

PEMROGRAMAN BERBASIS OBJEK 2

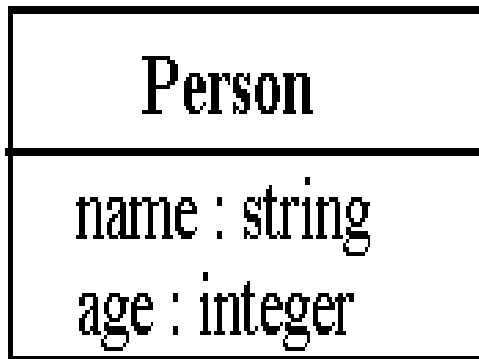
KONSEP PBO

Class dan Object

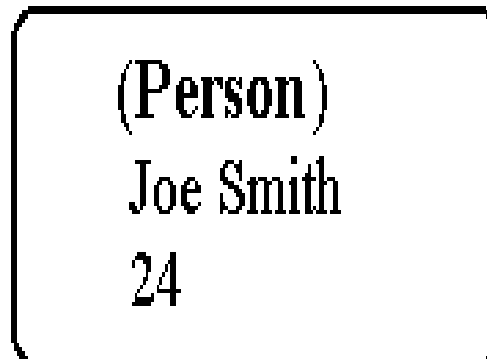
- Class: **konsep** dan **deskripsi** dari sesuatu
 - Class mendeklarasikan **method** yang dapat digunakan (dipanggil) oleh object
- Object: **instance dari class**, bentuk (contoh) nyata dari class
 - Object memiliki sifat **independen** dan dapat digunakan untuk memanggil method
- Contoh Class dan Object:
 - Class: **mobil**
 - Object: **mobilnya pak Joko, mobilku, mobil berwarna merah**

Class dan Object

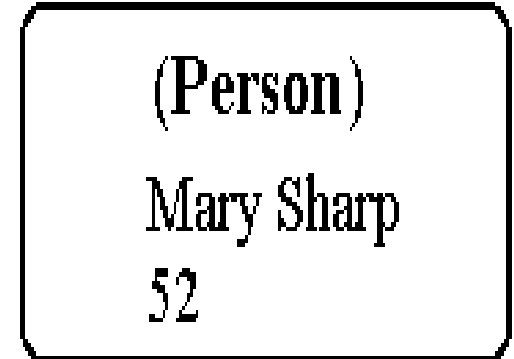
- Class seperti **cetakan kue**, dimana kue yg dihasilkan dari cetakan kue itu adalah **object**
- Warna kue bisa bermacam-macam meskipun berasal dari cetakan yang sama (**object memiliki sifat independen**)



Class with Attributes

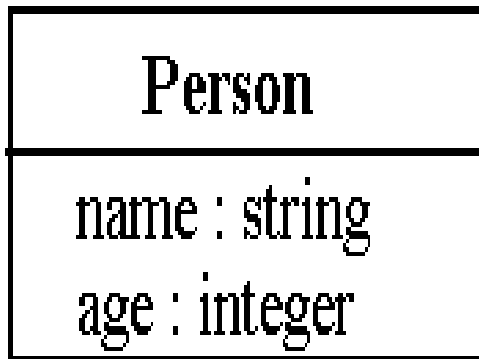


Objects with Values

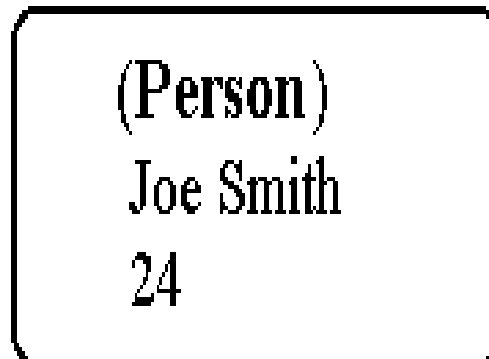


Attribute

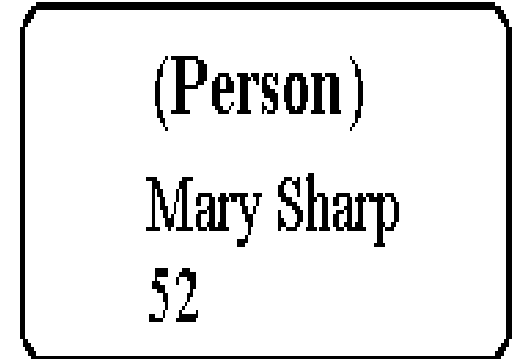
- **Variable** yang mengitari class, dengan **nilai datanya bisa ditentukan di object**
- Variable digunakan untuk **menyimpan nilai** yang nantinya akan digunakan pada program
- Variable memiliki **jenis (tipe), nama** dan **nilai**
- Name, age, dan weight adalah atribute (variabel) dari class Person



Class with Attributes



Objects with Values



Class = Method + Variable

Class Sepeda

gir

kecepatan

tampilkan kecepatan

ubah gir

variable

method

Object = Method + Variable yg Memiliki Nilai

Object Sepedaku

gir = 3

kecepatan = 10km/jam

tampilkan kecepatan ()
kecepatan = 10 km/jam

ubah gir (2)
gir = 5

**instance
variable**

**instance
method**

Membuat Class, Object dan Memanggil Atribut

```
public class Mobil {  
    String warna;  
    int tahunProduksi;  
}
```

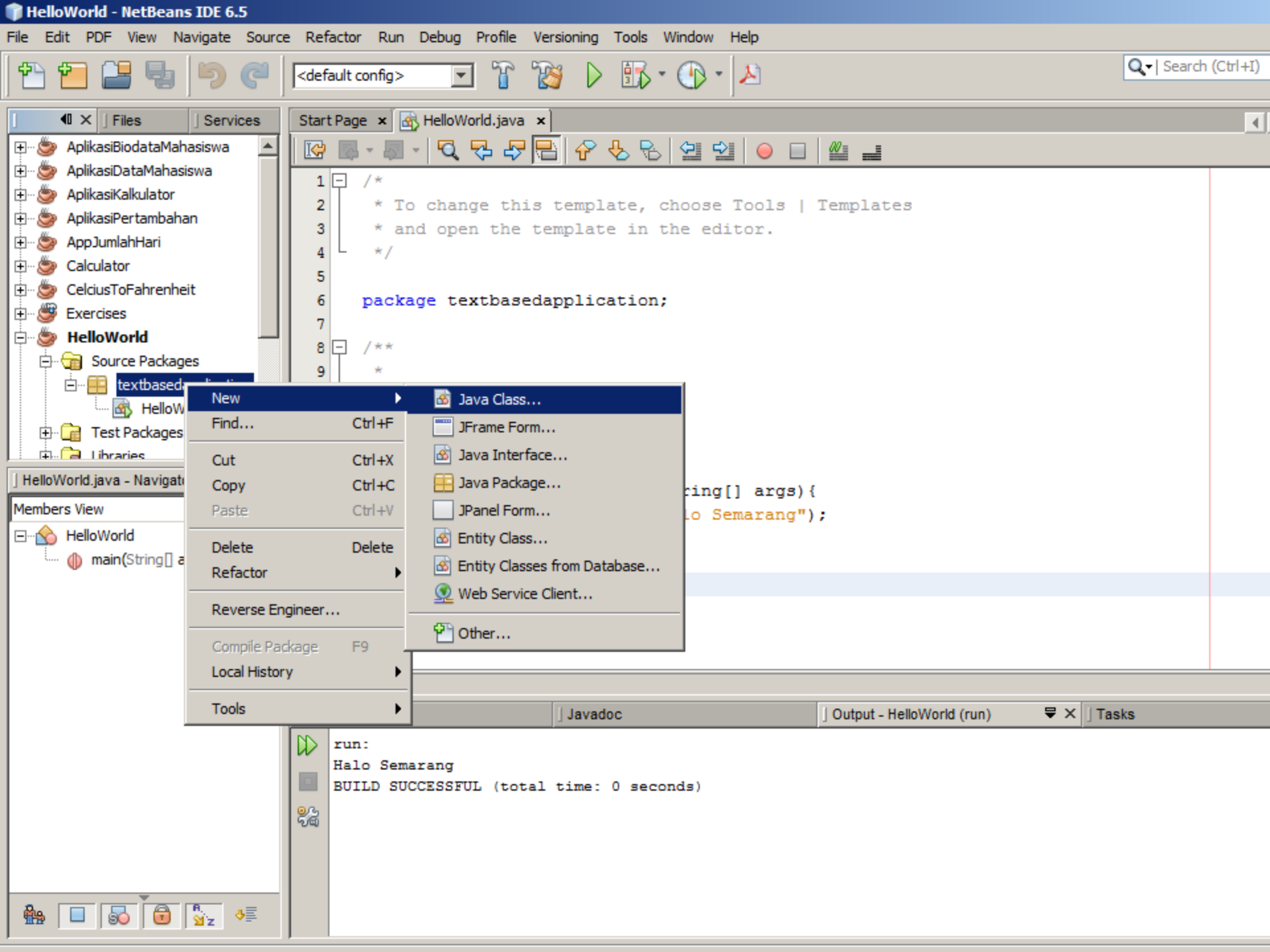
Mobil.java

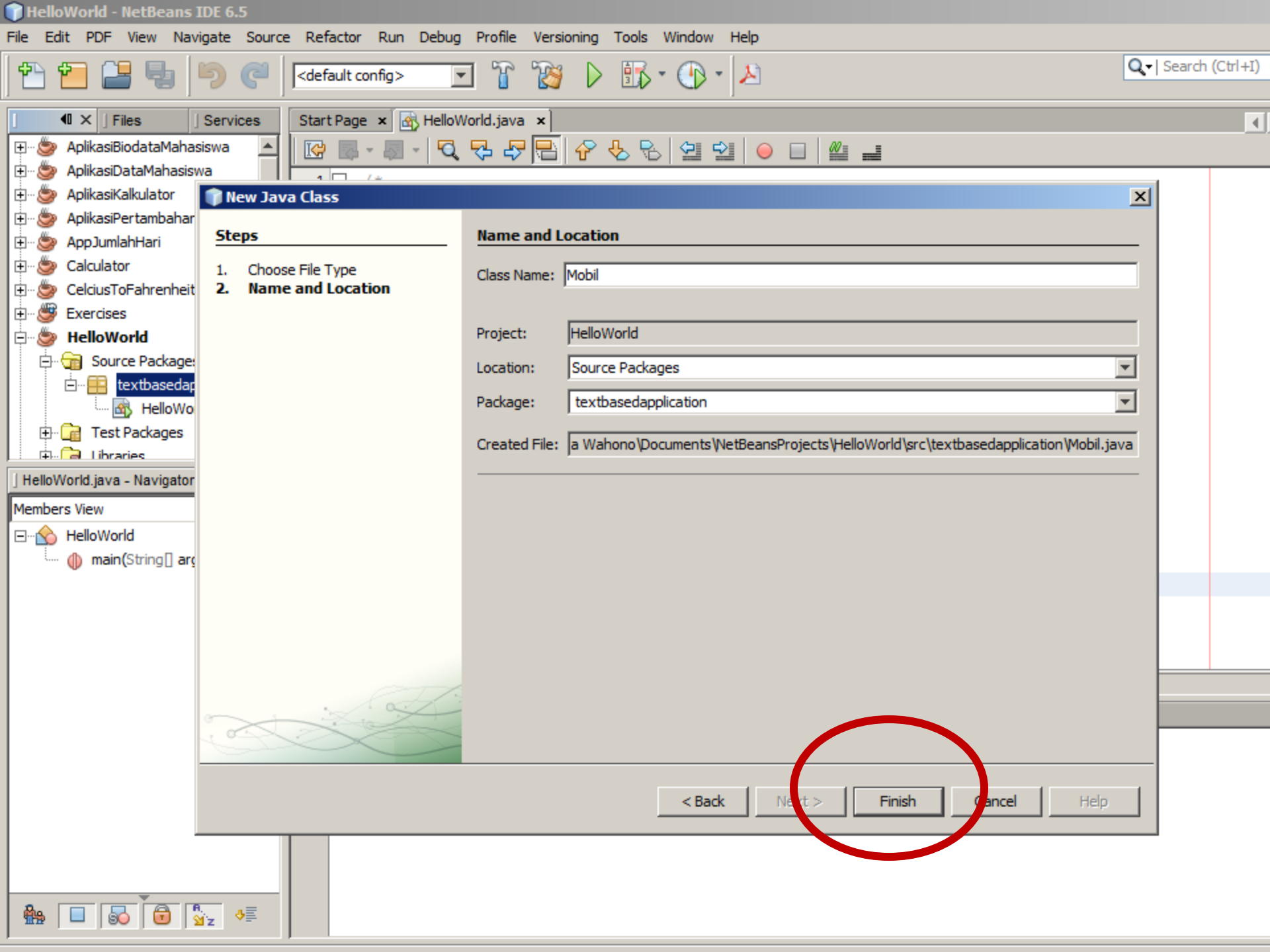
```
public class MobilBeraksi{  
    public static void main(String[] args){  
        // Membuat object  
        Mobil mobilku = new Mobil();  
  
        /* memanggil atribut dan memberi nilai */  
        mobilku.warna = "Hitam";  
        mobilku.tahunProduksi = 2006;  
        System.out.println("Warna: " + mobilku.warna);  
        System.out.println("Tahun: " + mobilku.tahunProduksi);  
    }  
}
```

MobilBeraksi.java

Latihan: Membuat Program dg Netbeans

1. Buka Netbeans IDE
2. Ikuti langkah berikut





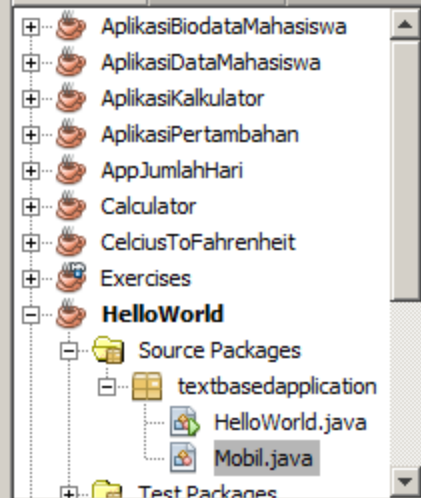


<default config>



Search (Ctrl+I)

Files Services



Navigator

Members View

Mobil

Start Page x HelloWorld.java x Mobil.java x



```
1  /*
2      * To change this template, choose Tools | Templates
3      * and open the template in the editor.
4      */
5
6  package textbasedapplication;
7
8  /**
9      *
10     * @author Romi Satria Mahono
11     */
12     public class Mobil {
13
14     }
15
```

1:1 INS

Usages

Javadoc

Output - HelloWorld (run)

Tasks

```
run:
Halo Semarang
BUILD SUCCESSFUL (total time: 0 seconds)
```



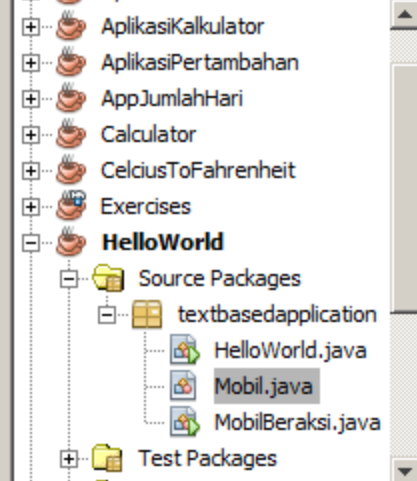


<default config>



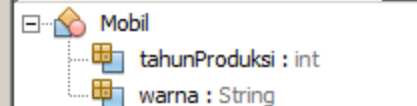
Search (Ctrl+I)

Files Services



Mobil.java - Navigator

Members View



Start Page x HelloWorld.java x Mobil.java x MobilBeraksi.java x



```
1  /*
2      * To change this template, choose Tools | Templates
3      * and open the template in the editor.
4      */
5
6  package textbasedapplication;
7
8  /**
9      *
10     * @author Romi Satria Wahono
11     */
12     public class Mobil {
13         String warna;
14         int tahunProduksi;
15     }
16
```

16:1 INS

Usages

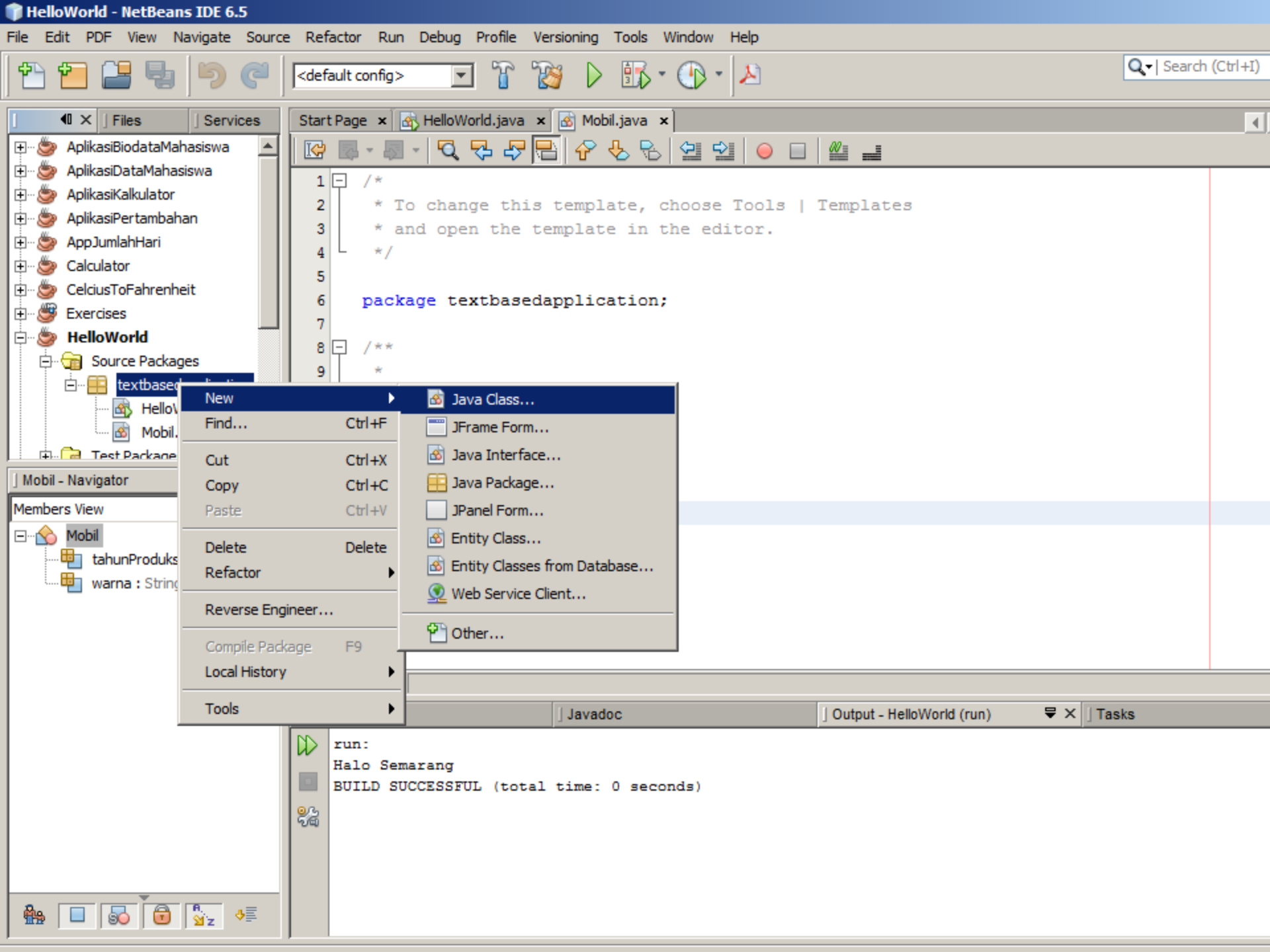
Javadoc

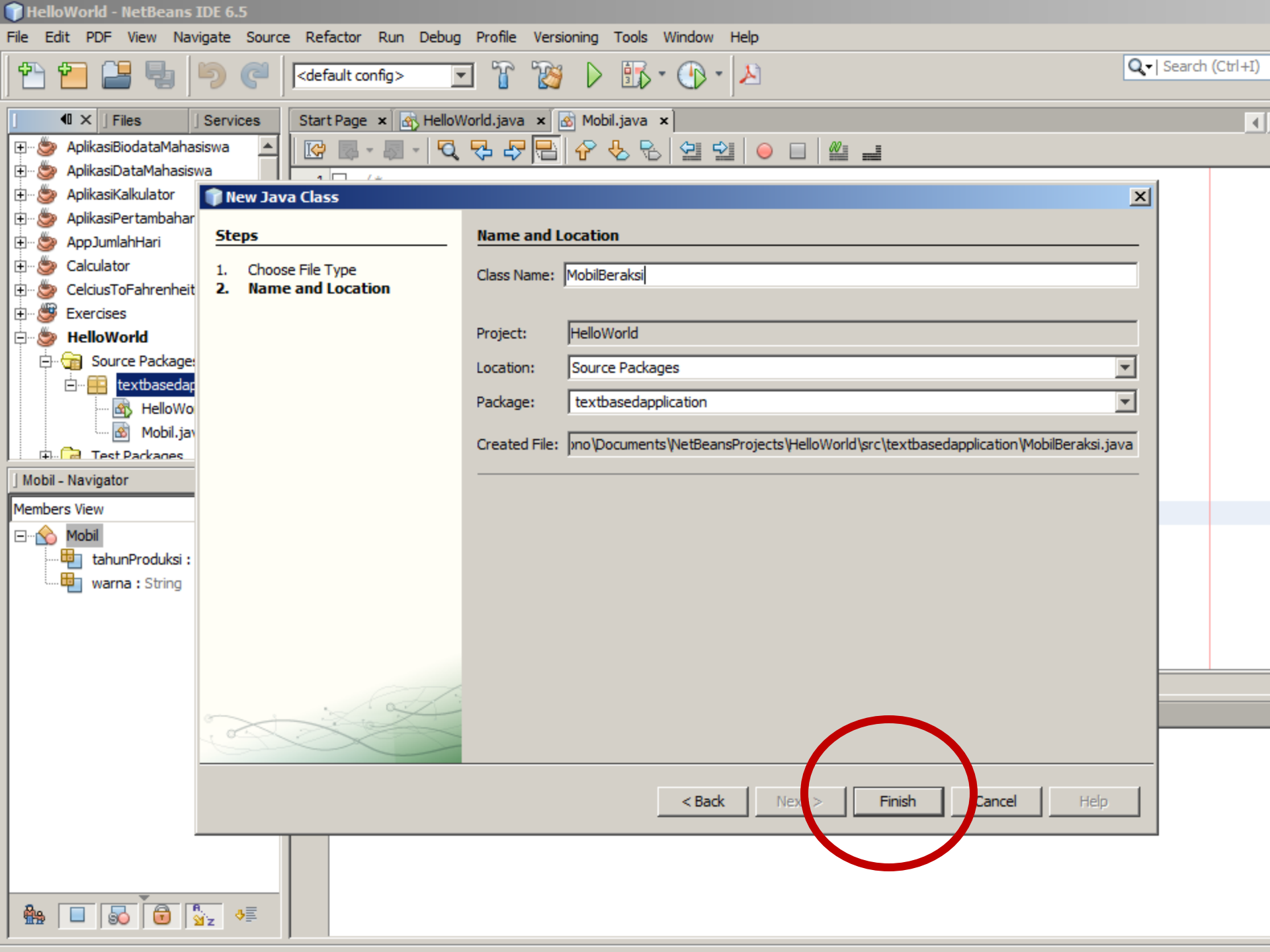
Output - HelloWorld (run)

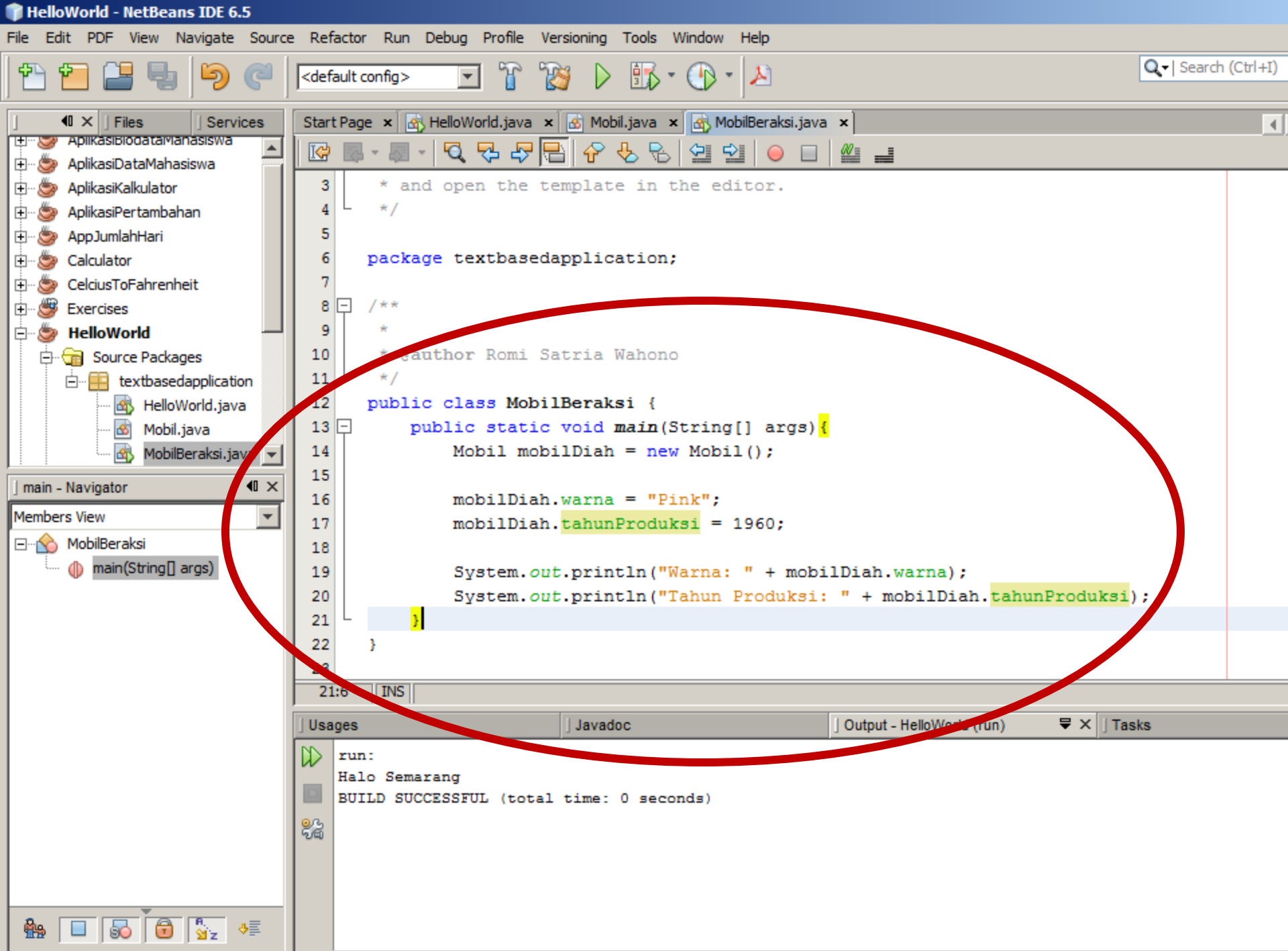
Tasks

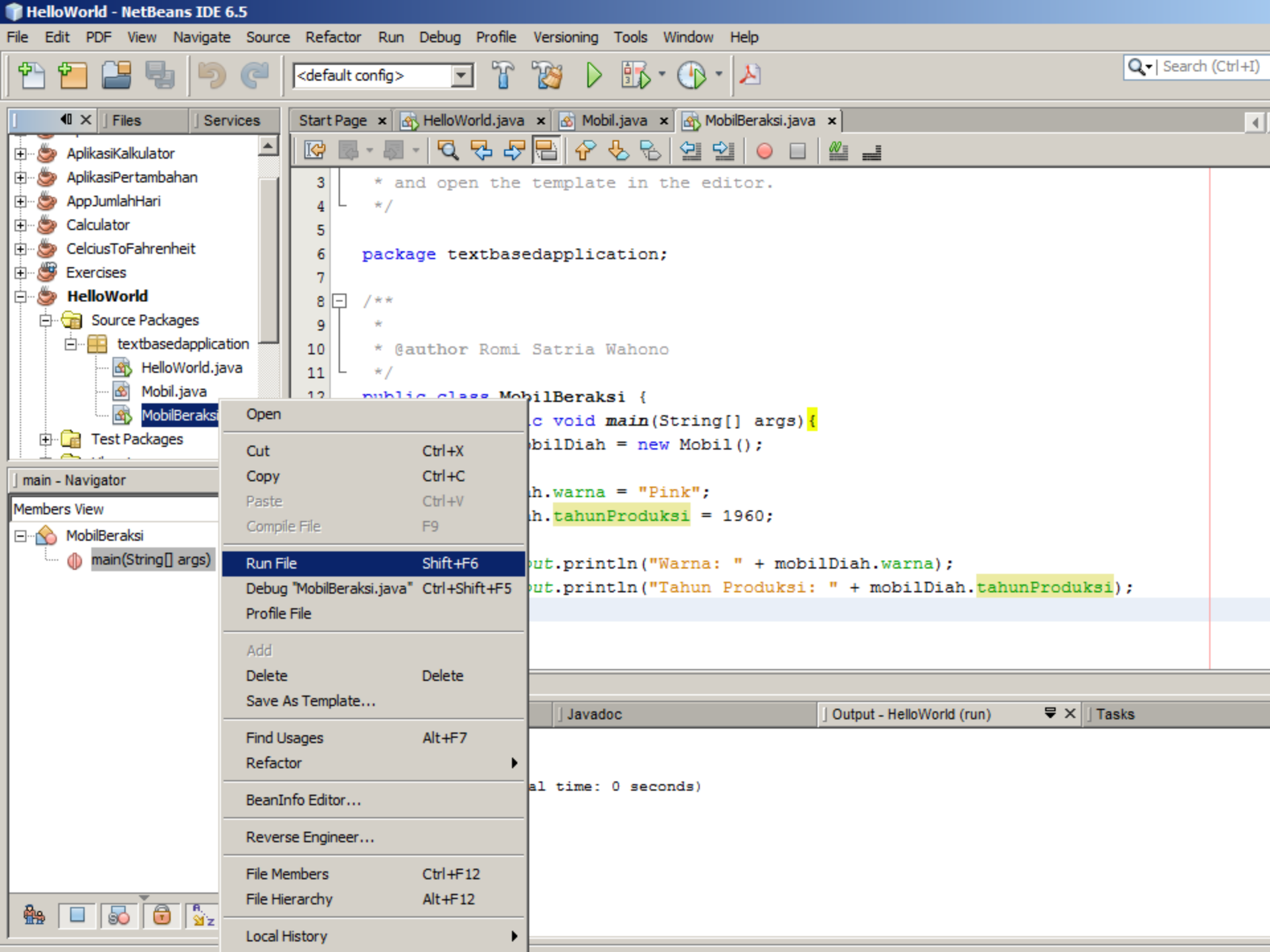
```
run:
Warna: Pink
Tahun Produksi: 1960
BUILD SUCCESSFUL (total time: 0 seconds)
```











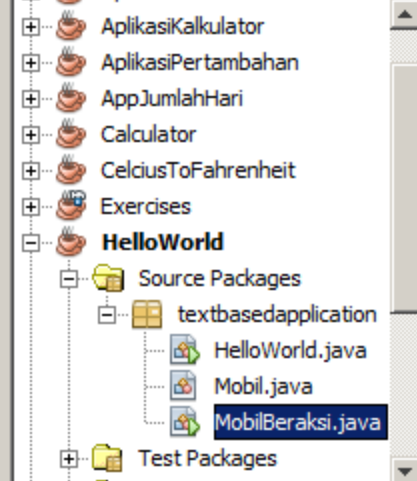


<default config>



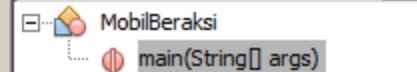
Search (Ctrl+I)

Files Services



main - Navigator

Members View



Start Page x HelloWorld.java x Mobil.java x MobilBeraksi.java x



```
3      * and open the template in the editor.
4      */
5
6      package textbasedapplication;
7
8      /**
9       *
10      * @author Romi Satria Wahono
11      */
12      public class MobilBeraksi {
13          public static void main(String[] args) {
14              Mobil mobilDiah = new Mobil();
15
16              mobilDiah.warna = "Pink";
17              mobilDiah.tahunProduksi = 1960;
18
19              System.out.println("Warna: " + mobilDiah.warna);
20              System.out.println("Tahun Produksi: " + mobilDiah.tahunProduksi);
21          }
22      }
23
```

21:6

Usages

Java

Output - HelloWorld (run)

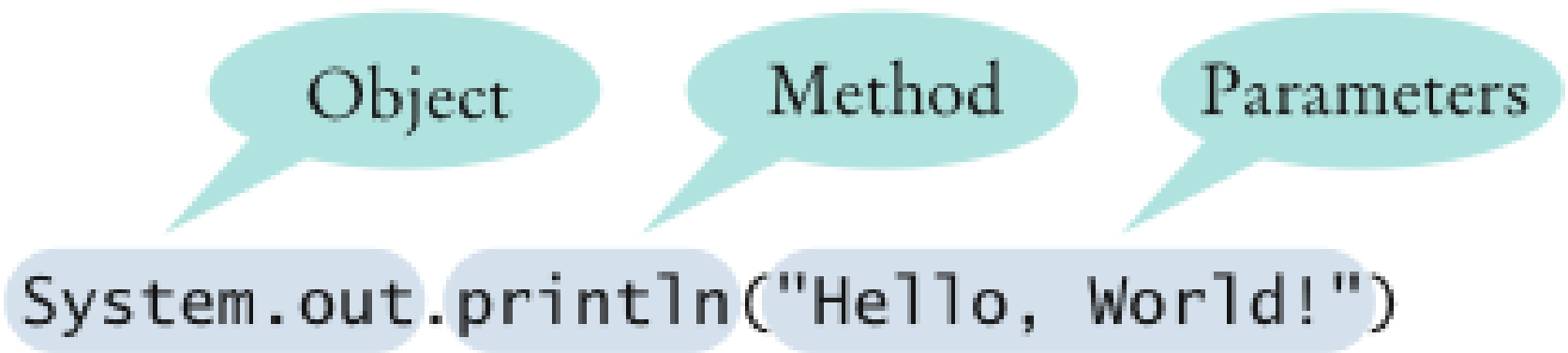
X

Tasks

```
run:
Warna: Pink
Tahun Produksi: 1960
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method

- Method adalah **urutan instruksi** yang mengakses data dari object
- Method melakukan:
 1. **Manipulasi data**
 2. **Perhitungan** matematika
 3. **Memonitor kejadian** dari suatu event



Method

Syntax *object.methodName(parameters)*

Example

The method is
invoked on this object.

This is the
name of the method.

This parameter is
passed to the method.

System.out.println("Hello")

Parameters are enclosed in parentheses.
Multiple parameters are separated by commas.

Membuat dan Memanggil Method

```
public class Mobil2{  
    String warna;  
    int tahunProduksi;
```

Mobil2.java

```
    void printMobil(){  
        System.out.println("Warna: " + warna);  
        System.out.println("Tahun: " + tahunProduksi);  
    }  
}
```

```
public class Mobil2Beraksi{  
    public static void main(String[] args){  
        Mobil2 mobilku = new Mobil2();  
  
        mobilku.warna = "Hitam";  
        mobilku.tahunProduksi = 2006;  
        mobilku.printMobil();  
    }  
}
```

Mobil2Beraksi.java

Jenis Method: Mutator dan Accessor

Syntax *accessSpecifier returnType methodName(parameterType parameterName, . . .)*
 {
 method body
 }

Example

These methods
are part of the
public interface.

public void deposit(double amount)
{
 balance = balance + amount;
}

This method does
not return a value.

A mutator method modifies
an instance variable.

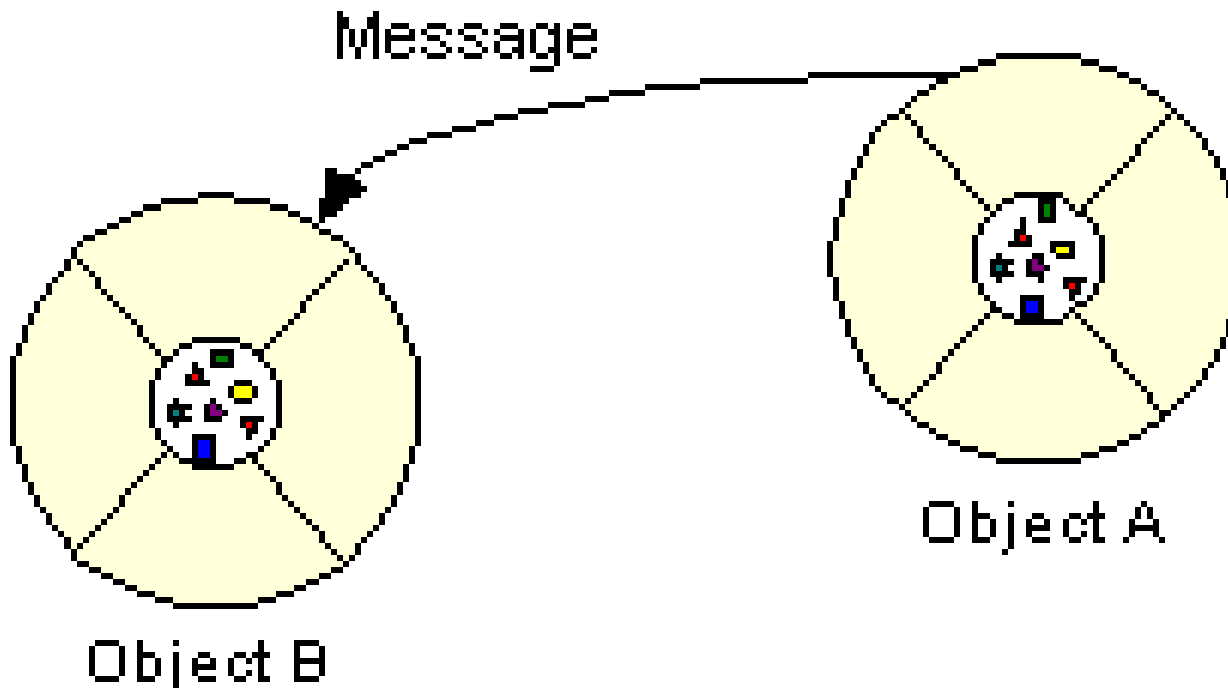
public double getBalance()
{
 return balance;
}

This method has
no parameters.

An accessor method returns a value.

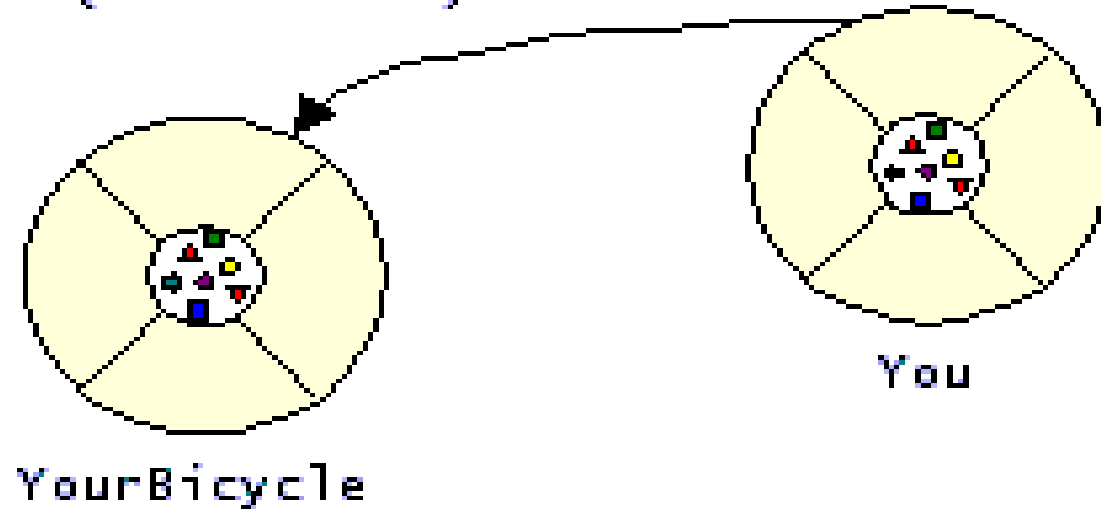
Parameter

- Sepeda akan berguna apabila ada object lain **yang berinteraksi dengan sepeda tersebut**
- Object software berinteraksi dan berkomunikasi dengan object lain dengan cara mengirimkan **message atau pesan**
- Pesan adalah suatu method, dan informasi dalam pesan dikenal dengan nama **parameter**



Pengiriman Pesan dan Parameter

`changeGears(lowerGear)`



1. **You** → object pengirim
2. **YourBicycle** → object penerima
3. **changeGears** → pesan berupa method yang dijalankan
4. **lowerGear** → parameter yang dibutuhkan method (pesan) untuk dijalankan

Sepeda.java

```
public class Sepeda{  
    int gir;  
  
    // method (mutator) dengan parameter  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    // method (accessor)  
    int getGir() {  
        return gir;  
    }  
}
```


SepedaBeraksi.java

```
public class SepedaBeraksi{  
    public static void main(String[] args) {  
        Sepeda sepedaku = new Sepeda();  
  
        sepedaku.setGir(1); // menset nilai gir = 1 (sebelumnya 0)  
        System.out.println("Gir saat ini: " + sepedaku.getGir());  
  
        sepedaku.setGir(3); // menambahkan 3 pada posisi gir saat ini (1)  
        System.out.println("Gir saat ini: " + sepedaku.getGir());  
    }  
}
```

Konstruktor -1-

- Method yang digunakan untuk memberi nilai awal pada saat object diciptakan
- Dipanggil secara otomatis ketika **new** digunakan untuk membuat instan class
- Sifat konstruktor:
 - Nama konstruktor **sama dengan nama class**
 - **Tidak memiliki nilai balik** dan tidak boleh ada kata kunci void

Konstruktor -2-

```
public class Mobil {  
    String warna;  
    int tahunProduksi;  
    public Mobil(String warna, int tahunProduksi){  
        this.warna = warna;  
        this.tahunProduksi = tahunProduksi;  
    }  
    public void info(){  
        System.out.println("Warna: " + warna);  
        System.out.println("Tahun: " + tahunProduksi);  
    }  
}
```

Mobil.java

```
public class MobilKonstruktor{  
    public static void main(String[] args){  
        Mobil mobilku = new Mobil("Merah", 2003);  
        mobilku.info();  
    }  
}
```

MobilKonstruktor.java

Kata Kunci this

Digunakan pada pembuatan class dan digunakan untuk **menyatakan object sekarang**

```
public class Mobil{  
    String warna;  
    int tahunProduksi;  
  
    void isiData(String aWarna,  
                  int aTahunProduksi){  
  
        warna = aWarna;  
        tahunProduksi = aTahunProduksi;  
    }  
}
```

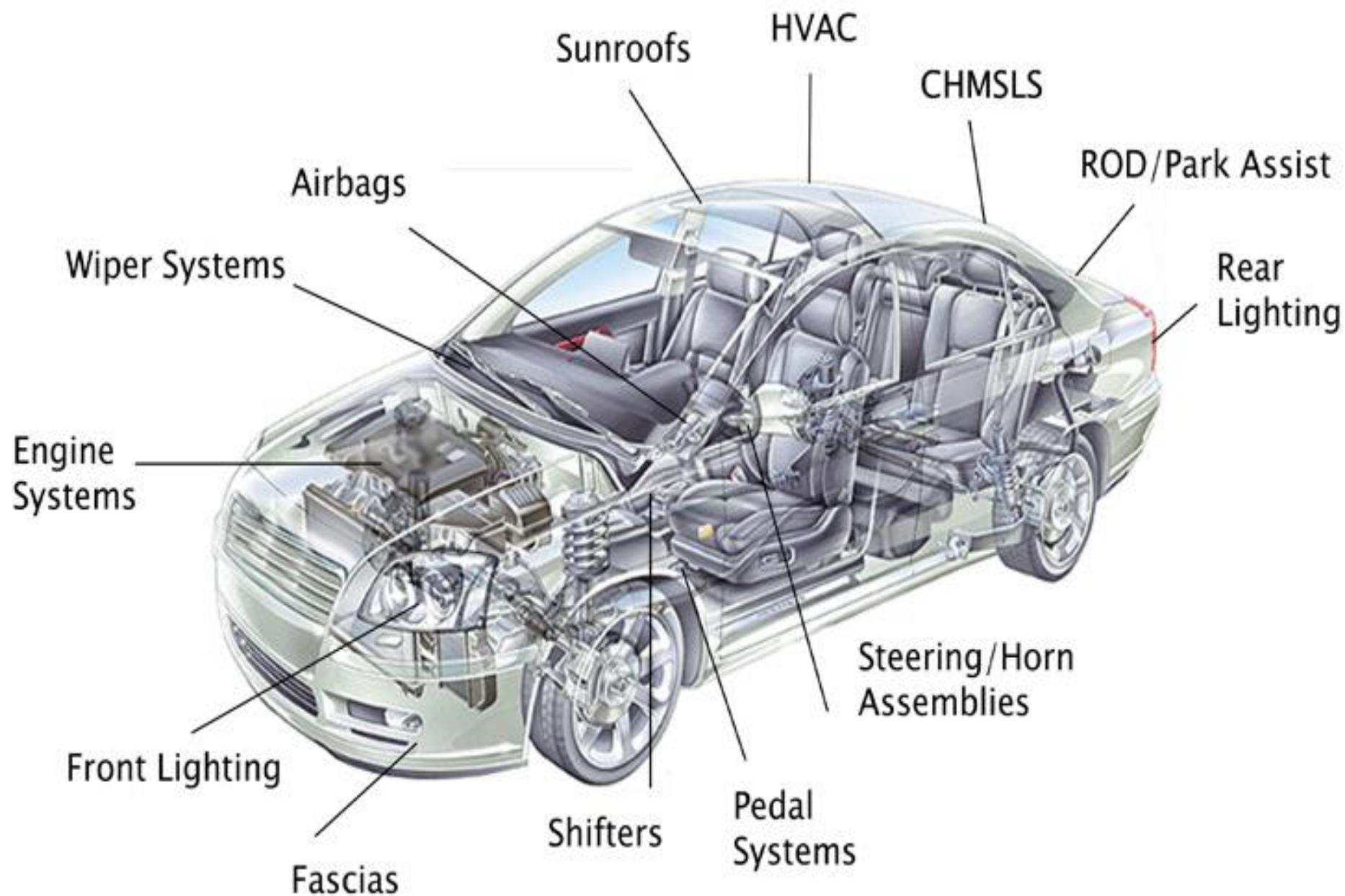
```
public class Mobil{  
    String warna;  
    int tahunProduksi;  
  
    void isiData(String warna,  
                  int tahunProduksi){  
  
        this.warna = warna;  
        this.tahunProduksi = tahunProduksi;  
    }  
}
```

PEMROGRAMAN BERBASIS OBJEK 2

KARAKTERISTIK PBO

Abstraction

- Cara kita melihat suatu sistem dalam **bentuk yang lebih sederhana**, yaitu sebagai suatu kumpulan subsistem (object) yang saling berinteraksi.
 - Mobil adalah kumpulan sistem pengapian, sistem kemudi, sistem pengereman
- Alat meng-abstraksikan sesuatu adalah **class**
- Object bersifat **modularity**. Object dapat ditulis dan **dimaintain terpisah (independen)** dari object lain



Encapsulation

- Mekanisme menyembunyikan suatu proses dan data dalam sistem untuk menghindari interferensi, dan menyederhanakan penggunaan proses itu sendiri
 - Tongkat transmisi (gigi) pada mobil
 - Tombol on/off/pengaturan suhu pada AC
- Class access level (public, protected, private) adalah implementasi dari konsep encapsulation
- Enkapsulasi data dapat dilakukan dengan cara:
 1. mendeklarasikan instance variable sebagai private
 2. mendeklarasikan method yang sifatnya public untuk mengakses variable tersebut

Encapsulation dan Access Modifier

Modifier	Dalam Class yang Sama	Dalam Package yang Sama	Dalam SubClass	Dalam Package Lain
private	✓			
tanpa tanda	✓	✓		
protected	✓	✓	✓	
public	✓	✓	✓	✓

Encapsulation

- Enkapsulasi data juga dapat dilakukan dengan cara:
 1. mendeklarasikan **instance variable** sebagai **private**
 2. mendeklarasikan **method** yang sifatnya **public** untuk mengakses variable tersebut

Syntax

```
accessSpecifier class ClassName  
{  
    instance variables  
    constructors  
    methods  
}
```

Example

```
public class Counter  
{
```

```
    private int value;
```

```
    public Counter(double initialValue) { value = initialValue; }
```

```
    public void count() { value = value + 1; }
```

```
    public int getValue() { return value; }
```

```
}
```

Public interface

Private
implementation

Sepeda.java

```
public class Sepeda{  
    int gir;  
  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir() {  
        return gir;  
    }  
}
```

SepedaBeraksi.java

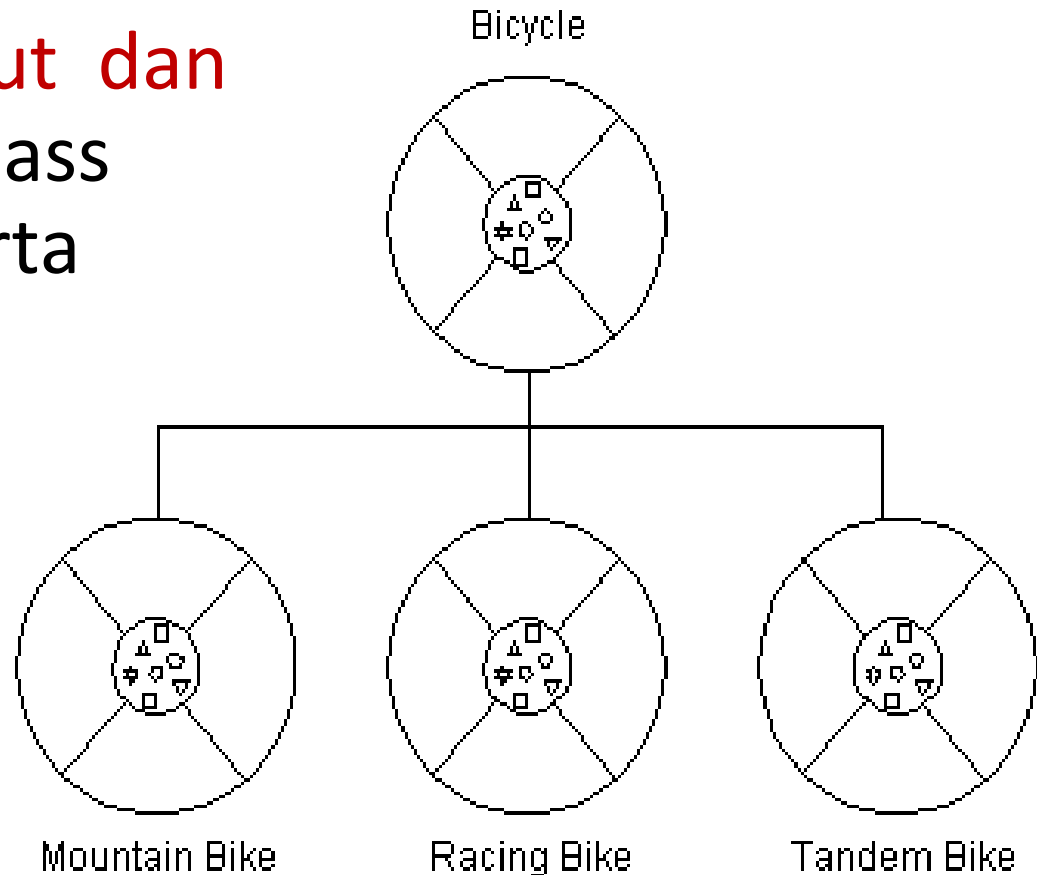
```
public class SepedaBeraksi{  
    public static void main(String[] args) {  
        Sepeda sepedaku = new Sepeda();  
  
        sepedaku.setGir(1);  
        /* Variabel bisa diubah atau tidak sengaja diubah.  
        Hal ini berbahaya dan sering menimbulkan bug.  
        Berikan access modifier private pada instance variable */  
        sepedaku.gir = 3;  
        System.out.println("Gir saat ini: " + sepedaku.getGir());  
    }  
}
```

Sepeda.java

```
public class Sepeda{  
    private int gir; // access modifier private pada instance variable  
  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir() {  
        return gir;  
    }  
}
```

Inheritance (Pewarisan)

- Suatu class dapat **mewariskan atribut dan method** kepada class lain (subclass), serta membentuk class hierarchy
- Penting untuk **Reusability**
- Java Keyword: **extends**



Sepeda.java

```
public class Sepeda{  
    private int gir;  
  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir() {  
        return gir;  
    }  
}
```

Class SepedaGunung Mewarisi Class Sepeda

```
public class SepedaGunung extends Sepeda{  
  
    private int sadel;  
  
    void setSadel (int jumlah) {  
        sadel = getGir() - jumlah;  
    }  
  
    int getSadel(){  
        return sadel;  
    }  
}
```

SepedaGunung.java

```
public class SepedaGunungBeraksi {  
    public static void main(String[] args) {  
  
        SepedaGunung sg=new SepedaGunung();  
  
        sg.setGir(3);  
        System.out.println(sg.getGir());  
  
        sg.setSadel(1);  
        System.out.println(sg.getSadel());  
    }  
}
```

SepedaGunungBeraksi.java

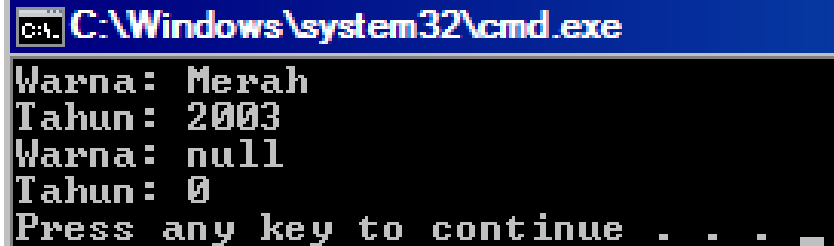
Polymorphism

- Kemampuan untuk **memperlakukan object yang memiliki perilaku (bentuk) yang berbeda**
- Implementasi konsep polymorphism:
 1. **Overloading**: Kemampuan untuk menggunakan **nama yang sama** untuk beberapa **method yang berbeda parameter (tipe dan atau jumlah)**
 2. **Overriding**: Kemampuan subclass untuk **menimpa method** dari superclass, yaitu dengan cara menggunakan nama dan parameter yang sama pada method

Polymorphism – Overloading

```
class Mobil {  
    String warna;  
    int tahunProduksi;  
  
    public Mobil(String warna, int tahunProduksi){  
        this.warna = warna;  
        this.tahunProduksi = tahunProduksi;  
    }  
  
    public Mobil(){  
    }  
  
    void info(){  
        System.out.println("Warna: " + warna);  
        System.out.println("Tahun: " + tahunProduksi);  
    }  
}
```

```
public class MobilKonstruktor{  
    public static void main(String[] args){  
        Mobil mobilku = new Mobil("Merah", 2003);  
        mobilku.info();  
  
        Mobil mobilmu = new Mobil();  
        mobilmu.info();  
    }  
}
```



C:\Windows\system32\cmd.exe

```
Warna: Merah  
Tahun: 2003  
Warna: null  
Tahun: 0  
Press any key to continue . . . _
```

Polymorphism – Overloading

```
class Lingkaran{  
    void gambarLingkaran(){  
    }  
    void gambarLingkaran(int diameter){  
    ...  
    }  
    void gambarLingkaran(double diameter){  
    ...  
    }  
    void gambarLingkaran(int diameter, int x, int y){  
    ...  
    }  
    void gambarLingkaran(int diameter, int x, int y, int warna, String  
    namaLingkaran){
```

Polymorphism - Overriding

```
public class Sepeda{  
    protected int gir;  
  
    void setGir(int pertambahanGir) {  
        gir= gir+ pertambahanGir;  
    }  
  
    int getGir() {  
        return gir;  
    }  
}
```

Polymorphism - Overriding

```
public class SepedaGunung extends Sepeda{  
  
    @override  
    void setGir(int pertambahanGir) {  
        super.setGir(pertambahanGir);  
        gir = 2*getGir();  
    }  
}
```

SepedaGunung.java

```
public class SepedaGunungBeraksi {  
    public static void main(String[] args) {  
  
        SepedaGunung sg=new SepedaGunung();  
  
        sg.setGir(2);  
        System.out.println(sg.getGir());  
  
        sg.setGir(3);  
        System.out.println(sg.getGir());  
    }  
}
```

SepedaGunungBeraksi.java

PEMROGRAMAN BERBASIS OBJEK 2

PENGORGANISASIAN CLASS

Packages

- Package adalah **koleksi dari beberapa class dan interface yang berhubungan, dan menyediakan proteksi akses dan pengelolaan namespace**
- 1 package adalah 1 folder di file system
- Package berguna untuk **mengorganisir file** dalam suatu project atau library
- Nama package menggunakan **lowercase**
- Nama package mengikuti **nama domain** (perusahaan) dengan **susunan terbalik**
 - Contoh: com.brainmatics.kendaraan
- Keyword: ***package name;***

Packages

Syntax `package packageName;`

Example

`package com.horstmann.bigjava;`

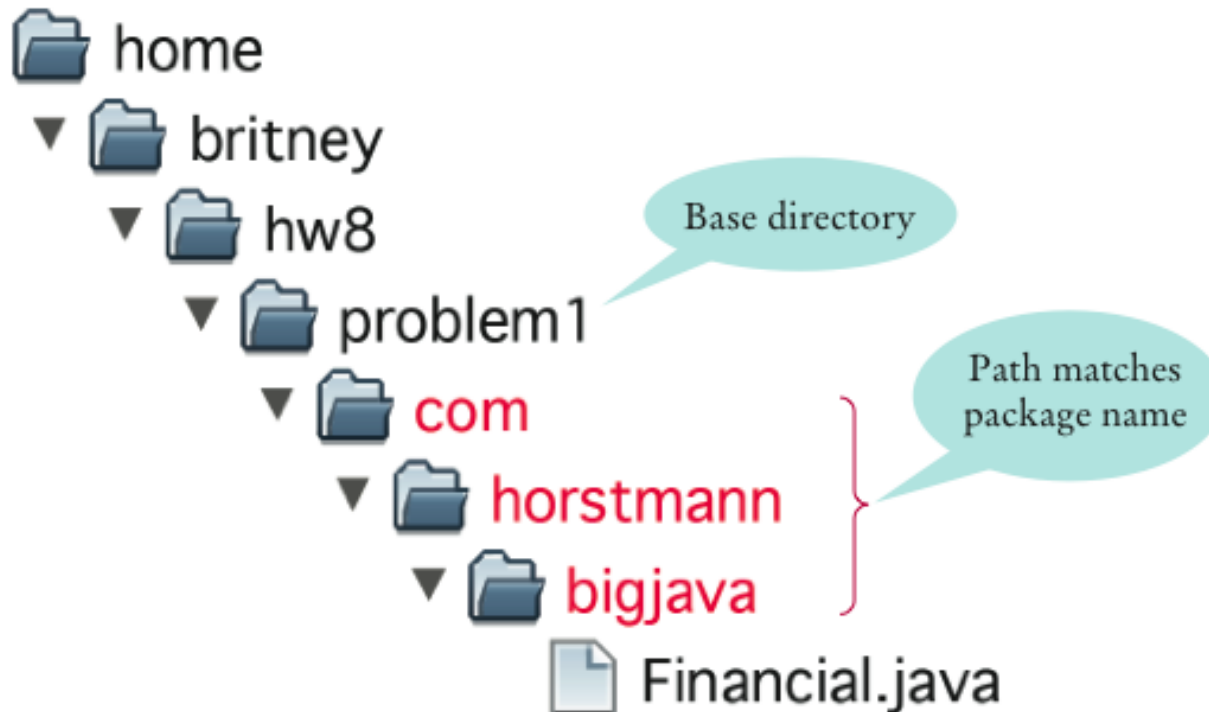
The classes in this file
belong to this package.

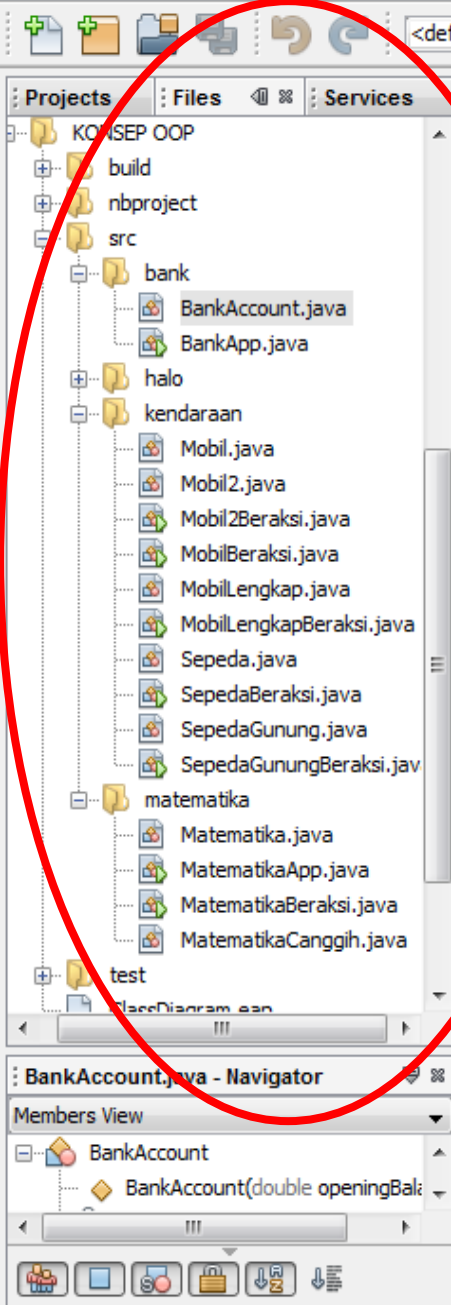
A good choice for a package name
is a domain name in reverse.

Packages

- **Base directory:** holds your program's Files
- **Path name**, relative to base directory, must match **package name**:

`com/horstmann/bigjava/Financial.java`





The screenshot shows the NetBeans IDE interface with the following components:

- Projects, Files, Services:** The top-left pane showing the project structure. The 'bank' package is expanded, showing 'BankAccount.java' and 'BankApp.java'. The 'matematika' package is also expanded, showing several Java files including 'Matematika.java', 'MatematikaApp.java', 'MatematikaBeraksi.java', and 'MatematikaCanggih.java'.
- BankAccount.java - Navigator:** The middle-left pane showing the 'BankAccount' class and its member 'BankAccount(double openingBal...'.
- Members View:** The bottom-left pane showing the 'BankAccount' class and its member 'BankAccount(double openingBal...'.
- Source Editor:** The main editor window displaying the code for 'BankAccount.java'. The code includes a package declaration, a class declaration, and a constructor.
- Output - KONSEP OOP (run):** The bottom-right pane showing the output of the run command. It displays an exception message: 'Exception in thread "main" java.lang.RuntimeException: Uncompilable source code' and a 'BUILD SUCCESSFUL' message.

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package bank;
7
8  /**
9   *
10  * @author ROMI SATRIA WAHONO
11  */
12  public class BankAccount {
13
14      private double balance;
15      String test;
16
17      public BankAccount(double openingBalance) {
18          balance = openingBalance;
19          System.out.println("Balance awal sebelum transaksi = " + openingBalance);
20      }
21
22      public void deposit(double amount) {
```

run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code
at kendaraan.SepedaGunungBeraksi.main(SepedaGunungBeraksi.java:15)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)

Budi.java

```
package kelasku;
```

```
public class Budi{
```

```
    public void info(){
```

```
        System.out.println("Kelas Budi");
```

```
    }
```

```
}
```

Joko.java

```
package kelasku;
```

```
public class Joko{
```

```
    public void info(){
```

```
        System.out.println("Kelas Joko");
```

```
    }
```

```
}
```

PaketBeraksi.java

```
import kelasku.Joko;
```

```
public class PaketBeraksi{  
    public static void main(String[] args){  
        Joko objectJoko = new Joko();  
        objectJoko.info();  
    }  
}
```

PaketBeraksi.java

```
import kelasku.*;
```

```
public class PaketBeraksi{  
    public static void main(String[] args){  
        Budi objectBudi = new Budi();  
        objectBudi.info();  
        Joko objectJoko = new Joko();  
        objectJoko.info();  
    }  
}
```

Interface

- Interface digunakan apabila kita ingin menentukan apa yang harus dilakukan oleh suatu class tapi **tidak menentukan bagaimana cara untuk melakukannya**
- Interface sebenarnya sama dengan class, tapi hanya memiliki **deklarasi method tanpa implementasi**

Interface dan Implementation

Syntax

```
public interface InterfaceName
{
    method signatures
}
```

Example

```
public interface Measurable
{
    double getMeasure();
}
```

The methods of an interface are automatically public.

No implementation is provided.

Syntax

```
public class ClassName implements InterfaceName, InterfaceName, . . .
{
    instance variables
    methods
}
```

Example

```
public class BankAccount implements Measurable
{
    . . .
    public double getMeasure()
    {
        return balance;
    }
    . . .
}
```

BankAccount instance variables

Other BankAccount methods

List all interface types that this class implements.

This method provides the implementation for the method declared in the interface.

InterfaceLampu.java

```
interface InterfaceLampu{  
    public static final int KEADAAN_HIDUP=1;  
    public static final int KEADAAN_MATI=0;  
  
    public abstract void hidupkan();  
    public abstract void matikan();  
}
```

Lampu.java

```
public class Lampu implements InterfaceLampu{
    int statusLampu;

    public void hidupkan(){
        if (statusLampu == KEADAAN_MATI){
            statusLampu = KEADAAN_HIDUP;
            System.out.println("Hidupkan Lampu! --> Lampu Hidup");
        }else{
            System.out.println("Hidupkan Lampu! --> Lampu Sudah Hidup Kok");
        }
    }

    public void matikan(){
        if (statusLampu == KEADAAN_HIDUP){
            statusLampu = KEADAAN_MATI;
            System.out.println("Matikan Lampu! --> Lampu Mati");
        }else{
            System.out.println("Matikan Lampu! --> Lampu Sudah Mati Kok");
        }
    }
}
```

LampuBeraksi.java

```
public class LampuBeraksi{  
    public static void main(String[] args){  
  
        Lampu lampuKamar = new Lampu();  
  
        System.out.println("Status Lampu Saat Ini: Mati");  
  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
        lampuKamar.matikan(); //Matikan Lampu  
        lampuKamar.matikan(); //Matikan Lampu  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
        lampuKamar.hidupkan(); //Hidupkan Lampu  
    }  
}
```

Files

Test Libraries

OOP Concepts

Source Packages

- buku
 - Buku.java
 - BukuApp.java
- halo
 - HaloJakarta.java
- kendaraan
 - Mobil.java
 - Mobil2.java
 - Mobil2App.java
 - MobilApp.java
- lampu
 - InterfaceLampu.java
 - Lampu.java
 - LampuBeraksi.java**
- matematika

InterfaceLampu.java Lampu.java LampuBeraksi.java

```

10  * @author ROMI SAIKIA WAHONO
11  */
12  public class LampuBeraksi {
13      public static void main(String[] args) {
14
15          Lampu lampuKamar = new Lampu();
16
17          System.out.println("Status Lampu Saat Ini: Mati");
18
19          lampuKamar.hidupkan(); //Hidupkan Lampu
20          lampuKamar.matikan(); //Matikan Lampu
21          lampuKamar.matikan(); //Matikan Lampu
22          lampuKamar.hidupkan(); //Hidupkan Lampu
  
```

main - Navigator

Members View

- main(String[] args)

Output - OOP Concepts (...

run:

```

Status Lampu Saat Ini: Mati
Hidupkan Lampu! --> Lampu Hidup
Matikan Lampu! --> Lampu Mati
Matikan Lampu! --> Lampu Sudah Mati Kok
Hidupkan Lampu! --> Lampu Hidup
Hidupkan Lampu! --> Lampu Sudah Hidup Kok
BUILD SUCCESSFUL (total time: 0 seconds)
  
```

Kompresi dengan JAR

- JAR atau Java Archive adalah metode **kompresi standard** dari file-file yang berisi program Java
- JAR menampung file **.class** dan file lain yang dibutuhkan supaya program bisa berjalan dengan baik
- Kompresi dapat dilakukan setelah **semua class dikompilasi**

Penggunaan JAR

- Perintah **Membuat** file JAR:

```
jar -cvf namafile.jar file1.class file2.class
```

- Perintah **Melihat** isi dalam file JAR:

```
jar -tvf namafile.jar
```

- Perintah **Mengekstraksi** isi file JAR:

```
jar -xvf namafile.jar
```

- **Keterangan** Pilihan:

- **c** = create (membuat file JAR)
- **v** = verbose (menampilkan informasi pada layar)
- **f** = filename (daftar nama file yang akan dikompresi)

JAR Manifest

- JAR Manifest dibuat secara otomatis dan diletakkan di dalam folder META-INF pada file kompresi yang kita buat
- JAR Manifest digunakan untuk mendeskripsikan file-file yang terdalem dalam file JAR

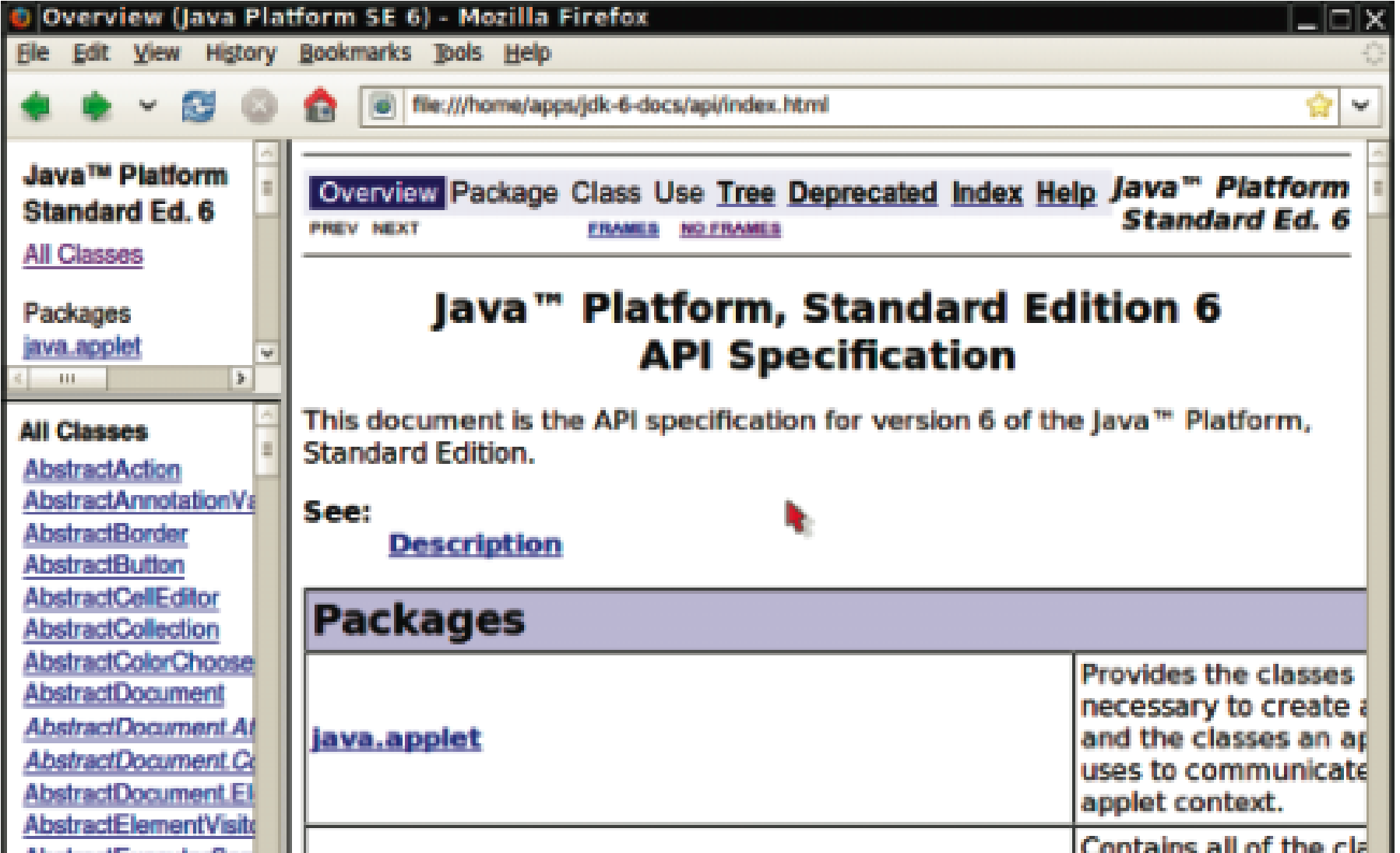
Java API Library and Documentation

- **API**: Application Programming Interface
- **API documentation**: daftar class dan method di java library
- <http://java.sun.com/javase/7/docs/api/index.html>

Important Packages in the Java Library

Package	Purpose	Sample Class
java.lang	Language support	Math
java.util	Utilities	Random
java.io	Input and output	PrintStream
java.awt	Abstract Windowing Toolkit	Color
java.applet	Applets	Applet
java.net	Networking	Socket
java.sql	Database Access	ResultSet
javax.swing	Swing user interface	JButton
org.w3c.dom	Document Object Model for XML documents	Document

API Documentation of the Java Library



API Documentation for the Rectangle Class

The screenshot shows the Java Platform Standard Ed. 6 API documentation for the `Rectangle` class. The left sidebar contains a navigation pane with links to 'All Classes', 'Packages', and a list of classes including `AbstractAction`, `AbstractAnnotationVa`, `AbstractBorder`, `AbstractButton`, `AbstractCellEditor`, `AbstractCollection`, `AbstractColorChoose`, `AbstractDocument`, `AbstractDocumentAt`, `AbstractDocument Co`, `AbstractDocument El`, `AbstractElementVisi`, `AbstractExecutorSer`, `AbstractInterruptibleC`, `AbstractLayoutCache`, and `AbstractLayovtCache`. The main content area displays the class hierarchy for `java.awt.Rectangle`, showing it extends `java.awt.geom.Rectangle2D`, which in turn extends `java.awt.geom.RectangularShape`, which extends `java.lang.Object`. Below the hierarchy, it lists 'All Implemented Interfaces: `Shape`, `Serializable`, `Cloneable`' and 'Direct Known Subclasses: `DefaultCaret`'. The class declaration is shown as `public class Rectangle extends Rectangle2D implements Shape, Serializable`. A description follows: 'A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x,y) in the coordinate space, its width, and its height.'

Java™ Platform Standard Ed. 6

All Classes

Packages

java.applet

All Classes

AbstractAction

AbstractAnnotationVa

AbstractBorder

AbstractButton

AbstractCellEditor

AbstractCollection

AbstractColorChoose

AbstractDocument

AbstractDocumentAt

AbstractDocument Co

AbstractDocument El

AbstractElementVisi

AbstractExecutorSer

AbstractInterruptibleC

AbstractLayoutCache

AbstractLayovtCache

Overview Package **Class** Use Tree Deprecated Index Help Java™ Platform Standard Ed. 6

PREV CLASS NEXT CLASS

FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

java.awt

Class Rectangle

java.lang.Object

- ↳ java.awt.geom.RectangularShape
 - ↳ java.awt.geom.Rectangle2D
 - ↳ java.awt.Rectangle

All Implemented Interfaces:
[Shape](#), [Serializable](#), [Cloneable](#)

Direct Known Subclasses:
[DefaultCaret](#)

```
public class Rectangle
extends Rectangle2D
implements Shape, Serializable
```

A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x,y) in the coordinate space, its width, and its height.

Method Summary

Java™ Platform
Standard Ed. 6

[All Classes](#)

Packages
[java.applet](#)

All Classes

- [AbstractAction](#)
- [AbstractAnnotationVa](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChoose](#)
- [AbstractDocument](#)
- [AbstractDocumentAt](#)
- [AbstractDocumentCo](#)
- [AbstractDocumentEl](#)
- [AbstractElementVisite](#)
- [AbstractExecutorServ](#)

Method Summary	
void	add (int newx, int newy) Adds a point, specified by the integer arguments newx,newy to the bounds of this Rectangle.
void	add (Point pt) Adds the specified Point to the bounds of this Rectangle.
void	add (Rectangle r) Adds a Rectangle to this Rectangle.
boolean	contains (int x, int y) Checks whether or not this Rectangle contains the point at the specified location (x,y).
boolean	contains (int X, int Y, int W, int H) Checks whether this Rectangle entirely contains the Rectangle at the specified location (X,Y) with the specified dimensions (W,H).
boolean	contains (Point p) Checks whether or not this Rectangle contains the specified Point.
boolean	contains (Rectangle r) Checks whether or not this Rectangle entirely contains the specified Rectangle.