subclass dari class A. Jadi, variabel referensi `aa` bukan mereferensi ke instance dari kelas A.

- Contoh anonymous inner class sebagai implementasi suatu interface :

```
interface A {
    public void methodDiA();
}

interface B extends A {
    public void methodDiB();
}

public class CC {
    public static void main(String[] args) {
        B b = new B() {
            public void methodDiA() {
                System.out.println("Ini method di A");
```

```
            }

            public void methodDiB() {
                System.out.println("Ini method di B");
            }
        };

        b.methodDiA();
        b.methodDiB();
    }
}
```

- Anonymous inner class merupakan salah satu bentuk polymorphisme, oleh karena itu, method yang dapat dipanggil dari anonymous inner class adalah method yang dideklarasikan di super class atau interfacenya (meskipun di dalam anonymous inner class dapat dideklarasikan method-method yang tidak ada di super class atau interfacenya).

  Contoh :

```
class A {
    int i = 10;
    void lakukanSesuatu() {
        i--;
        System.out.println("i = " + i);
    }
}

public class Test {
    public static void main(String[] args) {
        A a = new A() {
            void lakukanSesuatu() {
                i++;
                System.out.println("i = " + i);
            }

            //Di bawah ini adalah method yang tidak ada di class A
            void newMethod() {
                System.out.println("Ini adalah method baru");
            }
        };

        a.lakukanSesuatu(); //Tidak error
        a.newMethod(); //ERROR !!!!!!!
    }
}
```

- Anonymous inner class dapat diletakkan sebagai argument dari suatu method.

  Contoh :

```
class A {
    void lakukanSesuatu() {
        System.out.println("Ini isi aslinya");
    }
```

```
}

class B {
    static void demoAnonymousInnerClassSebagaiArgument(A a) {
        a.lakukanSesuatu();
    }
}

public class BelajarAnonymous3 {
    public static void main(String[] args) {
        B.demoAnonymousInnerClassSebagaiArgument(new A() {
            void lakukanSesuatu() {
                System.out.println("Ini method di anonymous inner
class");
            }
        });
    }
}
```

Tugas:

- Cobalah contoh-contoh penggunaan inner class!
- Di dalam sebuah Mobil terdapat MusicPlayer. MusicPlayer dapat dilihat sebagai sebuah object baru yang merupakan bagian dari mobil dan tidak bisa terpisahkan dari mobil. Implementasikan dalam OOP dengan menerapkan inner class.

Mengapa ada inner class?

Sampai di sini kita sudah mengenal syntax dan semantic bagaimana cara membuat inner class dapat bekerja tetapi belum terjawab kenapa harus ada inner class. Mengapa desainer Java harus repot-repot menambahkan fitur ini?

Secara umum, inner class meng-inherits sebuah class atau meng-implement sebuah interface, dan dan kode-kode pada inner class memanipulasi object-object outer class tempat inner class dibuat. Sehingga kita dapat katakan bahwa sebuah inner class menyediakan seperti sebuah jendela untuk outer class.

Tapi bukankah jika kita ingin mereference ke sebuah interface, bukankah kita bisa cukup melakukannya pada outer class untuk mengimplement sebuah interface? Jawabanya adalah, jika memang itu cukup maka lakukan saja.

At this point you've seen a lot of syntax and semantics describing the way inner classes work, but this doesn't answer the question of why they exist. Why did the Java designers go to so much trouble to add this fundamental language feature?

Typically, the inner class inherits from a class or implements an interface, and the code in the inner class manipulates the outer-class object that it was created within. So you could say that an inner class provides a kind of window into the outer class.

A question that cuts to the heart of inner classes is this: If I just need a reference to an interface, why don't I just make the outer class implement that interface? The answer is "If that's all you need, then that's how you should do it." So what is it that distinguishes an inner class implementing an interface from an outer class implementing the same interface? The answer is that you can't always have the convenience of interfaces—sometimes you're working with implementations.

Jadi alasan adanya sebuah inner class adalah:

Masing-masing class dapat secara bebas men

So the most compelling reason for inner classes is:

> *Each inner class can independently inherit from an implementation. Thus, the inner class is not limited by whether the outer class is already inheriting from an implementation.*

Without the ability that inner classes provide to inherit—in effect—from more than one concrete or **abstract** class, some design and programming problems would be intractable. So one way to look at the inner class is as the rest of the solution of the multiple-inheritance problem. Interfaces solve part of the problem, but inner classes effectively allow "multiple implementation inheritance." That is, inner classes effectively allow you to inherit from more than one non-interface.