# Bab 8. Class

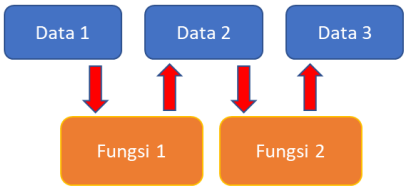
**OBJEKTIF :**

1. Mahasiswa mampu memahami mengenai materi Class pada Java.
2. Mahasiswa mampu memahami mengenai penggunaan Class pada program Java.
3. Mahasiswa mampu mensimulasikan penggunaan Class pada program Java untuk kejadian di dunia nyata.

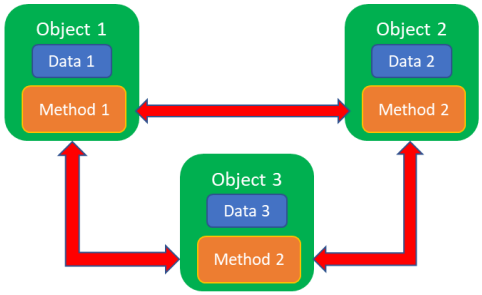
---

## 8.1 Pemrograman Berorientasi Object

Terdapat dua teknik pemrograman yang digunakan saat ini: pemrograman prosedural dan pemrograman berorientasi object. Pada pemrograman prosedural, program dipecah menjadi prosedur-prosedur (fungsi-fungsi), sedangkan pada pemrograman berorientasi object program dibangun dengan membuat object-object dan melakukan interaksi antara object-object tersebut. Gambar berikut mengilustrasikan perbedaan teknik pemrograman prosedural dengan teknik pemrograman berorientasi object:




Pada **pemrograman procedural**, pengolahan data dilakukan dengan memroses data ke prosedur-prosedur (fungsi-fungsi).




Pada **OOP**, pengolahan data dilakukan dengan interaksi antar object-object.

Pada topik-topik sebelum ini kita menggunakan teknik pemrograman prosedural. Kita memecah program menjadi sejumlah method yang masing-masing method mempunyai tugas tersendiri. Teknik pemrograman prosedural ini lebih cocok ketika kita membuat sebuah program kecil. Untuk membuat sebuah program yang besar dan kompleks umumnya kita menggunakan teknik pemrograman berorientasi object.

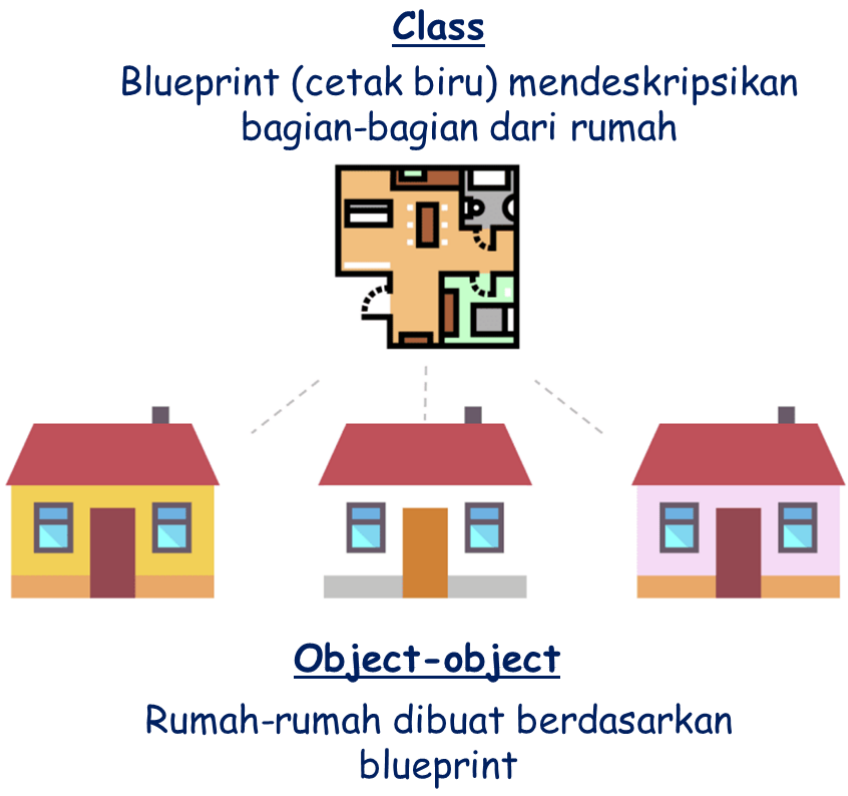
Bahasa Java adalah bahasa pemrograman yang menekankan teknik pemrograman berorientasi object. Dalam Java, kita membuat program dengan membangunnya dari object-object. Object dalam pemrograman adalah komponen program yang memiliki dua kemampuan:

- Object dapat menyimpan data. Data yang disimpan dalam object disebut sebagai ***field***.
- Object dapat melakukan aksi atau operasi-operasi. Aksi-aksi atau operasi-operasi ini disebut sebagi ***method***.

### Class

Object tidak langsung tersedia dalam program. Sebelum sebuah object dapat digunakan oleh program, object-object tersebut harus dibuat di dalam memori komputer. Dan, sebelum sebuah object dapat dibuat dalam memori, di dalam program harus terdapat sebuah class untuk object tersebut.

Ketika kita membangun sebuah rumah, sebelum kita memulai membangunnya kita membuat sebuah blueprint (cetak biru) yang merinci bagian-bagian dari rumah. Kita dapat membayangkan class sebagai blueprint (cetak biru) untuk membuat object. Gambar berikut mengilustrasikan ini:




Class adalah kode program yang mendeskripsikan suatu object. Class mendefinisikan *attribute* (karaketeristik/sifat) dan *behavior* (perilaku) dari object. Misalkan kita mendefinisikan class yang merepresentasikan object mobil. Dalam class tersebut kita dapat mendefinisikan karakteristik-karakteristik dari mobil terdiri dari model, warna, dan bahan bakar. Perilaku-perilaku dari mobil juga dapat didefinisikan sebagai berakselerasi, berhenti, dan berbelok. Class mobil ini dapat digunakan untuk membuat object mobil dengan suatu attribute tertentu. Proses pembuatan object dari class disebut sebagai **instansiasi** (pewujudan) dan object yang dibuat disebut juga sebagai **instance** (wujud) dari class tersebut. Gambar berikut mengilustrasikan class Mobil dan instansiasi dari class Mobil ini:

Class Mobil

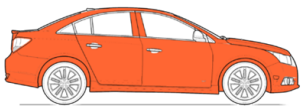
| Mobil | |
|---|---|
| **Attribute** | |
| Model | ... |
| Warna | ... |
| Bahan Bakar | ... |
| **Behavior** | |
| Berakselerasi<br>Berhenti<br>Berbelok | |

Instansiasi

(Pewujudan)

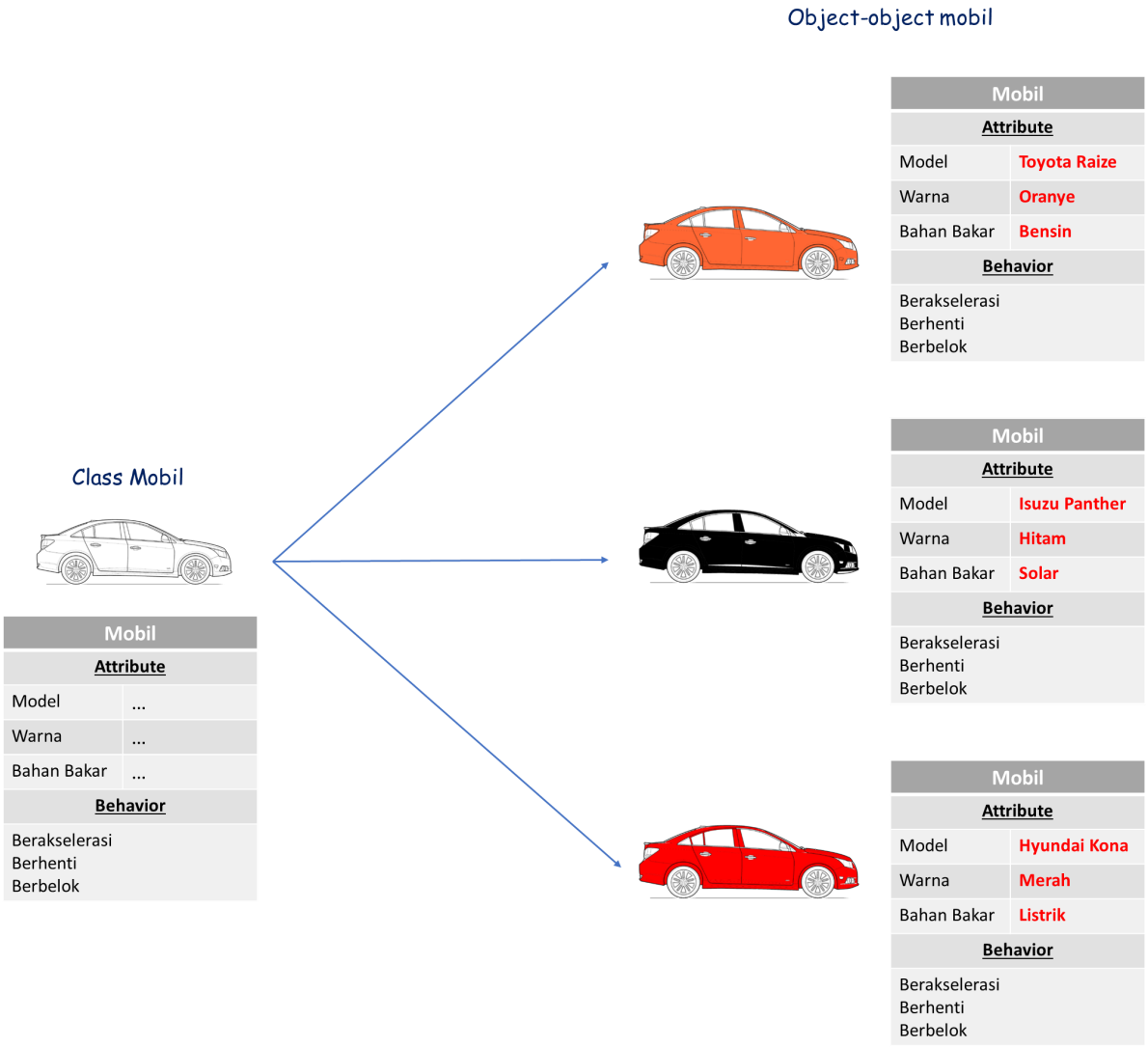
Ketika kita membuat object dari class, kita menyebutnya kita meng-***instansiasi*** (mewujudkan) class tersebut. Dan object yang terbentuk dari class tersebut seringkali disebut sebagai ***instance*** (wujud) dari class tersebut.

Object Mobil (Instance)



| Mobil | |
|---|---|
| **Attribute** | |
| Model | Toyota Raize |
| Warna | Oranye |
| Bahan Bakar | Bensin |
| **Behavior** | |
| Berakselerasi<br>Berhenti<br>Berbelok | |

Sebuah class tidak hanya dapat digunakan untuk membuat sebuah object. Namun, kita dapat membuat berapapun object dari suatu class. Gambar berikut mengilustrasikan pembuatan tiga object Mobil:




Pada gambar dapat dilihat, setiap object (instance) dari class Mobil memiliki nilai-nilai attribute tersendiri. Attribute-attribute dari object ini adalah data-data yang dimiliki object. Data-data ini disebut juga sebagai **field**. Sedangkan behavior dari object adalah hal-hal yang dapat dikerjakan object. Behavior ini disebut sebagai **method**.

## 8.2 Menulis Class Sederhana

Pada bagian ini kita akan menulis sebuah class bernama `PersegiPanjang`. Setiap object yang dibuat dari class `PersegiPanjang` akan menyimpan data mengenai sebuah persegi panjang. Secara spesifik, sebuah object `PersegiPanjang` akan memiliki field-field berikut:

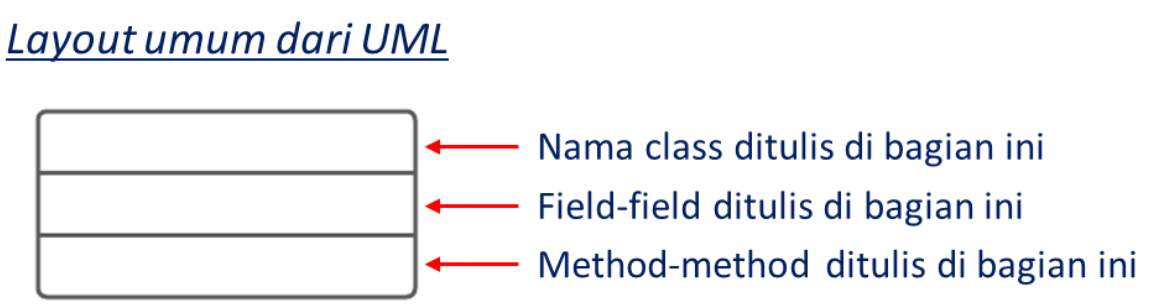
- `panjang`. Field `panjang` digunakan untuk menyimpan panjang dari persegi panjang.
- `lebar`. Field `lebar` digunakan untuk menyimpan lebar dari persegi panjang.

Class `PersegiPanjang` juga akan memiliki method-method berikut:

- `setPanjang`. Method `setPanjang` menetapkan suatu nilai ke field `panjang` dari object.
- `setLebar`. Method `setLebar` menetapkan suatu nilai ke field `lebar` dari object.
- `getPanjang`. Method `getPanjang` memgembalikan nilai yang disimpan dalam field `panjang` dari object.

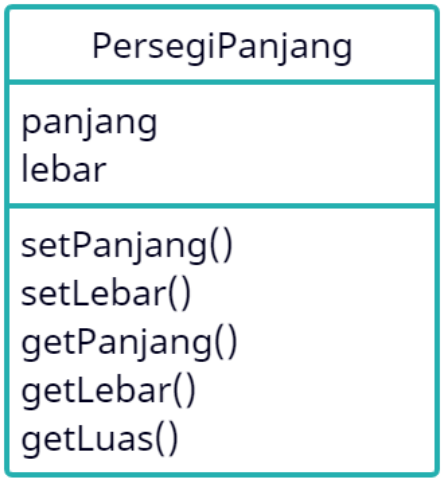
- `getLebar`. Method `getLebar` mengembalikan nilai yang disimpan dalam field `lebar` dari object.
- `getLuas`. Method `getLuas` mengembalikan luas dari persegi panjang, yang merupakan hasil kali dari nilai pada field `panjang` dan nilai pada field `lebar` dari object.

Saat mendesain sebuah class, seringkali memudahkan kita untuk menggambar diagram UML dari class tersebut. UML adalah singkatan dari Unified Modeling Language. Layout umum dari diagram UML dapat dilihat pada gambar berikut:




UML digambarkan sebagai sebuah kotak dengan tiga bagian. Bagian paling atas adalah tempat kita menuliskan nama dari class. Bagian tengah adalah tempat menuliskan field-field dari class. Dan bagian bawah adalah tempat untuk menuliskan method-method dari class.

Diagram UML dari class `PersegiPanjang` dapat digambarkan seperti gambar berikut:




## Menuliskan Kode untuk Class

Langkah selanjutnya setelah mengidentifikasikan field-field dan method-method yang dimiliki oleh class `PersegiPanjang` adalah menuliskan kode untuk class tersebut. Pertama buat sebuah file baru bernama `PersegiPanjang.java`. Dalam file `PersegiPanjang.java` kita akan memulai kode dengan menuliskan "kerangka" umum dari class seperti berikut:

```
public class PersegiPanjang
{

}
```

Keyword `public` yang dituliskan di awal dari header class adalah access specifier. Access specifier menyatakan bagaimana class tersebut dapat diakses. Access specifier `public` berarti bahwa class ini tersedia untuk publik atau dengan kata lain class ini tersedia untuk semua kode yang ditulis di luar file `PersegiPanjang.java`.

Setelah keyword `public` kita menuliskan keyword `class` yang diikuti dengan `PersegiPanjang`, yang merupakan nama dari class. Pada baris setelah header dari class kita menuliskan kurung kurawal buka yang lalu diikuti dengan kurung kurawal tutup. Isi dari class, yang terdiri dari field-field dan method-method, akan dituliskan di dalam tanda kurung kurawal ini. Format umum penulisan definisi sebuah class adalah sebagai berikut:

```
AccessSpecifier class NamaClass
{
    member-member
}
```

Semua field-field dan method-method yang dimiliki oleh class diistilahkan sebagai **member** (anggota) dari class.

## Menuliskan Kode untuk Field-field Class

Mari kita lanjutkan menuliskan class `PersegiPanjang` dengan menuliskan kode untuk field-field dari class tersebut. Class `PersegiPanjang` mempunyai dua field, `panjang` dan `lebar`. Field adalah variabel yang digunakan untuk menyimpan data dari object. Kita mendefinisikan field pada suatu class dengan mendefinisikan variabel. Untuk field `panjang` dan `lebar` kita akan menggunakan variabel tipe `double`. Kode untuk mendefinisikan field `panjang` dan `lebar` dapat dilihat pada kode berikut:

```
public class PersegiPanjang
{
    private double panjang;
    private double lebar;
}
```

Perhatikan pada kode di atas, kita mendeklarasikan dua variabel `panjang` dan `lebar`. Namun deklarasi ini sedikit berbeda dengan deklarasi-deklarasi variabel yang sebelumnya kita gunakan. Disini, kita menambahkan keyword `private` sebelum tipe data. Keyword `private` adalah access specifier yang menandakan bahwa variabel yang dideklarasikan dengan keyword ini tidak dapat diakses oleh statement-statement di luar class.

Dengan menggunakan access specifier `private`, sebuah class dapat menyembunyikan datanya dari kode di luar class. Ketika field dari class disembunyikan dari kode di luar class, data terlindungi dari pengubahan yang tidak disengaja dan tidak diinginkan. Praktik yang umum dalam menuliskan definisi class adalah dengan membuat semua field-field class sebagai `private` dan menyediakan method `public` yang digunakan untuk mengakses field-field tersebut. Dengan kata lain, sebuah class umumnya memiliki field-field `private` dan method-method `public` untuk mengakses field-field tersebut. Tabel berikut menjelaskan perbedaan access specifier `private` dan `public`:

| Access Specifier | Keterangan |
| --- | --- |
| `private` | Jika access specifier `private` digunakan ke member dari class, member tersebut tidak dapat diakses oleh kode di luar class. Member ini hanya dapat diakses oleh method-method yang merupakan member dari class yang sama. |
| `public` | Jika access specifier `public` digunakan ke sebuah member dari class, member tersebut dapat diakses oleh kode di dalam kelas dan juga oleh kode di luar class. |

## Menuliskan Method `setPanjang`

Sekarang kita akan menuliskan method-method dari class `PersegiPanjang`. Kita akan mulai dari method `setPanjang`. Method `setPanjang` ini memungkinkan kode di luar class untuk menyimpan sebuah nilai ke field `panjang`. Kode berikut menambahkan kode untuk method `setPanjang` ke kode definisi class `PersegiPanjang` sebelumnya:

```
/*
     Class PersegiPanjang, tahap 1
     Sedang dalam pengerjaan!
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;

    /*
         Method setPanjang menyimpan sebuah nilai
         ke field panjang.
         @param pjg Nilai untuk disimpan dalam panjang.
    */
    public void setPanjang(double pjg)
    {
        panjang = pjg;
    }
}
```

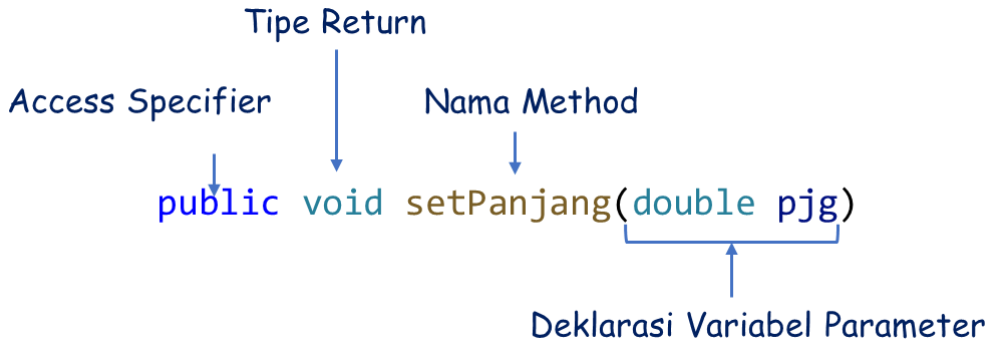
Pada baris 10 sampai dengan 14 kita menuliskan comment yang memberikan penjelasan singkat mengenai method `setPanjang`. Definisi method `setPanjang` berada di baris 15 sampai dengan 18. Perhatikan header dari method `setPanjang`:

```
public void setPanjang(double pjg)
```

Penjelasan bagian-bagian header method di atas adalah sebagai berikut:

- `public`. Keyword `public` adalah access specifier yang menandakan bahwa method ini dapat dipanggil oleh statement-statement di luar class.
- `void`. Ini adalah tipe return dari method. Keyword `void` menandakan bahwa method ini tidak mengembalikan data ke statement yang memanggilnya.
- `setPanjang`. Ini adalah nama dari method.
- `(double pgj)`. Ini adalah deklarasi dari sebuah variabel parameter dari tipe `double`, bernama `pjg`.

Gambar berikut menjelaskan bagian-bagian dari header method `setPanjang`:




Header ini mirip dengan header dari method yang telah kita pelajari sebelumnya dengan satu perbedaan: pada header dari method `setPanjang` ini kita tidak menuliskan keyword `static`. Method-method yang kita pelajari sebelumnya dimana kita menuliskan keyword `static` pada headernya, disebut dengan method static. Pada method static kita tidak memerlukan sebuah instance dari suatu class untuk memanggil method tersebut. Jika kita menuliskan method untuk bekerja pada instance dari class, kita tidak menuliskan `static` pada headernya. Method yang bekerja pada instance dari suatu class disebut sebagai **method instance**. Karena method `setPanjang` digunakan untuk menyimpan sebuah nilai dalam field `panjang` dari sebuah instance dari class `PersegiPanjang`, maka method ini adalah method instance. Kita akan membahas mengenai perbedaan method instance dan method static nanti. Untuk saat ini yang perlu kita ingat adalah pada header dari method instance dari sebuah class, kita tidak menuliskan keyword `static`.

Setelah header, kita menuliskan body dari method `setPanjang` seperti berikut:

```
{
    panjang = pjg;
}
```

Body dari method `setPanjang` hanya terdiri dari satu statement, yang menugaskan nilai `pjg` ke field `panjang`. Ketika method ini dieksekusi, variabel parameter `pjg` akan menyimpan nilai dari argument yang diberikan ke method. Lalu nilai tersebut ditugaskan ke field `panjang`.

## Menuliskan Method `setLebar`

Kode untuk method `setLebar` mirip dengan kode method `setPanjang`, hanya saja pada method `setLebar` kita menugaskan nilai argument yang diberikan ketika method ini dipanggil ke field `lebar`. Kode definisi class `PersegiPanjang` setelah ditambahkan kode definisi method `setLebar` akan seperti berikut:

```
/*
     Class PersegiPanjang, tahap 2
     Sedang dalam pengerjaan!
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;

    /*
        Method setPanjang menyimpan sebuah nilai
```

```
        dalam field panjang.
        @param pjg Nilai yang disimpan dalam field panjang.
    */
    public void setPanjang(double pjg)
    {
        panjang = pjg;
    }

    /*
        Method setLebar menyimpan sebuah nilai
        dalam field lebar.
        @param lbr Nilai yang disimpan dalam field lebar.
    */
    public void setLebar(double lbr)
    {
        lebar = lbr;
    }
}
```

## Menuliskan Method `getPanjang` dan Method `getLebar`

Karena field `panjang` dan field `lebar` pada class kita tuliskan dengan access specifier `private`, yang berarti hanya method member dari class yang dapat mengaksesnya, kita harus menuliskan method `setPanjang` dan `setLebar` sehingga kode di luar class `PersegiPanjang` dapat menyimpan nilai-nilai dalam field-field tersebut. Kita juga harus menuliskan method-method untuk memungkinkan kode di luar class untuk mendapatkan nilai-nilai field-field tersebut. Ini yang akan dilakukan oleh method `getPanjang` dan method `getLebar`. Method `getPanjang` akan mengembalikan nilai yang disimpan dalam field `panjang`, dan method `getLebar` akan mengembalikan nilai yang disimpan dalam field `lebar`.

Kode definisi method `getPanjang` kita tuliskan seperti berikut:

```
public double getPanjang()
{
    return panjang;
}
```

Perhatikan pada header dari method `getPanjang` kita menuliskan tipe return `double`. Ini karena method ini mengembalikan field `panjang` yang bertipe `double`. Dan karena method ini hanya digunakan untuk mendapatkan nilai field `panjang`, method ini tidak memerlukan argument, sehingga kita tidak menuliskan parameter dalam tanda kurung.

Kode definisi method `getLebar` mirip dengan definisi method `getPanjang` hanya saja pada method `getLebar` kita mengembalikan nilai field `lebar`. Kode definisi method `getLebar` dituliskan seperti berikut:

```
public double getLebar()
{
    return lebar;
}
```

Sehingga, kode class `PersegiPanjang` dengan tambahkan kode method `getPanjang` dan method `getLebar` akan seperti berikut:

```java
/*
     Class PersegiPanjang, tahap 3
     Sedang dalam pengerjaan!
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;

    /*
        Method setPanjang menyimpan sebuah nilai
        dalam field panjang.
        @param pjg Nilai yang disimpan dalam field panjang.
    */
    public void setPanjang(double pjg)
    {
        panjang = pjg;
    }

    /*
        Method setLebar menyimpan sebuah nilai
        dalam field lebar.
        @param lbr Nilai yang disimpan dalam field lebar.
    */
    public void setLebar(double lbr)
    {
        lebar = lbr;
    }

    /*
        Method getPanjang mengembalikan panjang dari
        object PersegiPanjang.
        @return Nilai dalam field panjang
    */
    public double getPanjang()
    {
        return panjang;
    }

    /*
        Method getLebar mengembalikan lebar dari
        object PersegiPanjang.
        @return Nilai dalam field lebar
    */
    public double getLebar()
    {
        return lebar;
    }
}
```

## Menuliskan Method `getLuas`

Method terakhir yang akan kita tulis untuk class `PersegiPanjang` adalah method `getLuas`. Method ini mengembalikan luas dari sebuah persegi panjang, yang dihitung dengan panjang dikalikan lebar dari persegi panjang. Berikut adalah kode untuk method `getLuas`:

```
public double getLuas()
{
    return panjang * lebar;
}
```

Method ini mengembalikan hasil dari ekspresi aritmatika `panjang * lebar`. Kode di bawah mencontohkan penggunaan method `getLuas`:

```
PersegiPanjang boks = new PersegiPanjang();
boks.setPanjang(20.0);
boks.setLebar(10.0);
double luas = boks.getLuas();
```

Statement terakhir dari kode di atas menugaskan nilai yang dikembalikan oleh method `getLuas` ke variabel `luas`. Setelah statement ini dieksekusi, variabel `luas` akan berisi nilai `200.0`.

Kode lengkap dari class `PersegiPanjang` setelah ditambahkan method `getLuas` dapat dilihat pada kode  berikut:

***Definisi Class (PersegiPanjang.java)***

```
/*
     Class PersegiPanjang, tahap 4
     Sedang dalam pengerjaan!
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;

    /*
        Method setPanjang menyimpan sebuah nilai
        dalam field panjang.
        @param pjg Nilai yang disimpan dalam field panjang.
    */
    public void setPanjang(double pjg)
    {
        panjang = pjg;
    }

    /*
        Method setLebar menyimpan sebuah nilai
        dalam field lebar.
        @param lbr Nilai yang disimpan dalam field lebar.
    */
    public void setLebar(double lbr)
    {
        lebar = lbr;
    }
```

```
    /*
        Method getPanjang mengembalikan panjang dari
        object PersegiPanjang.
        @return Nilai dalam field panjang
    */
    public double getPanjang()
    {
        return panjang;
    }

    /*
        Method getLebar mengembalikan lebar dari
        object PersegiPanjang.
        @return Nilai dalam field lebar
    */
    public double getLebar()
    {
        return lebar;
    }

    /*
        Method getLuas mengembalikan luas dari
        object PersegiPanjang.
        @return Hasil dari panjang kali lebar.
    */
    public double getLuas()
    {
        return panjang * lebar;
    }
}
```

Perhatikan pada kode class `PersegiPanjang`, kita tidak menuliskan method `main`. Ini karena class bukanlah program komplit, tetapi merupakan sebuah definisi class `PersegiPanjang`. Program yang membuat dan menggunakan object-object `PersegiPanjang` akan mempunyai method `main` sendiri. Sebelum kita membuat sebuah program yang membuat dan menggunakan object `PersegiPanjang`, kita harus menyimpan kode definisi class `PersegiPanjang` ini dalam sebuah file `PersegiPanjang.java`.

## Menuliskan Program yang Menggunakan Class `PersegiPanjang`

Program berikut mendemonstrasikan penggunaan class `PersegiPanjang` untuk membuat object `PersegiPanjang`:

***Program (DemoPersegiPanjang.java)***

```
/*
    Program berikut mendemonstrasikan penggunaan
    method setPanjang dan setLebar dari class PersegiPanjang
*/
public class DemoPersegiPanjang
{
    public static void main(String[] args)
    {
        // Buat sebuah object PersegiPanjang dan
        // tugaskan alamatnya ke variabel boks.
        PersegiPanjang boks = new PersegiPanjang();

        // Panggil method setPanjang, berikan nilai 20.0
        // sebagai argument
        boks.setPanjang(20.0);    // Menugaskan field panjang dengan nilai 20.0

        // Panggil method setLebar, berikan nilai 10.0
        // sebagai argument
        boks.setLebar(10.0);      // Menugaskan field lebar dengan nilai 10.0

        // Tampilkan panjang dan lebar dari object PersegiPanjang
        System.out.println("Panjang boks adalah " + boks.getPanjang());
        System.out.println("Lebar boks adalah " + boks.getLebar());
    }
}
```

Program `DemoPersegiPanjang.java` harus disimpan di dalam folder yang sama dengan file `PersegiPanjang.java`. Perintah berikut dapat digunakan untuk mengkompilasi program:

```
javac DemoPersegiPanjang.java
```

Ketika compiler membaca source code `DemoPersegiPanjang.java` dan melihat bahwa class bernama `PersegiPanjang` digunakan pada program tersebut, compiler mencari file `PersegiPanjang.class` di dalam folder tempat source code `DemoPersegiPanjang.java`. Dan karena karena kita belum mengkompilasi `PersegiPanjang.java`, maka tidak terdapat file `PersegiPanjang.class` dalam folder tersebut. Sehingga, compiler mencari file `PersegiPanjang.java` dan mengkompilasinya. Setelah compiler selesai mengkompilasi, kita dapat menjalankan program dengan menjalankan file `DemoPersegiPanjang.class` dengan perintah berikut:

```
java DemoPersegiPanjang
```

Output dari program:

```
Panjang boks adalah 20.0
Lebar boks adalah 10.0
```

Mari kita lihat kode program `DemoPersegiPanjang.java`. Pada baris 11, di dalam method `main` dari program, kita menuliskan statement berikut untuk membuat sebuah object `PersegiPanjang` dan mereferensikannya dengan sebuah variabel:

```
PersegiPanjang boks = new PersegiPanjang();
```

Statement di atas adalah penyingkat dari dua statement berikut:

```
PersegiPanjang boks;
boks = new PersegiPanjang();
```

Statement pertama:

```
PersegiPanjang boks;
```

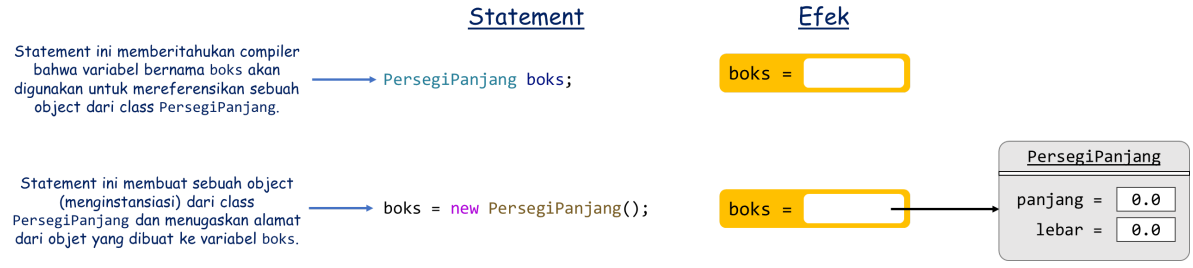
adalah statement deklarasi variabel bernama `boks` dengan tipe data berupa class `PersegiPanjang`. Karena tipe data pada statement deklarasi ini bukanlah tipe data primitif, maka statement ini mendeklarasikan sebuah variabel referensi. Variabel referensi adalah variabel yang menyimpan alamat memori dari sebuah object. Ketika sebuah variabel referensi menyimpan alamat memori dari suatu object, kita menyebutnya variabel referensi ini mereferensikan object tersebut. Sehingga, statement di atas adalah statement deklarasi dari sebuah variabel referensi bernama `boks` yang akan digunakan untuk mereferensikan sebuah object `PersegiPanjang`.

Statement kedua:

```
boks = new PersegiPanjang();
```

adalah statement assignment (penugasan). Pada ruas kanan operator assignment (=), kita menuliskan keyword `new` yang diikuti oleh nama class `PersegiPanjang` dengan tanda kurung buka dan kurung tutup, `PersegiPanjang()`. Keyword `new` ini adalah sebuah operator yang digunakan untuk membuat sebuah object dalam memori. Karena setelah operator `new` kita menuliskan nama class `PersegiPanjang`, maka sebuah object dari class `PersegiPanjang` dibuat di dalam memori. Setelah object dari class `PersegiPanjang` dibuat, alamat memori dari object tersebut ditugaskan ke variabel `boks`. Setelah statement ini dieksekusi, variabel `boks` akan mereferensikan sebuah object dari class `PersegiPanjang`.

Gambar berikut mengilustrasikan efek dari dua statement di atas:



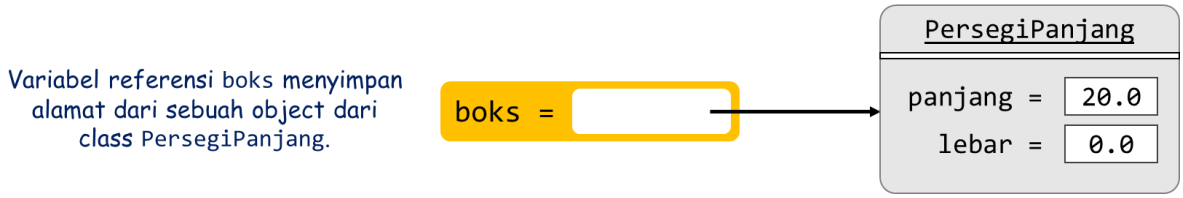

Perhatikan pada gambar di atas, field `panjang` dan field `lebar` dari object `PersegiPanjang` yang direferensikan oleh variabel `boks` ditetapkan ke `0`. Ini karena field-field numerik secara otomatis akan diinisialisasi ke 0.

Variabel referensi yang telah mereferensikan sebuah object dapat kita anggap sebagai nama dari object yang direferensikannya. Sebagai contoh, kita dapat menyebut object yang direferensikan oleh variabel `boks` sebagai object `boks`.

Pada baris 15, kita menuliskan statement:

```
boks.setPanjang(20.0);
```

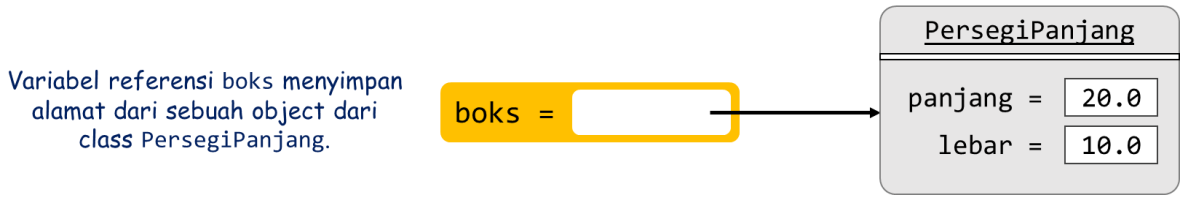
Statement ini memanggil method `setPanjang` dari object `boks` dengan argument `20.0`. Ketika statement ini dieksekusi, method `setPanjang` menugaskan nilai argument yaitu `20.0` ke field `panjang`. Sehingga, setelah statement ini dieksekusi, object `boks` akan mempunyai nilai-nilai field seperti terlihat pada gambar berikut:




Pada baris 19, kita menuliskan statement:

```
boks.setLebar(10.0);
```

Statement ini memanggil method `setLebar` dari object `boks` dengan argument `10.0`. Ketika statement ini dieksekusi, method `setLebar` menugaskan nilai argument yaitu `10.0` ke field `lebar`. Sehingga, setelah statement ini dieksekusi, object `boks` akan mempunyai nilai-nilai field seperti terlihat pada gambar berikut:




Lalu statement pada baris 22:

```
System.out.println("Panjang boks adalah " + boks.getPanjang());
```

memanggil method `getPanjang` dari object `boks`. Method ini mengembalikan nilai yang disimpan pada field `panjang` yaitu nilai `20.0`. Output dari statement di atas:

```
Panjang boks adalah 20.0
```

Statement pada baris 23:

```
System.out.println("Lebar boks adalah " + boks.getLebar());
```

memanggil method `getLebar` dari object `boks`. Statement ini mengembalikan nilai yang disimpan pada field `lebar` yaitu nilai `10.0`. Output dari statement di atas:

```
Lebar boks adalah 10.0
```

## Data Hiding (Penyembunyian Data)

*Data hiding* (penyembunyian data) adalah konsep penting dalam pemrograman berorientasi object. Object harus didesain untuk menyembunyikan data internalnya dari kode di luar class yang membentuk object tersebut. Hanya method-method yang dituliskan dalam class yang mempunyai akses langsung dan dapat mengubah data internal dari object. Kita menyembunyikan data internal object dengan membuat field-field class menjadi `private` dan membuat method-method yang mengakses field-field object sebagai `public`.

Mungkin Anda bertanya-tanya apa tujuan menyembunyikan data dalam class. Jika Anda menulis class untuk Anda gunakan sendiri mungkin ini terlihat terlalu berlebihan. Namun, jika Anda terlibat dalam sebuah tim yang mengembangkan software besar dan programmer-programmer lain menggunakan class-class yang Anda tulis, dengan menyembunyikan data, Anda dapat memastikan bahwa class yang Anda desain digunakan dan bekerja sesuai dengan peruntukkan yang Anda inginkan.

## Accessor dan Mutator

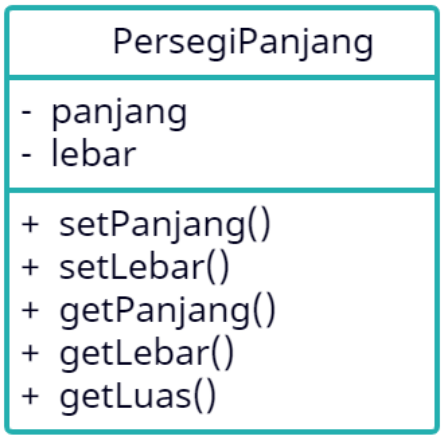
Seperti yang telah disebutkan sebelumnya, class harus didesain dengan memperhatikan data hiding. Data hiding diterapkan dengan membuat field-field class `private` dan menyediakan method-method `public` untuk mengakses dan mengubah nilai-nilai pada field-field tersebut. Data hiding memastikan bahwa data dalam object konsisten dengan tujuan dari object tersebut.

Dalam pemrograman berorientasi object terdapat istilah-istilah yang digunakan untuk method-method yang mengakses field `private` dari object, yaitu ***accessor*** dan ***mutator***. Method yang mengembalikan nilai dalam sebuah field dan tidak mengubah nilai dari field disebut sebagai method accessor. Pada class `PersegiPanjang`, method `getPanjang`, `getLebar` adalah method accessor. Sedangkan method yang mengubah nilai field disebut sebagai method mutator. Pada class `PersegiPanjang`, method `setPanjang` dan `setLeabar` adalah method mutator.

Umumnya method-method accessor dinamakan dengan awalan `get` yang diikuti dengan nama field dan method-method mutator dinamakan dengan awalan `set` yang diikuti dengan nama field. Sehingga method mutator terkadang disebut juga sebagai "*setter*" dan method accessor disebut juga sebagai "*getter*".

## Diagram UML dengan Spesifikasi Access

Diagram UML yang kita lihat pada bagian awal saat kita mendesain class `PersegiPanjang` mendaftar semua member-member dari class tetapi tidak menandakan member-member mana yang `private` dan member-member mana yang `public`. Dalam diagram UML, kita menggunakan tanda `-` sebelum nama member untuk menandakan member tersebut `private` dan menggunakan tanda `+` untuk menandakan member tersebut `public`. Gambar di bawah adalah diagram UML dengan informasi `public` dan `private` dari member-member untuk class `PersegiPanjang`:




## Tipe Data dan Notasi Parameter dalam Diagram UML

UML juga menyediakan notasi yang dapat kita gunakan untuk menandakan tipe-tipe data dari field-field, method-method, dan variabel-variabel parameter. Untuk menandakan tipe data dari sebuah field kita menuliskan titik dua yang diikuti dengan nama dari tipe data setelah nama dari field. Sebagai contoh, field `panjang` pada class `PersegiPanjang` adalah tipe `double`. Kita dapat menuliskannya seperti berikut:

```
- panjang : double
```

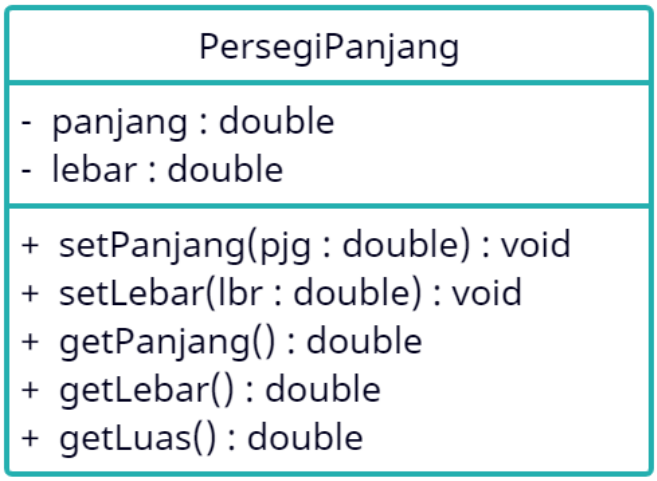
Tipe return dari sebuah method juga dapat dituliskan dengan cara yang sama. Setelah nama method tuliskan titik dua yang diikuti dengan tipe return dari method tersebut. Method `getPanjang` dari class `PersegiPanjang` mengembalikan sebuah nilai bertipe `double`, kita dapat menuliskan informasi ini pada diagram UML dengan menuliskan:

```
+ getPanjang() : double
```

Variabel parameter dan tipe datanya juga dapat dituliskan di UML dengan menuliskannya dalam tanda kurung setelah nama method. Sebagai contoh method `setPanjang` dari class `PersegiPanjang` mempunyai sebuah parameter `double` bernama `pjg`. Kita dapat menuliskan method ini pada UML seperti berikut:

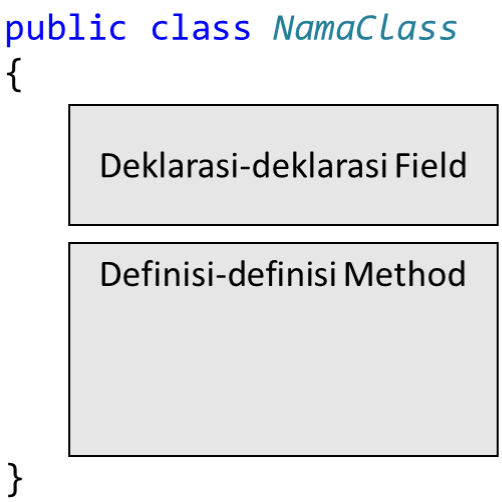
```
+ getPanjang(pjg : double) : void
```

Diagram UML dari class `PersegiPanjang` beserta notasi tipe data dan parameter dapat dilihat pada gambar berikut:




### Layout Penulisan Definisi Class

Pada class `PersegiPanjang` yang kita tulis, kita menuliskan deklarasi-deklarasi variabel-variabel field terlebih dahulu sebelum definisi-definisi method. Urutan penulisan ini tidak diharuskan. Kita dapat saja menuliskan definisi-definisi method terlebih dahulu lalu setelahnya deklarasi-deklarasi variabel-variabel field. Namun, urutan penulisan deklarasi-deklarasi field terlebih dahulu lalu setelahnya definisi-definisi method ini yang paling banyak diikuti oleh programmer-programmer. Gambar berikut mengilustrasikan layout penulisan definisi class:




## 8.3 Field Instance dan Method Instance

Class adalah blueprint atau cetakan dari object. Ketika kita membuat objet dari suatu class, kita menyebutnya kita melakukan **instansiasi** class tersebut. Instansiasi berarti pewujudan. Object yang terbentuk dari class tersebut seringkali disebut sebagai **instance** (wujud) dari class tersebut.

Kita dapat membuat lebih dari satu instance dari suatu class dalam sebuah program. Setiap instance tersebut akan mempunyai **state** masing-masing. State adalah nilai-nilai field dari suatu instance dalam satu waktu. Sebagai contoh, program berikut membuat tiga instance dari class `PersegiPanjang` yang masing-masing instance mempunyai state-state tersendiri:

***Program (LuasRuangan.java)***

```
import java.util.Scanner;

/*
    Program ini membuat tiga instance dari
    class PersegiPanjang.
*/
public class LuasRuangan
{
    public static void main(String[] args)
    {
        double totalLuas;
        Scanner keyboard = new Scanner(System.in);

        // Buat tiga object PersegiPanjang
        PersegiPanjang dapur = new PersegiPanjang();
        PersegiPanjang kamarTidur = new PersegiPanjang();
        PersegiPanjang ruangKerja = new PersegiPanjang();

        // Dapatkan dimensi dari dapur
        System.out.print("Masukkan panjang dapur: ");
        dapur.setPanjang(keyboard.nextDouble());

        System.out.print("Masukkan lebar dapur: ");
        dapur.setLebar(keyboard.nextDouble());

        // Dapatkan dimensi kamar tidur
        System.out.print("Masukkan panjang kamar tidur: ");
        kamarTidur.setPanjang(keyboard.nextDouble());

        System.out.print("Masukkan lebar kamar tidur: ");
        kamarTidur.setLebar(keyboard.nextDouble());

        // Dapatkan dimensi ruang kerja
        System.out.print("Masukkan panjang ruang kerja: ");
        ruangKerja.setPanjang(keyboard.nextDouble());

        System.out.print("Masukkan lebar ruang kerja: ");
        ruangKerja.setLebar(keyboard.nextDouble());

        // Tampilkan total luas
                    totalLuas   =   dapur.getLuas()   +   kamarTidur.getLuas()   +
ruangKerja.getLuas();

        System.out.println("Total luas dari tiga ruang adalah " + totalLuas);
    }
}
```

***Output Program (LuasRuangan.java)***

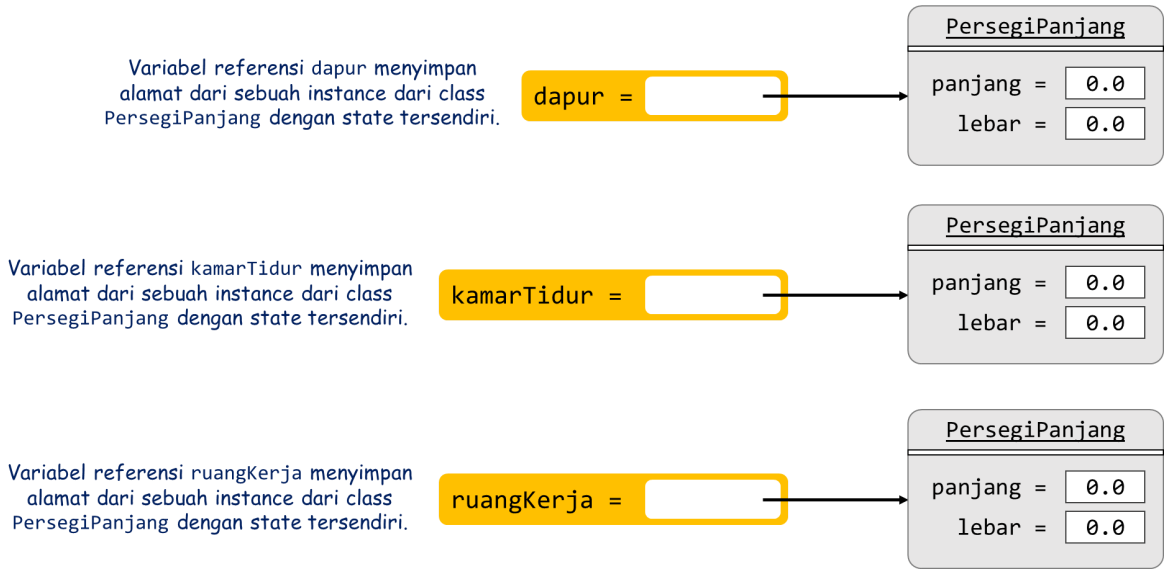
```
Masukkan panjang dapur: 8
Masukkan lebar dapur: 5
Masukkan panjang kamar tidur: 12
Masukkan lebar kamar tidur: 8
Masukkan panjang ruang kerja: 8
Masukkan lebar ruang kerja: 4
Total luas dari tiga ruang adalah 168.0
```

Pada baris 15, 16, dan 17 di kode program `LuasRuangan.java` kita mempunyai statement-statement berikut:

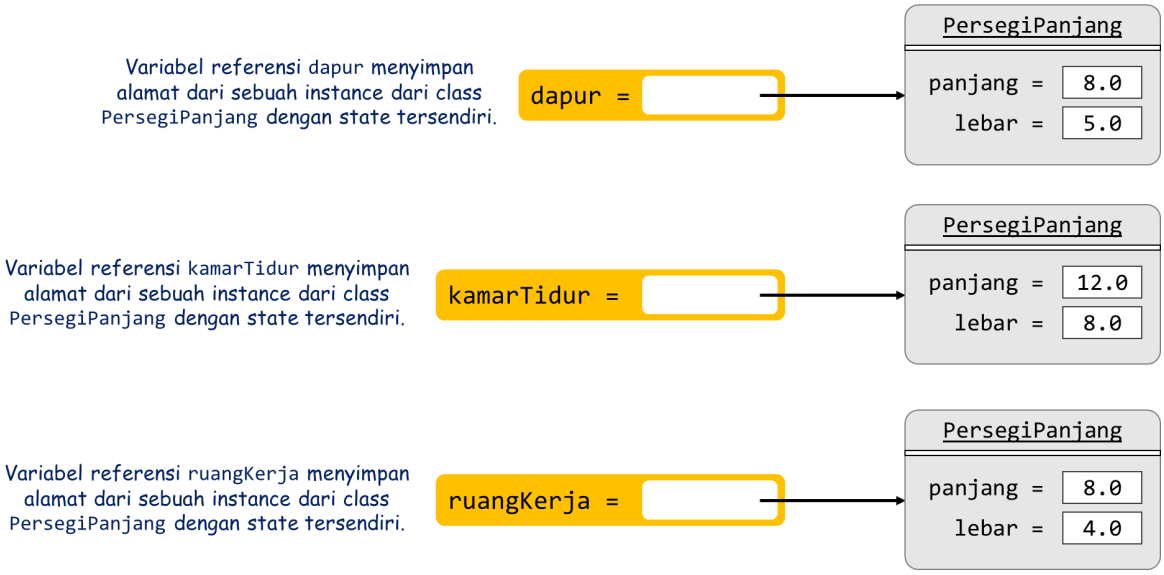
```
PersegiPanjang dapur = new PersegiPanjang();
PersegiPanjang kamarTidur = new PersegiPanjang();
PersegiPanjang ruangKerja = new PersegiPanjang();
```

Tiga statement di atas membuat tiga object yang dimana setiap object adalah sebuah instance dari class `PersegiPanjang`.

Gambar berikut mengilustrasikan bagaimana variabel-variabel referensi `dapur`, `kamarTidur`, dan `ruangKerja` mereferensikan object-object tersendiri dengan state-state tersendiri setelah eksekusi dari tiga statement di atas:




Setelah program menginstansiasi tiga instance dari class `PersegiPanjang`, program meminta input pengguna untuk memasukkan nilai-nilai ke field `panjang` dan field `lebar` dari masing-masing instance. Pada contoh output, kita memasukkan nilai 8 dan 5 sebagai panjang dan lebar dari dapur, nilai 12 dan 8 sebagai panjang dan lebar dari kamar tidur, dan nilai 8 dan 4 sebagai panjang dan lebar dari ruang kerja. Gambar berikut mengilustrasikan state-state dari masing-masing instance setelah nilai-nilai ini ditugaskan ke field-field dari instance-instance tersebut:




Perhatikan pada gambar di atas, masing-masing instance dari class `PersegiPanjang` mempunyai variabel `panjang` dan `lebar` tersendiri. Variabel-variabel ini disebut sebagai **variabel instance** atau **field instance**. Setiap instance dari sebuah class mempunyai field-field instance masing-masing dan dapat menyimpan nilai-nilainya sendiri dalam field-field tersebut.

Method-method yang beroperasi pada sebuah instance dari class disebut sebagai **method instance**. Semua method-method dalam class `PersegiPanjang` adalah method instance karena mereka melakukan operasi-operasi pada instance tertentu dari class. Sebagai contoh, lihat statement berikut yang ada pada baris 25 dari program `LuasRuangan.java`:

```
dapur.setPanjang(angka);
```

Statement di atas memanggil method `setPanjang` pada object `dapur`. Statement ini menyebabkan field `panjang` dalam object `dapur` ditugaskan dengan nilai `angka`. Sekarang perhatikan statement berikut pada baris 35 dari program `LuasRuangan.java`:

```
kamarTidur.setPanjang(angka);
```

Statement di atas memanggil method `setPanjang` pada object `kamarTidur`. Statement ini menyebabkan field `panjang` dari object `kamarTidur` ditugaskan dengan nilai `angka`. Hal yang sama juga dilakukan oleh statement pada baris 45 dari program `LuasRuangan`:

```
ruangKerja.setPanjang(angka);
```

Statement di atas memanggil method `setPanjang` pada object `ruangKerja`. Statement ini menyebabkan field `panjang` dari object `ruangKerja` ditugaskan dengan nilai `angka`.

# 8.4 Constructor

Constructor (pengkonstruksi) adalah method yang secara otomatis dipanggil ketika sebuah object dibuat. Constructor umumnya melakukan proses inisialisasi seperti menginisialisasi field-field instance ke suatu nilai. Dinamakan dengan constructor (pengkonstruksi) karena constructor membantu mengkonstruksi object.

Pada class `PersegiPanjang` sebelumnya kita memerlukan tiga statement unutk membuat sebuah object `PersegiPanjang` dan menginisialisasi nilai-nilai `field` dengan memanggil method `setPanjang` dan `setLebar`, seperti contoh berikut:

```
PersegiPanjang boks = new PersegiPanjang();
boks.setPanjang(20.0);
boks.setLebar(10.0);
```

Akan lebih efisien jika kita dapat menginisialisasi object `PersegiPanjang` saat pembuatannya sehingga kita dapat menyingkat proses pembuatan object `PersegiPanjang` dan penginisialisasian nilai-nilai field `panjang` dan `lebar` dalam satu statement, seperti contoh berikut:

```
PersegiPanjang boks = new PersegiPanjang(20.0, 10.0);
```

Kita dapat membuat class `PersegiPanjang` melakukan inisialisasi saat proses pembuatan object-objectnya dengan mendefinisikan sebuah method constructor yang menerima argument. Kita mendefinisikan method constructor dengan menuliskan definisi method yang mempunyai nama yang sama dengan nama class. Sebagai contoh, kode berikut menambahkan sebuah method constructor pada class `PersegiPanjang`:

***Definisi Class (PersegiPanjang.java)***

```
/*
    Class PersegiPanjang Versi 2 (dengan constructor).
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;

    /*
        Constructor
        @param pjg Panjang dari persegi panjang.
        @param lbr Lebar dari persegi panjang.
    */
    public PersegiPanjang(double pjg, double lbr)
    {
        panjang = pjg;
        lebar = lbr;
    }

...Kode-kode lain sama seperti sebelumnya.
```

Baris 14 sampai dengan 18 pada kode class `PersegiPanjang` di atas adalah constructor dari class `PersegiPanjang`:

```
public PersegiPanjang(double pjg, double lbr)
{
    panjang = pjg;
    lebar = lbr;
}
```

Constructor ini menerima dua argument, yang diberikan melalui variabel parameter `pjg` dan `lbr`. Nilai kedua variabel parameter ini kemudian ditugaskan ke field `panjang` dan field `lebar`.

Perhatikan pada header dari constructor, kita tidak menuliskan tipe return apapun (tidak juga `void`). Ini karena constructor tidak diperuntukkan untuk dieksekusi melalui pemanggilan method dan tidak dapat mengembalikan nilai.

Header method constructor mempunyai format umum seperti berikut:

```
AccessSpecifier NamaClass(Parameter...)
```

Berikut adalah contoh statement yang mendeklarasikan variabel `boks`, membuat sebuah object `PersegiPanjang`, dan memberikan nilai 14.0 dan 7.0 sebagai argument ke constructor:

```
PersegiPanjang boks = new PersegiPanjang(14.0, 7.0);
```

Ketika statement di atas dieksekusi, `new PersegiPanjang(14.0, 7.0)` menyebabkan constructor dipanggil dengan argument 14.0 dan 7.0. Sehingga, setelah statement di atas dieksekusi, variabel `boks` akan mereferensikan sebuah object `PersegiPanjang` dengan field `panjang` ditetapkan dengan nilai 14.0 dan field `lebar` ditetapkan dengan nilai 7.0. Gambar berikut mengilustrasikan ini:


Program berikut mendemonstrasikan pembuatan object `PersegiPanjang` dengan constructor:

***Program (DemoConstructor.java)***

```
/*
    Program ini mendemonstrasikan constructor
    dari class PersegiPanjang.
*/
public class DemoConstructor
{
    public static void main(String[] args)
    {
        // Buat sebuah object PersegiPanjang, berikan 15.0
        // dan 5.0 sebagai argument ke constructor
        PersegiPanjang boks = new PersegiPanjang(15.0, 5.0);

        // Tampilkan panjangnya
        System.out.println("Panjang boks adalah " + boks.getPanjang());

        // Tampilkan lebarnya
        System.out.println("Lebar boks adalah " + boks.getLebar());

        // Tampilkan luasnya
```

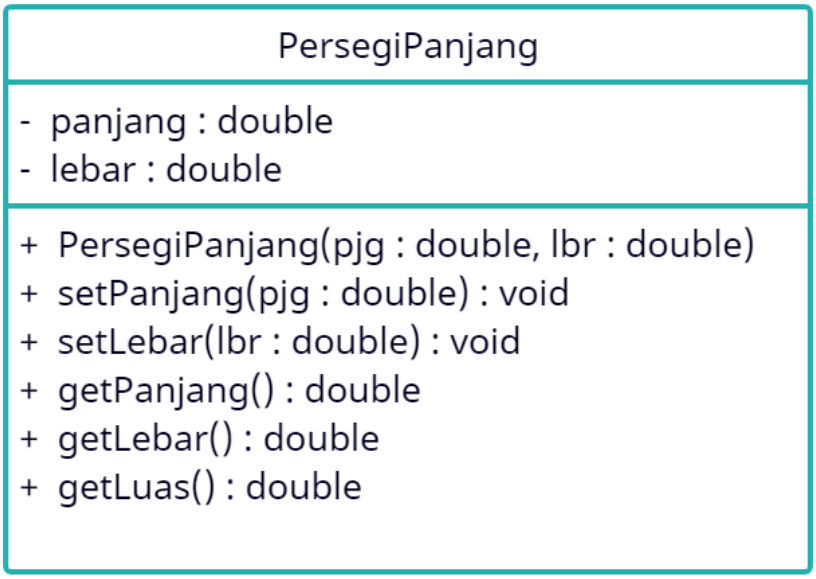
```
        System.out.println("Luas boks adalah " + boks.getLuas());
    }
}
```

***Output Program (DemoConstructor.java)***

```
Panjang boks adalah 15.0
Lebar boks adalah 5.0
Luas boks adalah 75.0
```

## Penulisan Constructor pada Diagram UML

Pada diagram UML kita menuliskan constructor seperti method-method lainnya. Gambar berikut adalah diagram UML untuk class `PersegiPanjang` dengan tambahan constructor:




## Constructor Default

Saat sebuah object dibuat, constructor dari class dari object tersebut selalu dipanggil. Namun, bagaimana jika kita tidak menuliskan constructor pada class dari object tersebut? Jika kita tidak menuliskan constructor dalam sebuah class, compiler Java secara otomatis menuliskan sebuah constructor ketika class tersebut dikompilasi. Constructor yang ditulis otomatis oleh compiler Java disebut sebagai ***constructor default***. Constructor default tidak menerima argument. Constructor ini menetapkan nilai pada field-field numerik object ke 0 dan field `boolean` ke `false`. Jika object tersebut mempunyai field-field referensi (bukan tipe primitif), constructor default menetapkannya ke nilai spesial `null`, yang berarti field-field referensi tersebut tidak mereferensikan apapun.

Constructor default hanya dituliskan oleh compiler Java ketika sebuah class tidak mempunyai constructor. Sebagai contoh, sebelumnya kita menuliskan class `PersegiPanjang` tanpa constructor. Ketika class tersebut dikompilasi, compiler menuliskan constructor default yang menetapkan nilai field `panjang` dan `lebar` ke 0.0. Sehingga kita dapat menggunakan statement berikut untuk membuat sebuah object dari class `PersegiPanjang`:

```
// Membuat object dari class PersegiPanjang tanpa constructor
PersegiPanjang boks = new PersegiPanjang();    // Memanggil constructor default
```

Saat kita membuat sebuah object dari class `PersegiPanjang` tanpa constructor kita tidak memberikan argument ke constructor default karena constructor default tidak menerima arugment apapun.

Setelah kita menambahkan constructor yang menerima dua argument untuk field `panjang` dan `lebar` ke class `PersegiPanjang` (pada class `PersegiPanjang` versi 2), saat kode class `PersegiPanjang` tersebut dikompilasi, compiler Java tidak menambahkan constructor default. Constructor yang kita tulis menjadi satu-satunya constructor yang dimiliki class `PersegiPanjang`. Sehingga ketika kita membuat object dari class `PersegiPanjang` kita harus memberikan argument untuk field `panjang` dan untuk field `lebar`. Jika kita mencoba membuat object dari class `PersegiPanjang` tanpa argument, compiler akan memberikan pesan error:

```
// Terdapat sebuah constructor PersegiPanjang yang menerima argument
PersegiPanjang boks = new PersegiPanjang();       // ERROR!! Harus memberikan
argument
```

Pesan error yang diberikan oleh compiler:

```
.\DemoConstructor.java:11:   error:   constructor   PersegiPanjang   in   class
PersegiPanjang cannot be applied to given types;
        PersegiPanjang boks = new PersegiPanjang();
                              ^
  required: double,double
  found:    no arguments
  reason: actual and formal argument lists differ in length
1 error
```

Compiler Java akan memberikan pesan bahwa compiler mengharapkan argument saat memanggil constructor dan pada kode kita tidak terdapat argument untuk constructor tersebut.

## Menuliskan Constructor Tanpa Argument

Kita dapat juga menuliskan sebuah constructor yang tidak menerima argument apapun. Sebagai contoh, kita dapat menuliskan constructor berikut ke class `PersegiPanjang`:

```
public PersegiPanjang()
{
    panjang = 1.0;
    lebar = 1.0;
}
```

Kode berikut adalah contoh untuk membuat sebuah object dari class `PersegiPanjang`:

```
// Sekarang class PersegiPanjang mempunyai constructor tanpa argument
PersegiPanjang r = new PersegiPanjang();       // Memanggil constructor tanpa
argument
```

Setelah statement di atas diekseksi, object dari class `PersegiPanjang` yang direferensikan oleh variabel `r`, nilai-nilai dari field `panjang` dan `lebar` dari object tersebut akan ditetapkan ke 1.0.

### Constructor Class `String`

Kita telah melihat bahwa untuk menginstansiasi suatu class kita menggunakan operator `new` yang diikuti dengan nama class tersebut. Operator `new` ini lalu memanggil constructor dari class tersebut. Terdapat satu class yang dapat diinstansiasi tanpa operator `new` yaitu class `String`.

Karena operasi string sangat umum, Java memungkinkan kita untuk membuat sebuah object dari class `String` dengan cara yang sama seperti kita membuat variabel primitif. Berikut adalah contohnya:

```
String nama = "Herman Susilo";
```

Statement di atas membuat sebuah object dari class `String` dalam memori, menginisialisasinya dengan literal string `"Herman Susilo"`. Lalu, object tersebut direferensikan oleh variabel `nama`. Kita dapat juga menggunakan operator `new` untuk membuat sebuah object dari class `String`, dan menginisialisasi object tersebut dengan literal string ke constructornya, seperti terlihat berikut:

```
String nama = new String("Herman Susilo");
```

## 8.5 Method Overloading dan Constructor Overloading

Method overloading adalah bagian penting dalam object oriented programming. Method overloading adalah istilah ketika kita mempunyai lebih dari satu method dengan nama yang sama, namun menggunakan parameter-parameter dengan tipe-tipe data berbeda. Kita menggunakan method overloading ketika kita memerlukan sejumlah cara untuk melakukan operasi yang sama. Sebagai contoh, misalkan sebuah class mempunyai dua method berikut:

```
public int tambah(int a, int b)
{
    int jumlah = a + b;
    return tambah;
}

public String tambah(String str1, String str2)
{
    String kombinasi = str1 + str2;
    return kombinasi;
}
```

Kedua method di atas mempunyai nama `tambah`. Keduanya menerima dua argument, yang keduanya dijumlahkan. Method pertama menerima dua argument `int` dan mengembalikan jumlah kedua argument tersebut. Method kedua menerima dua referensi `String` dan mengembalikan sebuah referensi ke sebuah `String` yang merupakan hasil konkatenasi dari dua argument. Ketika kita memanggil method `tambah`, compiler harus menentukan method mana yang cocok dengan pemanggilan tersebut. Jika kita memanggil method `tambah` dengan argument dua `int` maka method pertama yang didefinisikan dengan dua parameter `int` yang dieksekusi. Sedangkan jika kita memanggil method `tambah` dengan dua argument `String`, method kedua yang didefinisikan dengan dua parameter `String` yang dieksekusi.

Bagaimana compiler Java menentukan method mana yang dieksekusi ketika method yang dioverloading dipanggil? Compiler Java menggunakan signature dari method untuk membedakan antara method bernama sama. Signature dari method terdiri dari nama method dan tipe data dari parameter-parameter method. Sebagai contoh, berikut adalah signature dari method `tambah` pada contoh sebelumnya:

```
tambah(int, int)
tambah(String, String)
```

Perhatikan bahwa tipe return method bukan merupakan bagian signature dari method. Oleh karena ini, method `add` berikut tidak dapat ditambahkan pada class yang sama dengan class yang mempunyai dua method sebelumnya:

```
public int tambah(String str1, String str2)
{
    int jumlah = Integer.parseInt(str1) + Integer.parseInt(str2)
    return jumlah;
}
```

Perhatikan method di atas mempunyai signature method yang sama dengan method kedua di atas yang mempunyai dua parameter `String`. Karena compiler Java hanya melihat signature dari method untuk membedakan method bernama sama satu sama lainnya, maka compiler Java tidak bisa membedakan method ini dengan method kedua pada contoh di atas, sehingga sebuah error akan didapatkan ketika kita mencoba mengkompilasi class dengan ketiga method-method di atas.

Constructor dapat juga di-overload, yang berarti sebuah class dapat mempunyai lebih dari satu constructor. Aturan untuk melakukan overloading constructor sama dengan aturan overloading method-method lain. Setiap versi dari constructor harus mempunyai daftar parameter-parameter yang berbeda. Selama setiap constructor mempunyai signature yang unik, compiler dapat membedakan satu dengan yang lainnya. Sebagai contoh, class `PersegiPanjang` yang kita bahas sebelumnya dapat memiliki dua constructor berikut:

```
public PersegiPanjang()
{
    panjang = 0.0;
    lebar = 0.0;
}

public PersegiPanjang(double pjg, double lbr)
{
    panjang = pjg;
    lebar = lbr;
}
```

Constructor pertama tidak menerima argument dan menugaskan 0.0 ke field `panjang` dan field `lebar`. Constructor kedua menerima dua argument, yang masing-masing ditugaskan ke field `panjang` dan field `lebar`. Kode berikut mencontohkan bagaimana setiap constructor dipanggil:

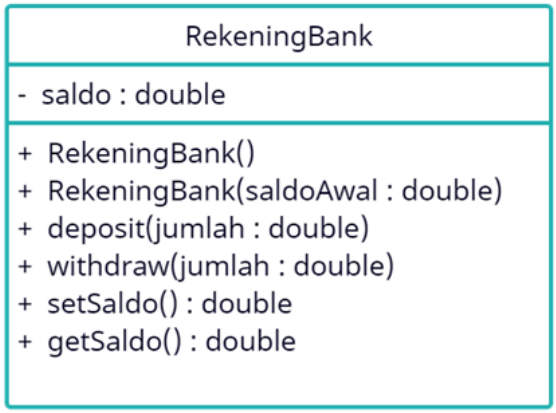
```
PersegiPanjang boks1 = new PersegiPanjang();
PersegiPanjang boks2 = new PersegiPanjang(5.0, 10.0);
```

Statement pertama membuat sebuah object `PersegiPanjang`, mereferensikannya dengan variabel `boks1`, dan mengeksekusi contructor tanpa argument. Field `panjang` dan field `lebar` pada object tersebut akan ditetapkan ke 0.0. Statement kedua membuat sebuah object `PersegiPanjang` lain, mereferensikannya dengan variabel `boks2` dan mengeksekusi constructor kedua yang menerima dua argument `double`. Pada object yang direferensikan `boks2`, field `panjang` akan ditetapkan dengan nilai 5.0 dan field `lebar` akan ditetapkan dengan nilai 10.0.

Ingat bahwa compiler Java hanya menuliskan otomatis constructor default jika kita tidak menuliskan sebuah constructor untuk class. Jika sebuah class hanya memiliki sebuah constructor yang menerima argument, class tersebut tidak akan secara otomatis dituliskan constructor default.

## Class `RekeningBank`

Sekarang kita akan melihat contoh sebuah class yang memiliki constructor yang di-*overloading*. Kita akan menuliskan class `RekeningBank` yang mererpresentasikan rekening bank pada suatu bank. Object yang dibuat dari class ini akan mensimulasikan rekening bank, yang memungkinkan kita untuk mempunyai saldo awal, membuat deposit, menarik uang, dan mendapatkan saldo terkini. Diagram UML untuk class `RekeningBank` yang akan kita buat ini dapat dilihat pada gambar berikut:




Seperti yang dapat kita lihat pada gambar di atas, class `RekeningBank` mempunyai dua constructor teroverloading. Class tersebut juga mempunyai sebuah method bernama `deposit`, sebuah method bernama `withdraw`, sebuah method bernama `setSaldo()`, dan sebuah method bernama `getSaldo()`. Kode berikut adalah kode definisi class `RekeningBank`:

***Definisi Class (RekeningBank.java)***

```
/*
    Class RekeningBank mensimulasikan sebuah rekening bank.
*/
public class RekeningBank
{
    private double saldo;       // Saldo rekening.

    /*
        Constructor ini menetapkan saldo awal
        ke 0.0.
    */
    public RekeningBank()
    {
        saldo = 0.0;
```

```
    }

    /*
        Constructor ini menetapkan saldo awal
        ke nilai yang diberikan sebagai argument.
        @param saldoAwal Saldo awal.
    */
    public RekeningBank(double saldoAwal)
    {
        saldo = saldoAwal;
    }

    /*
        Method deposit menaruh sejumlah uang
        ke rekening.
        @param jumlah Jumlah yang ditambahkan ke
                      field saldo.
    */
    public void deposit(double jumlah)
    {
        saldo = saldo + jumlah;
    }

    /*
        Method withdraw menarik sejumlah uang
        dari rekening.
        @param jumlah Jumlah yang dikurangi dari
                      field saldo.
    */
    public void withdraw(double jumlah)
    {
        if (saldo >= jumlah)
        {
            saldo = saldo - jumlah;
        }
        else
        {
            System.out.println("Dana tidak mencukupi.");
        }
    }

    /*
        Method setSaldo menetapkan saldo dari rekening.
        @param b Nilai untuk disimpan ke field saldo.

    */
    public void setSaldo(double s)
    {
        saldo = s;
    }

    /*
        Method getSaldo mengembalikan saldo rekening.
        @return Nilai dalam field saldo.
    */
    public double getSaldo()
    {
        return saldo;
```

```
    }
}
```

Class `RekeningBank` mempunyai satu field, `saldo`, yang bertipe `double`. Field ini menyimpan saldo terkini dari rekening. Class `RekeningBank` mempunyai dua constructor ter-overloading:

- Constructor pertama adalah constructor tanpa arugment. Constructor ini menetapkan field `saldo` ke 0.0. Contoh instansiasi dari class `RekeningBank` yang mengeksekusi constructor ini adalah:
  `RekeningBank rekening = new RekeningBank();`
- Constructor kedua adalah constructor yang menerima sebuah tipe `double` sebagai argument. Nilai dari argument ini ditugaskan ke field `saldo`. Contoh instansiasi dari class `RekeningBank` yang mengeksekusi constructor ini adalah:
  `RekeningBank rekening = new RekeningBank(1000000);`

Method `deposit` digunakan ketika sejumlah uang ditabung ke rekening. Method ini mempunyai sebuah parameter, `jumlah`, yang bertipe `double`. Ketika method ini dipanggil, nilai argument yang diberikan ditambahkan ke field `saldo`.

Method `withdraw` digunakan ketika sejumlah uang ditarik dari rekening. Method ini mempunyai sebuah parameter, `jumlah`, yang bertipe `double`. Ketika method ini dipanggil, pertama-tama nilai field `saldo` dibandingkan dengan nilai argument yang diberikan. Jika nilai field `saldo` lebih besar atau sama dengan dari nilai argument yang diberikan, maka nilai field `saldo` dikurangi dengan nilai argument. Jika nilai field `saldo` lebih kecil, method ini mencetak pesan "Dana tidak mencukupi."

Method `setSaldo` digunakan untuk menetapkan nilai dari field `saldo`. Method ini menerima sebuah argument `double`. Ketika method ini dieksekusi, nilai argument ditugaskan ke field `saldo`.

Method `getSaldo` digunakan untuk mendapatkan saldo dari rekening. Method ini mengembalikan nilai field `saldo`.

Program berikut mendemonstrasikan penggunakan class `RekeningBank`:

```
import java.util.Scanner;

/*
    Program ini mendemonstrasikan penggunakan
    class RekeningBank
*/
public class UjiRekeningBank
{
    public static void main(String[] args)
    {
        String input;   // Untuk menyimpan input pengguna.

        // Buat sebuah object Scanner untuk menerima input pengguna
        Scanner keyboard = new Scanner(System.in);

        // Dapatkan saldo awal dari pengguna
        System.out.print("Berapa saldo awal rekening Anda? ");
        input = keyboard.nextLine();

        // Buat sebuah object RekeningBank
        RekeningBank rekening = new RekeningBank(Double.parseDouble(input));
```

```
        // Dapatkan jumlah yang ditabung
        System.out.print("Berapa gaji Anda bulan ini? ");
        input = keyboard.nextLine();

        // Tabung jumlah gaji yang diterima pengguna
        rekening.deposit(Double.parseDouble(input));

        // Tampilkan saldo baru setelah gaji ditabung
        System.out.println("Gaji Anda telah didepositokan.");
                    System.out.printf("Saldo   Anda   sekarang   =   Rp.%,.2f\n",
rekening.getSaldo());

        // Dapatkan jumlah uang yang ingin ditarik dari rekening.
        System.out.print("Berapa jumlah yang ingin Anda tarik? ");
        input = keyboard.nextLine();

        // Tarik jumlah yang ingin ditarik dari rekening.
        rekening.withdraw(Double.parseDouble(input));

        // Tampilkan saldo akhir
                    System.out.printf("Saldo   Anda   sekarang   =   Rp.%,.2f\n",
rekening.getSaldo());
    }
}
```

***Output Program (UjiRekeningBank.java)***

```
Berapa saldo awal rekening Anda? 1500000
Berapa gaji Anda bulan ini? 3500000
Gaji Anda telah didepositokan.
Saldo Anda sekarang = Rp.5,000,000.00
Berapa jumlah yang ingin Anda tarik? 2750000
```

# 8.6 Variabel Referensi `this`

Keyword `this` adalah variabel referensi yang dapat digunakan oleh sebuah object untuk mereferensikan dirinya sendiri.

## Menggunakan `this` untuk Mereferensikan Field

Ketika kita menuliskan method instance kita harus menggunakan nama variabel parameter yang berbeda dari nama field. Sebagai contoh, pada method `setPanjang` dari class `PersegiPanjang` kita menggunakan nama variabel parameter `pjg` untuk membedakannya dengan field `panjang`:

```
public void setPanjang(double pjg)
{
    panjang = pjg;
}
```

Kita tidak dapat menggunakan nama variabel parameter yang sama dengan nama field. Jika kita menggunakan nama variabel parameter yang sama dengan nama field kita akan kehilangan akses ke field tersebut.

Terkadang sulit dan memerlukan waktu yang tidak sebentar untuk memikirkan nama parameter yang berbeda dengan nama field. Untuk menghindari kesulitan ini, kita dapat menggunakan nama parameter yang sama dengan nama field yang terkait, dengan menambahkan keyword `this` untuk mereferensikan nama `field`. Sebagai contoh, method `setPanjang` dapat kita tulis ulang sebagai berikut:

```
public void setPanjang(double panjang)
{
    this.panjang = panjang;
}
```

## Menggunakan `this` untuk Memanggil Constructor Lain dari Sebuah Contructor

Selain untuk mereferensikan field, kita dapat juga menggunakan `this` untuk memanggil constructor lain dalam sebuah class. Pada class `PersegiPanjang`, kita mempunyai constructor yang menerima dua argument seperti berikut:

```
public PersegiPanjang(double pjg, double lbr)
{
    panjang = pjg;
    lebar = lbr;
}
```

Misalkan kita ingin menambahkan sebuah constructor lain yang menerima semua argument dan menugaskan nilai argument tersebut ke field `panjang` dan field `lebar`. Kita dapat menuliskan contructor tersebut seperti berikut:

```
public PersegiPanjang(double sisi)
{
    this(sisi, sisi);
}
```

Constructor di atas menggunakan variabel referensi `this` untuk memanggil constructor yang menerima dua argument. Constructor ini memberikan nilai dalam variabel parameter `sisi` sebagai argument ke parameter `pjg` dan parameter `lbr` dari constructor yang menerima dua argument. Hasil dari pemanggilan contrusctor ini adalah nilai dalam `sisi` ditugaskan ke field `panjang` dan field `lebar`.

Ketika kita menggunakan `this` untuk memanggil constructor lain, kita harus memperhatikan dua aturan berikut:

- `this` hanya dapat digunakan untuk memanggil sebuah constructor dari constructor lain dalam class yang sama.
- Statement pemanggilan constuctor lain dengan `this` harus dituliskan sebagai statement pertama dalam constructor yang memanggil. Jika tidak, error kompilasi akan terjadi.

Kode berikut adalah kode class `PersegiPanjang` yang ditulis ulang menggunakan `this` untuk mereferensikan field dan menggunakan `this` untuk memanggil constructor lain:

***Definisi Class*** `PersegiPanjang`

```
/*
     Class PersegiPanjang yang menggunakan keyword this.
*/
public class PersegiPanjang
{
    private double panjang;
    private double lebar;


    /*
        Constructor
        @param pjg Panjang dari persegi panjang.
        @param lbr Lebar dari persegi panjang.
    */
    public PersegiPanjang()
    {
        panjang = 0.0;
        lebar = 0.0;
    }

    /*
        Constructor
        @param pjg Panjang dari persegi panjang.
        @param lbr Lebar dari persegi panjang.
    */
    public PersegiPanjang(double panjang, double lebar)
    {
        this.panjang = panjang;
        this.lebar = lebar;
    }

    /*
        Constructor
        @param sisi Panjang dan lebar dari persegi panjang.
    */
    public PersegiPanjang(double sisi)
    {
        this(sisi, sisi);
    }

    /*
        Method setPanjang menyimpan sebuah nilai
        dalam field panjang.
        @param pjg Nilai yang disimpan dalam field panjang.
    */
    public void setPanjang(double panjang)
    {
        this.panjang = panjang;
    }

    /*
        Method setLebar menyimpan sebuah nilai
        dalam field lebar.
        @param lbr Nilai yang disimpan dalam field lebar.
    */
```

```
    public void setLebar(double lebar)
    {
        this.lebar = lebar;
    }

    /*
        Method getPanjang mengembalikan panjang dari
        object PersegiPanjang.
        @return Nilai dalam field panjang
    */
    public double getPanjang()
    {
        return panjang;
    }

    /*
        Method getLebar mengembalikan lebar dari
        object PersegiPanjang.
        @return Nilai dalam field lebar
    */
    public double getLebar()
    {
        return lebar;
    }

    /*
        Method getLuas mengembalikan luas dari
        object PersegiPanjang.
        @return Hasil dari panjang kali lebar.
    */
    public double getLuas()
    {
        return panjang * lebar;
    }
}
```

# 8.7 Menggunakan Object dengan Method

Object dapat diberikan ke sebuah method sebagai argument. Object juga dapat dikembalikan oleh sebuah method. Pada bagian ini kita akan membahas cara memberikan object ke method dan cara mengembalikan object dari sebuah method.

## Memberikan Object ke Method

Pada pembahasan method kita telah melihat bagaimana nilai-nilai tipe primitif, dan juga referensi-referensi ke object `String` dapat diberikan sebagai argument-argument ke method. Kita dapat juga memberikan referensi-referensi dari object-object selain `String` sebagai argument ke method. Sebagai contoh, program berikut mendemonstrasikan sebuah method yang menerima argument berupa referensi ke object `PersegiPanjang`:

***Program (PassObject.java)***

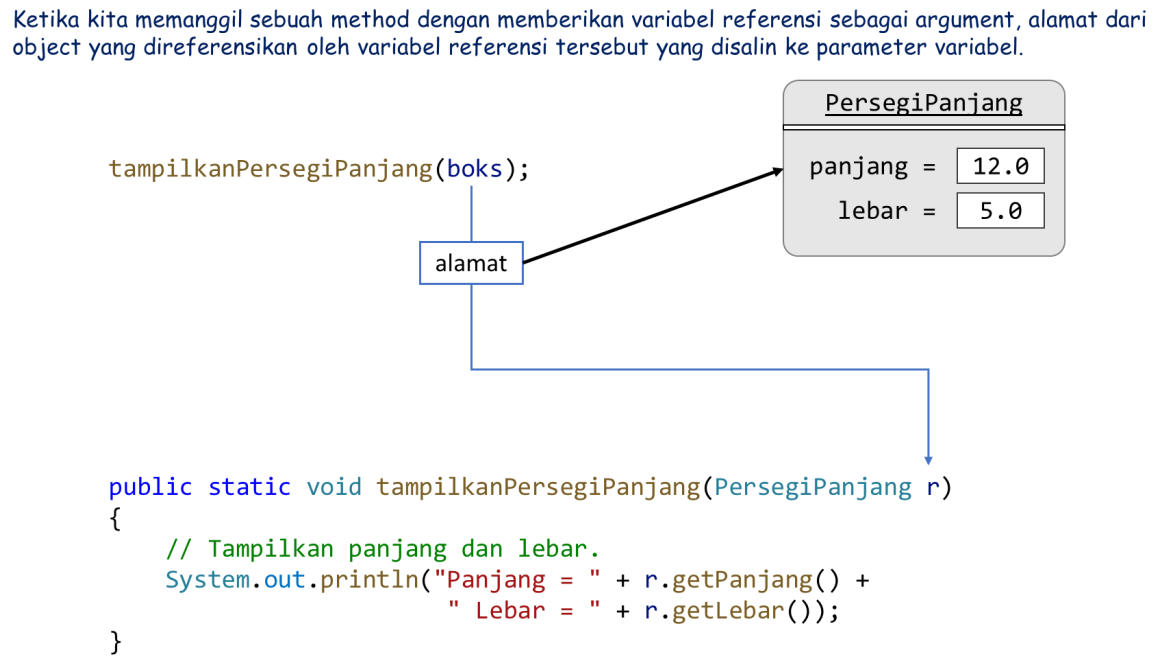
```
/*
    Program ini memberikan sebuah object sebagai argument
*/
public class PassObject
{
    public static void main(String[] args)
    {
        // Buat object PersegiPanjang
        PersegiPanjang boks = new PersegiPanjang(12.0, 5.0);

        // Berikan sebuah referensi ke object PersegiPanjang
        // sebagai argument ke method tampilkanPersegiPanjang.
        tampilkanPersegiPanjang(boks);
    }

    /*
        Method tampilkanPersegiPanjang menampilkan
        panjang dan lebar dari sebuah persegi panjang.
        @param r Sebuah referensi ke sebuah object PersegiPanjang.
    */
    public static void tampilkanPersegiPanjang(PersegiPanjang r)
    {
        // Tampilkan panjang dan lebar.
        System.out.println("Panjang = " + r.getPanjang() +
                           " Lebar = " + r.getLebar());
    }
}
```

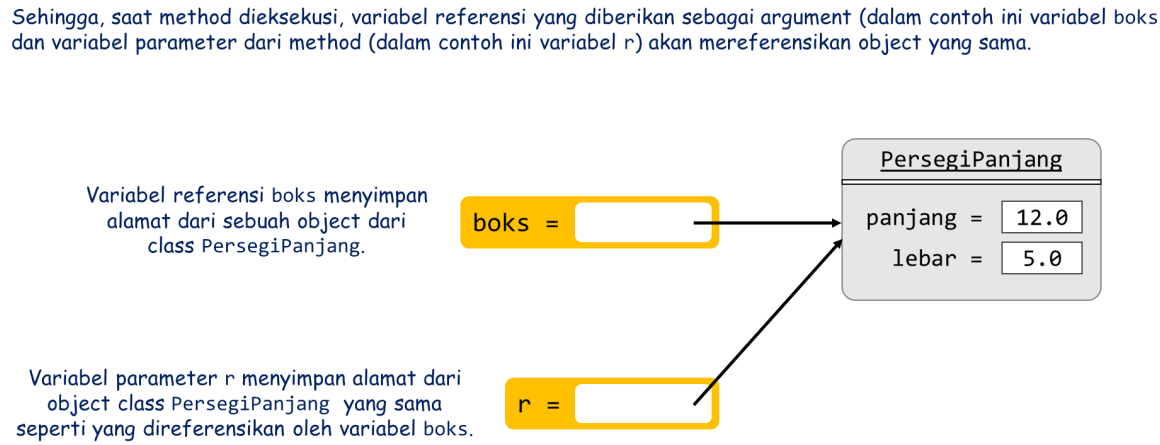
***Output Program (PassObject.java)***

```
Panjang = 12.0 Lebar = 5.0
```

Dalam method `main` dari program di atas, variabel `boks` adalah sebuah variabel referensi ke object `PersegiPanjang`. Pada baris 13, variabel `boks` diberikan sebagai argument ke method `tampilkanPersegiPanjang`. Saat method ini dieksekusi, alamat ke object `PersegiPanjang` yang diberikan melalui variabel `boks` disalin ke variabel parameter `r` yang juga merupakan variabel referensi ke class `PersegiPanjang`. Ini berarti saat method `tampilkanPersegiPanjang`

dieksekusi, variabel `boks` dan `r` keduanya mereferensikan object yang sama. Ini diilustrasikan oleh gambar berikut:




Ingat pada Topik Method, ketika kita memberikan sebuah variabel bertipe primitif sebagai argument ke sebuah method, nilai yang disimpan dalam variabel tersebut disalin ke variabel parameter dari method tersebut. Ini berarti, jika method tersebut mengubah nilai dari variabel parameter, hal ini tidak berpengaruh pada isi dari variabel yang diberikan sebagai argument. Namun, ketika kita memberikan sebuah variabel referensi sebagai argument ke sebuah method, variabel parameter yang menerimanya akan mereferensikan object yang sama. Ini berarti method tersebut mempunyai akses ke object yang direferensikan oleh variabel referensi tersebut. Sehingga, method yang menerima referensi object sebagai argument dapat memodifikasi isi dari object yang diterimanya. Program berikut mendemonstrasikan ini:

***Program (PassObject2.java)***

```
/*
    Program ini memberikan sebuah object sebagai argument.
    Object tersebut dimodifikasi oleh method yang menerimanya.
*/
public class PassObject2
{
    public static void main(String[] args)
    {
        // Buat object PersegiPanjang
        PersegiPanjang boks = new PersegiPanjang(12.0, 5.0);

        // Tampilkan isi object
        System.out.println("Isi dari object boks:");
        System.out.println("Panjang = " + boks.getPanjang() +
                           " Lebar = " + boks.getLebar());

        // Berikan referensi ke object ke method ubahPersegiPanjang.
        ubahPersegiPanjang(boks);

        // Tampilkan isi object
        System.out.println("\nIsi dari object boks sekarang:");
        System.out.println("Panjang = " + boks.getPanjang() +
                            " Lebar = " + boks.getLebar());

    }

    /*
        Method ubahPersegiPanjang menetapkan field panjang dan lebar
        dari object PersegiPanjang ke 0.0
    */
    public static void ubahPersegiPanjang(PersegiPanjang r)
    {
        r.setPanjang(0.0);
        r.setLebar(0.0);
    }
}
```

***Output Program (PassObject2.java)***

```
Isi dari object boks:
Panjang = 12.0 Lebar = 5.0

Isi dari object boks sekarang:
Panjang = 0.0 Lebar = 0.0
```

## Mengembalikan Object dari Method

Method juga dapat mengembalikan sebuah referensi ke suatu object. Sebagai contoh, program berikut menggunakan class `RekeningBank` dan menggunakan method `getRekening` yang mengeamblikan sebuah refrensi ke object `RekeningBank`:

***Program (ReturnObject.java)***

```
import java.util.Scanner;

/*
    Program ini mendemonstrasikan bagaimana sebuah method
    dapat mengembalikan sebuah referensi ke object.
*/
public class ReturnObject
{
    public static void main(String[] args)
    {
        RekeningBank rekening;

        // Dapat sebuah referensi ke sebuah object RekeningBank
        rekening = getRekening();

        // Tampilkan saldo dari rekening.
        System.out.printf("Rekening mempunyai saldo Rp.%,.2f\n",
                          rekening.getSaldo());
    }

    /*
        Method getRekening membuart sebuah object RekeningBank
        dengan saldo yang diberikan oleh pengguna.
        @return Sebuah referensi ke object.
    */
    public static RekeningBank getRekening()
    {
        double saldo;   // Untuk menyimpan saldo.

        // Dapatkan saldo dari pengguna
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Masukkan saldo awal: ");
        saldo = keyboard.nextDouble();

        // Buat sebuah object RekeningBank dan
        // kembalikan referensi ke object tersebut.
        return new RekeningBank(saldo);
    }
}
```

***Output Program (ReturnObject.java)***

```
Masukkan saldo awal: 2500000
Rekening mempunyai saldo Rp.2,500,000.00
```

Perhatikan method `getRekening` mempunyai tipe return `RekeningBank`. Gambar berikut menunjukkan tipe return dari method ini:


Method yang mempunyai tipe return `RekeningBank` berarti method tersebut mengembalikan sebuah referensi ke object `RekeningBank`. Statement berikut, pada baris 14 dari program di atas, menugaskan nilai return dari method `getRekening` ke variabel `rekening`:

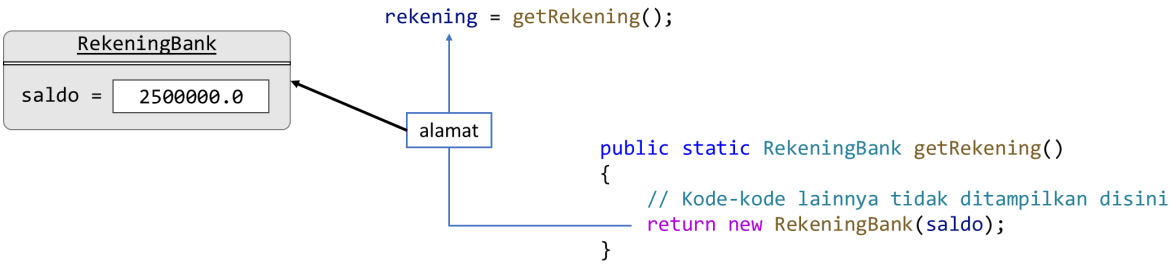
```
rekening = getRekening();
```

Setelah statement di atas dieksekusi, variabel `rekening` akan mereferensikan object `RekeningBank` yang dikembalikan oleh method `getRekening`.

Sekarang mari kita lihat method `getRekening`. Method ini meminta pengguna memasukkan saldo awal rekening, lalu menggunakan saldo yang diberikan untuk membuat sebuah object `RekeningBank`. Statement terakhir dari method `getRekening`:

```
return new RekeningBank(saldo);
```

Staement ini menggunakan keyword `new` untuk membuat sebuah object `RekeningBank`, memberikan `saldo` sebagai argument ke constructor. Alamat dari object yang dibuat ini kemudian dikembalikan dari method `getRekening`. Gambar berikut mengilustrasikan bagaimana method `getRekening` dipanggil dan mengembalikan alamat dari object yang dibuatnya:

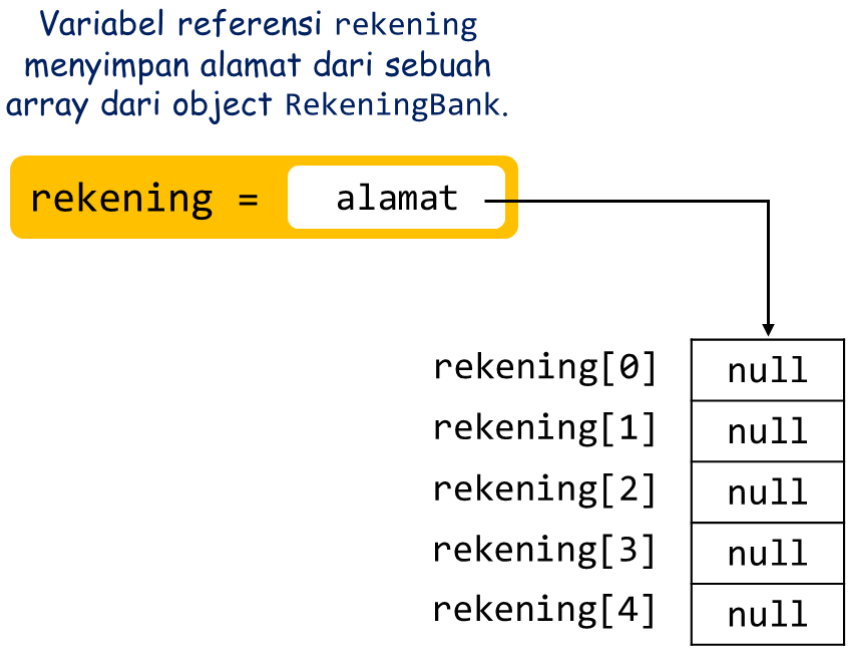


# 8.8 Array dari Object

Seperti tipe-tipe data lainnya, kita dapat membuat array dari object-object. Sebagai contoh, kita dapat membuat sebuah array yang berisi object-object dari class `RekeningBank`. Kode berikut mendeklarasikan sebuah array dari lima object `RekeningBank`:

```
final int BANYAK_REKENING = 5;
RekeningBank[] rekening = new RekeningBank(BANYAK_REKENING);
```

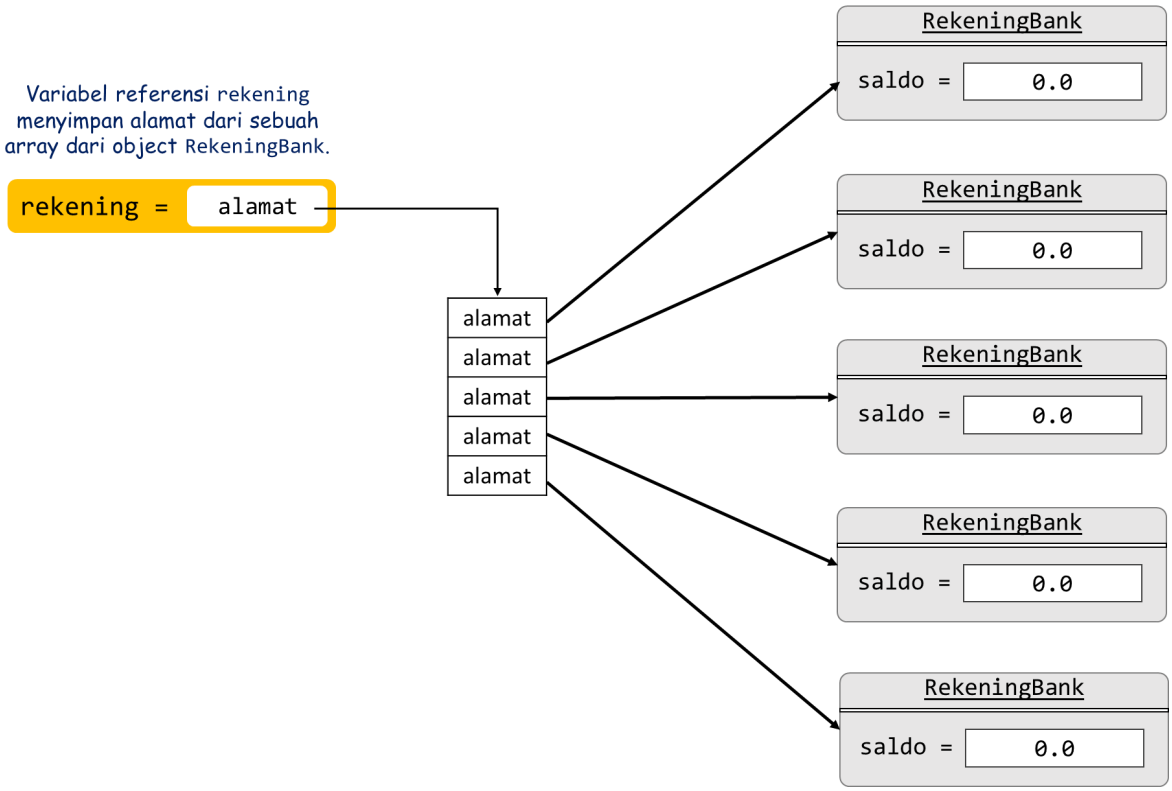
Variabel yang mereferensikan array dari object `RekeningBank` bernama `rekening`. Sama seperti array dari `String`, setiap elemen dari array ini adalah sebuah variabel referensi. Gambar berikut mengilustrasikan ini:




Perhatikan pada gambar, setiap elemen dari array diinisialisasi ke nilai `null`. Nilai `null` adalah nilai spesial dalam Java yang menandakan bahwa elemen-elemen array belum mereferensikan object. Kita harus secara individu membuat object-object yang akan direferensikan oleh elemen-elemen. Kode berikut menggunakan loop untuk membuat object-object `RekeningBank` untuk setiap elemen:

```
for (int index = 0; index < rekening.length; index++)
{
    rekening[index] = new RekeningBank();
}
```

Pada kode di atas, pembuatan object `RekeningBank` dilakukan melalui constructor tanpa argument. Ingat, constructor tanpa argument dari class `RekeningBank` yang kita tulis sebelumnya menugaskan nilai 0.0 ke field `saldo`. Setelah loop dieksekusi, setiap elemen dari array `rekening` akan mereferensikan sebuah object `RekeningBank`. Ini diilustrasikan pada gambar berikut:




Sama seperti pada array dari tipe data primitif, object-object dalam sebuah array diakses menggunakan notasi subscript. Sebagai contoh, kode berikut menggunakan elemen `rekening[2]` untuk memanggil method `setSaldo` dan method `withdraw`:

```
rekening[2].setSaldo(1500000);
rekening[2].withdraw(100000);
```

Program berikut mendemonstrasikan penggunaan array dari object-object:

***Program (ArrayObject.java)***

```
import java.util.Scanner;

/*
    Program ini bekerja dengan sebuah array dari
    tiga object RekeningBank.
*/
public class ArrayObject
{
    public static void main(String[] args)
    {
        final int BANYAK_REKENING = 3;  // Banyak rekening.

        // Buat sebuah array yang dapat mereferensikan
        // object-object RekeningBank.
        RekeningBank[] rekening = new RekeningBank[BANYAK_REKENING];

        // Buat object-object dari array
        buatRekening(rekening);

        // Tampilkan saldo dari masing-masing rekening.
        System.out.println("Berikut adalah saldo " +
                           "dari setiap rekening: ");
```

```
        for (int index = 0; index < rekening.length; index++)
        {
            System.out.print("Rekening " + (index + 1) + ": Rp.");
            System.out.printf("%,.2f\n", rekening[index].getSaldo());
        }
    }

    /*
        Method buatRekening membuat sebuah object RekeningBank
        untuk setiap elemen dari sebuah array. Pengguna diminta
        memasukkan saldo dari setiap rekeing.
        @param array Array yang mereferensikan rekening-rekening.
    */
    public static void buatRekening(RekeningBank[] array)
    {
        double saldo;   // Untuk menyimpan saldo yang diinput pengguna

        // Buat object Scanner
        Scanner keyboard = new Scanner(System.in);

        // Buat rekening
        for (int index = 0; index < array.length; index++)
        {
            // Dapatkan saldo rekening
            System.out.print("Masukkan saldo untuk " +
                             "rekening " + (index + 1) + ": ");
            saldo = keyboard.nextDouble();

            // Buat rekening
            array[index] = new RekeningBank(saldo);
        }
    }
}
```

***Output Program (ArrayObject.java)***

```
Masukkan saldo untuk rekening 1: 1250000
Masukkan saldo untuk rekening 2: 2750000
Masukkan saldo untuk rekening 3: 5425000
Berikut adalah saldo dari setiap rekening:
Rekening 1: Rp.1,250,000.00
Rekening 2: Rp.2,750,000.00
Rekening 3: Rp.5,425,000.00
```

# 8.9 Member Static dari Class

Setiap instance dari class mempunyai field-field tersendiri yang disebut sebagai field-field instance. Kita dapat membuat sejumlah instance dari suatu class dan setiap instance akan mempunyai field-field instance tersendiri. Sebagai contoh, misalkan variabel `boks` mereferensikan sebuah instance dari class `PersegiPanjang` dan kita mengeksekusi statetment berikut:

```
boks.setPanjang(10);
```

Statement di atas menyimpan nilai 10 ke field `panjang` dari instance yang direferensikan oleh `boks`.

Sebuah class juga dapat mempunyai method instance. Method instance adalah method yang bekerja pada suatu instance dari sebuah class dimana method tersebut didefinisikan. Ketika kita memanggil method instance, method tersebut melakukan suatu operasi pada instance tertentu dari class tersebut. Sebagai contoh, misalkan variabel `boks` mereferensikan sebuah instance dari class `PersegiPanjang` dan kita mempunyai statement berikut:

```
x = boks.getPanjang();
```

Statement di atas memanggil method `getPanjang`, yang mengembalikan nilai dari field `panjang` yang dimiliki oleh instance dari class `PersegiPanjang` yang direferensikan oleh variabel `boks`. Field instance dan method instance keduanya terasosiasikan ke sebuah instance tertentu dari suatu class, dan mereka tidak dapat digunakan sampai sebuah instance dari class dibuat.

## Member Static

Kita dapat membuat sebuah field atau method yang bukan merupakan bagian dari instance dari suatu class. Member-member yang bukan merupakan bagian dari instance disebut sebagai member static. Misalkan, kita dapat membuat field static dalam sebuah class. Field static ini tidak disimpan dalam sebuah instance dari class tersebut. Field ini tidak memerlukan sebuah instance untuk dapat menyimpan suatu nilai. Kita juga dapat membuat method static pada sebuah class. Method static tidak bekerja pada field-field dari instance tertentu dari class yang memiliki method tersebut. Kita tidak perlu membuat sebuah instance dari class tersebut untuk memanggil method static.

## Field Static

Field static dideklarasikan dengan menuliskan keyword `static`. Ketika sebuah field dideklarasikan sebagai static, hanya ada satu salinan field tersebut dalam memori meskipun terdapat lebih dari satu instance dari class tempat field tersebut dideklarasikan. Sebagai contoh, class `Countable` berikut menggunakan field `static` untuk menghitung banyaknya instance yang dibuat:

***Definisi Class `Countable` (Countable.java)***

```
/*
    Class ini mendemonstrasikan field static.
*/
public class Countable
{
    private static int counterInstance = 0;

    /*
        Constructor ini menginkrementasi fiels static
        counterInstance. Ini untuk menghitung banyaknya
        instance dari class ini yang dibuat.
    */
    public Countable()
    {
        counterInstance++;
    }

    /*
        Method getCounterInstance mengembalikan
        banyaknya instance dari class ini yang telah dibuat.
        @return Nilai dalam field counterInstance.
    */
    public int getCounterInstance()
    {
        return counterInstance;
    }
}
```

Pada baris 6, kita mendeklarasikan sebuah field static bernama `counterInstance` dan menginisialisasinya ke 0:

```
private static int counterInstance = 0;
```

Kita mendeklarasikan field static dengan menuliskan keyword `static` setelah access modifier dan sebelum tipe data dari field.

Pada baris 13 sampai dengan 16, kita menuliskan constructor dari class `Countable`. Constructor ini menggunakan operator inkrementasi `++` untuk menginkrementasi field `counterInstance`. Ini berarti, setiap kali sebuah instance dari class `Countable` dibuat, constructor ini akan dipanggil dan field `counterInstance` akan diinkrementasi. Sehingga, field `counterInstance` akan berisi banyaknya instance dari class `Countable` yang telah dibuat.

Pada baris 23 sampai dengan 26 adalah definisi method instance bernama `getCounterInstance`. Method ini mengembalikan nilai yang disimpan dalam `counterInstance`.

Program berikut mendemonstrasikan class `Countable`:

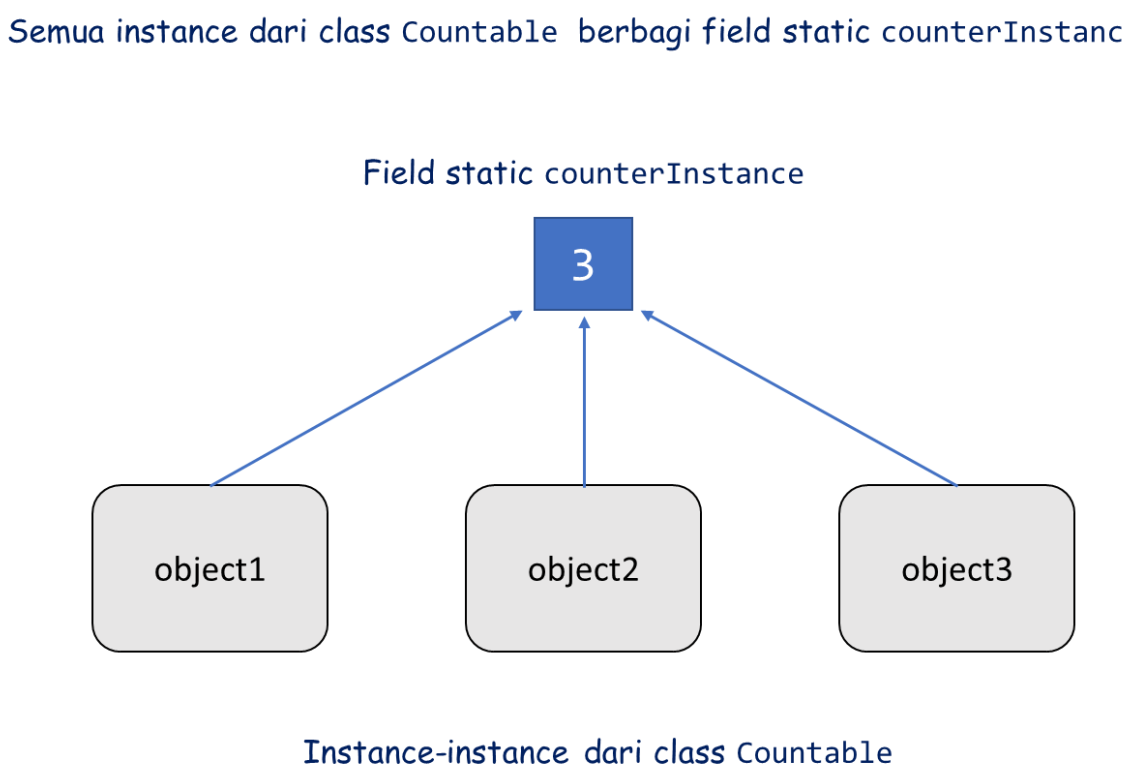
***Program (DemoStatic.java)***

```
/*
    Program ini mendemonstrasikan class Countable.
*/
public class DemoStatic
{
    public static void main(String[] args)
    {
        int banyakObject;

        // Buat tiga instance dari class Countable
        Countable object1 = new Countable();
        Countable object2 = new Countable();
        Countable object3 = new Countable();

        // Dapatkan banyaknya instance melalui
        // field static dari class
        banyakObject = object1.getCounterInstance();
        System.out.println("Terdapat " + banyakObject +
                           " instance dari class Countable yang telah dibuat.");
    }
}
```

***Output Program (DemoStatic.java)***

```
Terdapat 3 instance dari class Countable yang telah dibuat.
```

Program di atas membuat tiga instance dari class `Countable` dan mereferensikannya dengan variabel `object1`, `object2`, dan `object3`. Meskipun terdapat tiga instance dari class, hanya terdapat satu field static. Ini diilustrasikan pada gambar berikut:




Pada baris 17, program di atas memanggil method `getCounterInstance` untuk mendapatkan banyaknya instance yang telah dibuat:

```
banyakObject = object1.getCounterInstance();
```

Meskipun program ini memanggil method `getCounterInstance` dari `object1`, nilai yang sama akan dikembalikan jika program ini memanggilnya dari object-object lainnya.

## Method Static

Class dengan method static tidak perlu diinstansiasi untuk mengeksekusi method static dalam class tersebut. Perhatikan definisi class berikut:

***Definisi Class `Metrik` (Metrik.java)***

```
/*
    Class ini mendemonstrasikan method-method static.
*/
public class Metrik
{
    /*
        Method milKeKilometer mengkonversi jarak dalam mil
        ke kilometer.
        @param mil Jarak dalam mil.
        @return Jarak dalam km.
    */
    public static double milKeKilometer(double mil)
    {
        return mil * 1.609;
    }

    /*
        Method kilometerKeMil mengkonversi jarak dalam km
        ke mil.
        @param km Jarak dalam km.
        @return Jarak dalam mil.
    */
    public static double kilometerKeMil(double km)
    {
        return km / 1.609;
    }
}
```

Method static didefinisikan dengan menuliskan keyword `static` setelah access specifier pada header method. Class `Metrik` di atas mempunyai dua method static: `milKeKilometer` dan `kilometerKeMil`. Karena, kedua method ini static, keduanya dapat dipanggil tanpa harus membuat suatu instance dari class `Metrik` terlebih dahulu. Kita dapat memanggil method static dengan menuliskan nama class yang diikuti dengan titik sebelum nama method dan argument-argument dalam tanda kurung. Berikut adalah contoh pemanggilan method `milKeKilometer`:

```
km = Metrik.milKeKilometer(10.0);
```

`Metrik.milKeKilometer(10.0)` adalah pemanggilan method `milKeKilometer` dalam class `Metrik` dengan memberikan nilai 10 sebagai argument ke method tersebut. Nilai return dari pemanggilan method ini lalu ditugaskan ke variabel `km`. Perhatikan pada statement di atas, method static `milKeKilometer` tidak dipanggil dari sebuah instance dari class `Metrik`, tetapi

dipanggil langsung dari class `Metrik`. Program berikut mendemonstrasikan penggunaan class `Metrik`:

***Program (DemoMetrik.java)***

```
import java.util.Scanner;

/*
    Program ini mendemonstrasikan penggunaan
    method-method static dari class Metric.
*/
public class DemoMetrik
{
    public static void main(String[] args)
    {
        double mil;
        double km;

        // Buat object Scanner untuk mendapatkan input keyboard
        Scanner keyboard = new Scanner(System.in);

        // Dapatkan jarak dalam mil.
        System.out.print("Masukkan jarak dalam mil: ");
        mil = keyboard.nextDouble();

        // Konversi jarak dari mil ke km dan tampilkan hasilnya.
        km = Metrik.milKeKilometer(mil);
        System.out.printf("%.2f mil = %.2f km.\n", mil, km);

        // Dapatkan jarak dalam km.
        System.out.print("Masukkan jarak dalam km: ");
        km = keyboard.nextDouble();

        // Konversi jarak dari km ke mil dan tampilkan hasilnya.
        mil = Metrik.kilometerKeMil(km);
        System.out.printf("%.2f km = %.2f mil.\n", km, mil);
    }
}
```

***Output Program (DemoMetrik.java)***

```
Masukkan jarak dalam mil: 10
10.00 mil = 16.09 km.
Masukkan jarak dalam km: 100
100.00 km = 62.15 mil.
```

Static method umumnya digunakan untuk membuat class-class utilitas yang melakukan operasi-operasi kalkulasi terhadap data tetapi tidak memerlukan untuk menyimpan data tersebut. Class `Math` yang sudah tersedia dalam Java adalah contoh class utilitas. Kita telah melihat pada topik-topik sebelumnya, class `Math` tidak perlu diinstansiasi terlebih dahulu untuk menjalankan method-method di dalamnya.

**REFERENSI**

[1] Horstmann, Cay S. 2012. *Big Java: Late Objects, 1st Edition*. United States of America: John Wiley & Sons, Inc.

[2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.