



POLITEKNIK MASAMY INTERNASIONAL

SK Menristekdikti RI Nomor: 731/KPT/I/2018

Jalan Ikan Paus No.10-15 Kertosari Banyuwangi - 68411
Telp (0333) 3384593 – <http://polmain.info>

PROGRAM STUDI D3 TEKNIK KOMPUTER

Form:
B.Ak/eva/04/20

Nama Dosen : *Arif Fahmi, S.T.,M.T.*

Mata Kuliah : Pemrograman Berorientasi Object

Semester : 4 (Empat)

Kode Mata Kuliah : TKV4044

Th. Akdm : 2019/2020.

BAB

INHERITANCE DAN METHOD

SUB BAB :

1. Inheritance
2. Method

TUJUAN MATERI

1. Mahasiswa mampu memahami konsep inheritance dan method
2. Mahasiswa mampu membuat program dengan menggunakan konsep inheritance dan method

REFERENSI

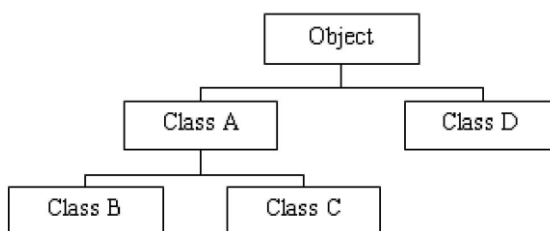
1. J.Eck, D. (2006). *Introduction to Programming Using Java*. Geneva.
2. An Object-Oriented Approach to Programming Logic and Design, Joyce Farrel, USA, 2013
3. An Introduction to Object Oriented Programming with Java, C. Thomas Wu, McGraw-Hill, New York, 2010.

INHERITANCE DAN METHOD

1. Inheritance (Pewarisan)

Pada dasarnya inheritance ini merupakan konsep pemrograman dimana sebuah class dapat mewariskan atau menurunkan property, method atau data data yang dimilikinya kepada class yang bertindak sebagai child. Jadi class tersebut dapat mengakses data-data dari class utamanya yang **bertindak sebagai parent**.

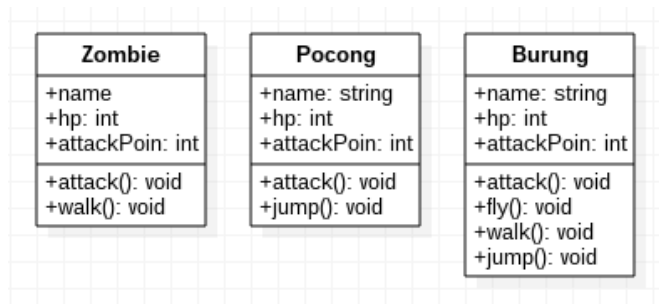
Sebuah class di Java, bisa memiliki satu atau lebih keturunan atau class anak. Class anak akan memiliki warisan properti dan method dari class ibu. Beberapa class di atas class utama dalam hirarki class dikenal sebagai superclass. Sementara beberapa class di bawah class pokok dalam hirarki class dikenal sebagai subclass dari class tersebut.



Pewarisan adalah keuntungan besar dalam pemrograman berbasis object karena suatu sifat atau method didefinisikan dalam *superclass*, sifat ini secara otomatis diwariskan dari semua *subclasses*. Jadi, Anda dapat menuliskan kode method hanya sekali dan mereka dapat digunakan oleh semua subclass. Subclass hanya perlu mengimplementasikan perbedaannya sendiri dan induknya

Contoh Studi kasus,

Misalkan dalam Game, kita akan membuat class-class musuh dengan perilaku yang berbeda.



Lalu kita membuat kode untuk masing-masing kelas seperti ini:

File: Zombie.java

```
class Zombie {
    String name;
    int hp;
    int
    attackPoin;

    void attack(){
        // ...
    }

    void walk(){
        //...
    }
}
```

File: Pocong.java

```
class Pocong {
    String name;
    int hp;
    int
    attackPoin;

    void attack(){
        // ...
    }

    void jump(){
        //...
    }
}
```

File: Burung.java

```
class Burung {
    String name;
    int hp;
    int attackPoin;
    void attack(){
        // ...
    }

    void walk(){
        //...
    }

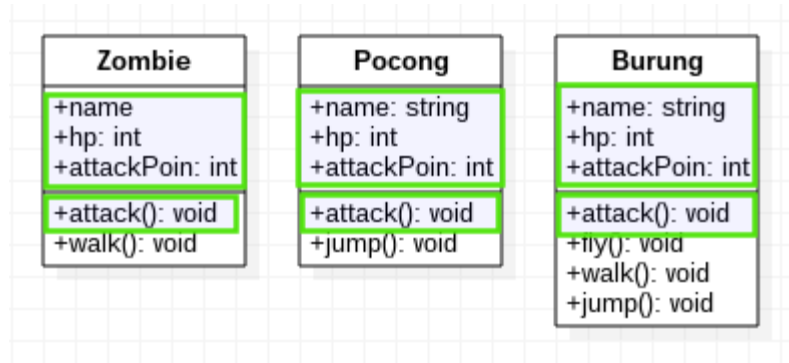
    void jump(){
        //...
    }

    void fly(){
        //...
    }
}
```

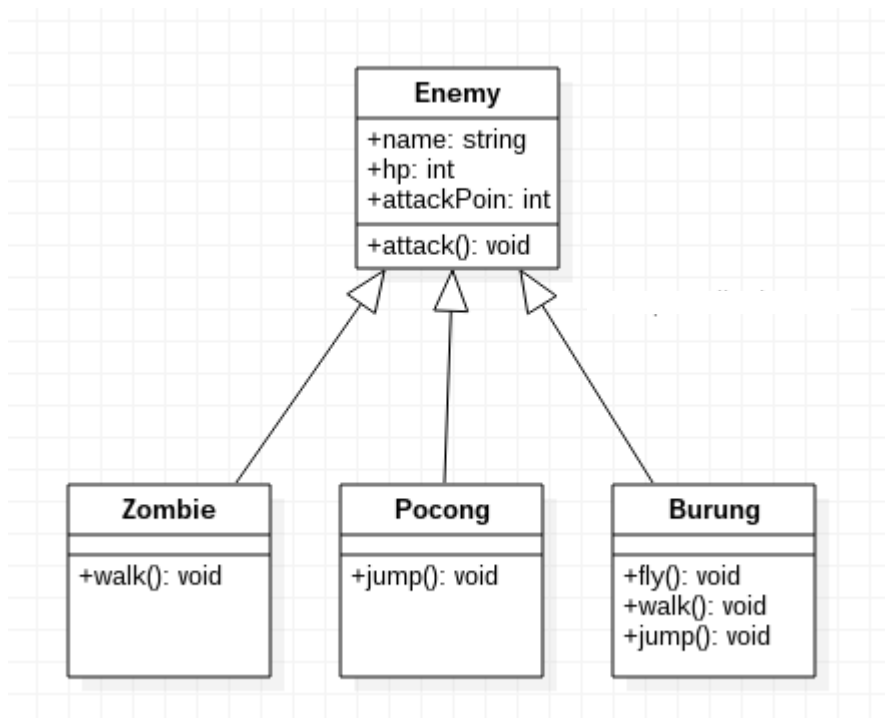
Apakah boleh seperti ini? Ya, boleh-boleh saja. Akan Tapi kurang efektif, karena kita menulis berulang-ulang properti dan method yang sama.

Bagaimana solusinya? Kita harus menggunakan *inheritance*.

Mari kita lihat memeber class yang sama:



Setelah menggunakan *inheritance*, maka akan menjadi seperti ini:



Class Enemy adalah class induk yang memiliki anak Zombie, Pocong, dan Burung. Apapun properti yang ada di class induk, akan dimiliki juga oleh class anak. Lalu bagaimana bentuk kodenya dalam Java?

Bentuk kodenya akan seperti ini:

File: Enemy.java

```
class Enemy {
    String name;
    int hp;
    int attackPoin;

    void attack() {
        System.out.println("Serang!");
    }
}
```

Pada class anak, kita menggunakan kata kunci `extends` untuk menyatakan kalau dia adalah class turunan dari `Enemy`.



Java Programming

File: Zombie.java

```
class Zombie extends Enemy {  
    void walk() {  
        System.out.println("Zombie jalan-jalan");  
    }  
}
```

File: Pocong.java

```
class Pocong extends Enemy {  
    void jump() {  
        System.out.println("loncat-loncat!");  
    }  
}
```

File: Burung.java

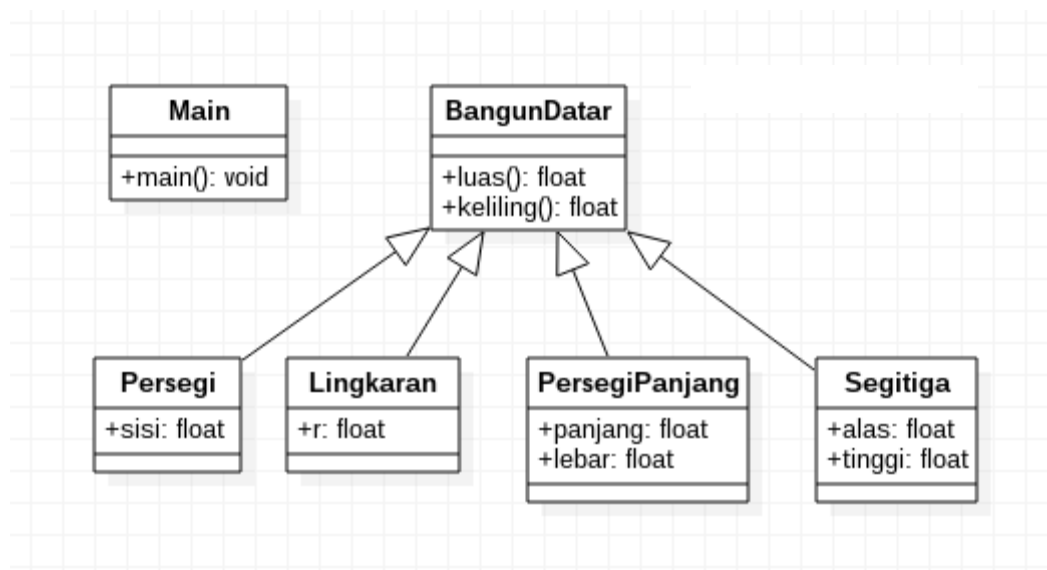
```
class Burung extends Enemy {  
    void walk() {  
        System.out.println("Burung berjalan");  
    }  
    void jump() {  
        System.out.println("Burung loncat-loncat");  
    }  
    void fly() {  
        System.out.println("Burung Terbang...");  
    }  
}
```

Lalu, bila kita ingin membuat objek dari class-class tersebut, Kita bisa membuatnya seperti ini:

```
Enemy monster = new Enemy();  
Zombie zumbi = new Zombie();  
Pocong hantuPocong = new Pocong();  
Burung garuda = new Burung();
```

Contoh program menggunakan konsep inheritance dengan IDE Netbeans

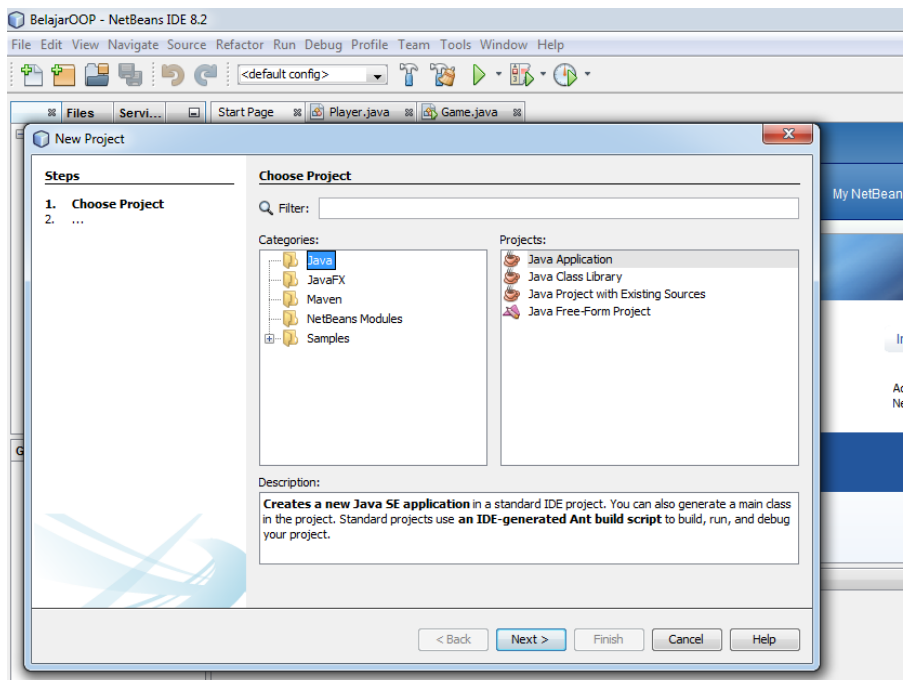
Setelah memahami konsep *inheritance*, sekarang mari kita buat contoh program sederhana. Program yang akan kita buat untuk berfungsi untuk menghitung luas dan keliling bangun datar. Bentuk class diagramnya seperti ini:



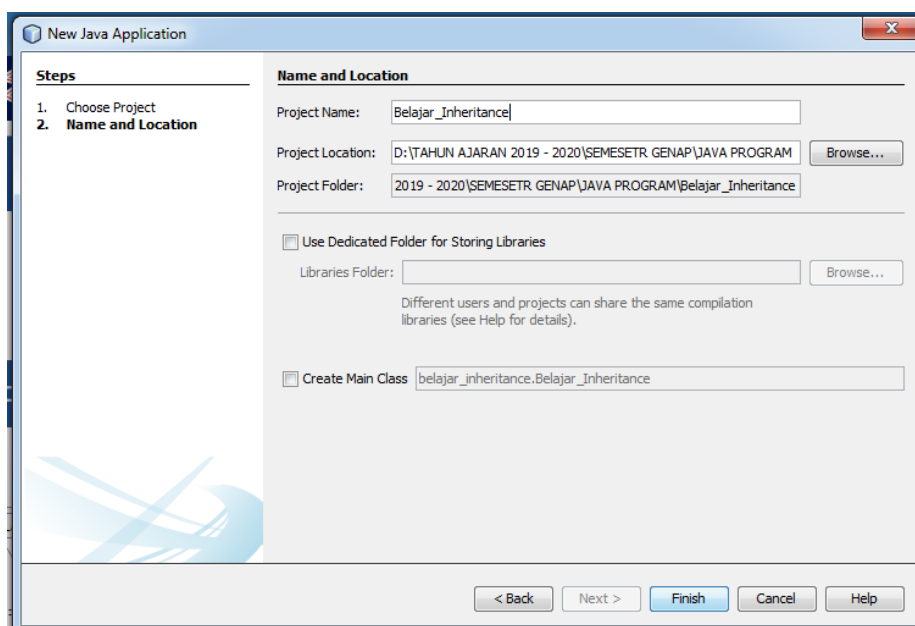


Langkah-langkah Pembuatan;

1. Pilih dan klik file → new project

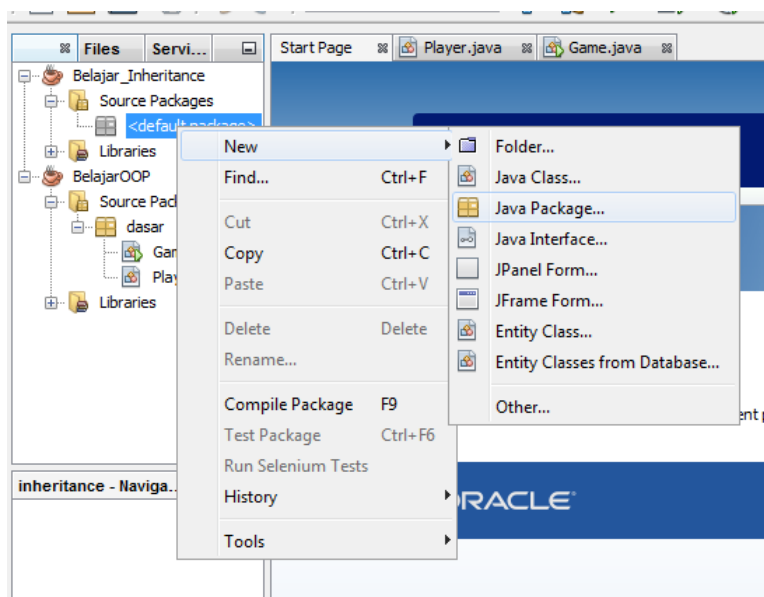


2. pilih next sehingga muncul tampilan berikut ini,



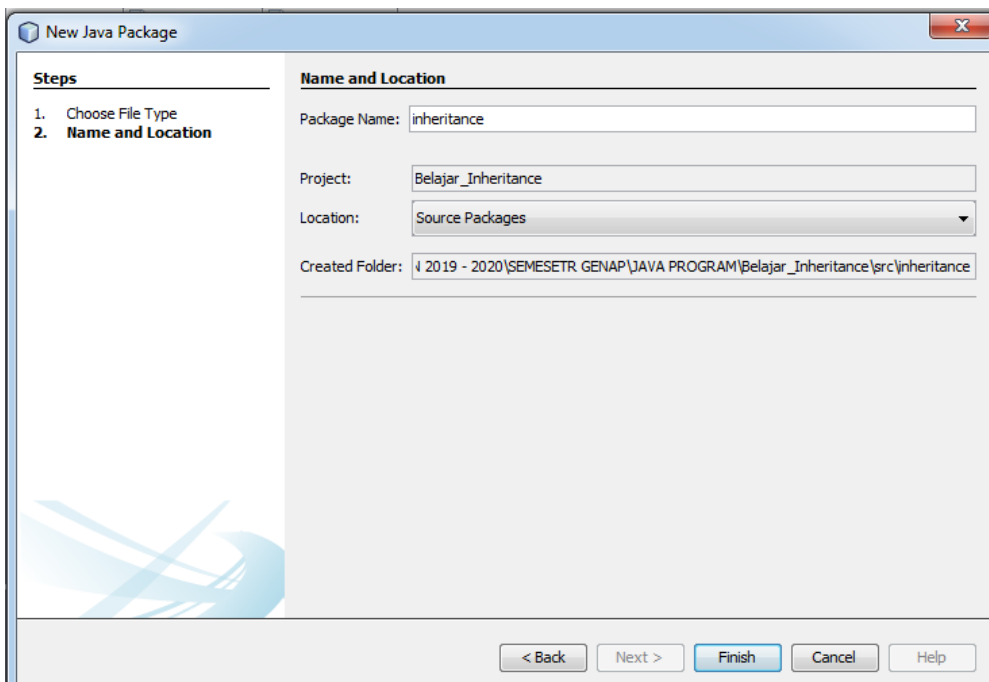
Isi project name “Belajar_Inheritance” sebagai lokasi penyimpanan file.

3. selanjutnya klik finish, sehingga muncul tampilan berikut ini.

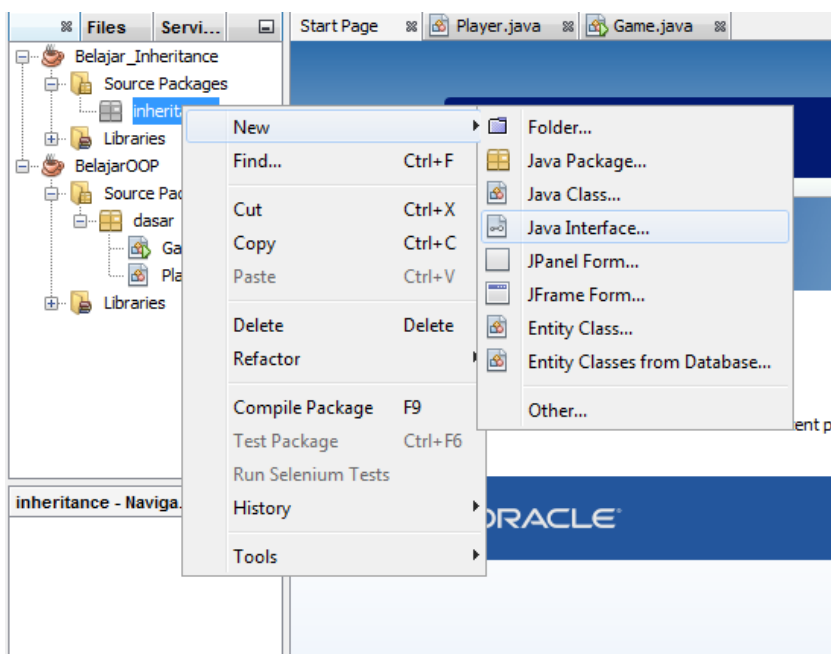


Buatlah package dengan nama “inheritance”

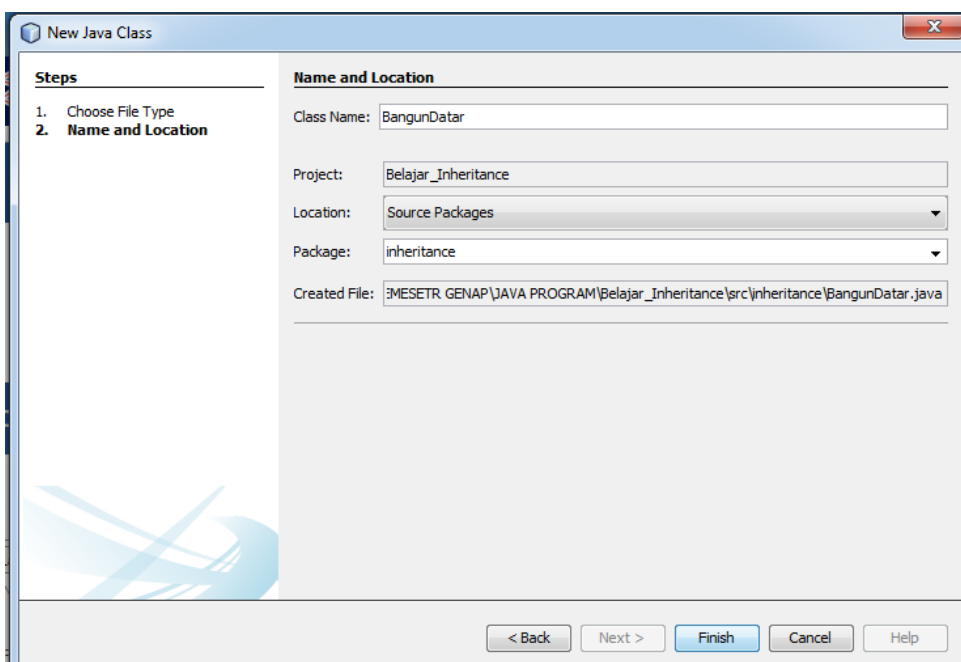
4. tampilan pembuatan package



5. kemudian buat java class



6. kemudian java class beri nama “BangunDatar ” Sebagaimana berikut,



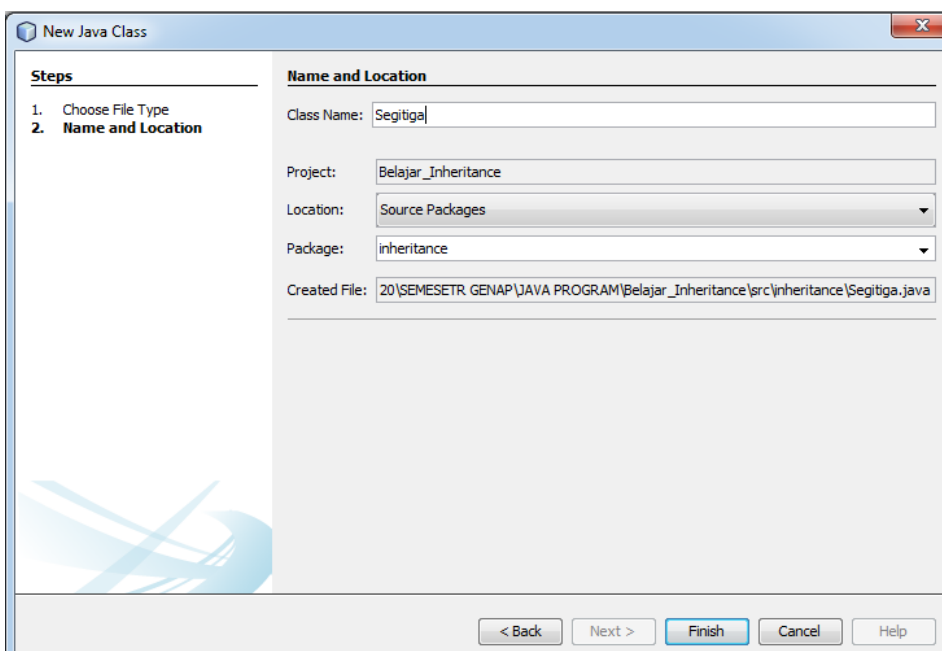


7. Buat juga java class dengan nama “Persegi”

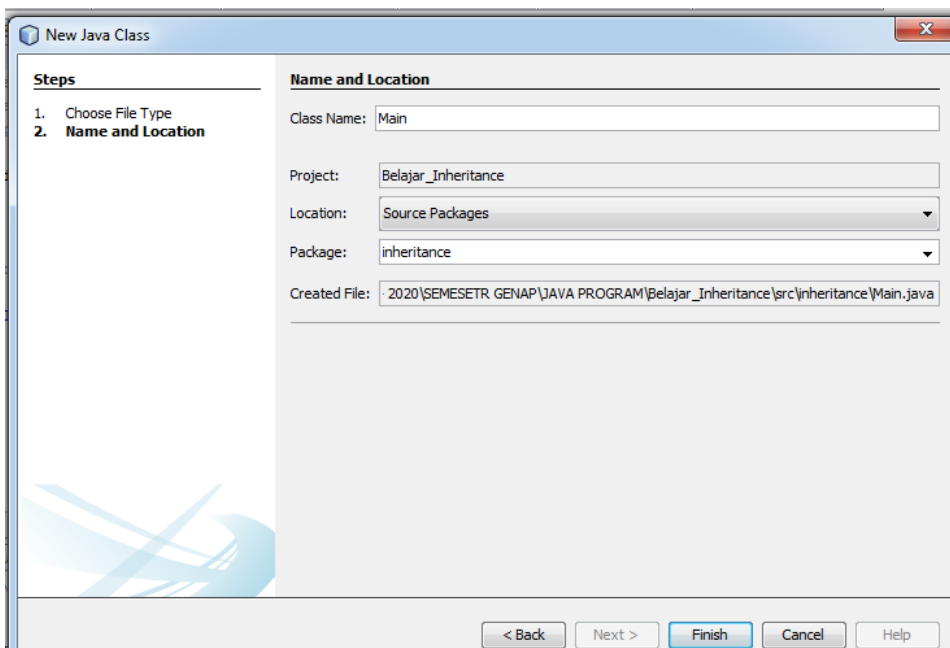
8. Buat juga java class dengan nama “Lingkaran”

9. Buat juga java class dengan nama “PersegiPanjang”

10. Buat juga java class dengan nama “Segitiga”

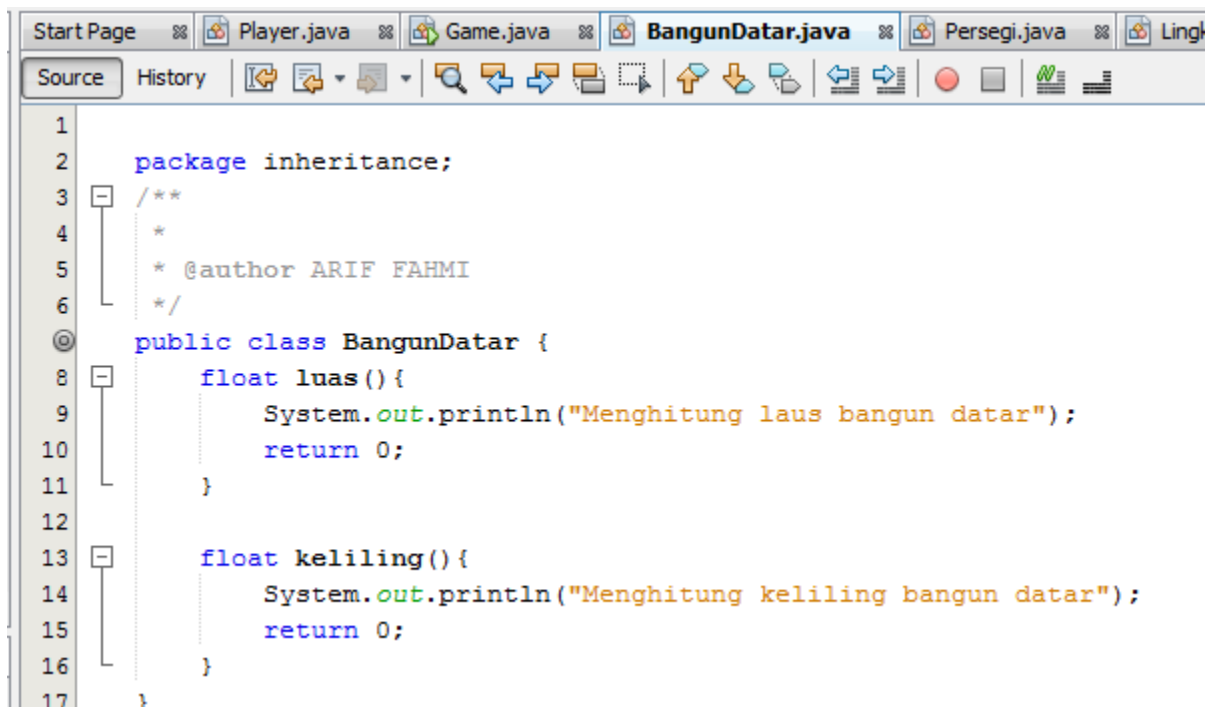


11. Buat juga java class dengan nama “Main”

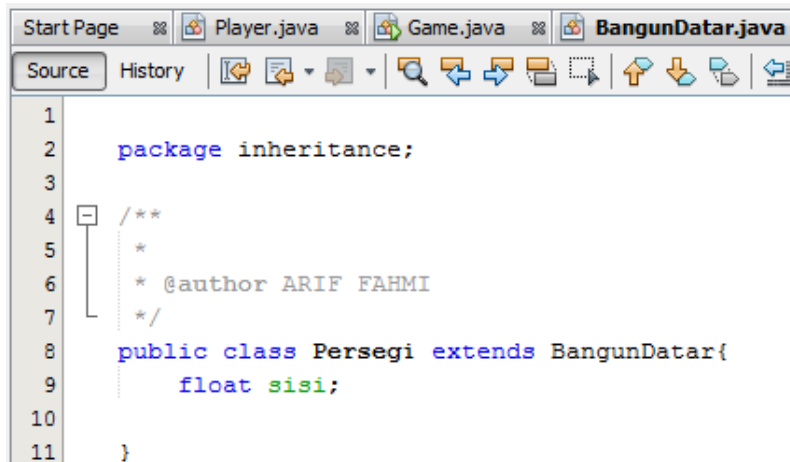


Berikut Listing program dari masing masing class

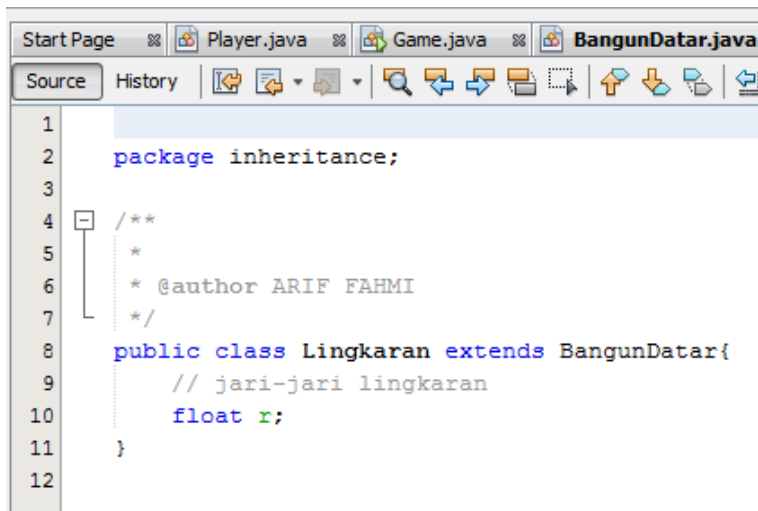
a. BangunDatar



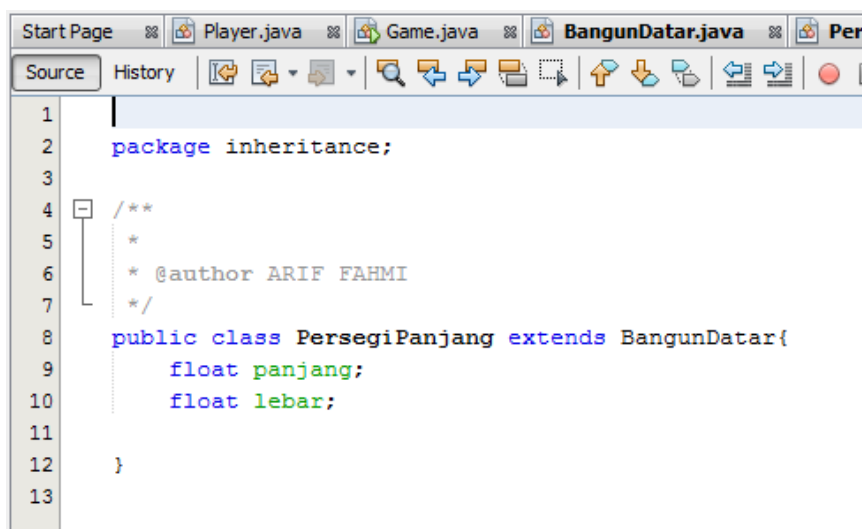
```
1
2  package inheritance;
3  /**
4   *
5   * @author ARIF FAHMI
6   */
7  public class BangunDatar {
8      float luas() {
9          System.out.println("Menghitung laus bangun datar");
10         return 0;
11     }
12
13     float keliling() {
14         System.out.println("Menghitung keliling bangun datar");
15         return 0;
16     }
17 }
```


b. Persegi

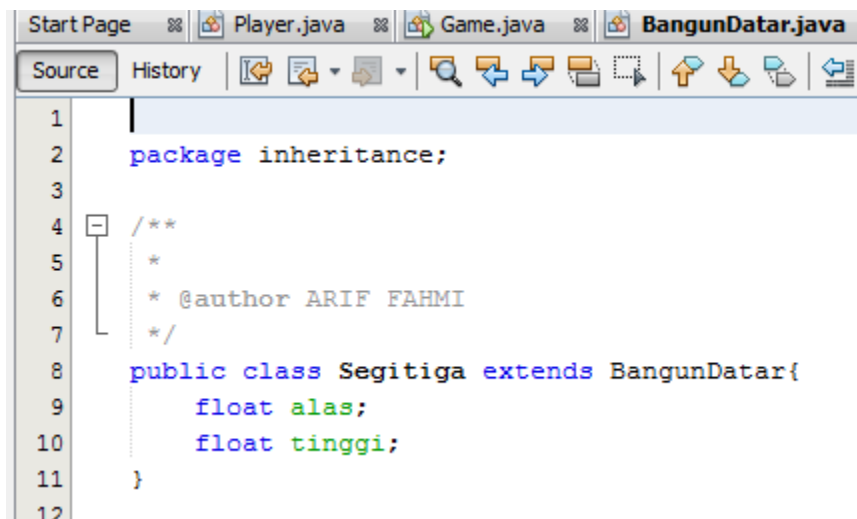
```
1
2 package inheritance;
3
4 /**
5  *
6  * @author ARIF FAHMI
7  */
8 public class Persegi extends BangunDatar{
9     float sisi;
10
11 }
```

c. Lingkaran

```
1
2 package inheritance;
3
4 /**
5  *
6  * @author ARIF FAHMI
7  */
8 public class Lingkaran extends BangunDatar{
9     // jari-jari lingkaran
10     float r;
11
12 }
```

d. PersegiPanjang

```
1
2 package inheritance;
3
4 /**
5  *
6  * @author ARIF FAHMI
7  */
8 public class PersegiPanjang extends BangunDatar{
9     float panjang;
10     float lebar;
11
12 }
13
```

e. Segitiga

```
1
2 package inheritance;
3
4 /**
5  *
6  * @author ARIF FAHMI
7  */
8 public class Segitiga extends BangunDatar{
9     float alas;
10     float tinggi;
11
12 }
```

f. Main

```
Start Page  Player.java  Game.java  BangunDatar.java  Persegi.java  Lingkaran.
Source  History  [Icons]
2  package inheritance;
3  /**
4   *
5   * @author ARIF FAHMI
6   */
7  public class Main {
8      public static void main(String[] args) {
9
10         // membuat objek bangun datar
11         BangunDatar bangunDatar = new BangunDatar();
12
13         // membuat objek persegi dan mengisi nilai properti
14         Persegi persegi = new Persegi();
15         persegi.sisi = 2;
16
17         // membuat objek Lingkaran dan mengisi nilai properti
18         Lingkaran lingkaran = new Lingkaran();
19         lingkaran.r = 22;
20
21         // membuat objek Persegi Panjang dan mengisi nilai properti
22         PersegiPanjang persegiPanjang = new PersegiPanjang();
23         persegiPanjang.panjang = 8;
24         persegiPanjang.lebar = 4;
25
26         // membuat objek Segitiga dan mengisi nilai properti
27         Segitiga mSegitiga = new Segitiga();
28         mSegitiga.alas = 12;
29         mSegitiga.tinggi = 8;
30
31
32         // memanggil method luas dan keliling
33         bangunDatar.luas();
34         bangunDatar.keliling();
35
36         persegi.luas();
37         persegi.keliling();
38
39         lingkaran.luas();
40         lingkaran.keliling();
41
42         persegiPanjang.luas();
43         persegiPanjang.keliling();
44
45         mSegitiga.luas();
46         mSegitiga.keliling();
47     }
48
49 }
```

Setelah itu, coba jalankan class Main, maka hasilnya:

```
>
Output - Belajar_Inheritance (run)
run:
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
Menghitung laus bangun datar
Menghitung keliling bangun datar
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kenapa hasilnya bisa begitu? Karena yang kita panggil sebenarnya adalah method `luas()` dan `keliling()` milik si induk (BangunDatar). Objek anak dari BangunDatar belum memiliki method `luas()` dan `keliling()`, akhirnya mengambil milik induknya.

Lalu bagaimana kalau kita ingin membuat agar semua class anak memiliki method `luas()` dan `keliling()` yang berbeda dari induk? Jawabanya: menggunakan *method overriding*. Tapi sebelum itu kita akan membahas mengenai konsep method secara terperinci.

2. Method

Apakah Method itu dan mengapa menggunakan Method?

Method dalam Java mirip dengan fungsi atau procedure dalam bahasa pemrograman yang lain. Pada konsep Object Oriented Programming, **Function lebih dikenal dengan istilah Method** dimana merupakan suatu bagian dari Object yang mendefinisikan apa yang bisa Object tersebut lakukan.

Jadi : Method adalah fungsi atau prosedur yang dibuat oleh seorang programmer didalam suatu Class. Method dapat dibagi menjadi fungsi dan prosedur. Fungsi adalah bagian atau sub dari program yang mempunyai algoritma tertentu dalam menyelesaikan suatu masalah dengan mengembalikan hasil.

Adapun method memiliki karakteristik sebagai berikut,

- Dapat mengembalikan satu nilai atau tidak sama sekali
- Dapat diterima beberapa parameter yang dibutuhkan atau tidak ada parameter sama sekali. Parameter bisa juga disebut sebagai argumen dari fungsi
- Setelah method telah selesai dieksekusi, dia akan kembali pada method yang memanggilnya.

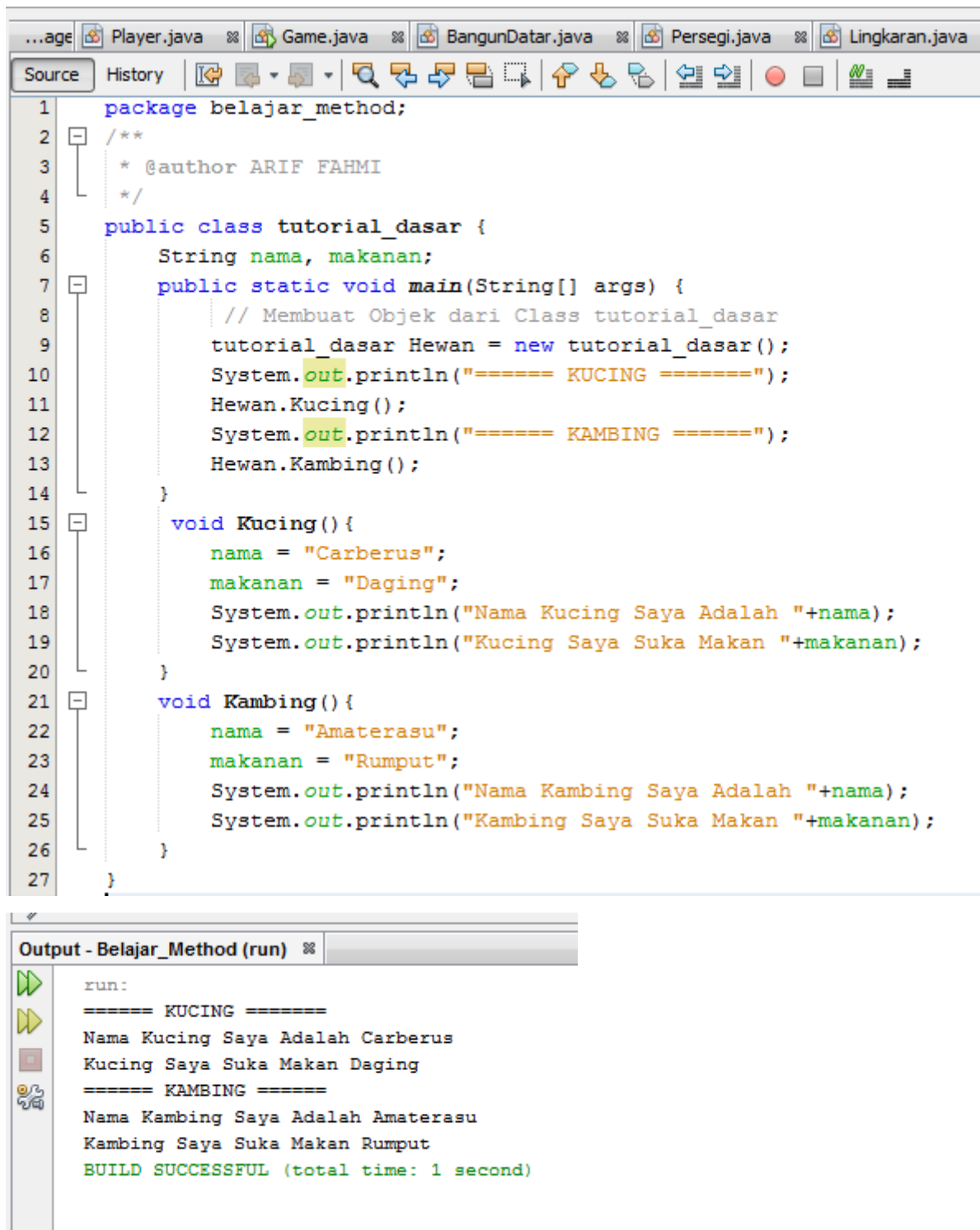
Terdapat beberapa jenis method pada bahasa pemrograman Java, diantaranya Method `main`, Method `void`, `return` (non-void) dan `static`, ketiga method tersebut mempunyai fungsi yang sama, hanya saja cara penulisannya saja yang berbeda.

A. Main Method

Main method merupakan method utama yang di jalankan pada saat aplikasi di run. Setiap class dalam sebuah aplikasi boeh memiliki main method tetapi hanya ada satu main method yang di run pada saat eksekusi program.

B. Void Method

Void adalah method yang tidak memiliki nilai kembali/return, bisanya digunakan tidak untuk mencari nilai dalam suatu operasi, untuk mendeklarasikannya kita harus menambahkan kata kunci *void*. Agar method tersebut dapat berjalan, kita perlu memanggilnya pada method *main*, kita harus membuat objek dari class yang kita gunakan terlebih dahulu, lalu panggil pada method *main*.

Contoh penerapan Void Method,

```
1 package belajar_method;
2 /**
3  * @author ARIF FAHMI
4  */
5 public class tutorial_dasar {
6     String nama, makanan;
7     public static void main(String[] args) {
8         // Membuat Objek dari Class tutorial_dasar
9         tutorial_dasar Hewan = new tutorial_dasar();
10        System.out.println("===== KUCING =====");
11        Hewan.Kucing();
12        System.out.println("===== KAMBING =====");
13        Hewan.Kambing();
14    }
15    void Kucing() {
16        nama = "Carberus";
17        makanan = "Daging";
18        System.out.println("Nama Kucing Saya Adalah "+nama);
19        System.out.println("Kucing Saya Suka Makan "+makanan);
20    }
21    void Kambing() {
22        nama = "Amaterasu";
23        makanan = "Rumput";
24        System.out.println("Nama Kambing Saya Adalah "+nama);
25        System.out.println("Kambing Saya Suka Makan "+makanan);
26    }
27 }
```

Output - Belajar_Method (run) %

```
run:
===== KUCING =====
Nama Kucing Saya Adalah Carberus
Kucing Saya Suka Makan Daging
===== KAMBING =====
Nama Kambing Saya Adalah Amaterasu
Kambing Saya Suka Makan Rumput
BUILD SUCCESSFUL (total time: 1 second)
```

Disana terdapat 2 buah method yang berbeda, pada masing-masing method mempunyai atribut yang berbeda pula, jadi method tersebut digunakan untuk mengkategorikan statement atau algoritma yang kita buat lalu memanggilnya pada method main menggunakan objek dari class.

C. Non-Void (Return) Method

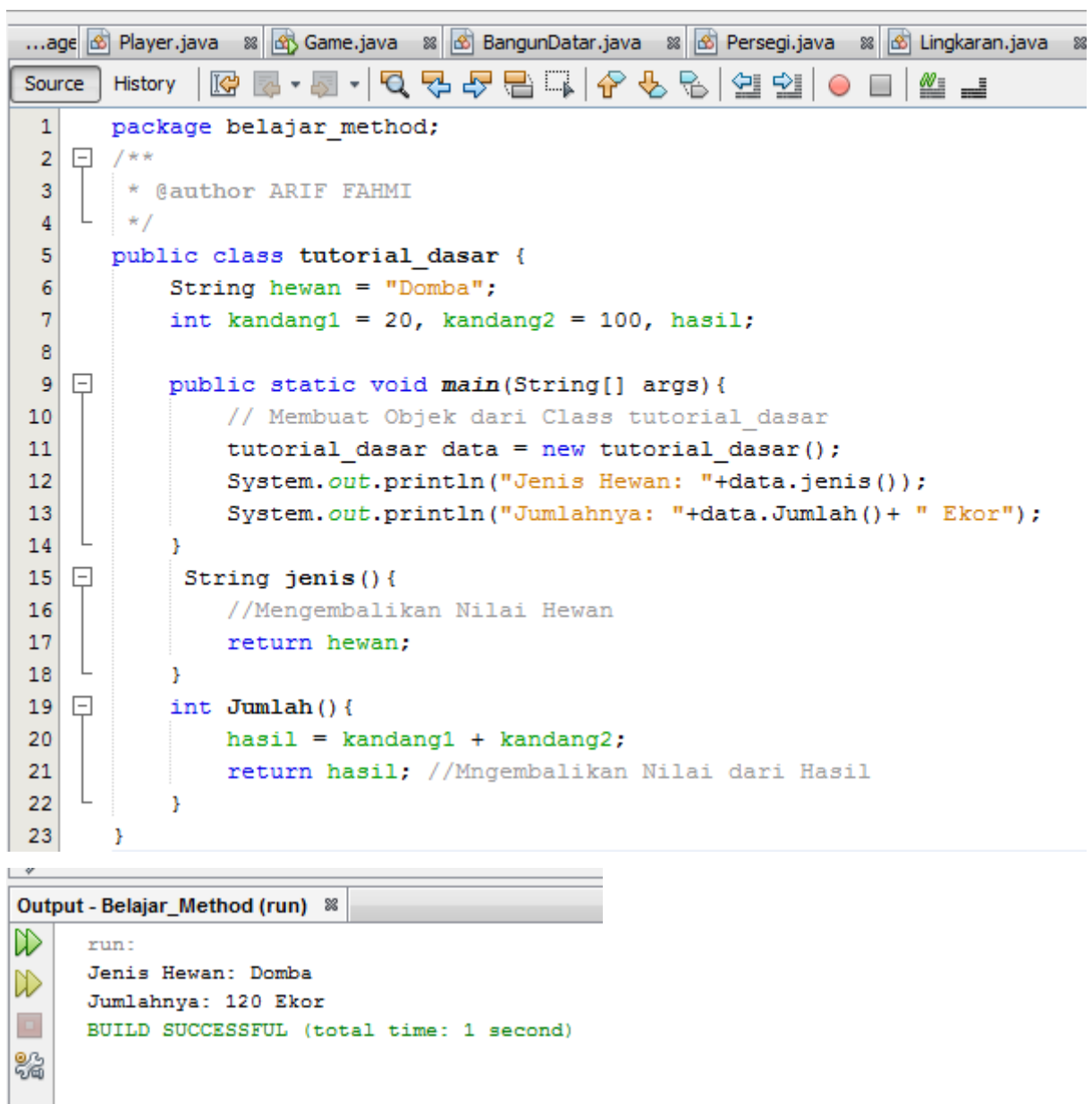
Return adalah method yang mengembalikan nilai secara langsung atau sebuah nilai dari variable. cara penulisan return method seperti berikut ini:

```
//TipeData //NamaMethod(){  
    return //Nilai yang ingin dikembalikan;  
}
```

Catatan ;

Tipe data pada method return harus sama dengan nilai yang ingin dikembalikan

Contoh penerapan Return Method,



```
1 package belajar_method;  
2 /**  
3  * @author ARIF FAHMI  
4  */  
5 public class tutorial_dasar {  
6     String hewan = "Domba";  
7     int kandang1 = 20, kandang2 = 100, hasil;  
8  
9     public static void main(String[] args){  
10         // Membuat Objek dari Class tutorial_dasar  
11         tutorial_dasar data = new tutorial_dasar();  
12         System.out.println("Jenis Hewan: "+data.jenis());  
13         System.out.println("Jumlahnya: "+data.Jumlah()+ " Ekor");  
14     }  
15     String jenis(){  
16         //Mengembalikan Nilai Hewan  
17         return hewan;  
18     }  
19     int Jumlah(){  
20         hasil = kandang1 + kandang2;  
21         return hasil; //Mngembalikan Nilai dari Hasil  
22     }  
23 }
```

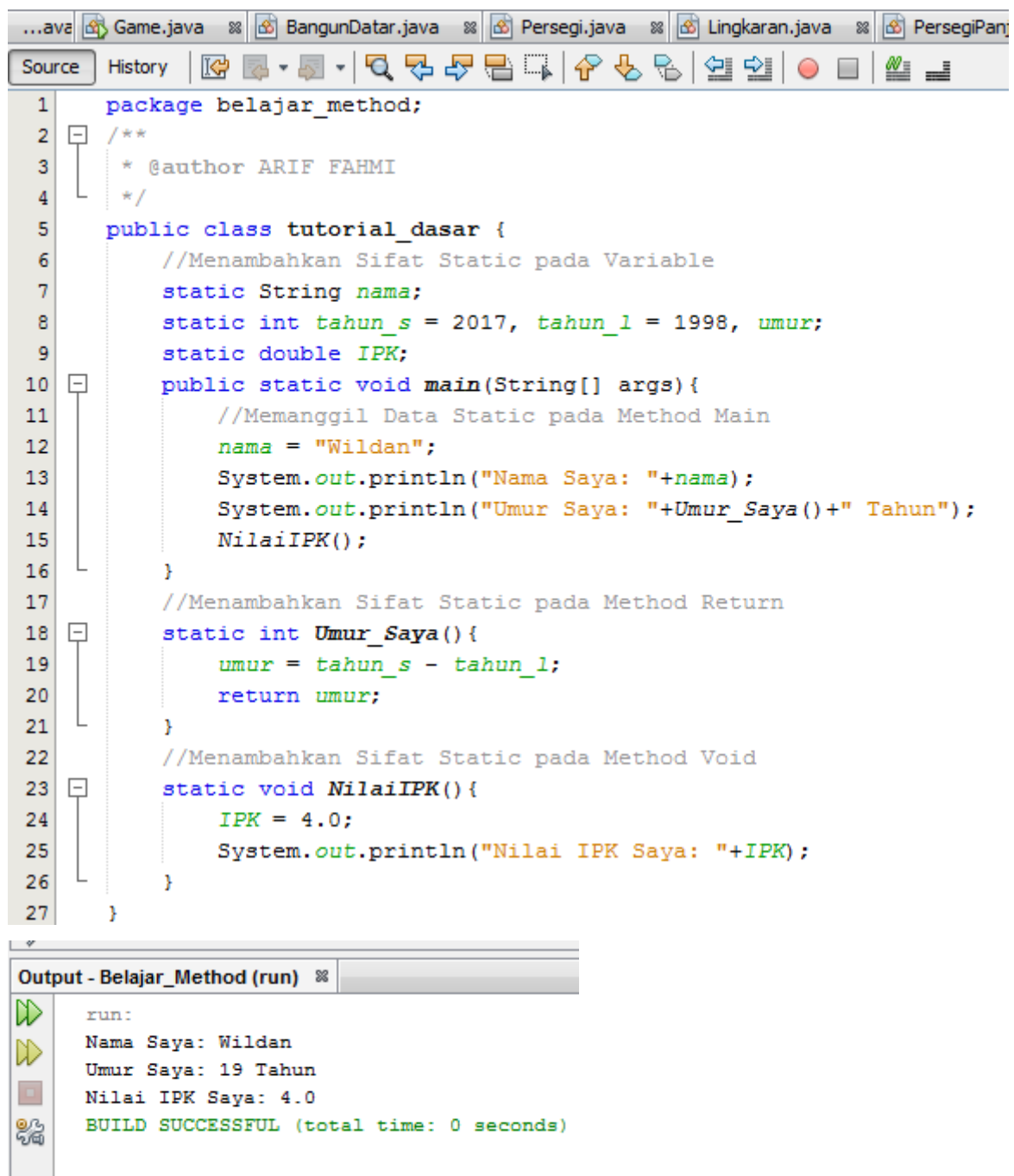
Output - Belajar_Method (run) %

```
run:  
Jenis Hewan: Domba  
Jumlahnya: 120 Ekor  
BUILD SUCCESSFUL (total time: 1 second)
```

D. Static Method

Static merupakan suatu sifat yang bisa kita gunakan pada variable atau method, jika kita menggunakan static pada sebuah variabel ataupun method, untuk memanggilnya kita tidak perlu menginisialisasi suatu class maksudnya kita tidak perlu membuat sebuah objek dari class, berbeda dengan jenis method sebelumnya, pada jenis method sebelumnya seperti Void dan Return kita harus membuat objek dari class terlebih dahulu untuk memanggil method tersebut.

Contoh penerapan Static Method,

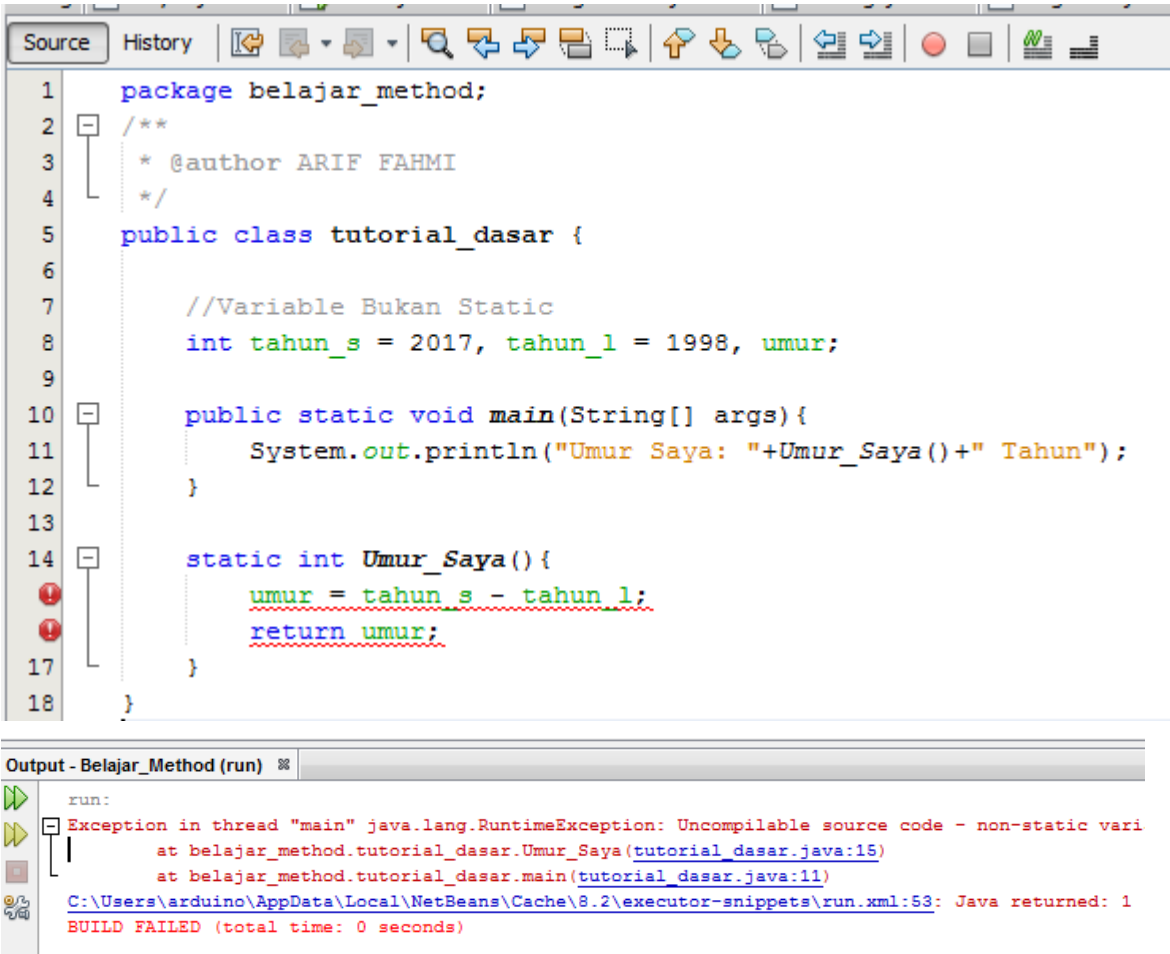


```
1 package belajar_method;
2 /**
3  * @author ARIF FAHMI
4  */
5 public class tutorial_dasar {
6     //Menambahkan Sifat Static pada Variable
7     static String nama;
8     static int tahun_s = 2017, tahun_l = 1998, umur;
9     static double IPK;
10    public static void main(String[] args) {
11        //Memanggil Data Static pada Method Main
12        nama = "Wildan";
13        System.out.println("Nama Saya: "+nama);
14        System.out.println("Umur Saya: "+Umur_Saya()+" Tahun");
15        NilaiIPK();
16    }
17    //Menambahkan Sifat Static pada Method Return
18    static int Umur_Saya() {
19        umur = tahun_s - tahun_l;
20        return umur;
21    }
22    //Menambahkan Sifat Static pada Method Void
23    static void NilaiIPK() {
24        IPK = 4.0;
25        System.out.println("Nilai IPK Saya: "+IPK);
26    }
27 }
```

Output - Belajar_Method (run)

```
run:
Nama Saya: Wildan
Umur Saya: 19 Tahun
Nilai IPK Saya: 4.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Perlu kalian ketahui, jika data yang kalian buat diubah menjadi static maka kalian harus memanggilnya pada method static juga, jika kalian memanggilnya seperti ini maka akan terjadi error.



```
1 package belajar_method;
2 /**
3  * @author ARIF FAHMI
4  */
5 public class tutorial_dasar {
6
7     //Variable Bukan Static
8     int tahun_s = 2017, tahun_l = 1998, umur;
9
10    public static void main(String[] args){
11        System.out.println("Umur Saya: "+Umur_Saya()+" Tahun");
12    }
13
14    static int Umur_Saya() {
15        umur = tahun_s - tahun_l;
16        return umur;
17    }
18 }
```

Output - Belajar_Method (run) %

run:

Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - non-static variable umur cannot be accessed from a static context

at belajar_method.tutorial_dasar.Umur_Saya(tutorial_dasar.java:15)

at belajar_method.tutorial_dasar.main(tutorial_dasar.java:11)

C:\Users\arduino\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1

BUILD FAILED (total time: 0 seconds)

Jika kalian jalankan, akan terjadi error, karena variable yang bukan static tidak bisa dipanggil pada method yang mempunyai sifat static, jadi keduanya harus mempunyai sifat static.

2.1 Modifier Method

Selanjutnya kitaakan membahas tentang modifier pada method di pemrograman Java. Secara umum ada 3 macam modifier yang digunakan dalam Java: public, private, dan protected.

Berikut ini tabel jangkauan untuk masing-masing modifier:

| Modifier | Class | Package | Subclass | World |
|-------------|-------|---------|----------|-------|
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| no modifier | Y | Y | N | N |
| private | Y | N | N | N |

Keterangan:

- Y artinya bisa diakses;
- N artinya tidak bisa diakses;
- Subclass artinya class anak;
- World artinya seluruh package di aplikasi.

A. Public

Access modifier public mempunyai hak akses paling luas dibanding yang lainnya. Karena aksesnya sangat luas, maka access modifier ini biasanya digunakan untuk method setter getter sesuai konsep OOP.

```
package modifier;

class Person {
    public String name;

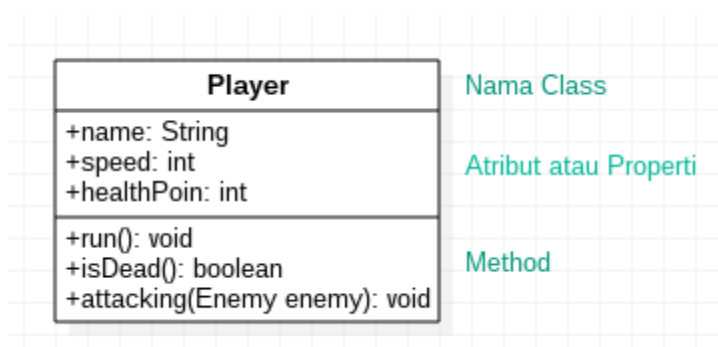
    public changeName(String newName) {
        this.name = newName;
    }
}
```

Pada class Preson terdapat dua member, yaitu:

1. atribut name
2. method changeName()

Kedua member tersebut kita berikan modifier public. Artinya mereka akan bisa diakses dari mana saja.

Pada class diagram, modifier public digambarkan dengan simbol plus (+).



B. Protected

Access modifier protected biasanya digunakan untuk mewariskan variabel yang ada di super class terhadap child class.

Modifier protected akan membuat member dan class hanya bisa diakses dari:

1. Class itu sendiri;
2. Sub class atau class anak;
3. Package (class yang berada satu package dengannya).

Modifier protected juga hanya boleh digunakan pada member saja.

```
package modifier;

public class Person {
    protected String name;

    public void setName(String name) {
        this.name = name;
    }

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

Pada contoh di atas, kita memberikan modifier protected pada atribut name.

Apabila kita coba mengakses dari class yang satu package dengannya, maka tidak akan terjadi error.

Namun, apabila kita mencoba mengakses dari luar package seperti ini:

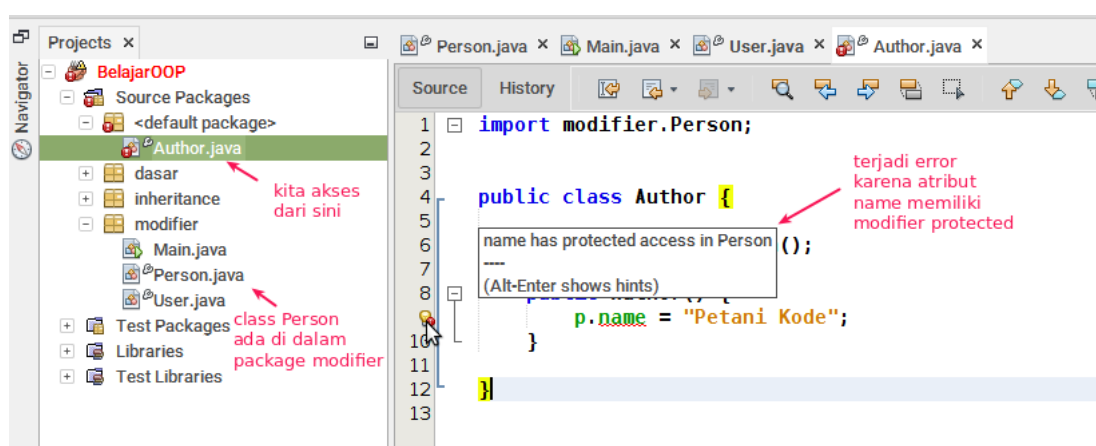
```
import modifier.Person;

public class Author {

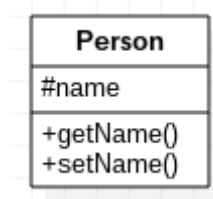
    Person p = new Person();

    public Author() {
        // akan terjadi error di sini karena atribut name
        // telah diberikan modifier protected
        p.name = "Petani Kode";
    }
}
```

Maka akan terjadi error.



Pada class diagram (di StarUML), modifier protected digambarkan dengan tanda pagar (#).



C. No Access Modifier

Sesuai namanya, hak akses yang satu ini tidak perlu dituliskan di method/variabelnya. Dengan hak akses ini, variabel/method dapat diakses dari class lain asalkan masih dalam satu package yang sama.

```
public class Kendaraan {
    int jumlahRoda;
    String warna;
}
```

D. Private

Access modifier private bersifat tertutup. Sesuai dengan konsep OOP Encapsulation, maka setiap variabel wajib untuk dilindungi hak aksesnya secara langsung dari luar. Oleh karena itu, variabel diberikan hak akses private dan untuk melakukan pengaksesan/perubahan data digunakan setter getter.

Modifier **private** akan membuat member hanya bisa diakses oleh dari dalam class itu sendiri.

Perlu diingat:

Modifier **private** tidak bisa diberikan kepada class, enum, dan interface. Modifier private hanya bisa diberikan kepada member class.

```
class Person {  
    private String name;  
  
    public void setName(name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
}
```

Pada contoh di atas, kita memberikan modifier private pada atribut name dan modifier public pada method setName() dan getName().

Apabila kita coba mengakses langsung atribut name seperti ini:

```
Person mPerson = new Person()  
mPerson.name = "Petani Kode"; //  
<- maka akan terjadi error di  
sini
```



Lalu, bagaimana cara mengakses member private dari luar class?

Kita bisa memanfaatkan [method setter dan getter](#). Karena, method ini akan selalu diberikan modifier public.

```
Person mPerson = new Person();  
mPerson.setName("Petani Kode");  
  
System.out.println("Person Name: " + mPerson.getName());
```

Pada class diagram, modifier private digambarkan dengan simbol minus (-).

