Pemrograman Berorientasi Objek

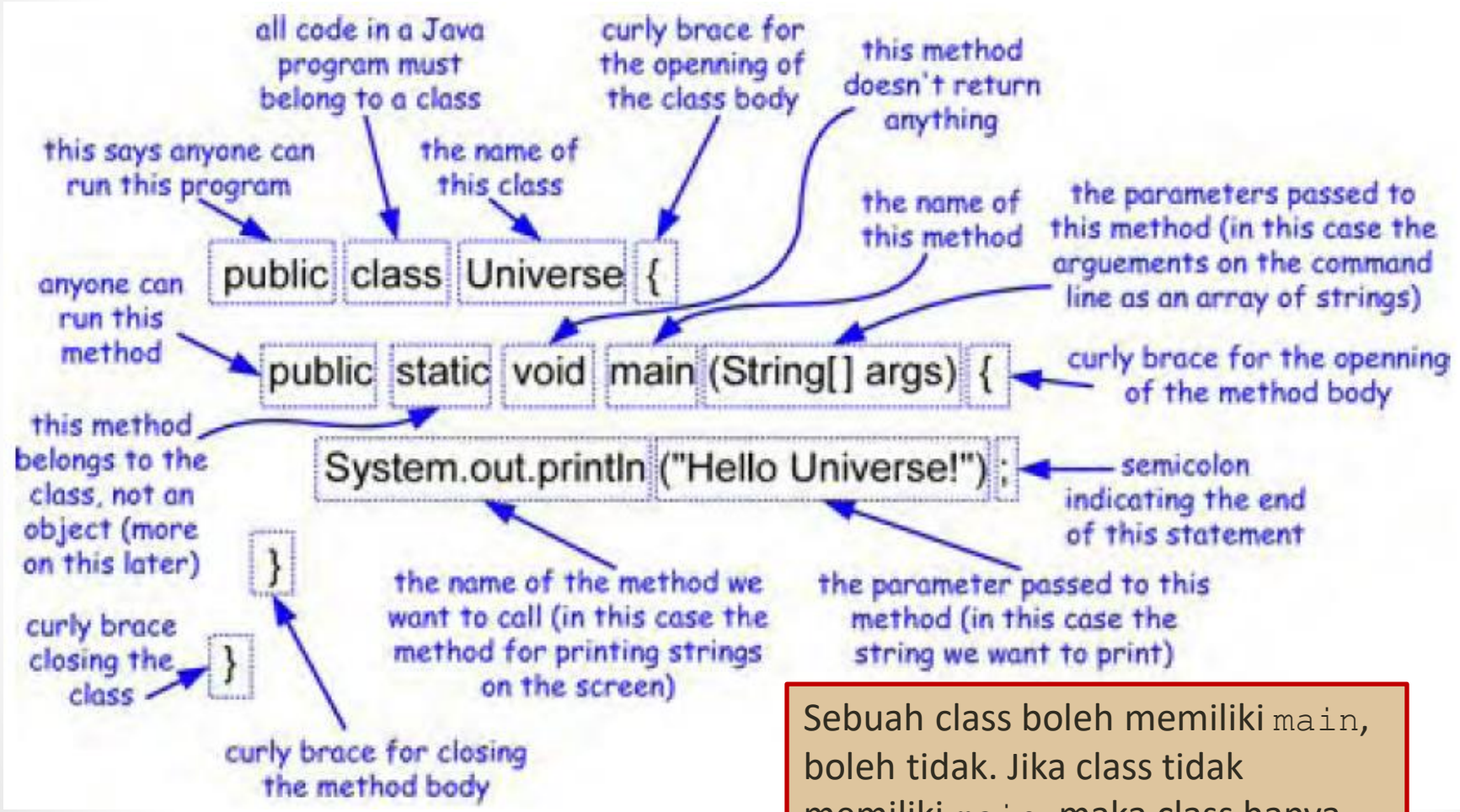
Pertemuan 3: Implementasi Class dan Objek dalam Java

Class

- Merupakan cara untuk mengikat data beserta fungsi-fungsi yang berkaitan dengan data
- Bila diperlukan, data dan fungsi dapat tersembunyi dari pihak eksternal class
- Ketika sebuah class dibuat, kita membuat sebuah Tipe Data Abstrak yang biasanya terdiri atas:
  - Deklarasi class
  - Definisi fungsi pada class
- Bentuk umum dari class:

```
<class_modifier> class class_name {
   deklarasi attribute
   deklarasi fungsi
}
```

Bentuk Umum Program Java



Sebuah class boleh memiliki `main`, boleh tidak. Jika class tidak memiliki `main`, maka class hanya dapat dipanggil/dijalankan dari class lain yang memiliki `main`

Sekilas tentang Class Modifier

- Merupakan kata kunci opsional yang mendahului penulisan sintaks *class*
- Beberapa macam class modifier:
  - Modifier **`abstract`** yang mendeskripsikan sebuah class yang memiliki method abstrak (method kosong yang harus diimplementasikan, akan dijelaskan kemudian di bagian akhir kuliah)
  - Modifier **`final`** yang mendeskripsikan class yang tidak memiliki subclass
  - Modifier **`public`** yang mendeskripsikan class yang dapat diinstansiasi (dibuat objeknya) atau diekstensi oleh class lain dalam satu package atau beda package (dengan menggunakan perintah **`imports`** class)

Class: Contoh

- Studi Kasus: Pembuatan sebuah aplikasi pemutar musik

```
public class Song {
    private String name;
    private String artist;
    private int duration;

    public void setName(String s){
        name=s;
    }
    public void setArtist(String a){
        artist=a;
}
    public void setDuration(int d){
        duration=d;
    }
}
```

Class: Contoh

- Setiap objek menyimpan 1 lagu dan atributnya disimpan di variable *instance*
- Contoh penggunaan class:

```
public static void main(String[] args){
    Song s1=new Song();
    s1.setName("Roar");
    s1.setArtist("Katy Perry");
    s1.setDuration(300);
}
```

- Suatu saat kita akan memerlukan untuk mencetak isi dari objek Song. Java memiliki perintah `System.out.println()` untuk mencetak ke layar

  `System.out.println(s1)` → `samples.Song@7de6385e`

- Cara di atas hanya dapat menampilkan atribut ID objek saja. Untuk mencetak semua isi atributnya, kita perlu membuat sebuah method (misal diberi nama `cetakLagu()`)

Objek dan Atribut

- Variabel *instance* (yang ada di dalam sebuah class) bersifat *private.* Artinya, yang boleh mengakses variabel tersebut hanyalah objek yang memilikinya. Objek lain (walaupun masih dalam class yang sama) tidak dapat mengaksesnya
- Misal kita ingin membaca isi dari variabel *instance*, maka kita perlu menambahkan method *getter* untuk mengaksesnya. Namanya biasanya menggunakan awalan `get` dan diikuti dengan nama variabel yang akan dibaca. Contoh method untuk membaca isi variabel `name`:

```
public String getName(){
  return this.name;
}
```

- Untuk mengganti isi variabel *instance*, sangat disarankan untuk menggunakan method *setter*. Contoh method untuk mengganti isi variabel `duration`:

```
public void setDuration(int d){
  this.duration=d;
}
```

Class: Contoh

- Contoh method `cetakLagu()`:

```
public class Song {
   private String name;
   private String artist;
   private int duration;

   public void setName(String s){
      name=s;
   }
   public void setArtist(String a){
      artist=a;
   }
   public void setDuration(int d){
      duration=d;
   }
   public void cetakLagu(){
      System.out.println("Song: "+this.name+"--
      "+this.artist+"("+this.duration+" seconds)");
   }
   public static void main(String[] args){
       Song s1=new Song();
       s1.setName("Roar");
       s1.setArtist("Katy Perry");
       s1.setDuration(300);
       s1.cetakLagu();
   }
}
```

Output:

Song: Roar--Katy Perry(300 seconds)

Class Variable dan *Class Method*

- ***Class Variable**:*
  - Variabel yang digunakan bersama oleh semua objek dalam 1 kelas
  - Di Java diawali dengan kata kunci **`static`**
  - Contoh jika kita ingin menambahkan variabel `play` pada class `Song` untuk menunjukkan total seluruh lagu yang pernah dimainkan oleh aplikasi pemutar lagu kita
- ***Class Method**:*
  - Method yang dimiliki oleh class (bukan objek) yang dapat diakses oleh semua objek pada class tersebut
  - Di Java diawali dengan kata kunci **`static`**
  - Pemanggilan class method dalam `main` langsung dengan namanya, tanpa didahului nama objek
  - Contoh jika kita ingin menambah method untuk mengecek apakah sebuah lagu memiliki durasi yang terlalu lama atau tidak. Method ini diletakkan di class baru yang bernama `SongList`

Contoh *Class Variable*

```
1  package samples;
2
3  public class Song {
4      private String name;
5      private String artist;
6      private int duration;
7      private int play;
8      private static int plays;
9
10     public void setName(String s){
11         name=s;
12         play=0;
13     }
14     public void setArtist(String a){
15         artist=a;
16     }
17     public void setDuration(int d){
18         duration=d;
19     }
20     public void cetakLagu(){
21         System.out.println("Song: "+this.name+"--"+this.artist+"("+this.duration+" seconds)");
22     }
23     public void play(){
24         play+=1;
25         plays+=1;
26         System.out.println("Lagu "+this.name+" telah diputar: "+this.play+" kali.");
27         System.out.println("Total semua lagu yang diputar: "+plays);
28     }
30     public static void main(String[] args){
31         Song s1=new Song();
32         s1.setName("Song1");
33         s1.setArtist("Artist1");
34         s1.setDuration(300);
35         s1.cetakLagu();
36         Song s2=new Song();
37         s2.setName("Song2");
38         s2.setArtist("Artist2");
39         s2.setDuration(450);
40         s2.cetakLagu();
41         s1.play();
42         s2.play();
43         s1.play();
44         s1.play();
45     }
46 }
```

Contoh *Class Method*

```
1  package samples;
2
3  public class SongList {
4      public static int MAX_TIME=5*60;
5      public static boolean isTooLong(Song s){
6          return s.getDuration()>MAX_TIME;
7      }
8
9      public static void main(String[] args){
10         Song s1=new Song();
11         s1.setName("Song1");
12         s1.setArtist("Artist1");
13         s1.setDuration(400);
14         System.out.println(isTooLong(s1));
15         Song s2=new Song();
16         s2.setName("Song2");
17         s2.setArtist("Artist2");
18         s2.setDuration(300);
19         System.out.println(isTooLong(s2));
20     }
21 }
```

Penulisan class method

Output:

```
true
false
```

Pengaturan Akses ke Method dari Class

- Akses ke method dari sebuah class dapat dibatasi dengan mode **public**, **protected**, atau **private**.
  - **Public:**
    - semua objek dapat mengakses (dalam 1 class ataupun di luar class)
    - Semua method memiliki mode public secara default
    - Dapat diwariskan ke kelas turunan
  - **Protected:**
    - hanya objek dalam class tersebut (beserta class turunannya) yang dapat mengakses
    - “Family-only” access
    - Dapat diwariskan ke kelas turunan
  - **Private:**
    - Tidak dapat dipanggil secara langsung, dan pengaksesnya hanya diperbolehkan dari objek *self* atau *this*
    - Di hampir semua bahasa PBO, method private tidak diwariskan ke kelas turunan

Skenario 1: *Public Method*

```
1  package samples;
2
3  class A {
4      public void public_method(){
5          System.out.println("Public method in A");
6          protected_method();
7          private_method();
8      }
9      protected void protected_method(){
10         System.out.println("Protected method in A");
11     }
12     private void private_method(){
13         System.out.println("Private method in A");
14     }
15
16     public static void main (String[] args){
17         A a = new A();
18         a.public_method();
19     }
20 }
```

Output:

```
Public method in A
Protected method in A
Private method in A
```

- Protected method dan private method dapat diakses dalam kelas A
- Protected dan private method hanya dapat dipanggil dari dalam method di kelas A

Skenario 2: *Private Method* tidak dapat diwariskan

```
1  package samples;
2
3  class B extends A {
4      public void public_method_in_b(){
5          public_method();
6          protected_method();
7          private_method();
8      }
9      public static void main(String[] args){
10         B b = new B();
11         b.public_method_in_b();
12     }
13 }
```

- Protected method dapat diakses dalam kelas B yang merupakan turunan A
- Private method tidak dapat dipanggil langsung dalam kelas B, hanya dapat dipanggil oleh A

Output1:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The method private_method() from the type A is not visible

        at samples.B.public_method_in_b(B.java:7)
        at samples.B.main(B.java:11)
```

Output2:
Jika baris 7 dihapus

```
Public method in A
Protected method in A
Private method in A
Protected method in A
```

Skenario 3a: Mengakses *protected* dan *private method* dari package yang sama

```
1  package samples;
2
3  public class C {
4      public void public_method(){
5          System.out.println("Public method in C");
6      }
7
8      public static void main(String[] args){
9          C c = new C();
10         A a = new A();
11         c.public_method();
12         a.protected_method();
13         a.private_method();
14     }
15 }
```

- Protected method dapat diakses di luar class yang mendefinisikan mereka dan di dalam package yang sama
- Private method tidak dapat diakses di luar class yang mendefinisikan mereka, walaupun di dalam package yang sama

Output1:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The method private_method() from the type A is not visible

        at samples.C.main(C.java:13)
```

Output2:
Jika baris 13 dihapus

```
Public method in C
Protected method in A
```

Skenario 3b: Mengakses *protected* dan *private method* dari package yang berbeda

```
1  package another;
2  import samples.A;
3
4  public class D {
5      public void public_method(){
6          System.out.println("Public method in D");
7      }
8      public static void main(String[] args){
9          D d = new D();
10         A a = new A();
11         d.public_method();
12         a.protected_method();
13         a.private_method();
14     }
15 }
```

- Protected dan private method TIDAK dapat diakses di luar class yang mendefinisikan mereka dan berada pada package yang berlainan

Output1:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The method protected_method() from the type A is not visible
        The method private_method() from the type A is not visible

        at another.D.main(D.java:12)
```

Output2:
Jika baris 12-13 dihapus

```
Public method in D
```

Skenario 4: Mengakses *private method* dengan *receiver* eksplisit

```java
package samples;

class B extends A {
    public void public_method_in_b(){
        public_method();
        System.out.println("Public method in B");
        protected_method();
    }
    public static void main(String[] args){
        B b = new B();
        A a = new A();
        a.private_method();
        b.public_method_in_b();
    }
}
```

Output1:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	The method private_method() from the type A is not visible

	at samples.B.main(B.java:12)
```

```java
package samples;

public class A {
    public void public_method(){
        System.out.println("Public method in A");
        //protected_method();
        //private_method();
    }
    protected void protected_method(){
        System.out.println("Protected method in A");
    }
    private void private_method(){
        System.out.println("Private method in A");
    }

    public static void main (String[] args){
        A a = new A();
        a.public_method();
        a.protected_method();
        a.private_method();
    }
}
```

Output2:

```
Public method in A
Protected method in A
Private method in A
```

- *Private method* **<u>TIDAK</u>** dapat diakses di luar *class* yang mendefinisikannya walaupun dengan *receiver* eksplisit
- *Private method* dapat diakses oleh *class* yang mendefinisikannya dengan *receiver* eksplisit

Next Week

- Konsep dan implementasi constructor dan destructor