UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Objects and Classes

Dasar – Dasar Pemrograman 2

Dinial Utami Nurul Qomariah

FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Introduction

- If we want create watermelon without class

```
//create first watermelon
String watermelon1 = white;
//set heavy for watermelon 1
double heavy_watermelon1 = 5.5;

//create second watermelon
String watermelon2 = yellow;
double heavy_watermelon2 = 10.0;

//create third watermolon
String watermelon3 = red;
double heavy_watermelon3 = 1.5;

System.out.format("Watermelon with color = %s and
heavy = %.2f ",watermelon1,heavy_watermelon1);
```

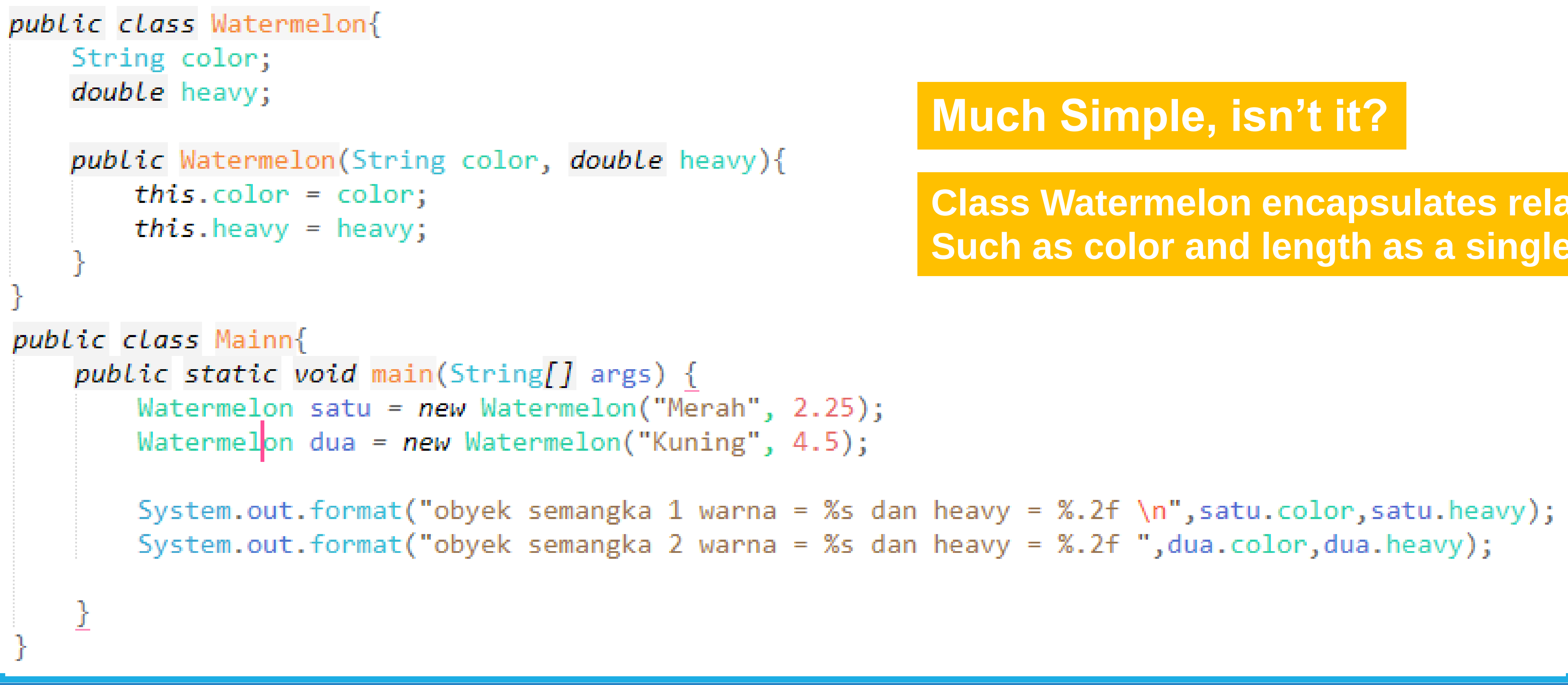
Soo Complicated

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS
ILMU
KOMPUTER

Introduction

- Try this one: Create **Watermelon** with **class**



```
public class Watermelon{
    String color;
    double heavy;

    public Watermelon(String color, double heavy){
        this.color = color;
        this.heavy = heavy;
    }
}
public class Mainn{
    public static void main(String[] args) {
        Watermelon satu = new Watermelon("Merah", 2.25);
        Watermelon dua = new Watermelon("Kuning", 4.5);

        System.out.format("obyek semangka 1 warna = %s dan heavy = %.2f \n",satu.color,satu.heavy);
        System.out.format("obyek semangka 2 warna = %s dan heavy = %.2f ",dua.color,dua.heavy);

    }
}
```

Much Simple, isn't it?

Class Watermelon encapsulates related data
Such as color and length as a single unit

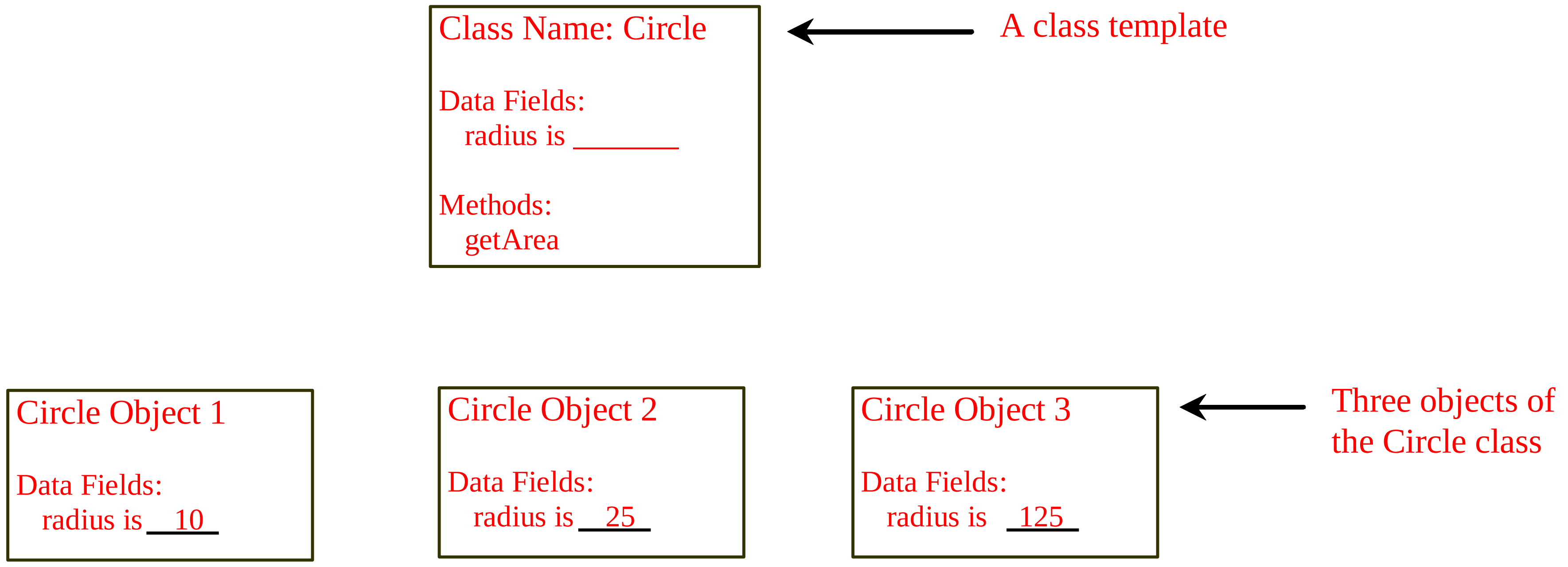
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS
ILMU
KOMPUTER

OO Programming Concepts

- ❖ Object-oriented programming (**OOP**) involves programming using objects.
- ❖ An ***object*** represents an entity in the real world that can be distinctly identified.
  - ❖ For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- ❖ An **object** has a unique identity, state, and behaviors.
- ❖ The ***state*** of an object **consists** of a set of ***data fields*** (also known as *properties*) with their current values.
- ❖ The ***behavior*** of an object is defined by a set of methods.
- ❖ **Class** berupa kerangka/cetakan dari suatu obyek, **obyek** adalah instance/perwujutan dari class.

Objects



- An **object** has both a **state** and **behavior**.
- The **state defines** the **object**
- The **behavior defines** what the **object does**.

Classes

- ***Classes*** are **constructs** that define **objects** of the **same type**.
- A Java class uses **variables** to **define data fields** and **methods** to **define behaviors**.
- Additionally, a **class provides** a **special** type of **methods**, known **as constructors**, which are invoked to construct objects from the class.
- dan **Instansiasi** proses **pembuatan obyek** dari **suatu class dengan** cara **memanggil contructor** dari **class tersebut**.

Object oriented thinking is natural

We have the class Home,
And the object are all those house instances!

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS
ILMU
KOMPUTER

And these hero and cute anime character?

They are objects as well, of type anime character.

FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS
INDONESIA

Veritas, Probitas, Iustitia

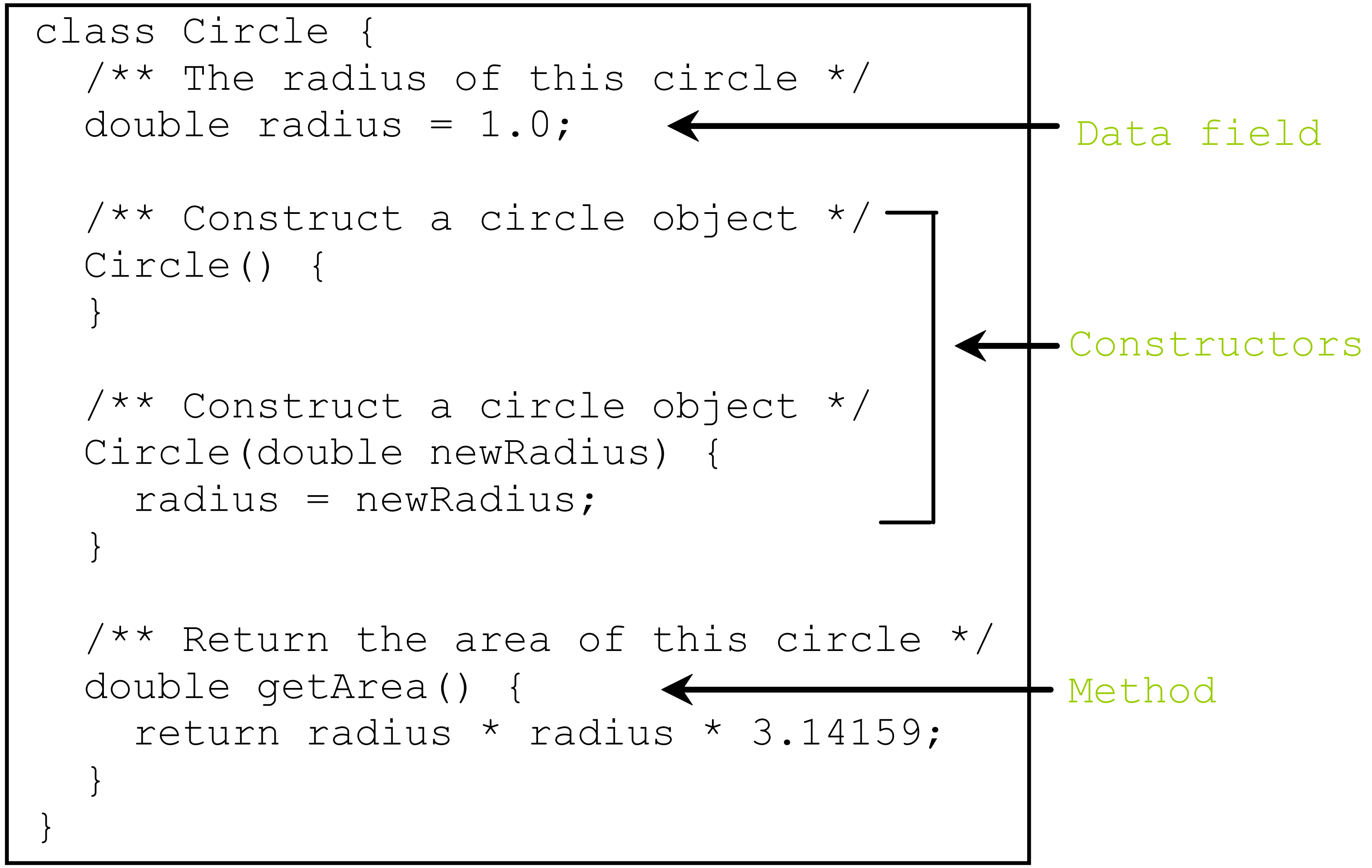
Object can be abstract Such as ideas, emotions, and knowledge.

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

The class cake mold
The object are the cake

Classes



```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Constructors

A constructor with no parameters is referred to as a *no-arg constructor*.

- a special kind of methods that are invoked to construct objects.
- must have the same name as the class itself.
- do not have a return type—not even void.
- invoked using the new operator when an object is created. Constructors play the role of initializing objects.

```
Circle() {
}

Circle(double newRadius) {
  radius = newRadius;
}
```

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

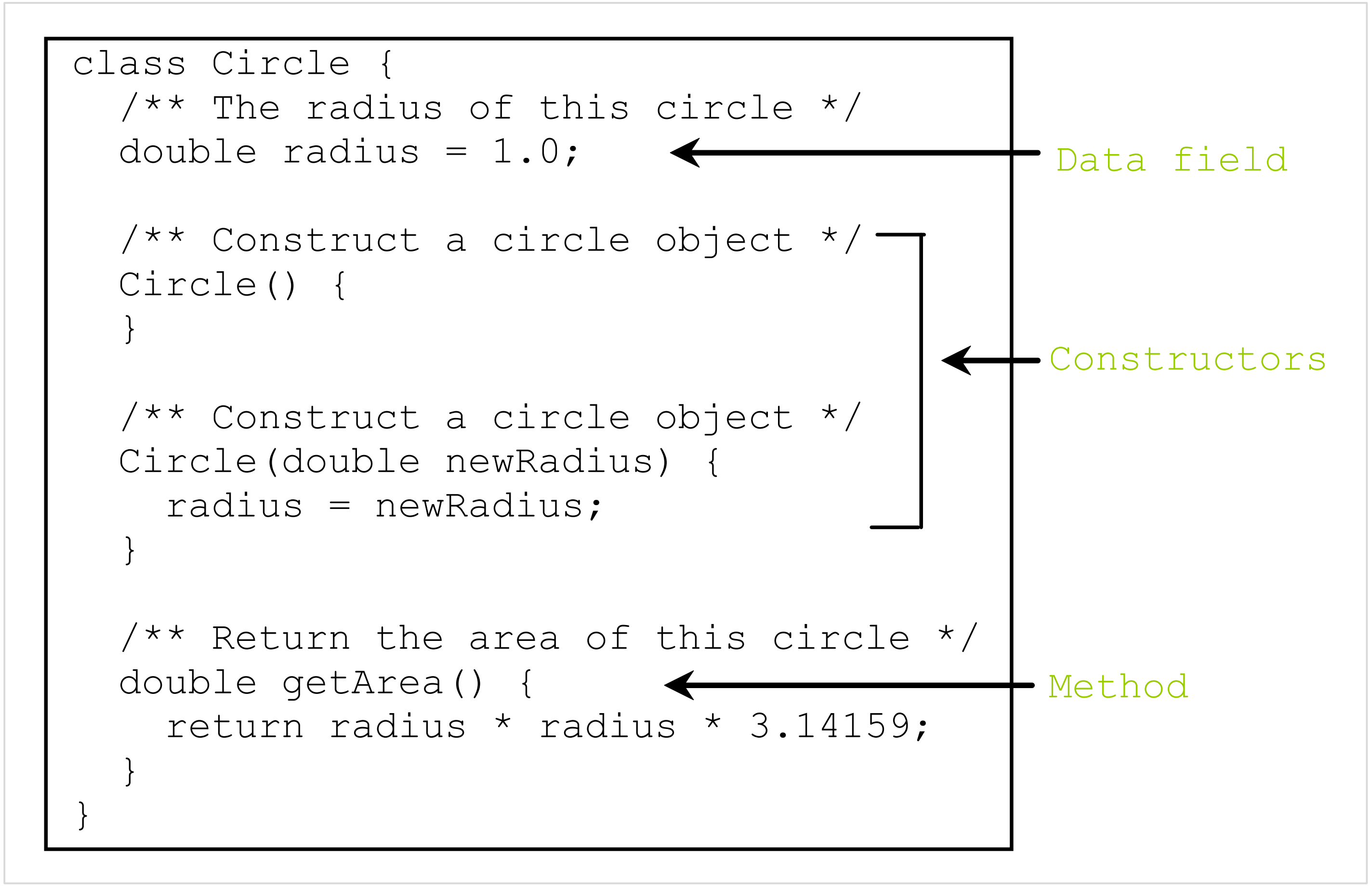
Veritas, Probitas, Iustitia

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();

new Circle(5.0);
```

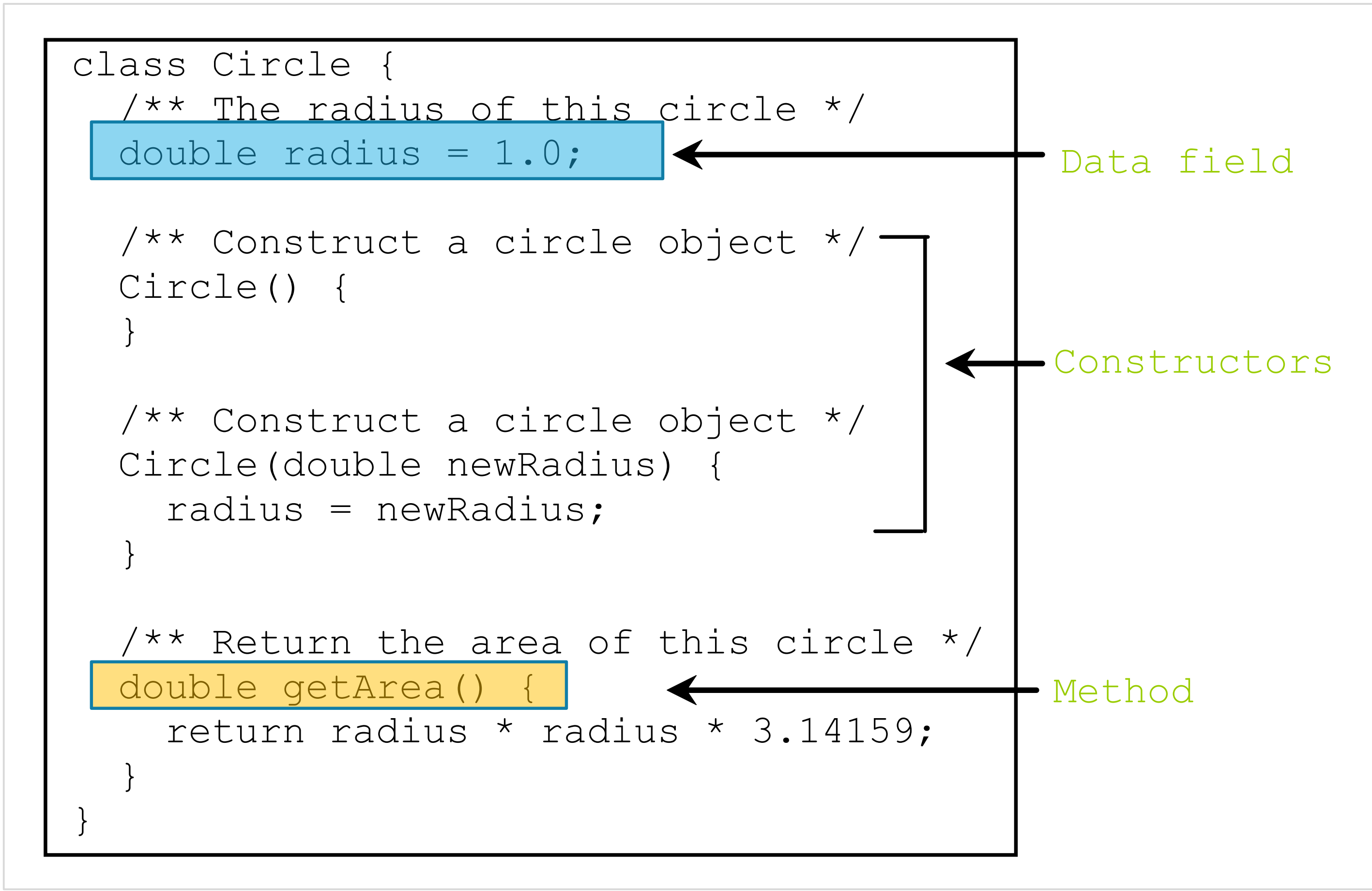


```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

Data Field and Methods

Data Field:
Identity/Properties of the objects.

Methods: behaviors of the objects.



```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** Construct a circle object */
  Circle() {
  }

  /** Construct a circle object */
  Circle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * 3.14159;
  }
}
```

this Keyword

- `this` keyword refers to an object itself.
- One common use of the this keyword is reference a class's *data fields*.

```
class Circle{
	double radius;
	Circle(double radius) {
		this.radius = radius;
	}

	double getArea(){
		return Math.PI * this.radius * this.radius;
	}
}
```

`this` Keyword

- Another common use of the **`this`** keyword is to enable a constructor to invoke another constructor of the same class.

```
class Circle{
   double radius;
   Circle() {
     this(0.0);
   }

   Circle(double radius){
     this.radius=radius;
   }
}
```

Declare and Initialization Object Reference Variables

To reference an object, assign the object to a reference variable.

To declare a reference variable, use the syntax:

```
ClassName objectRefVar;
objectRefVar = new ClassName();
```

Example:

```
Circle myCircle;
myCircle = new Circle();
```

FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Declaring/Creating Objects in a Single Step

```
ClassName objectRefVar = new ClassName();
```

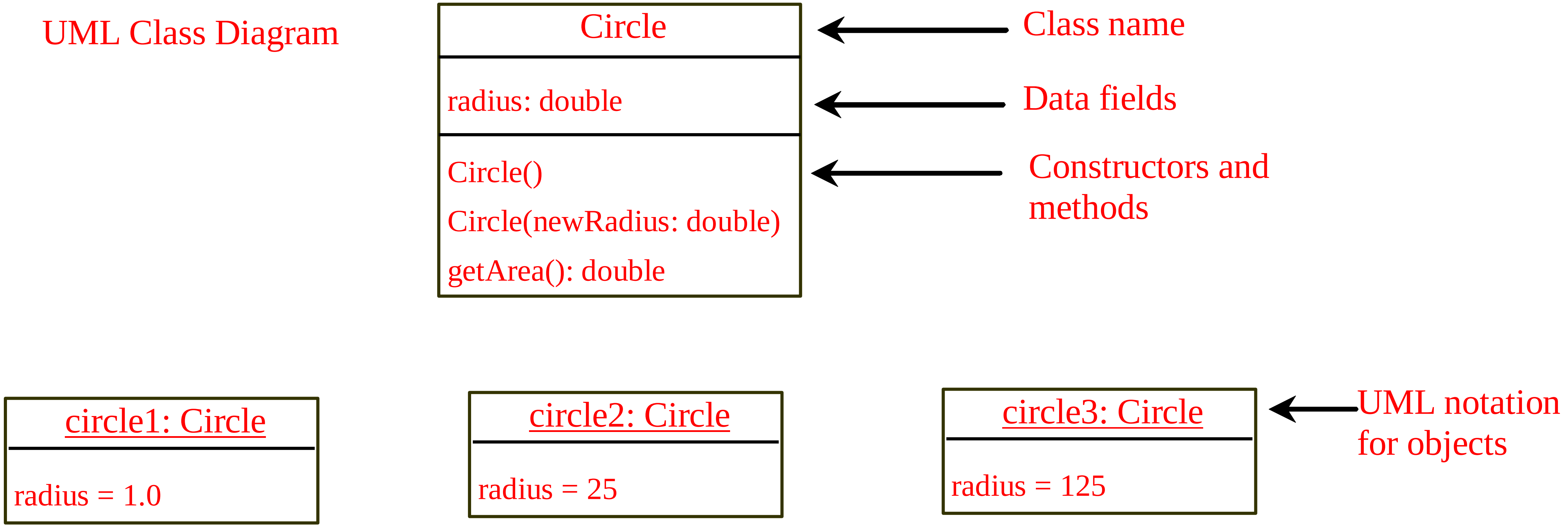
Create an object

Assign object reference

Example:

```
Circle myCircle = new Circle();
```

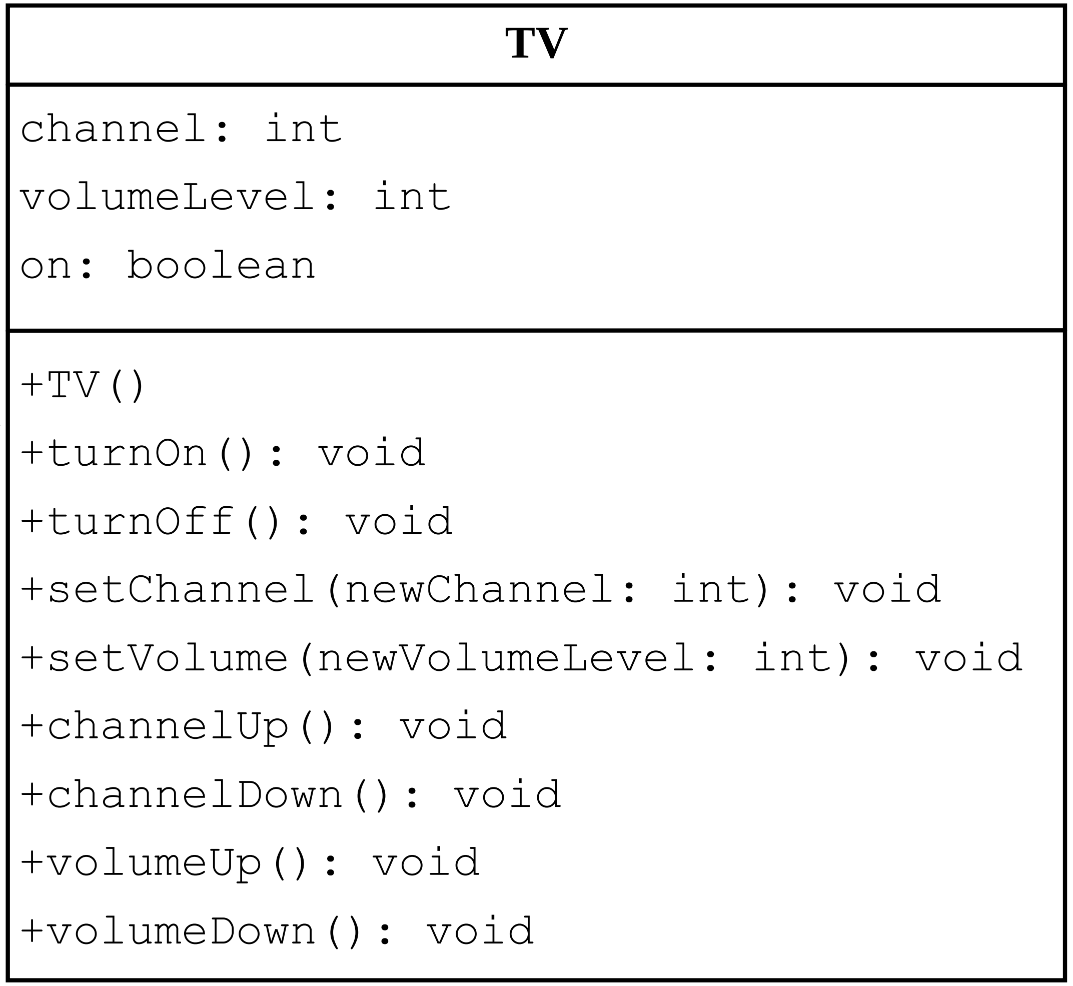
Unified Modelling Language (UML) Class Diagram



For modeling Class diagram Can Use Star UML, Power Designer and etc

FAKULTAS ILMU KOMPUTER

Example: Defining Classes and Creating Objects (UML)



TV

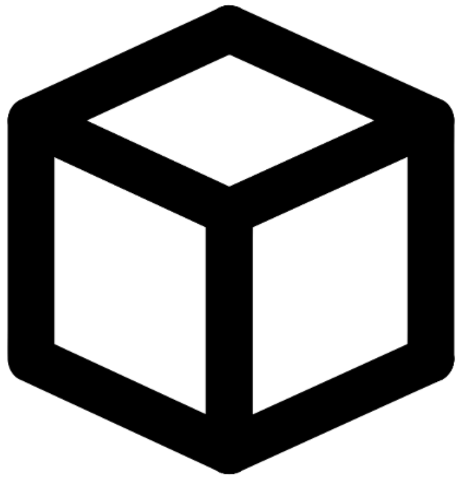
`channel: int`	The current channel (1 to 120) of this TV.
`volumeLevel: int`	The current volume level (1 to 7) of this TV.
`on: boolean`	Indicates whether this TV is on/off.
`+TV()`	Constructs a default TV object.
`+turnOn(): void`	Turns on this TV.
`+turnOff(): void`	Turns off this TV.
`+setChannel(newChannel: int): void`	Sets a new channel for this TV.
`+setVolume(newVolumeLevel: int): void`	Sets a new volume level for this TV.
`+channelUp(): void`	Increases the channel number by 1.
`+channelDown(): void`	Decreases the channel number by 1.
`+volumeUp(): void`	Increases the volume level by 1.
`+volumeDown(): void`	Decreases the volume level by 1.

The + sign indicates a public modifier. $\longrightarrow$

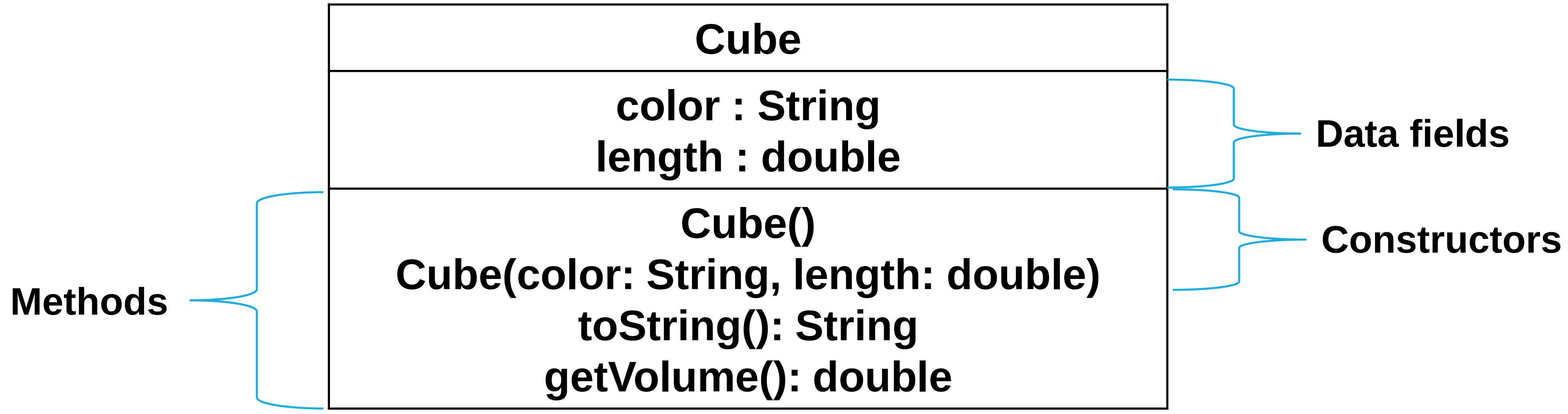
FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Say Hai To OOP!

- What can be the fields of a cube?
- What can be the methods of a cube?



UML Class Diagram



Cube

color : String
length : double

Cube()
Cube(color: String, length: double)
toString(): String
getVolume(): double

Objects of Cube

UML Class Diagram

Cube
color : String length : double

Cube()
Cube(color: String, length: double)
toString(): String
getVolume(): double

Cube1
color : “Blue” length : 1.0

Cube2
color : “Blue” length : 2.0

A class is like a factory or a blueprint to create objects

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

From UML become Code. Let’s Do it.

Data Fiels

Constructor

```
public class Cube{
    String color;
    double length;

    public Cube(){
        this.color = "White";
        this.length = 1.0;
    }

    public Cube(String color, double length){
        this.color = color;
        this.length = length;
    }
```

UML Class Diagram

Cube
color : String **length : double**

Cube()
Cube(color: String, length: double)
toString(): String
getVolume(): double

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

From UML become Code. Let's Do it.

```
public String toString(){
        return String.format("Cube with
length = %.2f and color = %s",
this.length, this.color);
    }

    public double getVolume(){
        return Math.pow(this.length, 3.0);
    }
}
```

Methods

UML Class Diagram

Cube
color : String **length : double**

Cube()
Cube(color: String, length: double)
toString(): String
getVolume(): double

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Recall the Cube class, let's instantiate it!

```
public class MainCube{
    public static void main(String[] args){
        Cube cube1 = new Cube("Purple", 5.0);
        Cube cube2 = new Cube();

        System.out.println(cube1);
    }
}
```

Objects are created using the **new** operator, which calls a relevant constructor

Recall the Cube class, let's instantiate it!

```
Cube cube1 = new Cube("Purple", 5.0);
Cube cube2 = new Cube();
```

code inside MainCube.java

```
public Cube(){
        this.color = "White";
        this.length = 1.0;
   }
```

code inside Cube.java

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Recall the Cube class, let's instantiate it!

```
Cube cube2 = new Cube();
Cube cube1 = new Cube("Purple", 5.0);
```

code inside MainCube.java

```
public Cube(String color,
double length){
     this.color = color;
     this.length = length;
  }
```

code inside Cube.java

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Let's access the data fields, and call the methods.

```
public class MainCube{
   public static void main(String[] args){
      Cube cube1 = new Cube("Purple", 5.0);
      Cube cube2 = new Cube();

      System.out.println(cube1);
      System.out.println(cube1.length);
      System.out.println(cube1.color);
      System.out.println(cube1.getVolume());
   }
}
```

Access data fields

Calling a method using “()”

FAKULTAS ILMU KOMPUTER

Let's access the data fields, and call the methods.

```
public class MainCube{
   public static void main(String[] args){
      Cube cube1 = new Cube("Purple", 5.0);
      Cube cube2 = new Cube();

      System.out.println(cube1);
      System.out.println(cube1.length);
      System.out.println(cube1.color);
      System.out.println(cube1.getVolume());
   }
}
```

```
Cube with length = 5.00 and color = Purple
5.0
Purple
125.0
```

Output

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Special method: toString()

```
Cube cube1 = new Cube("Purple", 5.0);
Cube cube2 = new Cube();

System.out.println(cube1);
System.out.println(cube2);
```

code inside MainCube.java

```
public String toString(){
      return String.format("Cube with length = %.2f
and color = %s", this.length, this.color);
   }
```

code inside Cube.java

```
Cube with length = 5.00 and color = Purple
Cube with length = 1.00 and color = White
```

Output

FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Variables store references of objects

```
Cube cube1 = new Cube("Purple", 5.0);
Cube cubecopy = cube1;
cubecopy.color = "Red";
System.out.println(cube1);
System.out.println(cubecopy);
```

```
Cube with length = 5.00 and color = Red
Cube with length = 5.00 and color = Red
```

Output

FAKULTAS
ILMU
KOMPUTER

Guess The Output ?

```
Cube cube1 = new Cube("Purple", 5.0);
Cube cubecopy = cube1;
System.out.println(cubecopy);
cubecopy.color = "Red";
System.out.println(cube1);
System.out.println(cubecopy);
```

```
?
?
?
```

Output

Variables store references of objects

```
Cube cube1 = new Cube("Purple", 5.0);
Cube cube2 = new Cube("Purple", 5.0);
cube2.color = "Red";
System.out.println(cube1);
System.out.println(cube2);
```

```
Cube with length = 5.00 and color = Purple
Cube with length = 5.00 and color = Red
```

Output

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Null value

It's a special value, meaning "no object“ (does not reference any object).

```
String str = null;
```

You can print it, but..
don't you ever try accessing an attribute or invoke a method of null!

```
System.out.println(str.length());
```

```
Exception in thread "main" java.lang.NullPointerException
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Default Value for a Data Field

- **null** for a reference type
- **0** for a numeric type
- **false** for a boolean type
- **'\u0000'** for a char type.
- However, Java assigns no default value to a local variable inside a method

```
public class Test {
   public static void main(String[] args){
      int x; // x has no default value
      String y; // y has no default value
      System.out.println("x is " + x);
      System.out.println("y is " + y);
   }
}
```

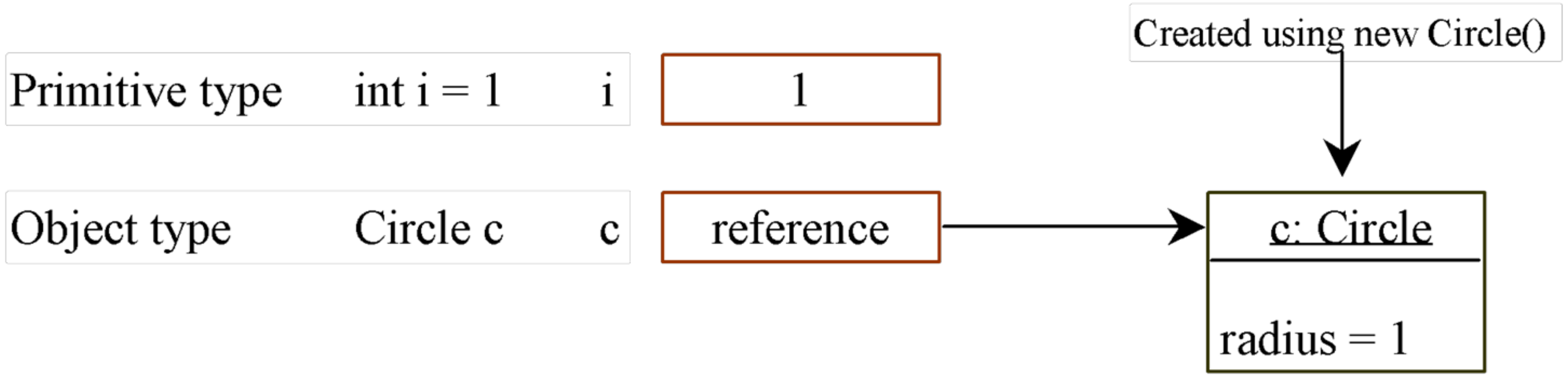
Guess What The Output ?

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

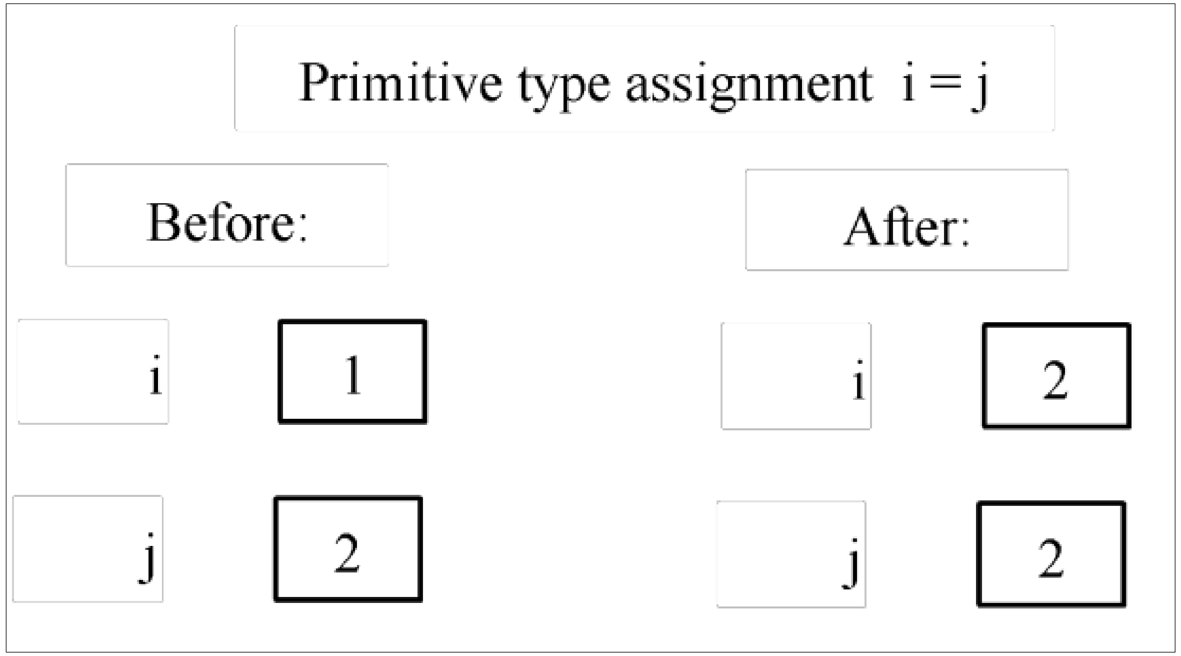
FAKULTAS ILMU KOMPUTER

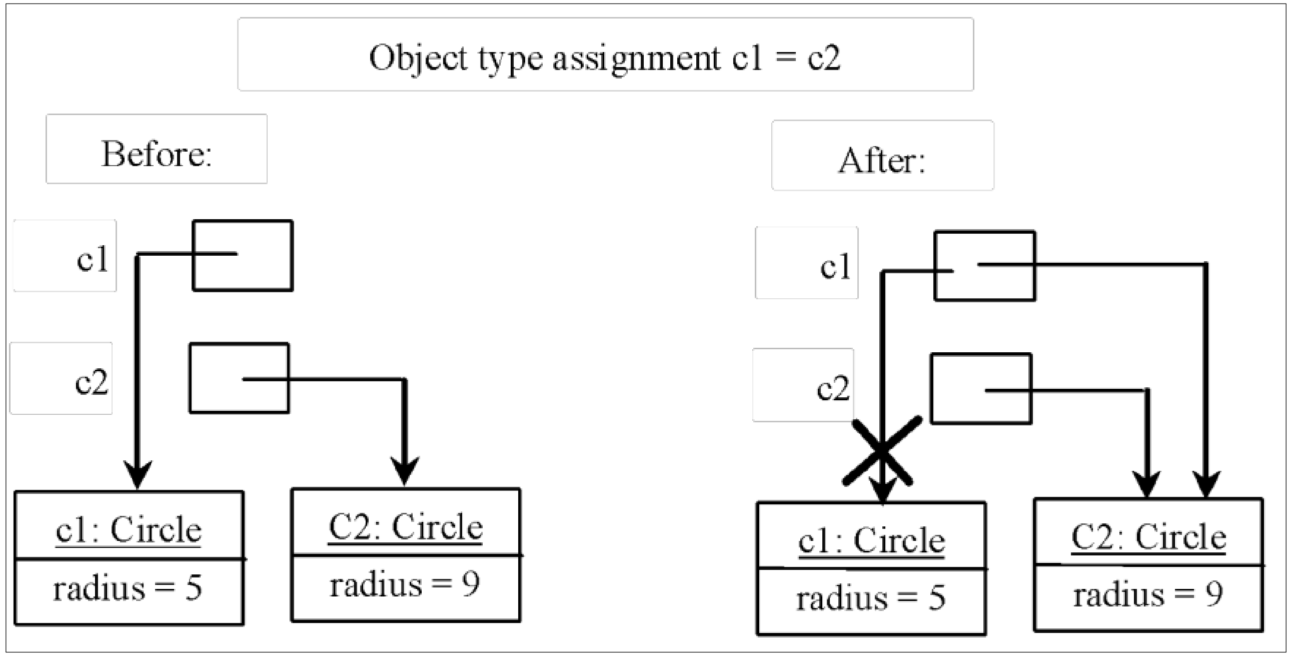
Primitive Data Types Vs Object Types

- Variabel dengan tipe data primitif menyimpan nilai data secara langsung,
- sedangkan variabel dengan tipe data reference menyimpan alamat penyimpanan objek tersebut di dalam heap memory.



Copying Variables of Primitive Data Types and Object Types





- The object previously referenced by c1 is no longer referenced (known as garbage). Garbage is automatically collected by JVM

FAKULTAS
ILMU
KOMPUTER

Real-world classes: String

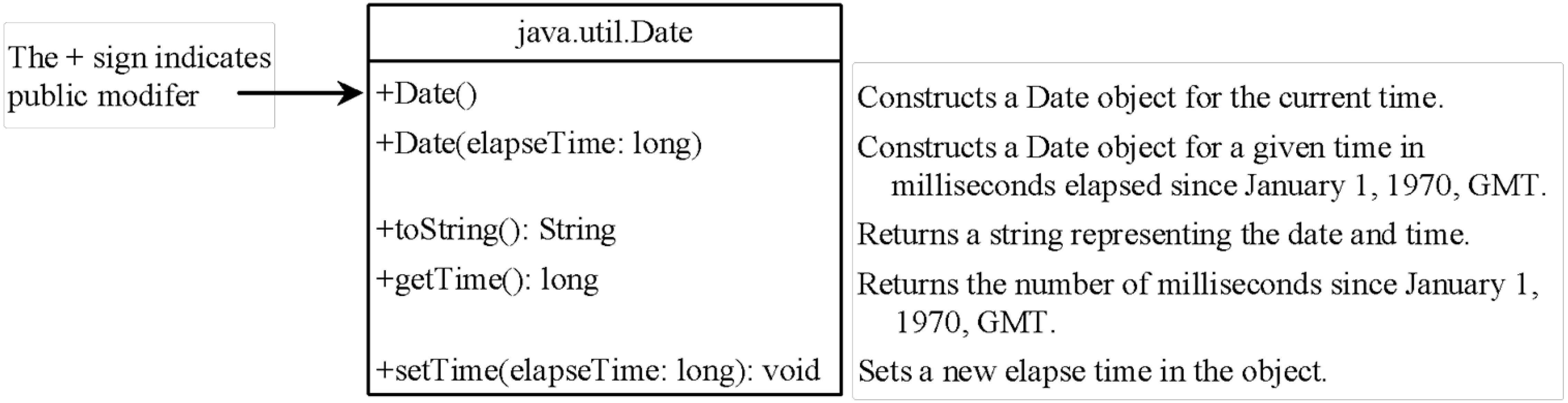
```
String csui = "Fasilkom UI";
System.out.println(csui.charAt(7));
System.out.println(csui.endsWith("UI"));
System.out.println(csui.indexOf("kom"));
System.out.println(csui.replaceAll("UI", "UB"));
System.out.println(csui.toUpperCase());
```

The Output

```
m
true
5
Fasilkom UB
FASILKOM UI
```

Real-world classes: `Date`

- You can use the **`Date`** class to create an instance for the current date and time and use its **`toString`** method to return the date and time as a string.


The + sign indicates public modifer
java.util.Date
+Date()
+Date(elapseTime: long)
+toString(): String
+getTime(): long
+setTime(elapseTime: long): void
Constructs a Date object for the current time.
Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
Returns a string representing the date and time.
Returns the number of milliseconds since January 1, 1970, GMT.
Sets a new elapse time in the object.

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA Veritas, Probitas, Iustitia

The Date Class Example

```
import java.util.Date;
```

```
Date date = new Date();
System.out.println(date.toString());
```

Output:

```
Sun Mar 09 13:50:19 EST 2003.
```

FAKULTAS
ILMU
KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Real-world classes: `Random`

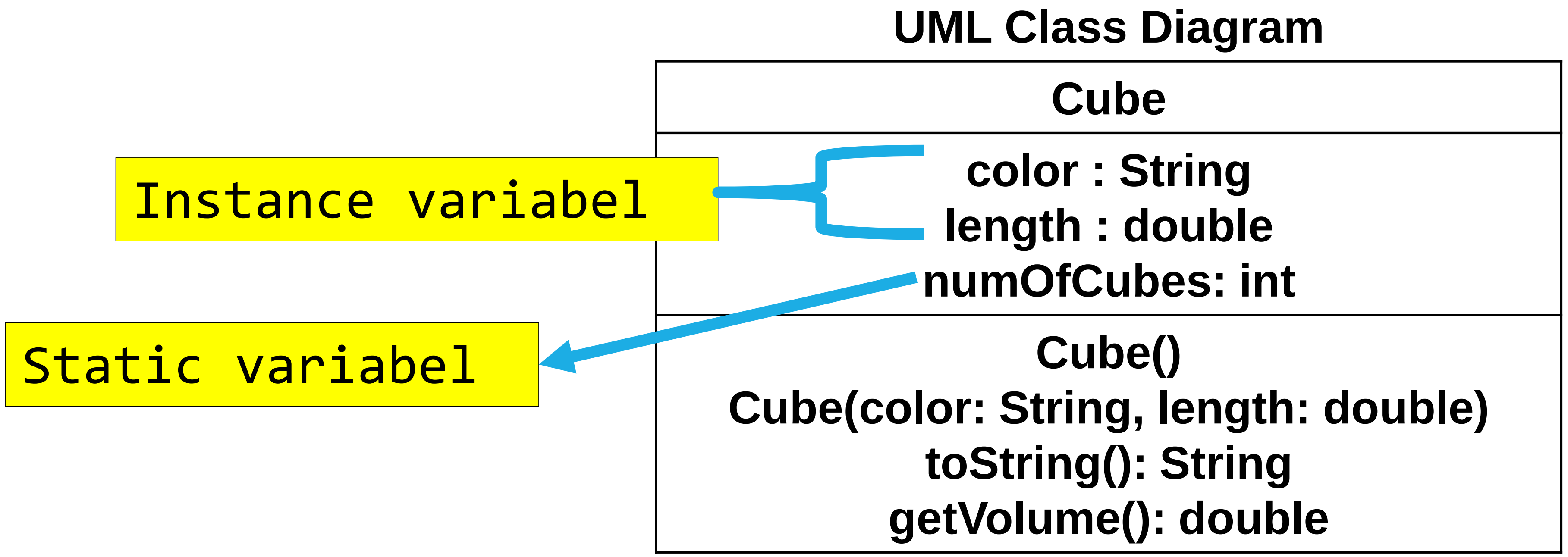
- Recall `Math.random()` !
- A more useful random number generator is provided in the `java.util.Random` class.

```
import java.util.Random;

public class GuessStarter {
   public static void main(String[] args) {
      // pick a random number from 1 to 100
      Random random = new Random();
      int number = random.nextInt(100) + 1;
      System.out.println(number);
   }
}
```

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Instance Variables, Static variables, Local Variables

- **Instance variables** belong to a specific instance (object).
- **Static variables** are shared by all objects of the class.
- **Static constants**: static variables that are final.
- **Local variables** are declared within a method.



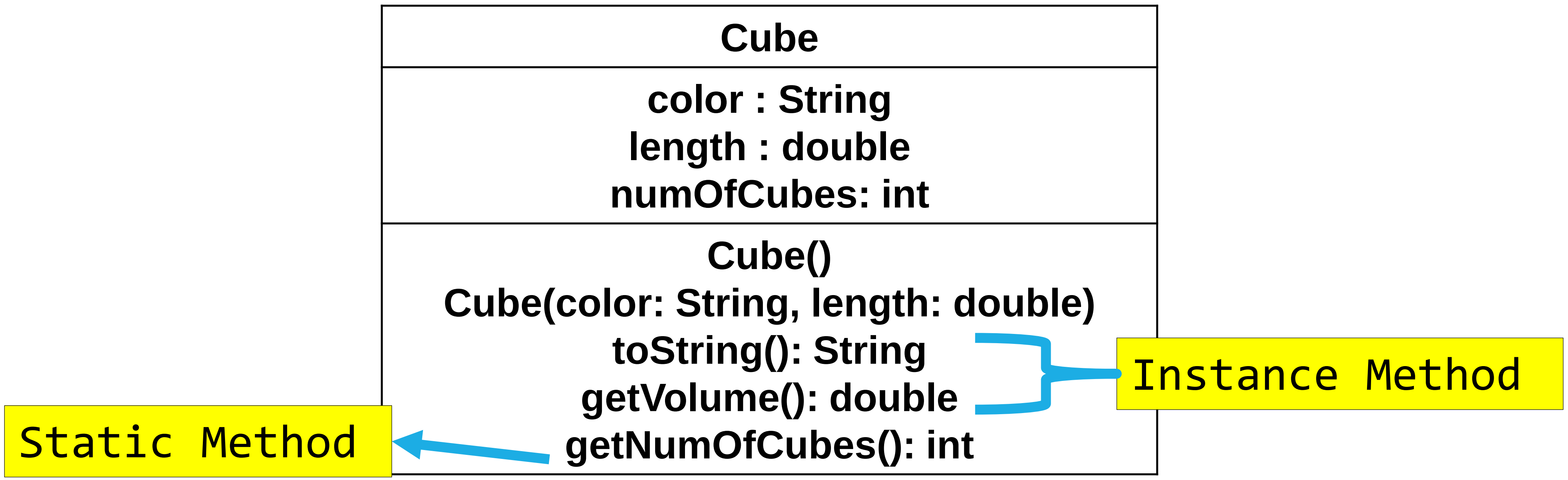
Scope of Variables

- The scope of *instance and static variables* is the **entire class**. They can be declared anywhere inside a class.
- The scope of a *local variable* **starts from its declaration and continues to the end of the block** that contains the variable. A local variable must be initialized explicitly before it can be used.

Instance methods, Static methods

- **Instance methods** are invoked by an instance of the class.
- **Static methods** are not tied to a specific object.

UML Class Diagram



Cube
color : String length : double numOfCubes: int
Cube() Cube(color: String, length: double) toString(): String getVolume(): double getNumOfCubes(): int

FAKULTAS ILMU KOMPUTER

Cube.java with static var and method

```
static int numOfCubes = 0; // add this variable
public Cube() {
   this.color = "White";
   this.length = 1.0;
   numOfCubes++; // add this line
}
public Cube(String color, double length) {
   this.color = color;
   this.length = length;
   numOfCubes++; // add this line
}
public static int getNumOfCubes() { // add this method
   return numOfCubes;
}
```

Edit the previous Cube.java accordingly

All variables appearing in a static method, must be static!

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Cube.java with static var and method

```java
System.out.println(Cube.getNumOfCubes());
Cube cube1 = new Cube();
Cube cube2 = new Cube("Blue", 4.0);
System.out.println(cube1.numOfCubes);
System.out.println(cube2.numOfCubes);
System.out.println(Cube.numOfCubes);
System.out.println(Cube.getNumOfCubes());
```

Call it in ‘main’ method

The Output

```
0
2
2
2
2
```

Quiz Time : Create a class of Time, storing hours (int), minutes (int), and seconds (double).

```
public class Time {
   //We declare the data fields
   int hour;
   int minute;
   double second;
}
```

Quiz Time : Create a class of Time, storing hours (int), minutes (int), and seconds (double).

```
public class Time {
   int hour;
   int minute;
   double second;

   //We create a constructor method
   public Time() {
      this.hour = 0;
      this.minute = 0;
      this.second = 0.0;
   }
}
```

this behaves like self in Python, it refers to the object we are creating

Quiz Time : Create a class of Time, storing hours (int), minutes (int), and seconds (double).

```
public class Time {
    int hour;
    int minute;
    double second;

    //We create a constructor method
    public Time() {
        this.hour = 0;
        this.minute = 0;
        this.second = 0.0;
    }
    //We create another constructor method
    public Time(int hour, int minute, double second) {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
    }
}
```

Quiz time: Given Time.java, what's the output?

```
Time t1 = new Time();
Time t2 = new Time(11, 30, 10.0);
System.out.println(t1.hour + ":" + t1.minute + ":" + t1.second);
System.out.println(t2.hour + ":" + t2.minute + ":" + t2.second);
```

Call it in ‘main’ method

The Output

```
0:0:0.0
11:30:10.0
```

Quiz Time : Now, make a toString method for Time.java!

```
    ... ... ...
    // inside Time.java
    public String toString() {
        return String.format("%02d:%02d:%04.1f",
        this.hour, this.minute, this.second);
    }
}
```

this always refers to the object that calls the **equals** method, the naming of **that**, however, is arbitrary.
You can replace **that** with **x, bla,** or **whatever**.

Quiz time: Given Time.java, what's the output?

```
Time t1 = new Time();
Time t2 = new Time(11, 30, 10.0);
System.out.println(t1);
System.out.println(t2);
```

Call it in 'main' method

The Output

```
00:00:00.0
11:30:10.0
```

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Quiz time: Given Time.java, what's the output?

```
Time t3 = new Time(1, 3, 2.1098);
System.out.println(t3);
```

Call it in 'main' method

The Output

```
01:03:02.1
```

equals method

- The == operator checks whether objects are identical; that is, whether they are the **same object** (= **same memory location**).
- The equals method checks whether they are equivalent; that is, whether they have the same **value**.
- **mean** by "**same**" in the **same value**? We define the **equals** method for our objects.

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Now, make the equals method for Time.java!

```
    ... ... ...
    // inside Time.java
    public boolean equals(Time that) {
       return this.hour == that.hour
          && this.minute == that.minute
          && this.second == that.second;
    }
}
```

Every object type has a method called **toString** that returns a string representation of the object. When you display an object using print or println, Java invokes the object's toString method

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Quiz time: Given Time.java, what's the output?

```
Time t1 = new Time(11, 30, 10.0);
Time t2 = new Time(11, 30, 10.0);
Time t3 = new Time(1, 10, 8.1);
System.out.println(t1 == t2);
System.out.println(t1.equals(t2));
System.out.println(t1 == t3);
System.out.println(t1.equals(t3));
```

Call it in ‘main’ method

The Output

```
false
true
false
false
```

Visibility modifiers

- Data fields inside a class are too "open".
- By default, they can be accessed by any class in the same package.
- We need **information hiding**: a way to control what can be accessed from outside, and what cannot.

```
public class Time {
   int hour;
   int minute;
   double second;
}
```

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Visibility Modifiers

- `public`
  The class, data, or method is **visible to any class in any package**.
- `default` (no modifier)
  The data or method is **visible to any class within a package**.
- `private`
  The data or methods can be accessed **only by the declaring class**.
- `protected`
  The class, data, or method is **visible to any its subclasses or any class within a package**.

Why Data Fields Should Be private?

- To protect data.
- To make code easy to maintain.

FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

Visibility Modifiers

```
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

Karena Berbeda Package maka modifier tipe **`default`** tidak dapat diakses

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Visibility Modifiers

```
package p1;

class C1 {
   ...
}
```

```
package p1;

public class C2 {
   can access C1
}
```

```
package p2;

public class C3 {
   cannot access C1;
   can access C2;
}
```

Karena Class Berbeda Package maka modifier tipe **`default`** tidak dapat diakses

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Visibility Modifiers in Time class

- Solution: Add **private** to the data fields.

```
public class Time {
   int hour;
   int minute;
   double second;
}
```

```
public class Time {
   private int hour;
   private int minute;
   private double second;
}
```

Visibility Modifiers in Time class

```
public class Time {
   private int hour;
   private int minute;
   private double second;
}
```

- Can you now **access** those **variables outside** the **Time class**?
- We can control what to access by providing:
  - **Getter** methods
  - **Setter** methods

Getters and setters

- Recall that the data fields (instance variables) of Time are private. We can access them from within the Time class, but if we try to access them from another class, the compiler generates an error.
- For example, here's a new class called TimeClient, trying to access the private data fields:

```
public class TimeClient {
   public static void main(String[] args) {
      Time time = new Time(11, 59, 59.9);
      System.out.println(time.hour); // compile error
   }
}
```

Getters (accessor)

```
    // inside Time.java
    public int getHour() {
        return this.hour;
    }
    public int getMinute() {
        return this.minute;
    }
    public double getSecond() {
        return this.second;
    }
}
```

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Setters (mutator)

```
    // inside Time.java
    public void setHour(int hour) {
    this.hour = hour;
    }
    public void setMinute(int minute) {
    this.minute = minute;
    }
    public void setSecond(int second) {
    this.second = second;
    }
}
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Calling Setter and Getter Method in 'main' method

```
Time t1 = new Time(11, 30, 10.0);
System.out.println(t1.getSecond());
System.out.println(t1.getHour());
t1.setMinute(50);
System.out.println(t1.getMinute());
```

The Output

```
10.0
11
50
```

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

Veritas, Probitas, Iustitia

Recall our Cube.java

- This is the original UML. Now suppose we want the data fields to be private, and all the methods to be public, what would change?

UML Class Diagram

Cube
color : String **length : double**

Cube()
Cube(color: String, length: double)
toString(): String
getVolume(): double

Recall our Cube.java

- This is the original UML. Now suppose we want the data fields to be private, and all the methods to be public, what would change?

UML Class Diagram

Cube
- color : String **- length : double**

+ Cube()
+ Cube(color: String, length: double)
+ toString(): String
+ getVolume(): double

- sign indicates **private**
+ sign indicates **public**

Calling Setter and Getter Method in 'main' method

```
Time t1 = new Time(11, 30, 10.0);
System.out.println(t1.getSecond());
System.out.println(t1.getHour());
t1.setMinute(50);
System.out.println(t1.getMinute());
```

The Output

```
10.0
11
50
```

TIP to Make OOP

- Defining a class creates a new object type.
- Every object belongs to a certain object type.
- A class definition is like a template for objects, it specifies:
  - what **attributes** the objects have; and
  - what **methods** can operate on them.
- The **new** operator creates new instances of a class.
- Think of a class like a blueprint for a house: you can use the same blueprint to build any number of houses.