CLASS DAN OBJECT

OBJEKTIF :

1. Mahasiswa Mampu Memahami *Class* dan *Object* pada Java.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA dalam Pembuatan *Class* dan *Object*.

PENDAHULUAN

Class adalah *blueprint/template* yang mendefinisikan tipe data suatu *object*. *Class* dapat digunakan untuk membuat satu atau lebih *object*. Contoh dari *class* adalah *class* Kendaraan, *class* Buah, *class* Hari, dan sebagainya.

Object adalah *instance* dari *class*. Sebuah *instance* adalah representasi nyata dari *class* itu sendiri. Contoh *object* adalah dari *class* Buah dapat dibuat *object* Apel, Mangga, Anggur, dan sebagainya.

7.1 CLASS

Class merupakan sarana untuk mengumpulkan fungsi dan variabel dalam satu tempat dan dapat saling berinteraksi sehingga membentuk sebuah program.

Sintaks pembuatan *class* pada Java :

```
class NamaClass {
    String atribut1;
    String atribut2;

    void namaMethod(){ ... }
    void namaMethodLain(){ ... }
}
```

Aturan penulisan pada *class* :

1. Nama *class* tidak boleh diambil dari nama *keyword* (kata kunci) dari Bahasa pemrograman Java.
2. Nama *class* boleh menggunakan huruf, angka (0-9), garis bawah (*underscore*) dan simbol *dollar* ($), namun penggunaan garis bawah dan simbol lebih baik dihindari, serta penggunaan angka tidak dapat digunakan pada karakter pertama.
3. Nama *class* sebaiknya diawali dengan huruf besar (kapital).

Sebelum kata kunci (*keyword*) *class*, dapat dideklarasikan suatu modifier. *Keyword public, abstract*, dan final dapat digunakan sebagai *modifier class*. *Class* hanya dapat diterapkan *access modifier* berupa *default* (tanpa *modifier*) atau *public* yang mengatur apakah *class* dapat diakses *class-class* lain di luar *package*.

1. *public*

Modifier public akan membuat *class* bisa di akses dari mana saja.

2. *abstract*

Pada kelas ini, tidak ada instan kelas yang dapat diciptakan. Kelas ini dapat berisi *method abstract.*

3. final

Modifier final digunakan agar suatu *class* bersifat final. Kelas ini tidak dapat diperluas atau diturunkan lagi.

7.2.1 KELAS PUBLIC

Kelas dapat diacu sembarang kelas di paket-paket manapun. Jika *modifier* public tidak digunakan, maka kelas hanya dapat diacu kelas-kelas lain yang sepaket.

7.2.2 KELAS ABSTRACT

Kelas abstract adalah kelas yang diimplementasikan secara parsial dan tujuannya untuk kenyamanan perancangan. Kelas-kelas abstract disusun dari satu *method* abstract atau lebih di mana *method* - *method* dideklarasikan tapi tanpa badan (tidak diimplementasikan). Terdapat batasan-batasan penggunaan abstract:

1. Kita tidak dapat membuat *constructor* yang abstract.
2. Kita tidak dapat membuat *method* static yang abstract. Batasan ini menyatakan bahwa *method* static dideklarasikan untuk seluruh kelas, maka tidak akan ada implementasi turunan untuk *method* static yang abstract.
3. Kita tidak diizinkan membuat *method private* yang abstract. Ketika kita menurunkan kelas dari superkelas dengan *method* abstract, maka kita harus melakukan penimpaan dan mengimplementasikan semua *method* abstract di kelas itu atau kita tidak akan dapat melakukan instanisasi kelas baru dan itu akan menyebabkan kelas masih tetap abstract karena kita tidak dapat melakukan penyimpangan terhadap *method private* yang abstract.

7.2.3 KELAS FINAL

Ketika kelas dideklarasikan dengan modifier final, maka kelas tidak dapat diperluas atau tidak membuat subkelas dari kelas itu sendiri. Contoh kelas final adalah java.lang.System. Pendeklarasian kelas final mencegah perluasan yang tidak diinginkan. Kelas final memungkinkan kompilator melakukan optimasi dalam menjalankan *method* di kelas tersebut.

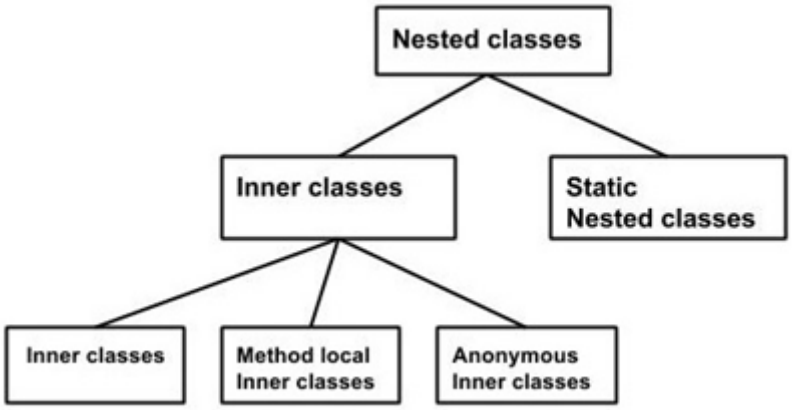
Di Java, menulis kelas di dalam kelas lain diperbolehkan. Kelas yang ditulis di dalam kelas disebut *nested class*, dan kelas yang menampung *inner class* disebut *outer class.* Berikut adalah sintaks untuk menulis *nested class :*

```
class Outer_Demo {
   class Inner_Demo {
   }
}
```

Di sini, kelas Outer_Demo adalah *outer class* dan kelas Inner_Demo adalah *nested class.*

Nested class dibagi menjadi dua jenis :

1. *Non static nested class*
2. *Static nested class*



7.2.4 INNER CLASS (NON STATIC NESTED CLASS)

Inner class adalah mekanisme keamanan di Java. Kelas tidak dapat dikaitkan dengan *access modifier private*, tetapi jika telah memiliki kelas sebagai anggota kelas lain, maka *inner class* dapat dijadikan *private*. Inner class terdiri dari tiga jenis tergantung pada bagaimana dan dimana pendefinisiannya, yaitu :

1. *Inner class*
2. *Method-local inner class*
3. *Anonymous inner class*

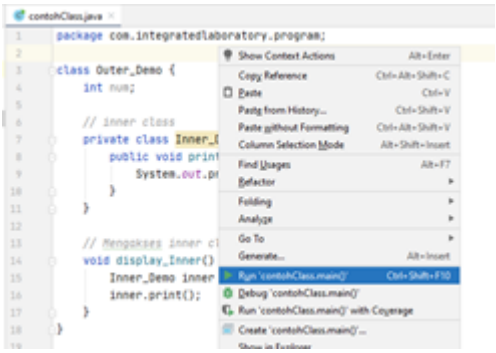
Inner class

Membuat *inner class* cukup sederhana, yaitu hanya perlu menulis kelas di dalam kelas. *Inner class* bisa menjadi *private*. Setelah mendeklarasikan *inner class* sebagai *private*, kelas tersebut tidak dapat diakses dari objek di luar kelas. Berikut adalah program untuk membuat *inner class* :

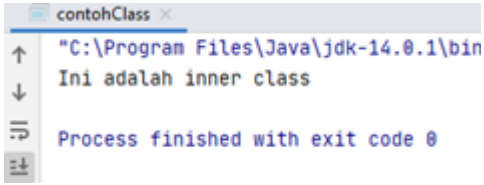
```
package com.integratedlaboratory.program;
class Outer_Demo {
    int num;
    // inner class
    private class Inner_Demo {
        public void print() {
        System.out.println("Ini adalah inner class");
        }
}
    // Mengakses inner class dari method
    void display_Inner() {
        Inner_Demo inner = new Inner_Demo();
        inner.print();
 }
}
public class contohClass{
    public static void main(String args[]) {
        // Instansiasi outer class
        Outer_Demo outer = new Outer_Demo();
        // Mengakses method display_Inner()
        outer.display_Inner();
    }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



Di sini, Outer_Demo adalah *outer class*, Inner_Demo adalah *inner class*, display_Inner () adalah *method* yang didalamnya kita gunakan untuk membuat instance *inner class*, dan *method* ini dipanggil dari *method* utama .

Method-local inner class

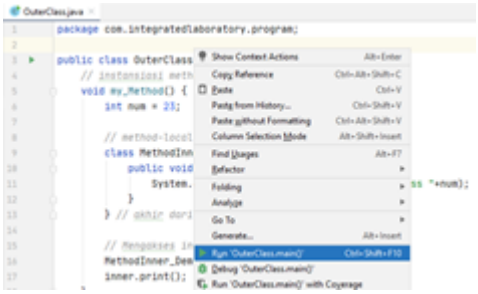
Di Java, kita dapat menulis kelas di dalam *method* dan akan menjadi tipe lokal. Seperti variabel lokal, ruang lingkup *inner class* dibatasi di dalam *method*. *Method-local inner class* bisa dibuat *instance*-nya hanya dalam *method* dimana *inner class* didefinisikan. Program berikut menunjukkan bagaimana menggunakan *method-local inner class*.

```
package com.integratedlaboratory.program;
public class OuterClass {
 // instansiasi method pada outer class
 void my_Method() {
   int num = 23;

  // method-local inner class
  class MethodInner_Demo {
    public void print() {
      System.out.println("This is method inner class "+num);
         }
      } // akhir dari inner class


   // Mengakses inner class
      MethodInner_Demo inner = new MethodInner_Demo();
      inner.print();
   }

   public static void main(String args[]) {
   OuterClass outer = new OuterClass();
```

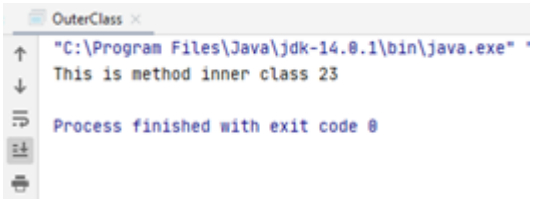
```
    outer.my_Method();
     }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



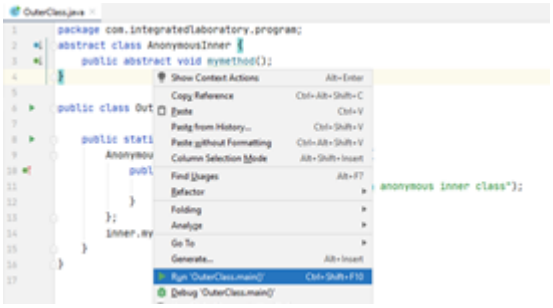
Anonymous Inner Class

Inner class yang dideklarasikan tanpa nama kelas dikenal sebagai *anonymous inner class*. Contoh *anonymous inner class* adalah sebagai berikut :

```
package com.integratedlaboratory.program;
abstract class AnonymousInner {
   public abstract void mymethod();
}

public class OuterClass {

   public static void main(String args[]) {
   AnonymousInner inner = new AnonymousInner() {
   public void mymethod() {
   System.out.println("Ini adalah contoh anonymous inner class");
      }
 }
        inner.mymethod();
    }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :

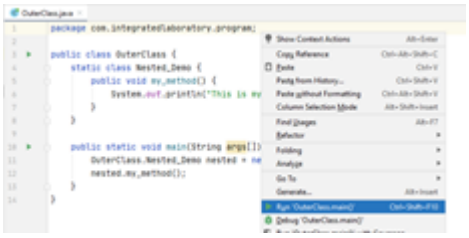
7.2.5 INNER CLASS (STATIC NESTED CLASS)

Static Nested Class tidak memiliki akses ke variabel instance dan method pada *outer class*. Sintaks *static nested class* adalah sebagai berikut :

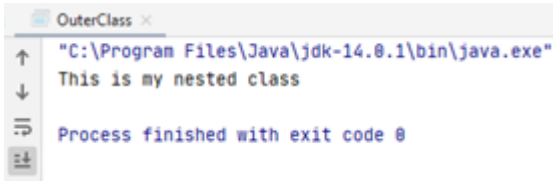
```
package com.integratedlaboratory.program;
public class OuterClass {
    static class Nested_Demo {
        public void my_method() {
          System.out.println("This is my nested class");
        }
}
public static void main(String args[]) {
OuterClass.Nested_Demo nested = new OuterClass.Nested_Demo();
nested.my_method();
    }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



7.2 OBJECT

Object adalah kesatuan entitas yang merupakan representasi nyata dari sebuah *class*. Di Java, kata kunci "*new*" digunakan untuk membuat *object* baru. Sintaks pembuatan object pada Java :

```
nama_class nama_object = new nama_constructor()
```

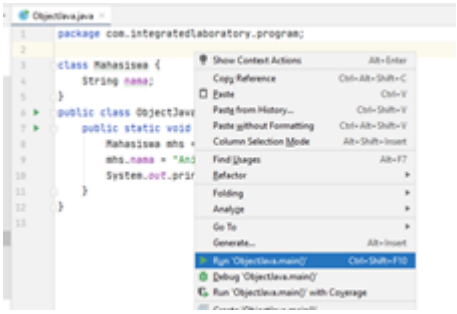
Setelah mendeklarasikan kata kunci *new*, terdapat satu metode *constructor* dengan nama yang sama dengan nama *class*. Tujuan dari *constructor* adalah untuk melakukan inisialisasi *object* baru.

Contoh program pembuatan *object* :

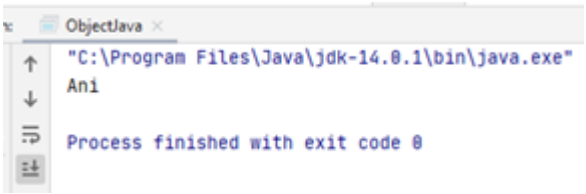
```
class Mahasiswa {
      String nama;
}
      public class ObjectJava {
      public static void main (String[] args) {
      Mahasiswa mhs = new Mahasiswa();
      mhs.nama = "Ani";
      System.out.println (mhs.nama);
    }
 }
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan *Run* pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* Java seperti berikut:



Output :



REFERENSI

[1] Hariyanto, Bambang. 2010. Esensi-Esensi Bahasa Pemrograman Java Revisi Ketiga. Bandung: Informatika.

[2] Tutorialsoint. Java - Inner Class. Diambil dari : https://www.tutorialspoint.com/java/java_innerclasses.htm (23 September 2020)