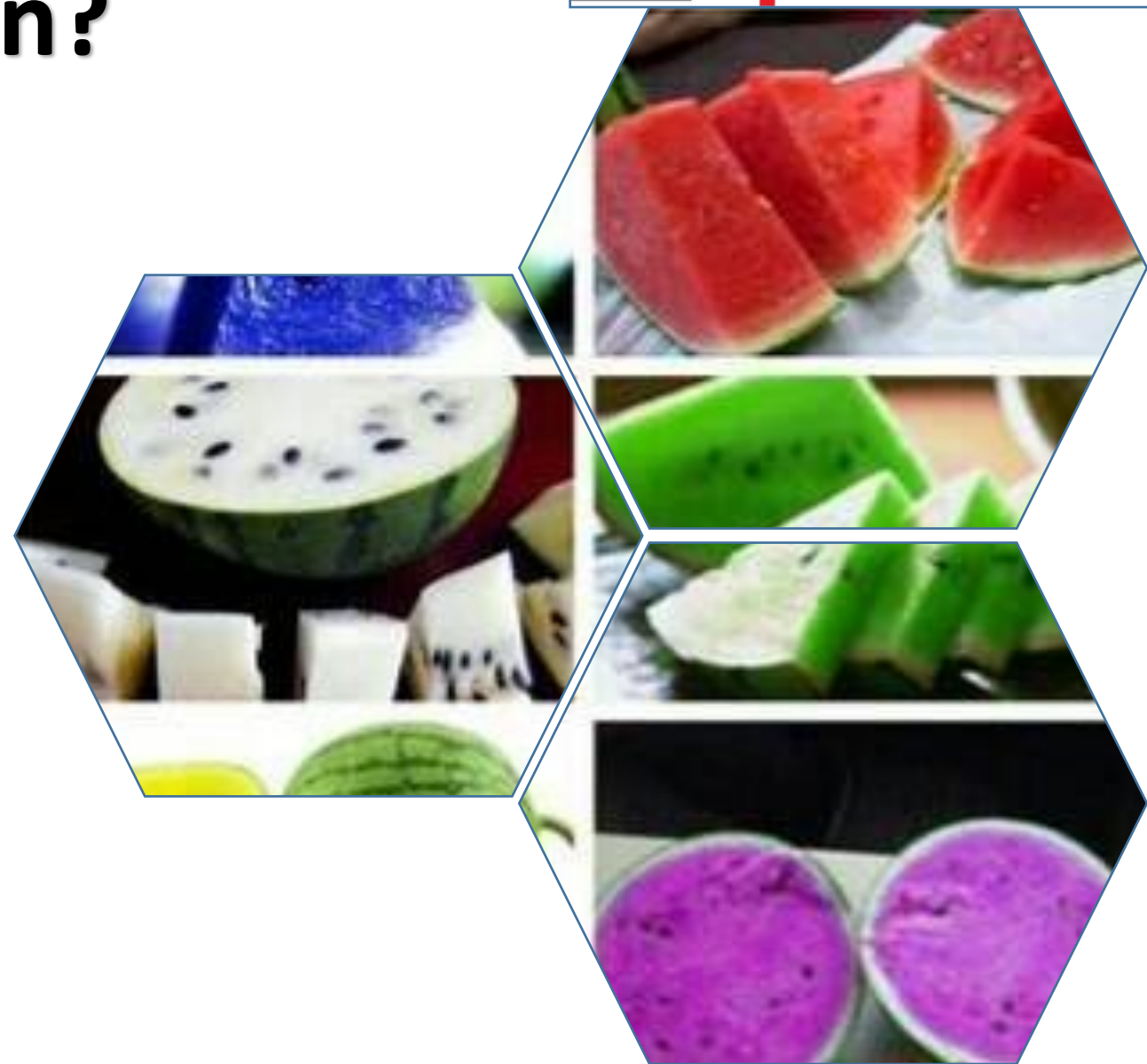
Objects and Classes

CSGE601021 Dasar-Dasar Pemrograman 2

Fakultas Ilmu Komputer Universitas Indonesia

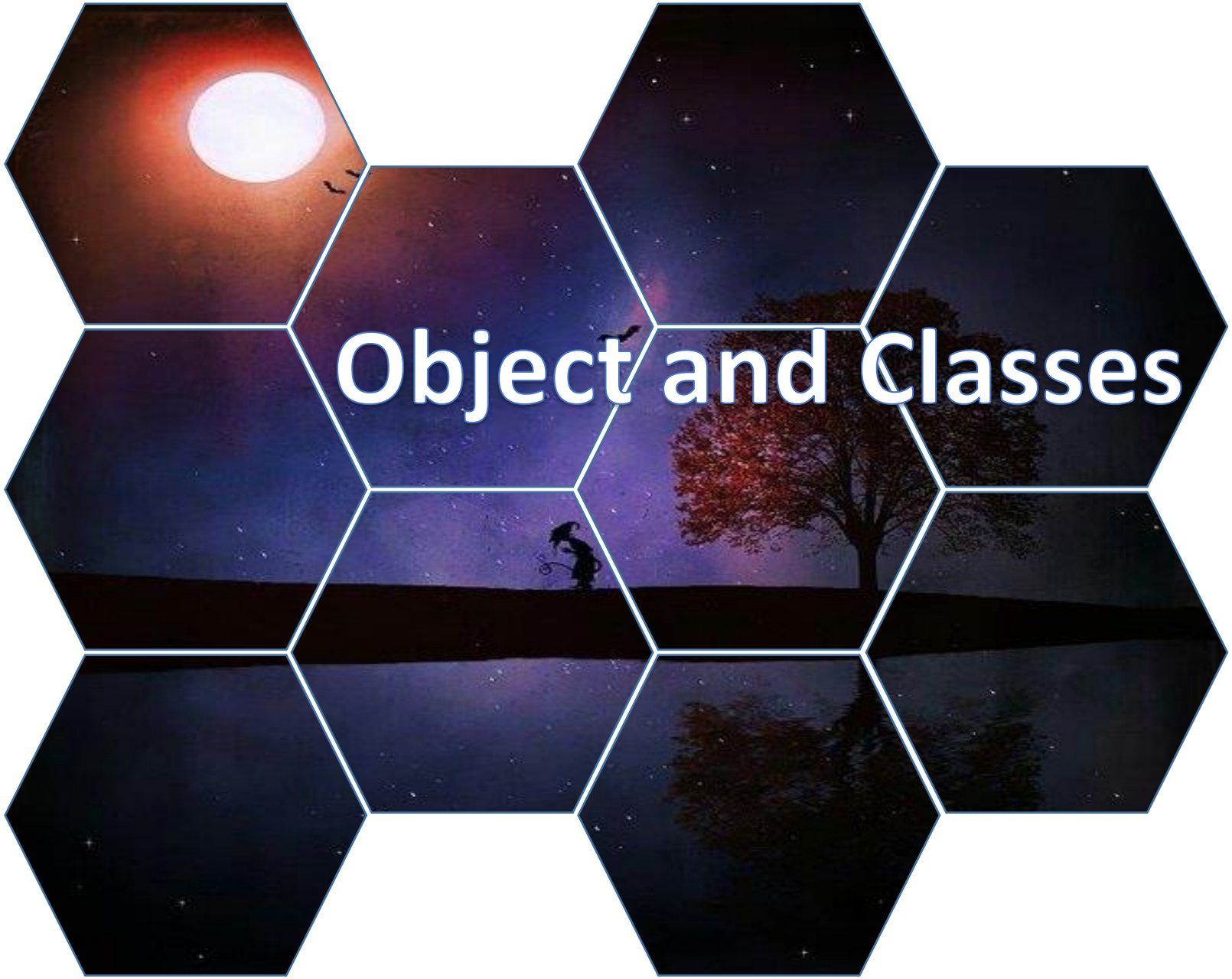
Do you like watermelon?

What properties they have?
what colors do they have?
And….
how much weight they have?



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);
```



Object and Classes

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**.

we can create Classes for Any Object

We can Create **Watermelon** with **class**

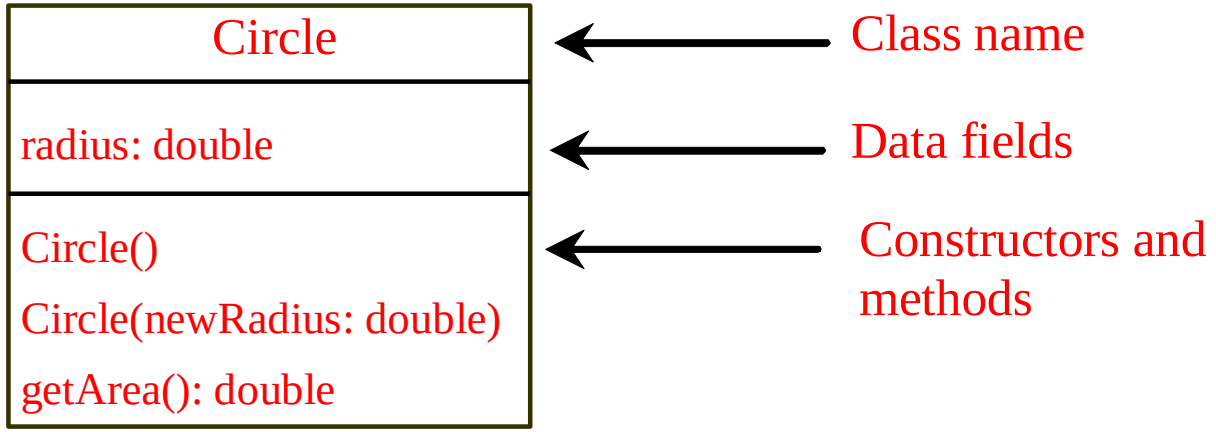
```
public class Watermelon{
    String color;
    double heavy;

    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);

    }
}
```

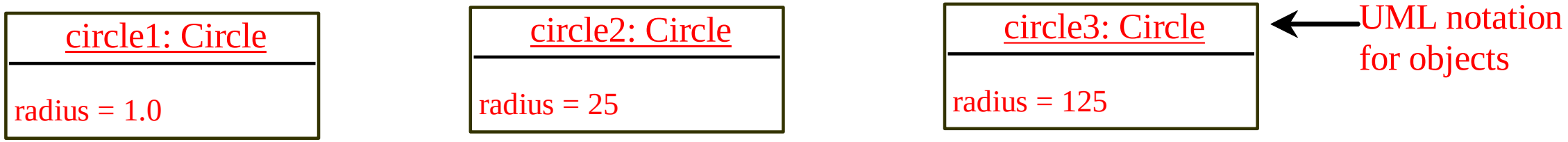
Much Simple, isn't it?

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

Unified Modelling Language (UML) Class Diagram

UML Class Diagram





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

Class : Circle

What does it **have**?
- radius

What does it **behave**?
- get area

UML class Diagram

Circle	
radius : double	**Data field**
Circle() Circle(radius : double)	**Constructors**
getArea(): double	**Method**

Class : Circle

```
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

Constructors

Method

Constructors

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

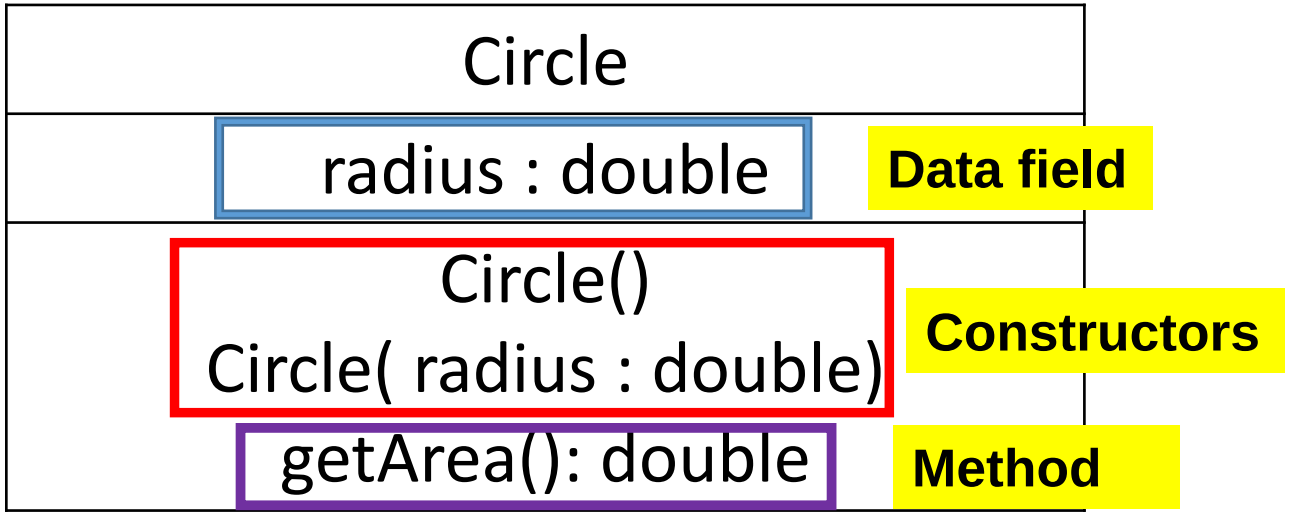
- a special kind of methods that are invoked to construct objects.
- must have the same name as the class itself.
- do not have a return type—not even void.
- invoked using the new operator when an object is created. Constructors play the role of initializing objects.

```
Circle() {
}

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

Creating Objects Using Constructors

Data Field: Identity/Properties of the objects.
what properties does circle have?
Methods: behaviors of the objects.
what can a circle do?

UML class Diagram




```
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

Constructors

Method

Creating Objects Using Constructors

```
className objectVar = new ClassName();
Circle circle1 = new Circle();
Circle circle1 = new Circle(5.0);
```

Class is a blueprint to create object

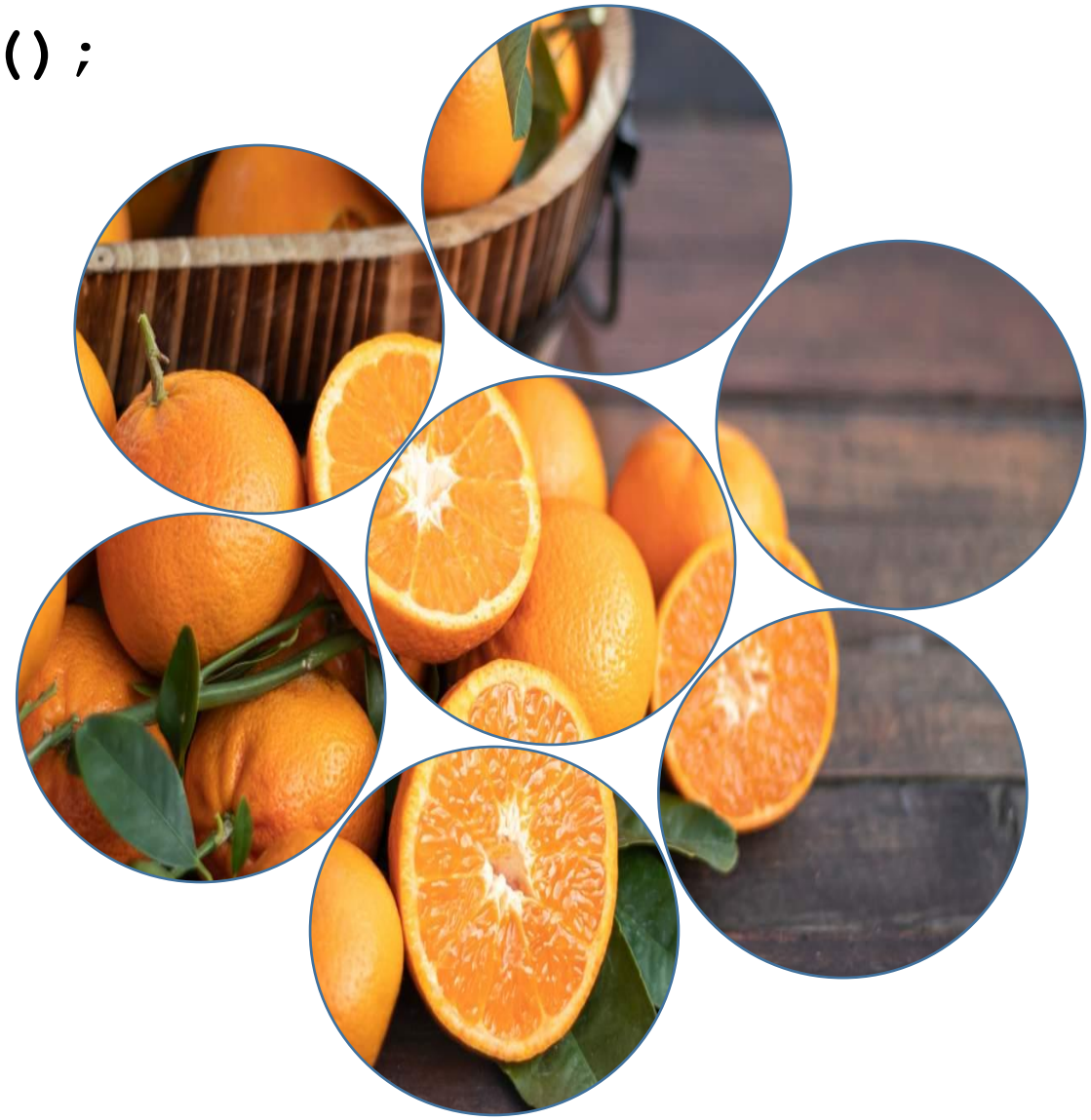
UML class Diagram

Circle
radius : double
Circle() Circle(radius : double) getArea(): double

Data field

Constructors

Method



Creating Objects Using Constructors

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

UML object

circle1 : Circle
radius : 1.0

circle2 : Circle
radius : 5.0

UML class Diagram

Circle
radius : double
Circle() Circle(radius : double)
getArea(): double

Data field

Constructors

Method

Circle object instantiation

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

```
Circle circle1 = new Circle();
Circle circle1 = new Circle(5.0);
```

```
Circle() {
}
```

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

Accessing object data and methods

Referencing the object's **data fields:**

objectVar.data

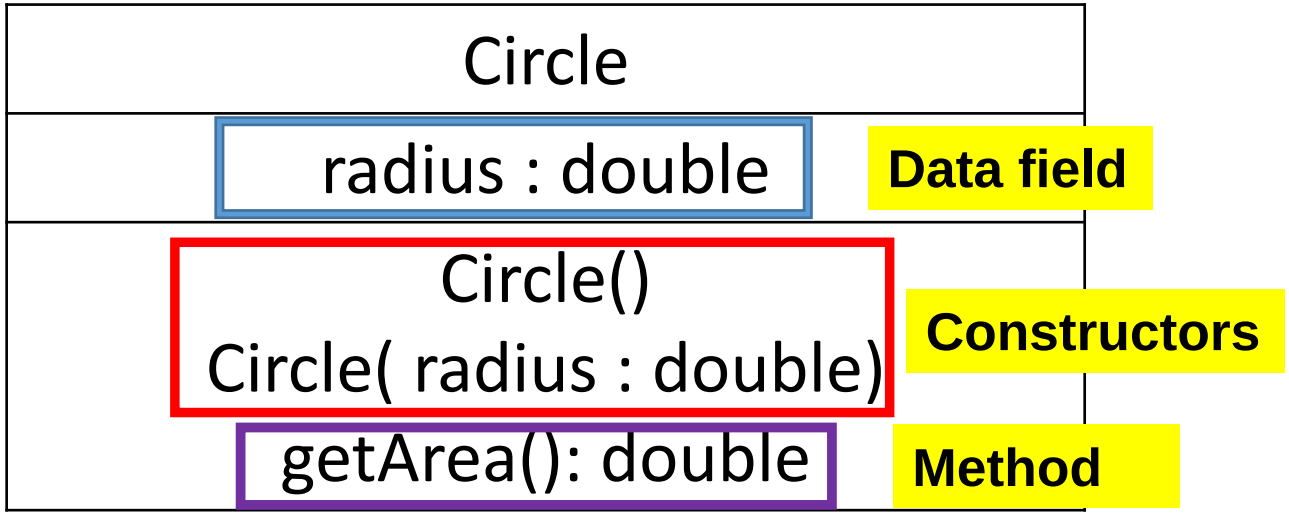
ex: circle1.radius

Invoking the object's **method:**

objectVar.methodName(args)

e.g., circle1.getArea()

UML class Diagram




```
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

Constructors

Method

Recall: Default Values of Variables

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);
  }
}
```

Compile error: variable not initialized

Default Value for Data Field

For Data Fields of Objects:

- **null** for a reference type
- **0** for a numeric type
- **false** for a boolean type
- **'\u0000'** for a char type.

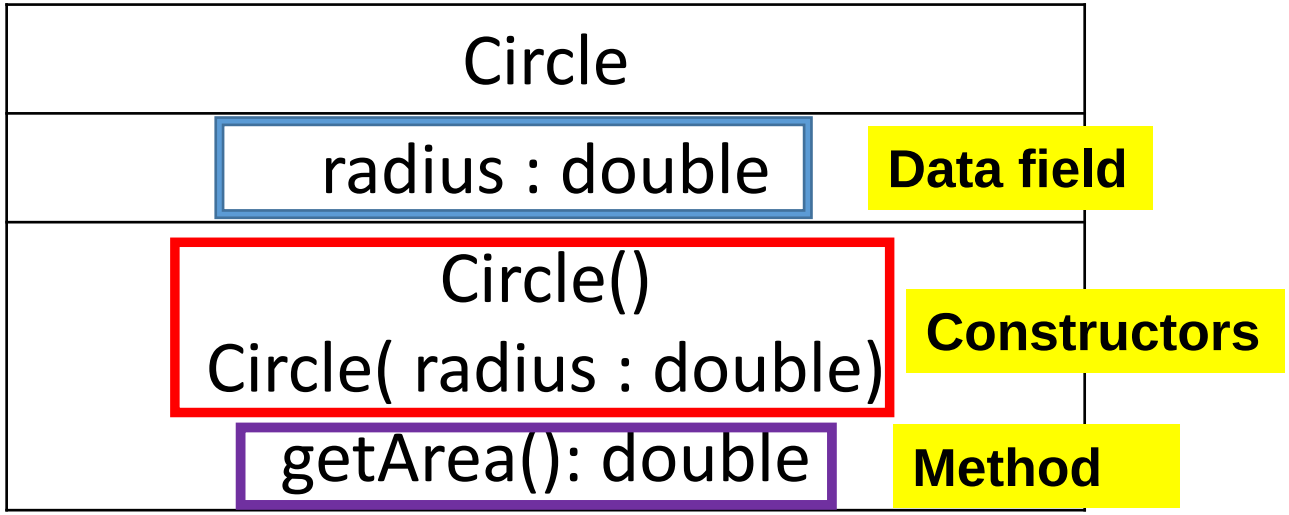
```
class Circle {
  /** The radius of this circle */
  double radius = 1.0;

  /** 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

Constructors

Method

UML class Diagram



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.NullPointer Exception

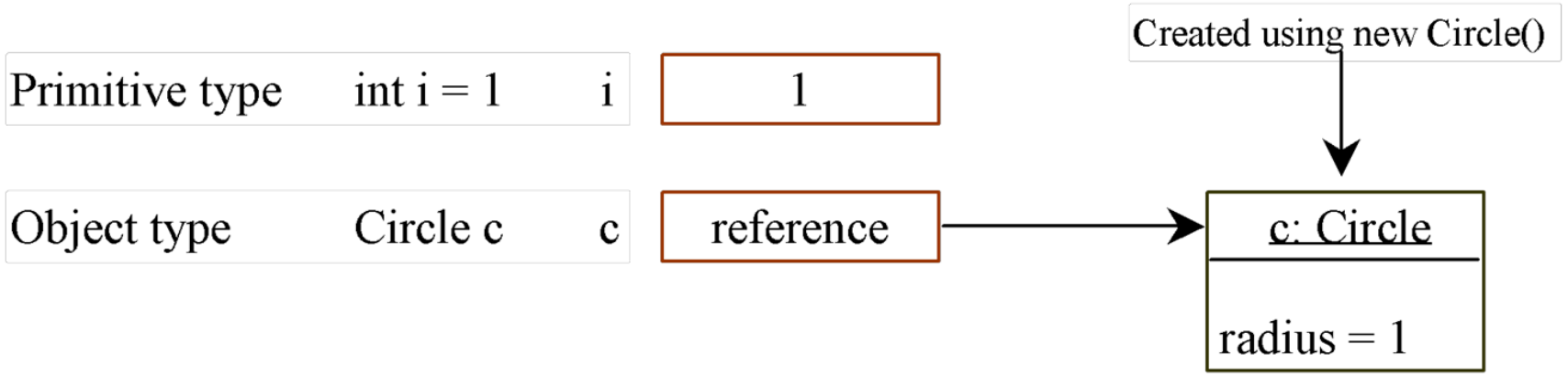
Reference data fields

The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

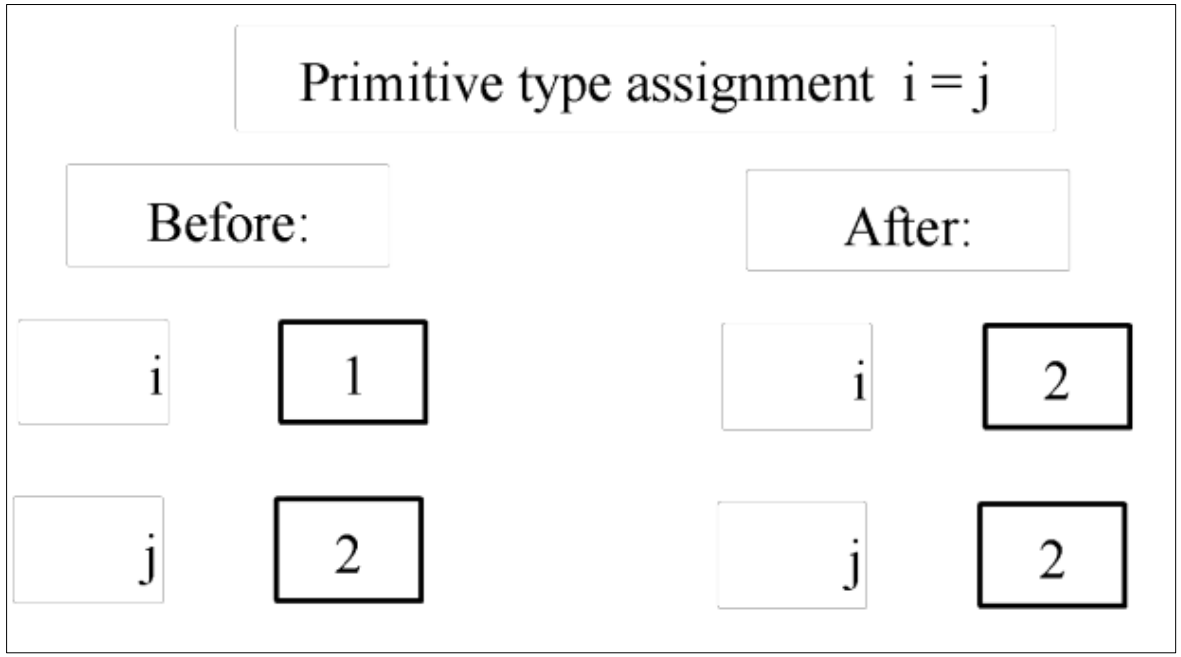
```
public class Cube {
   String color;          // name has default value null
   double length;         // radius has default value 0
   boolean isEmpty;       // isEmpty has default value false
   char label;            // label has default value '\u0000'
}
```

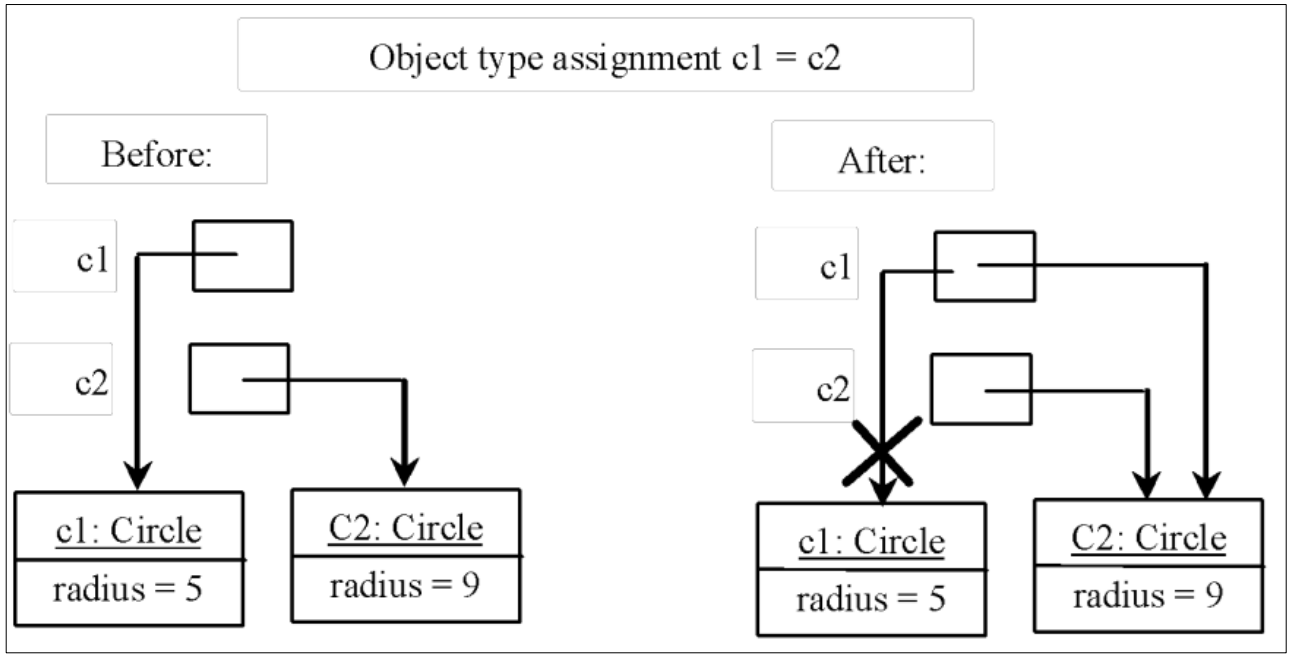
Primitive Data Types Vs Object Types

- 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

Reference data fields

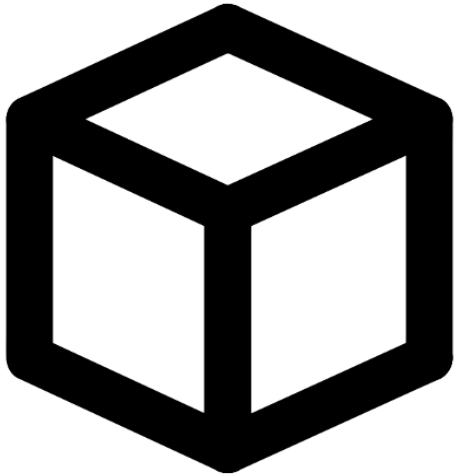
The data fields can be of reference types. For example, the following Student class contains a data field name of the String type.

That means that data fields of an object can be other objects as well!

```
public class Cube {
   String color;            // name has default value null
   double length;           // radius has default value 0
   boolean isEmpty;         // isEmpty has default value false
   char label;              // label has default value '\u0000'
   Circle shapeInCube;      // shapeInCube has default value null
}
```

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) getVolume(): double

Data fields

Constructors

Methods

Objects of Cube

UML Class Diagram

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

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) getVolume(): double

From UML become Code. Let's Do it.

```
    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) getVolume(): double

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

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

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 "()"

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());
      }
}
```

Output

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

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

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

```
?
?
?
```

`this` Keyword

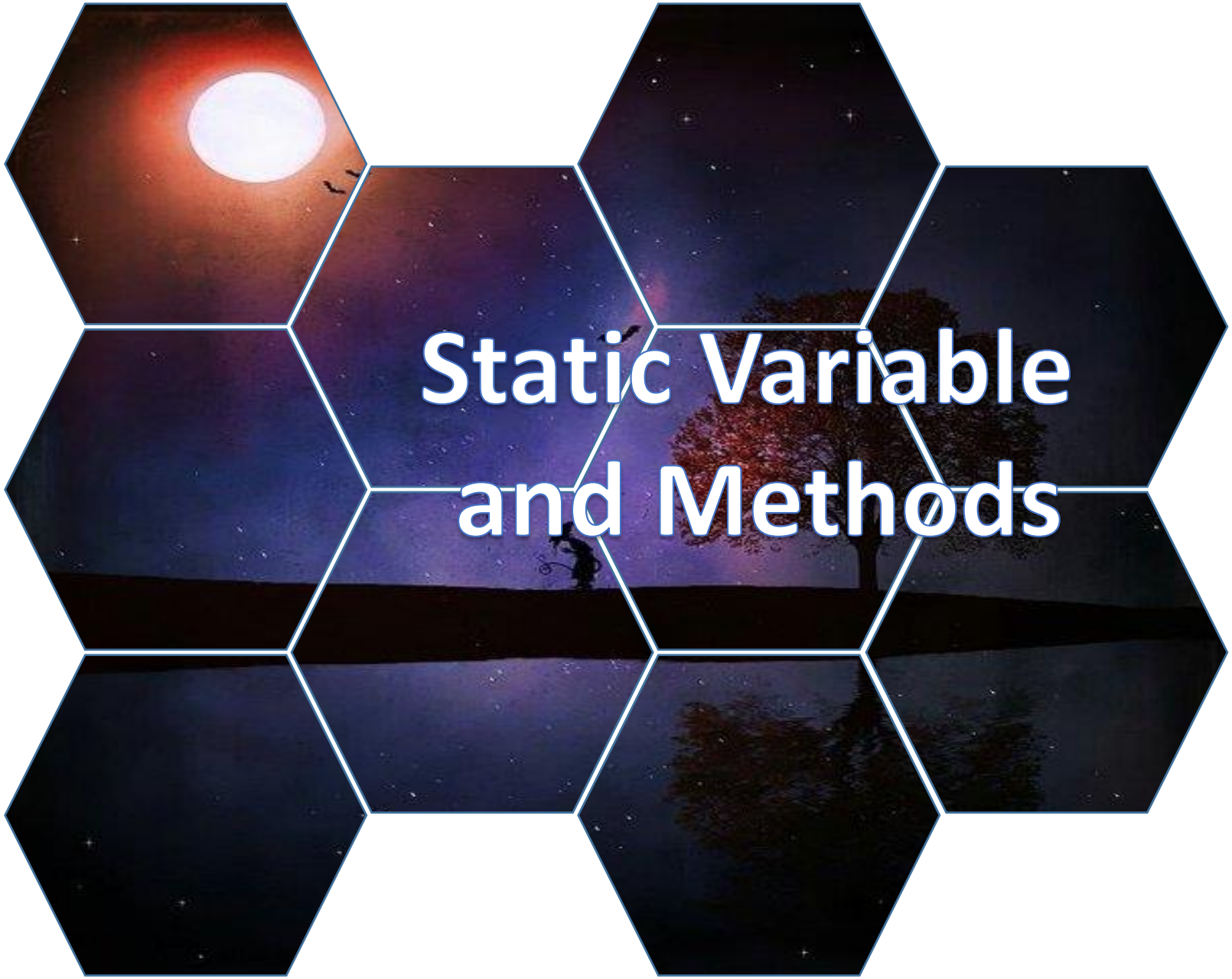
- `this` keyword refers to an object itself.
- One common use of the this keyword is reference a class's *data fields*.

```
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;
      }
}
```



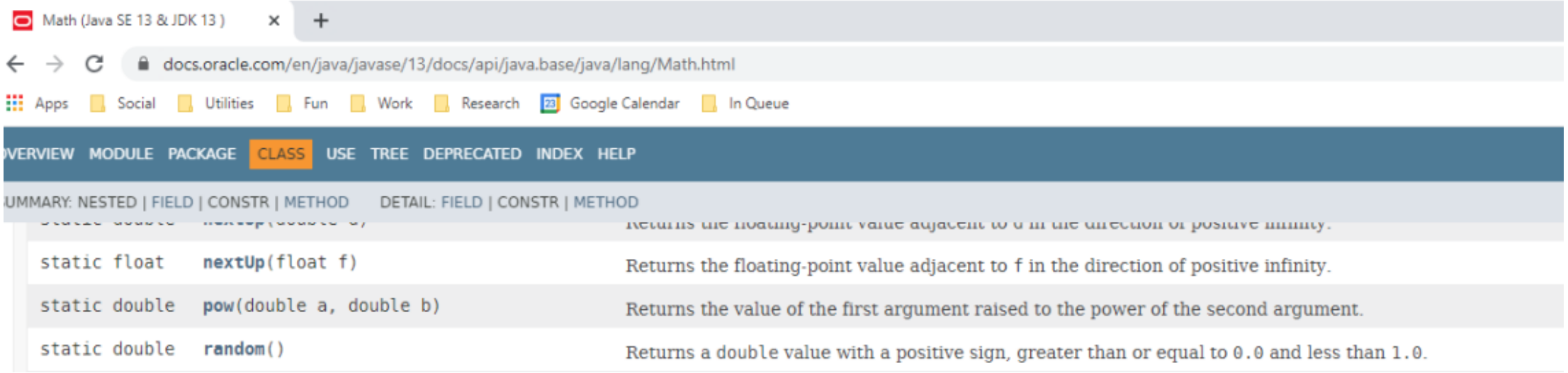
Static Variable and Methods

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Static and Non Static 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.

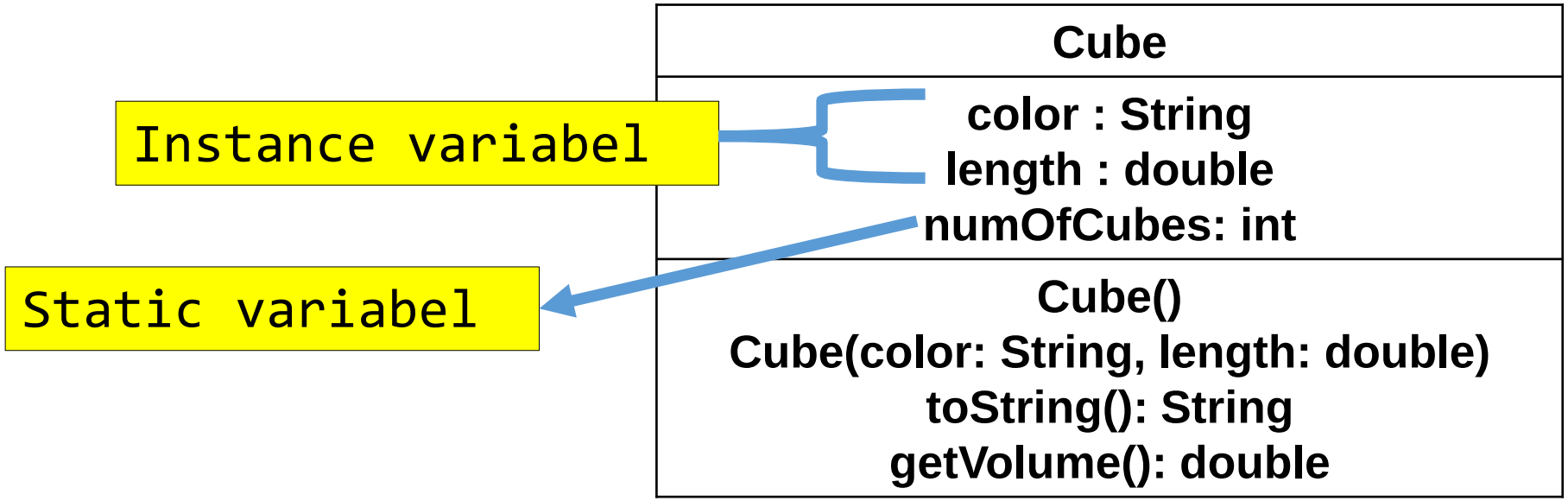
- Recall that you can use Math.pow(3,2.5) to invoke the method pow() from the Math class (without an instance).
- Math has static methods, which are defined using the static keyword.



Static and Non Static 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.

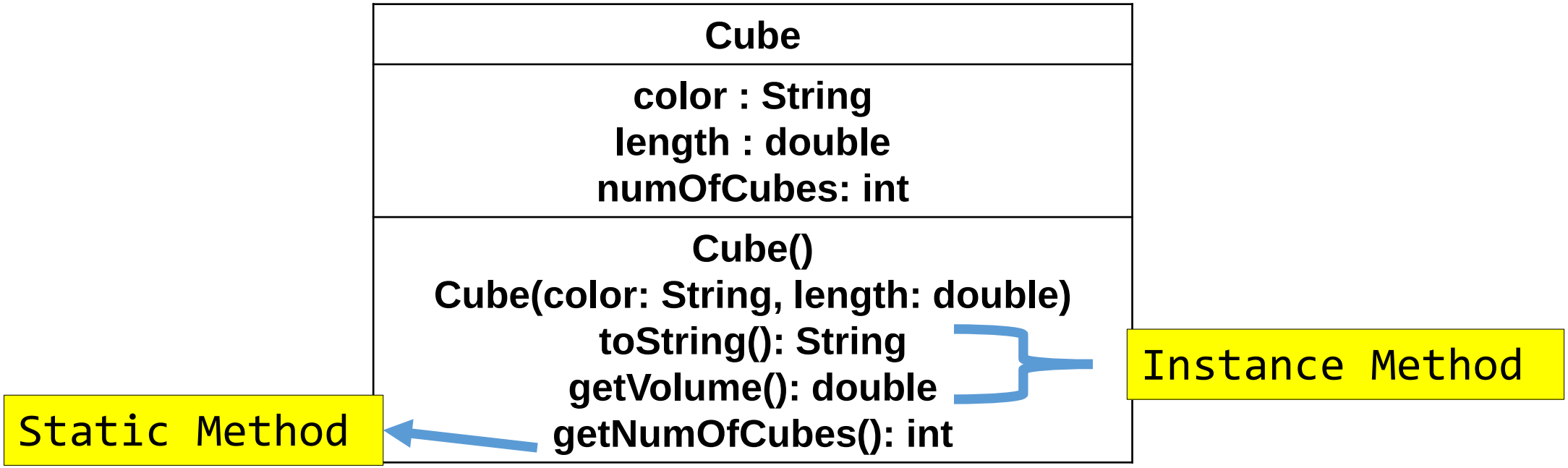
UML Class Diagram



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



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.

UML Class Diagram

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

When to Use a Static Variable

- If the value depends on each individual object → instance
  ex: each cube has its own length, it will differ for each object
- If the value doesn't depend on each individual object → static
  ex: sin, cos methods of class Math doesn't depend on any instance of math

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!

Cube.java with static var and method

```
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
```

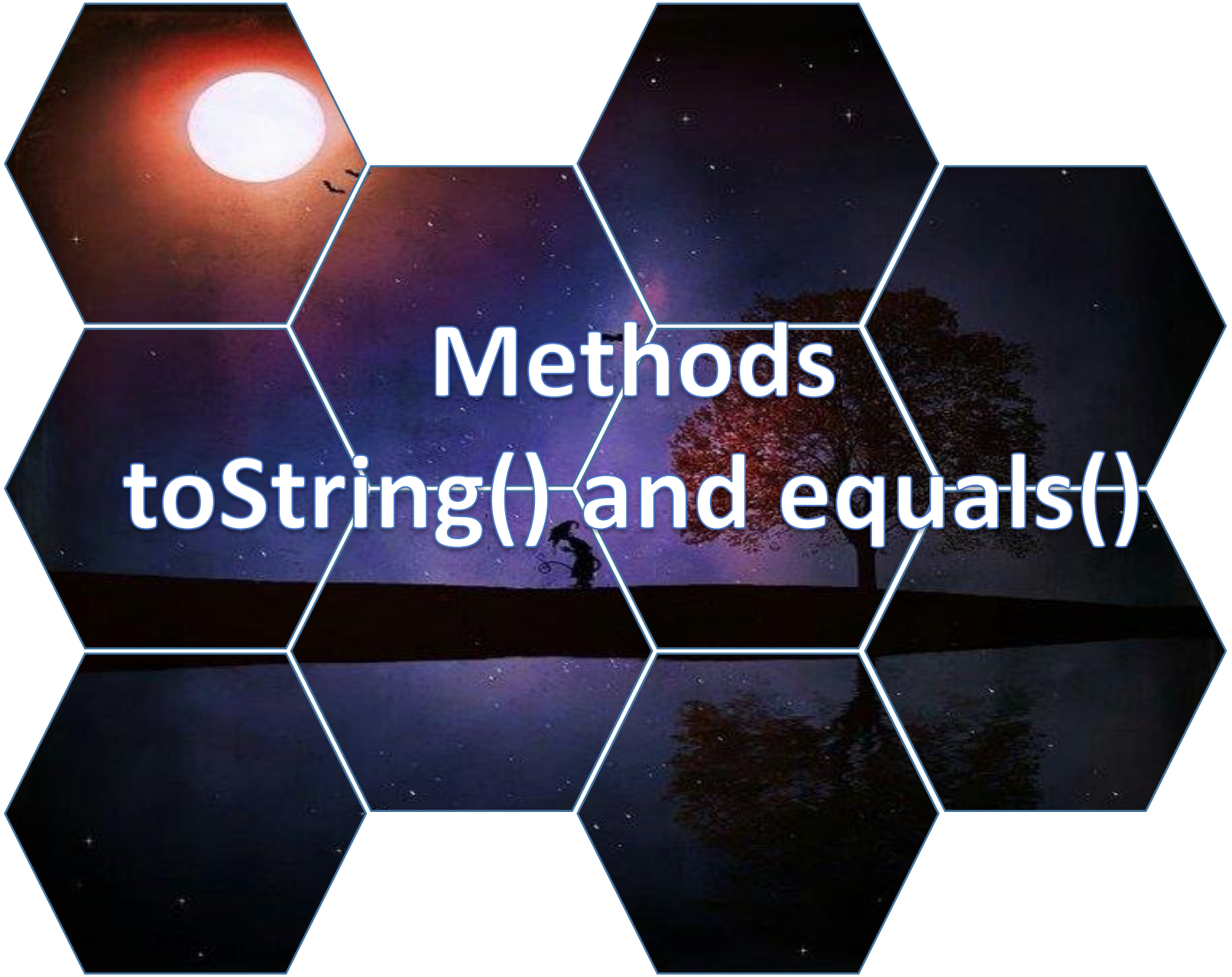
Cube.java with static constant

```
static final String CUBE_MATERIAL = "Silver";
```

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

The Output

```
Silver
Silver
Silver
```



Methods
toString() and equals()

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Printing Objects

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

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

Methods

Guess The Output ?

Output:
Cube@15db9742
Cube@6d06d69c

Printing objects calls the toString() method!

The default method prints out the identity hashcode of the object.

To String Method

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

Methods

UML Class Diagram

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

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

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.

Equals Method

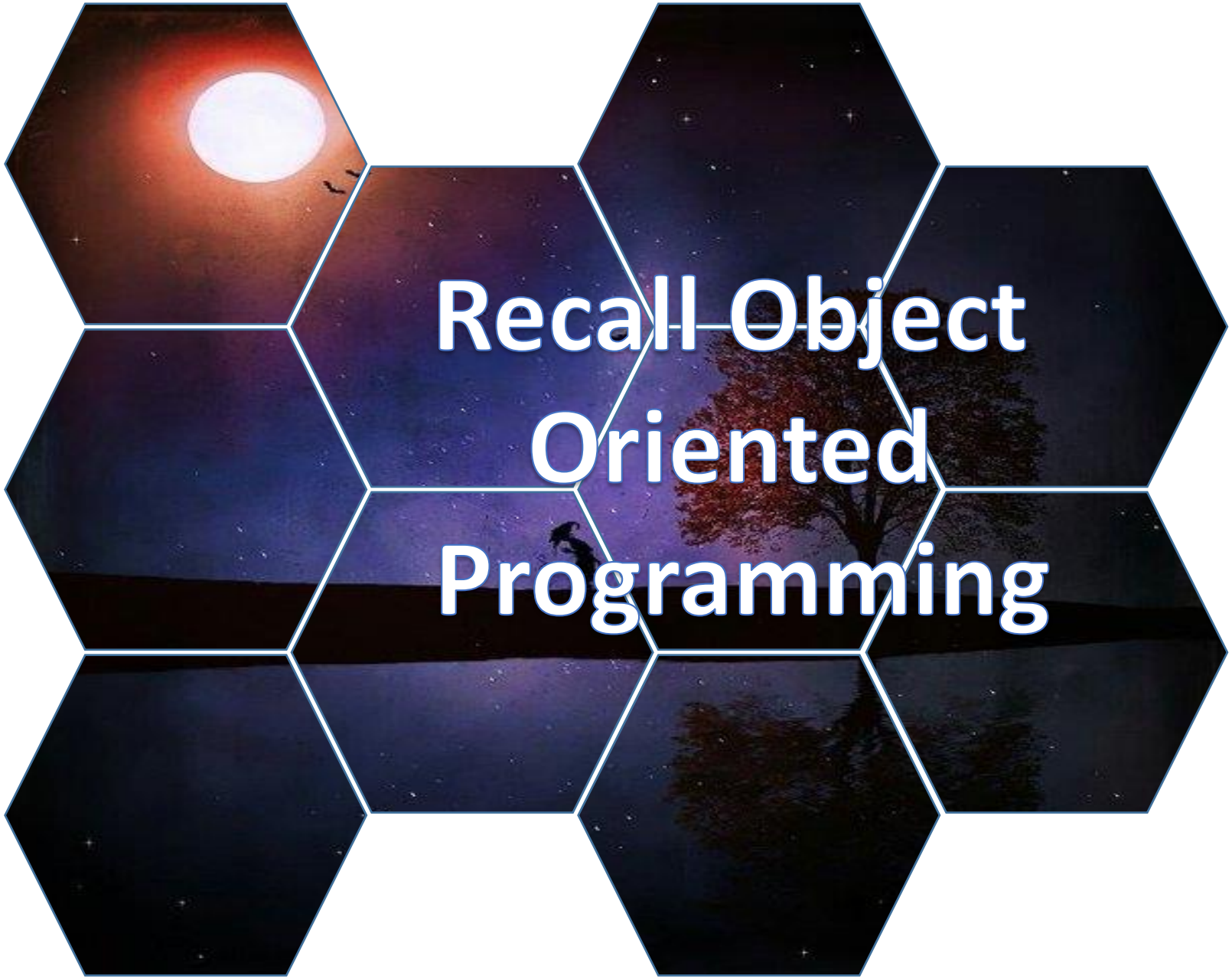
```
public boolean equals(Cube otherCube) {
      return this.color ==
      otherCube.color && this.length ==
      otherCube.length;
}
// adding this method in class of Cube
```

Methods

We consider 2 Cube objects equal when they have the same color and length.
You can define equal as you wish!

UML Class Diagram

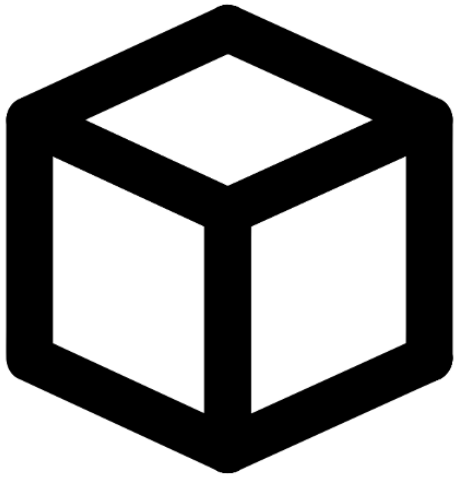
Cube
color : String length : double
Cube() Cube(color: String, length: double) toString(): String getVolume(): double equals(otherCube : Cube): Boolean



Recall Object Oriented Programming

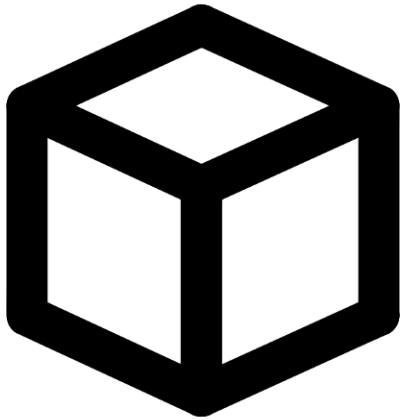
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER



UML Class Diagram

Cube	
color : String length : double	Data fields
Cube() Cube(color: String, length: double)	Constructors
getVolume(): double toString(): String	Methods



UML Class Diagram

Cube	
color : String length : double	Data fields
Cube() Cube(color: String, length: double)	Constructors
getVolume(): double toString(): String	Methods

- **Data Fields**
- **Constructors**
  - No-args/default
  - With Args
- **Methods**
  - toString()
  - Equals()
- **This Keyword**
- **Static and Non-Static**
- **Visibility Modifier**
  - Getter
  - Setter

Visibility Modifier

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FAKULTAS ILMU KOMPUTER

Visibility Modifier

Data Fields

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

UML Class Diagram

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

- 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.

Visibility Modifier

- `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**.

Visibility Modifiers

```java
package p1;

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

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

```java
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();
  }
}
```

```java
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

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;
}
```

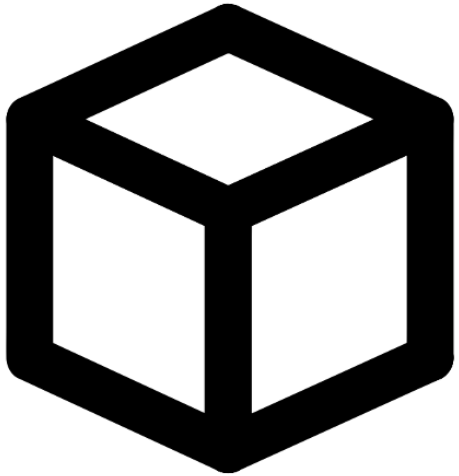
Karena Class Berbeda Package maka modifier tipe **`default`** tidak dapat diakses

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

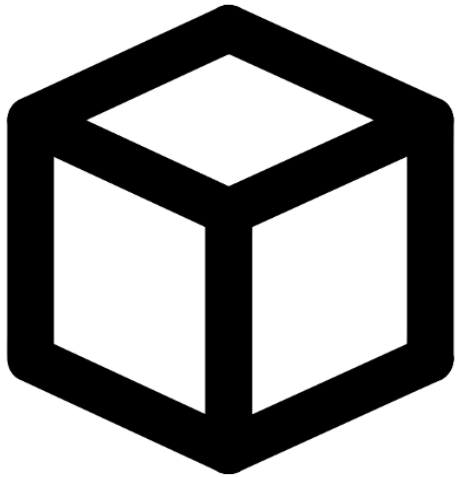


Recall our Cube.java

- 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**

Why Data Fields Should Be private?

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

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

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

When data field become private

- Now you can't access those variables outside the Cube class.

```
public class CubeMaker {
   public static void main(String[] args) {
      Cube myCube = new Cube("red", 2.0);
      System.out.println(myCube.color);
   }
}
```

Compile Error!

- We can control what to access by providing:
  - **Getter** methods
  - **Setter** methods

Access using Setter and Getter

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

```
// inside Cube.java
public String getColor() {
      return this.color;
}
public double getLength() {
      return this.length;
}
```

Getter or Accessors

```
// inside Cube.java
public void setColor(String color) {
      this.color = color;
}
public void setLength(double length){
      this.length = length;
}
```

Setter or Mutators

Calling Setter and Getter Method in 'main' method

```
Cube cube1 = new Cube("Red", 10.0);
System.out.println(cube1.getColor());
System.out.println(cube1.getLength());
cube1.setColor("Blue");
System.out.println(cube1.getColor());
```

The Output

```
Red
10.0
Blue
```

Let’s Practice

Time: UML Class Diagram

UML Class Diagram

Time
hour : int minute : int second : double
Time() Time(hour : int, minute : int, second : double) toString() equals()

Object of Time

Time : T1
hour : 0 minute : 0 second : 0.0

Time : T2
hour : 1 minute : 25 second : 30.0

Time : T3
hour : 2 minute : 25 second : 30.0

Time: Objects and Classes

Now let’s Make the code. ☺

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;
}
```

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;
      }
}
```

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;
        }
}
```

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);
      }
}
```

Time: 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;
        }
}
```

Time: Create Objects from class Time.java

```
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
```

Time: UML Class Diagram

UML Class Diagram

```
Time
- hour: int
- minute: int
- second: double
Time()
Time(hour: int, minute: int, second: double)
+ toString(): String
+ equals(): double
+ getHour: int
+ getMinute: int
+ getSecond: double
+ setHour: void
+ setMinute: void
+ setSecond: void
```

Object of Time

Time : T1
hour : 0 minute : 0 second : 0.0

Time : T2
hour : 1 minute : 25 second : 30.0

Time : T3
hour : 2 minute : 25 second : 30.0

Time: Objects and Classes

```
Now Please you Make the code. ☺
```

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.