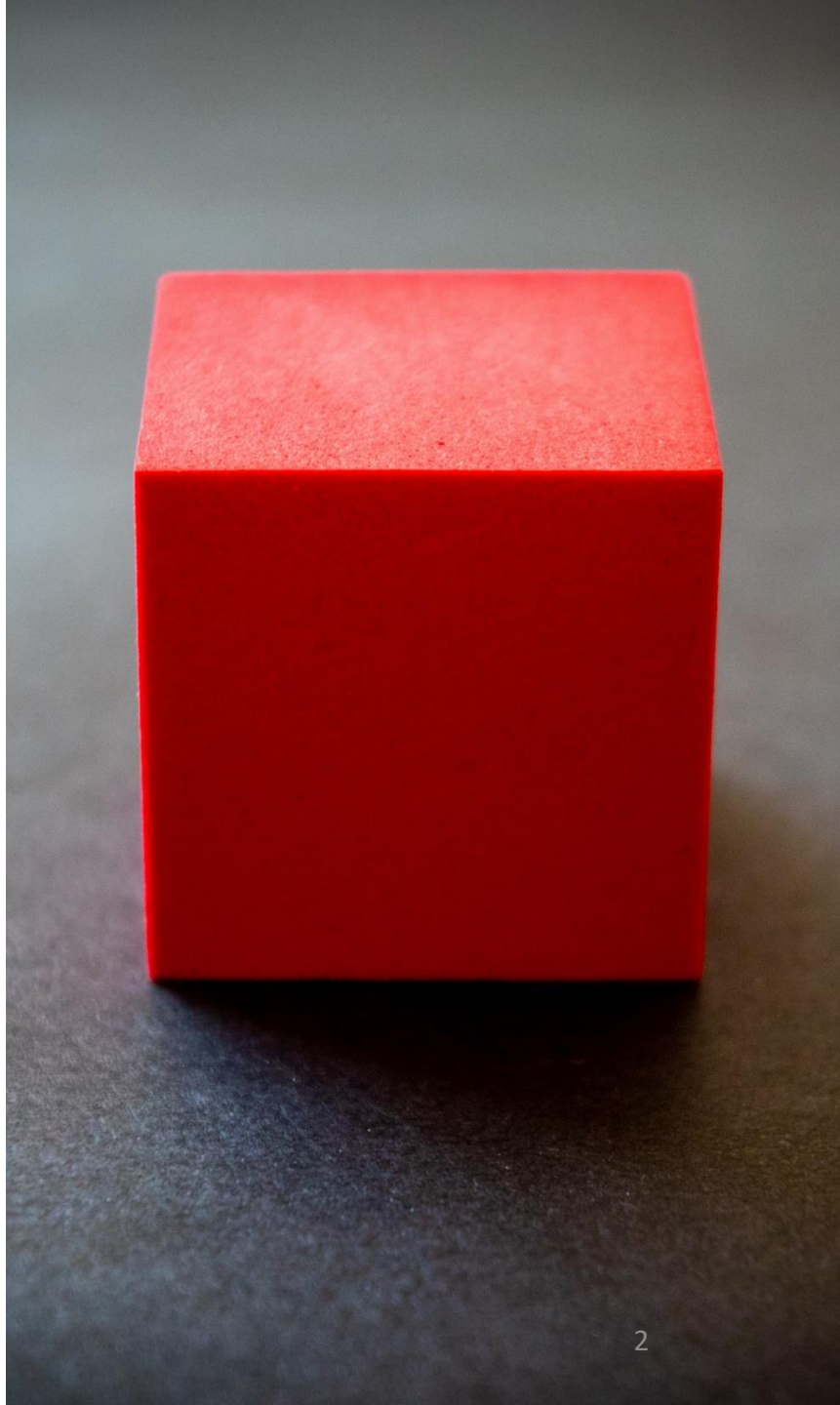
Dasar-Dasar Pemrograman 2: Pemrograman Berbasis Objek

Tim DDP 2 Fasilkom UI
Correspondence: Fariz Darari (fariz@cs.ui.ac.id)

UNIVERSITAS INDONESIA
FAKULTAS ILMU KOMPUTER

Why?

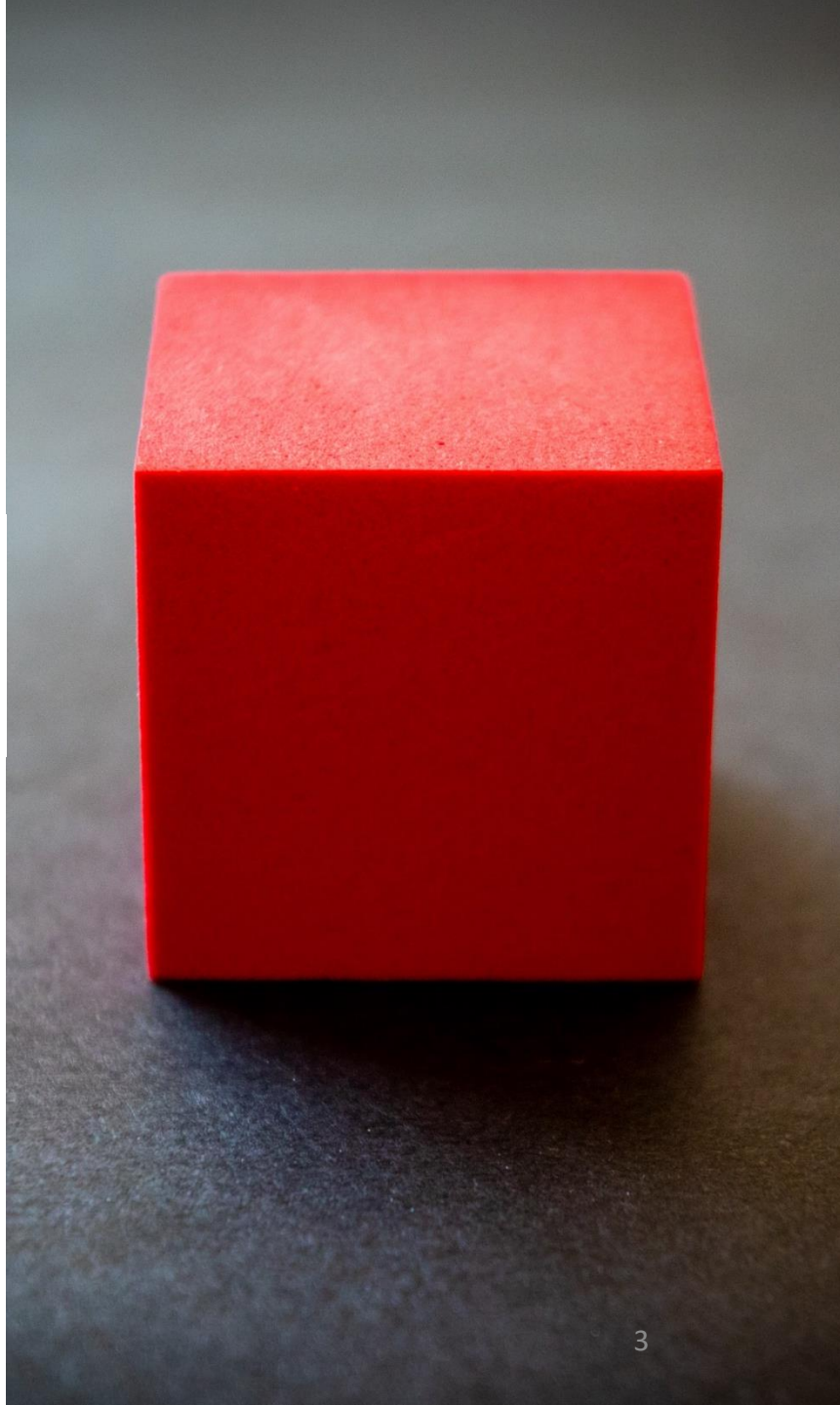
Try this one: Create **cubes**, without using classes.



Why?

Try this one: Create **cubes**, without using classes.

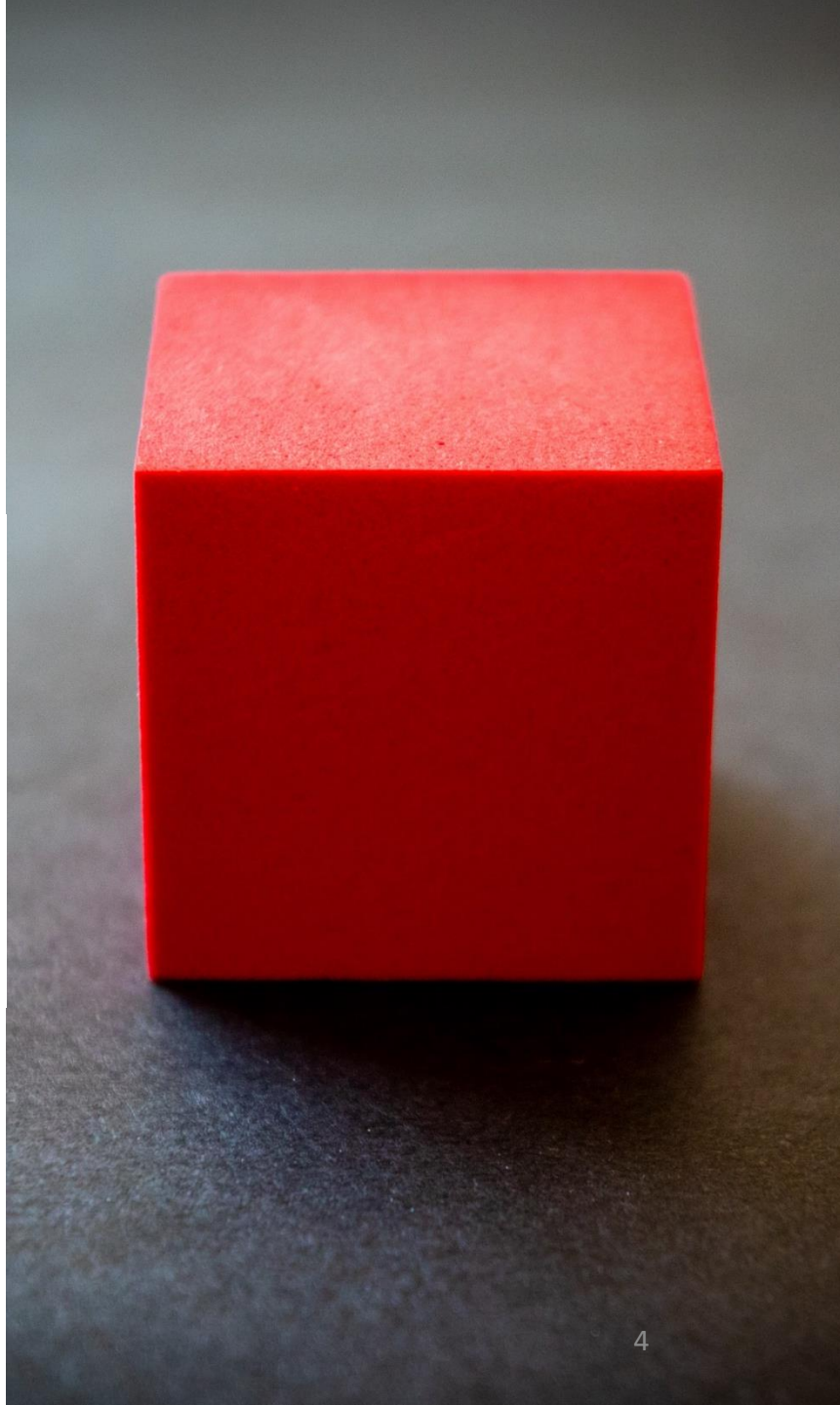
```
// create a cube
double cube1 = 3.0;

// create another cube
double cube2 = 4.0;
```



Why?

Try this one: Create **cubes**, without using classes.

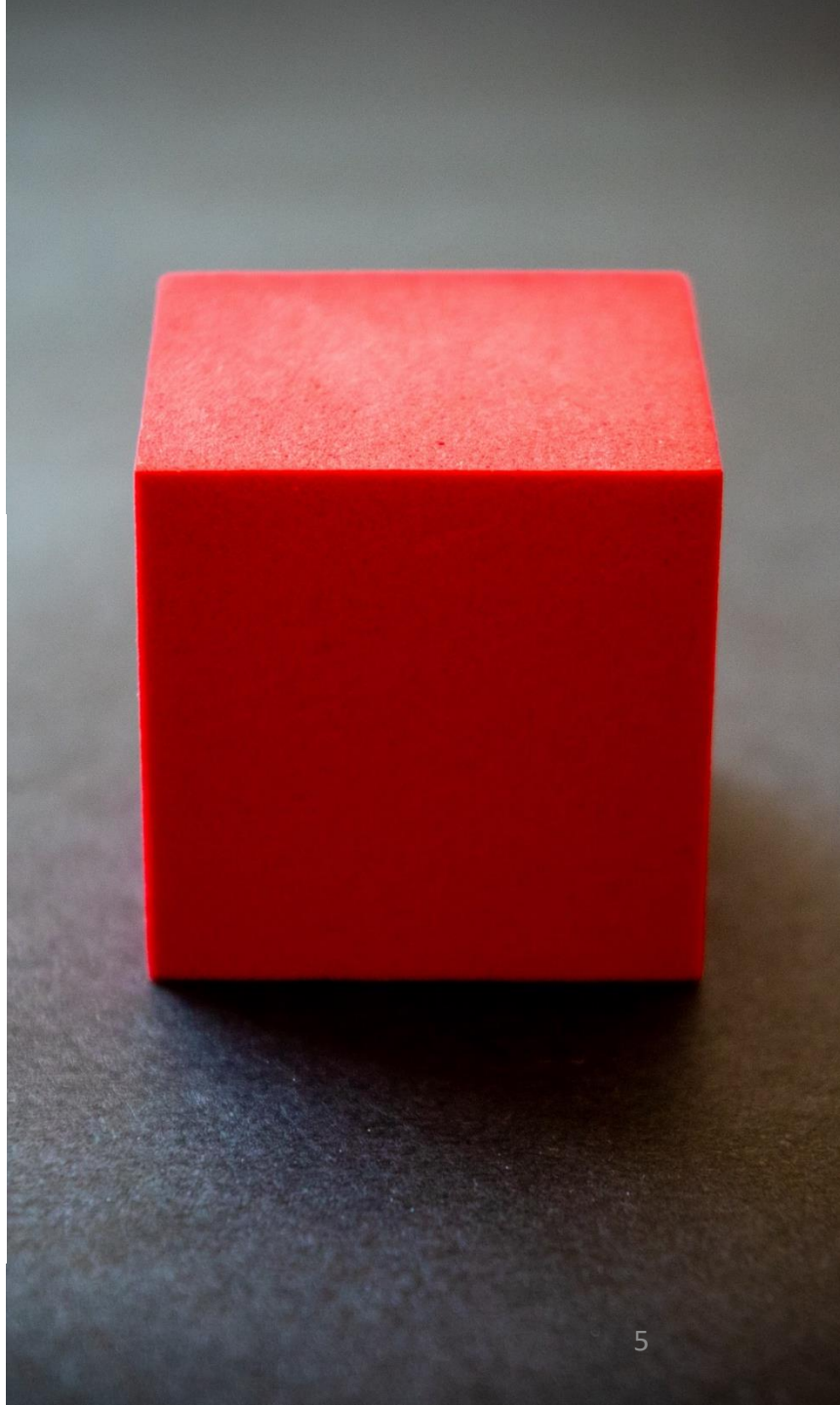
```
// create a cube
double cube1 = 3.0;

// create another cube
double cube2 = 4.0;

// set color for cube1
String cube1Color = "Red";

// set color for cube2
String cube2Color = "Blue";
```



Why?

Try this one: Create **cubes**, without using classes.

```
// create a cube
double cube1 = 3.0;

// create another cube
double cube2 = 4.0;

// set color for cube1
String cube1Color = "Red";

// set color for cube2
String cube2Color = "Blue";

// print cube1 and its color
System.out.format("Cube with length = %.2f and color = %s\n", cube1, cube1Color);

// print cube2 and its color
System.out.format("Cube with length = %.2f and color = %s\n", cube2, cube2Color);
```

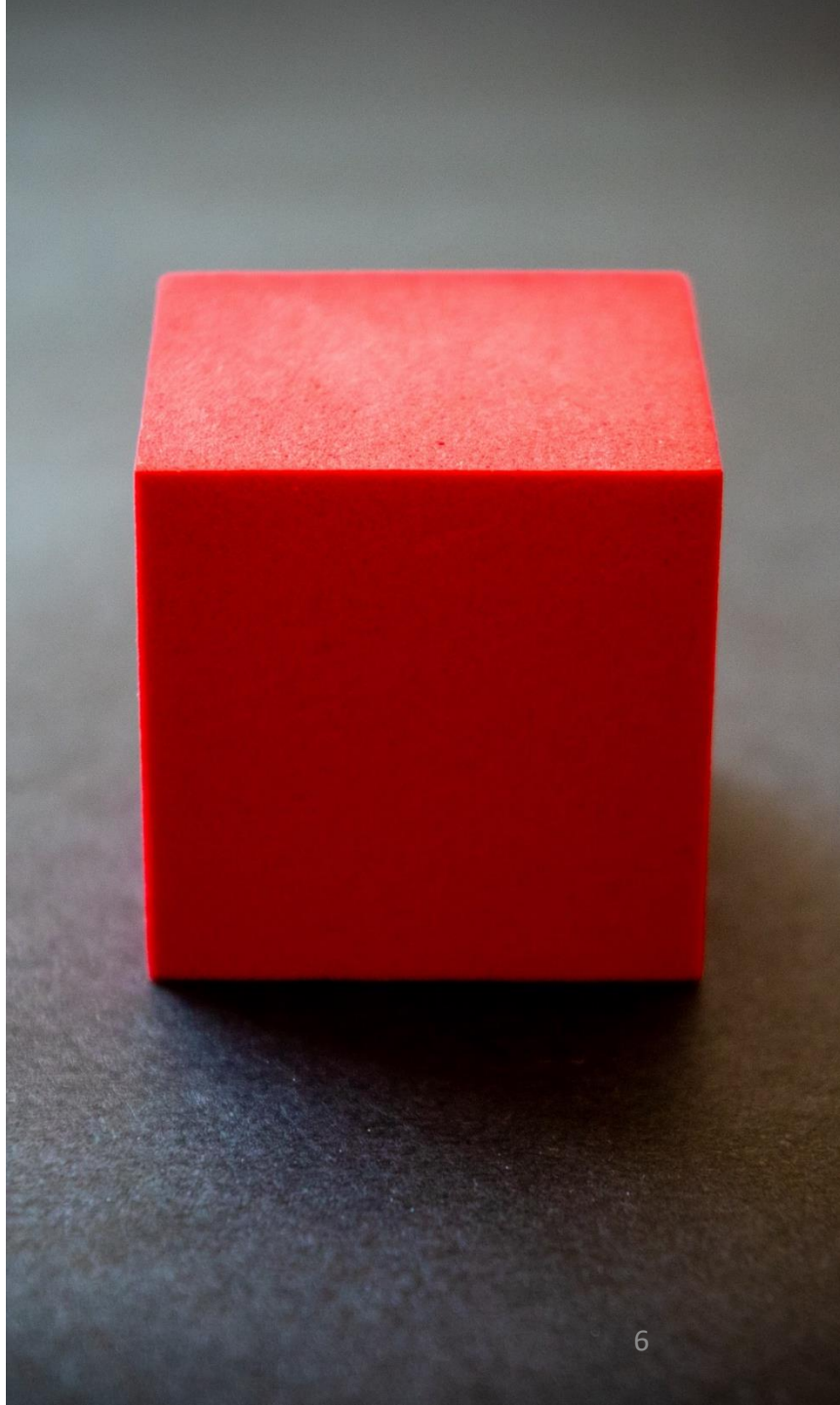


Why?

Try this one: Create **cubes**, without using classes.

```
// create a cube
double cube1 = 3.0;

// create another cube
double cube2 = 4.0;

// set color for cube1
String cube1Color = "Red";

// set color for cube2
String cube2Color = "Blue";

// print cube1 and its color
System.out.format("Cube with length = %.2f and color = %s\n", cube1, cube1Color);

// print cube2 and its color
System.out.format("Cube with length = %.2f and color = %s\n", cube2, cube2Color);
```

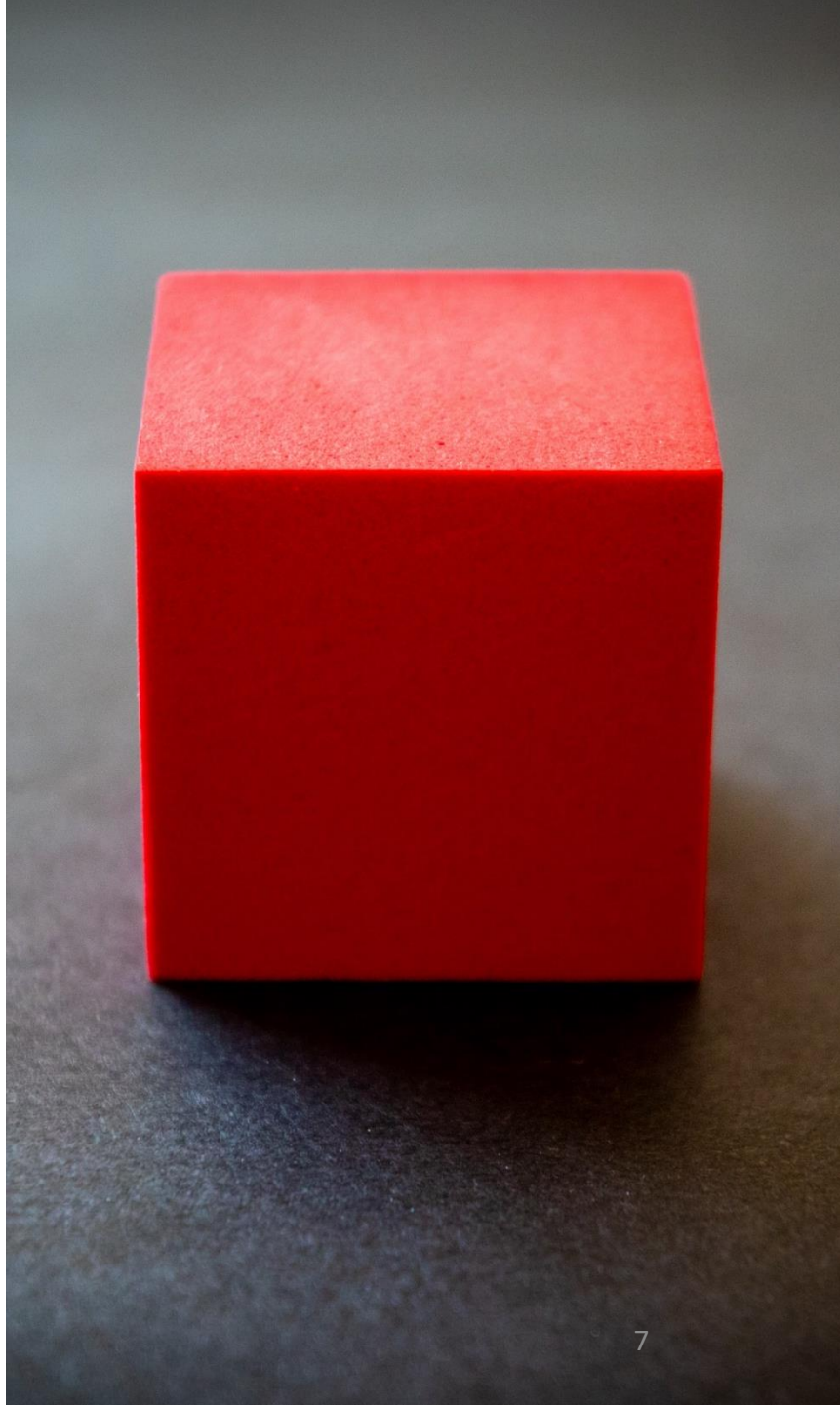
It's cumbersome (= ribet)!



Why?

Try this one: Create **cubes**, now with classes.

```
Cube cube1 = new Cube("Red", 3.0);
Cube cube2 = new Cube("Blue", 4.0);
System.out.println(cube1);
System.out.println(cube2);
```

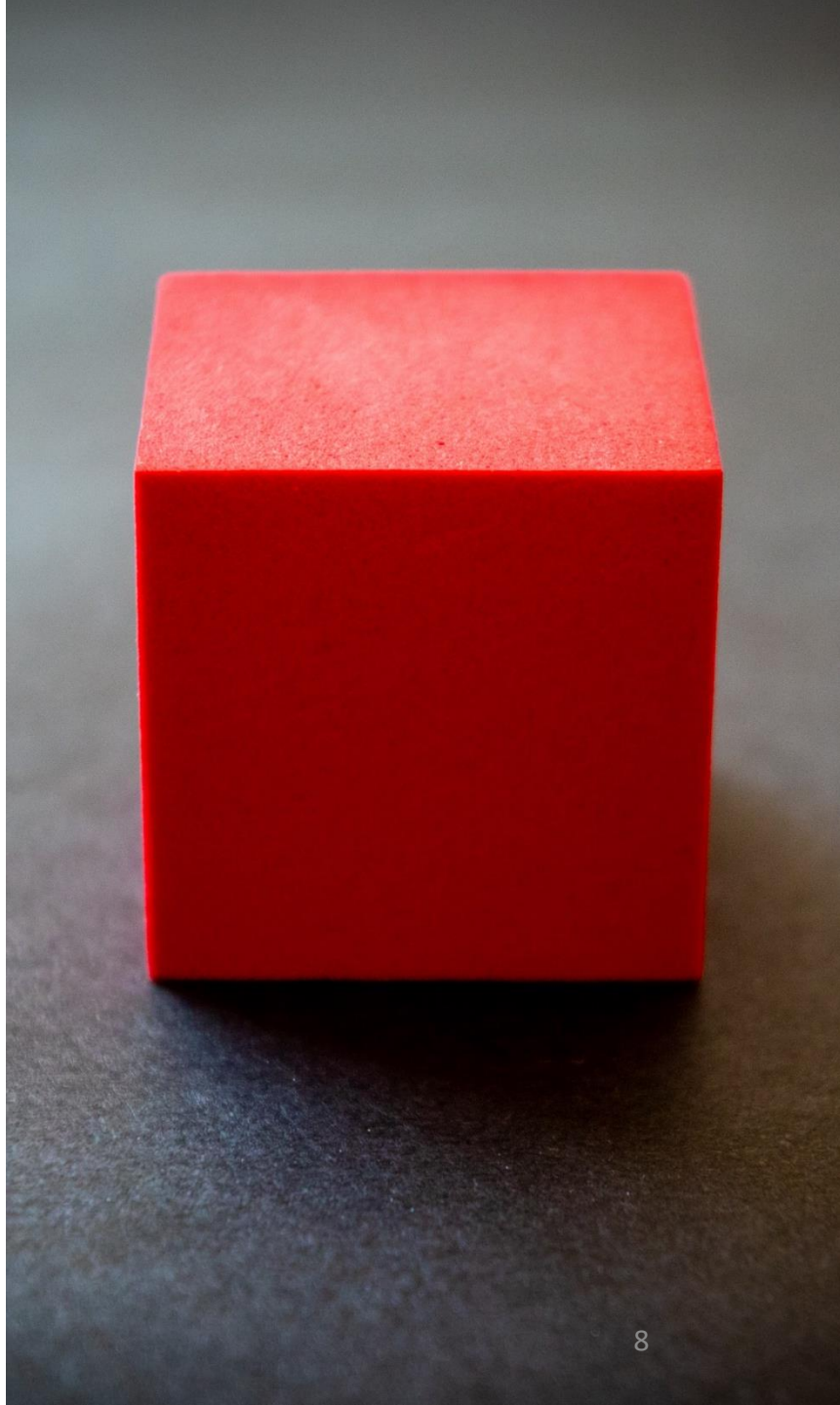


Why?

Try this one: Create **cubes**, now with classes.

```
Cube cube1 = new Cube("Red", 3.0);
Cube cube2 = new Cube("Blue", 4.0);
System.out.println(cube1);
System.out.println(cube2);
```

Much simpler, isn't it?



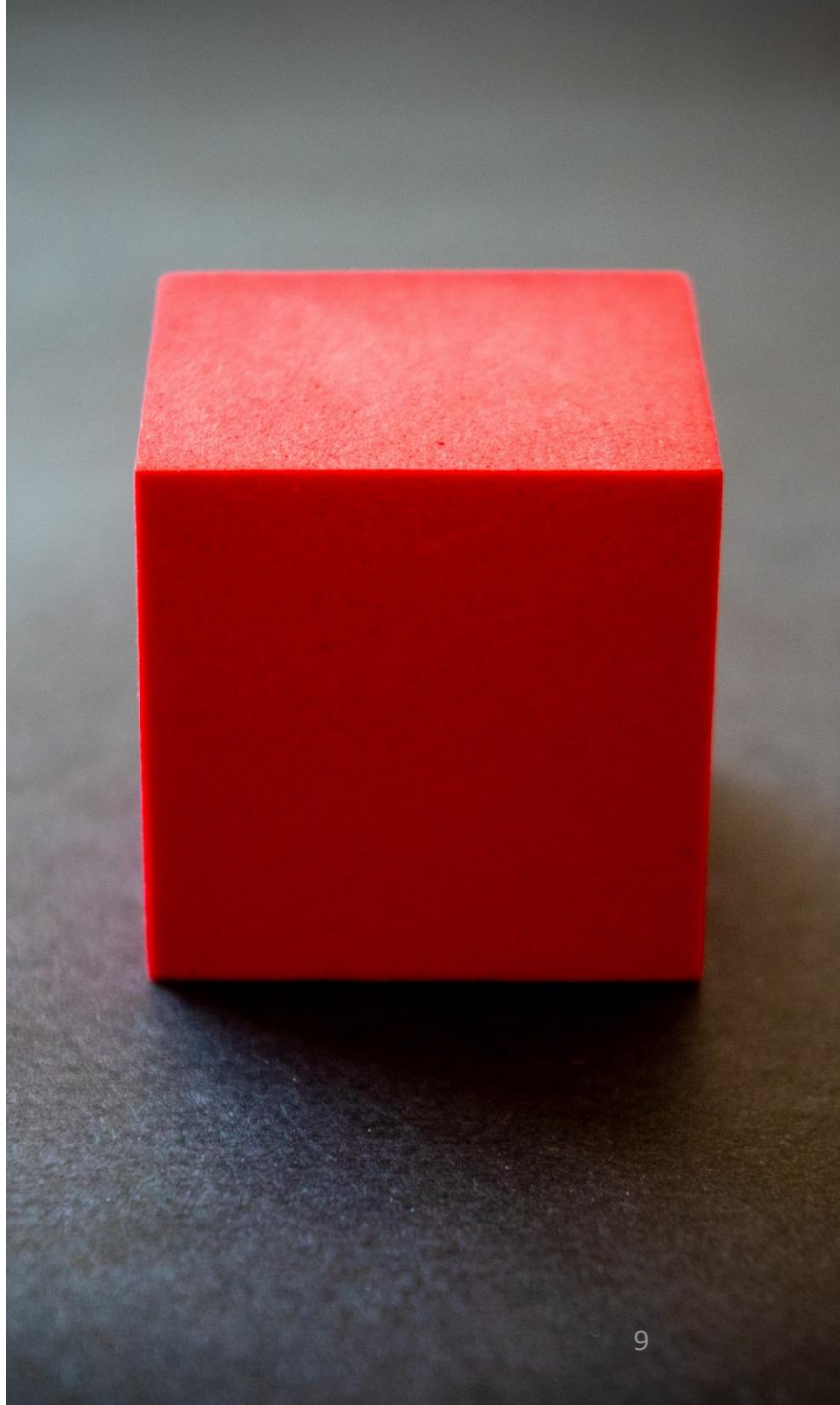
Why?

Try this one: Create **cubes**, now with classes.

```
Cube cube1 = new Cube("Red", 3.0);
Cube cube2 = new Cube("Blue", 4.0);
System.out.println(cube1);
System.out.println(cube2);
```

Much simpler, isn't it?

Here, the class Cube encapsulates (= membungkus) related data such as color and length as a single unit



Object-oriented thinking is natural!

Here, we have that the class is Car,
and the objects are, well, all those car instances!

And these cute cats?

They are objects as well, of type Cat!

Ahh, Human..

Beautiful, yet complicated objects :)


Objects can be abstract,
such as ideas, emotions, and knowledge

Hello, OOP!

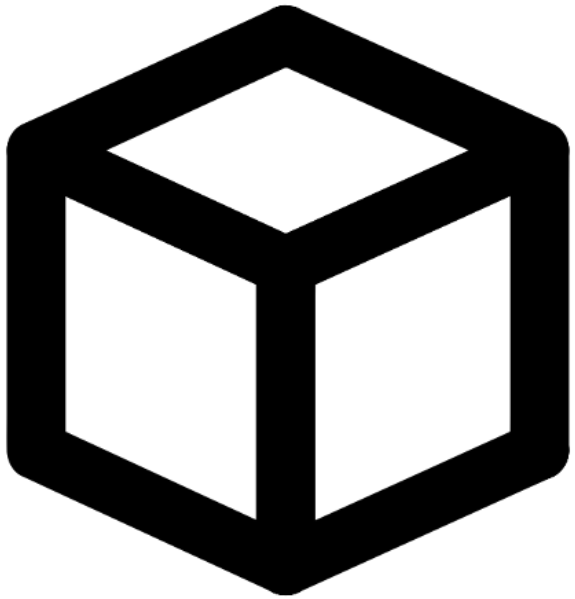
An object has:

- **states (aka. data fields/properties/attributes)**
- **behaviors (aka. methods)**

Hello, OOP!

An object has:

- **states (aka. data fields/properties/attributes)**
- **behaviors (aka. methods)**



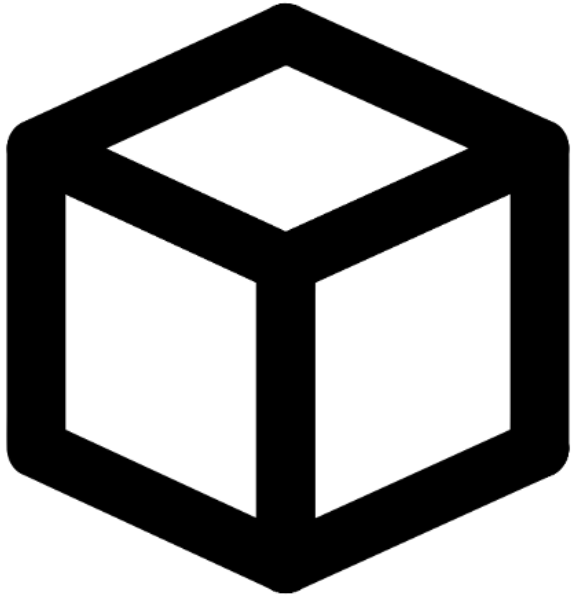
Hello, OOP!

An object has:

- **states (aka. data fields/properties/attributes)**
- **behaviors (aka. methods)**

UML Class Diagram

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



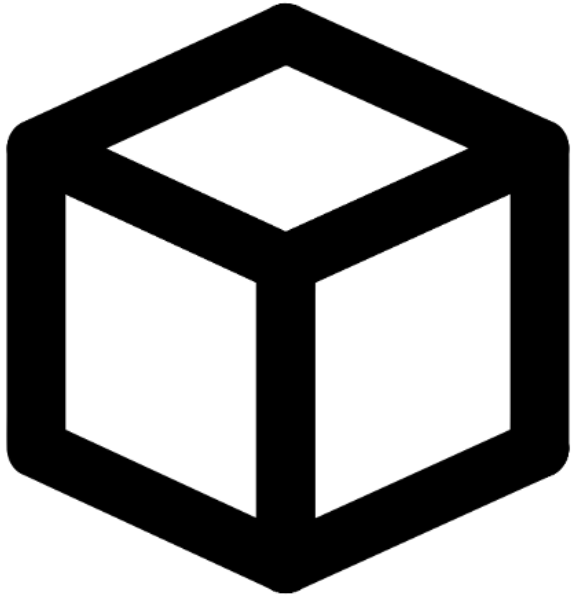
Hello, OOP!

An object has:

- **states (aka. data fields/properties/attributes)**
- **behaviors (aka. methods)**

UML Class Diagram

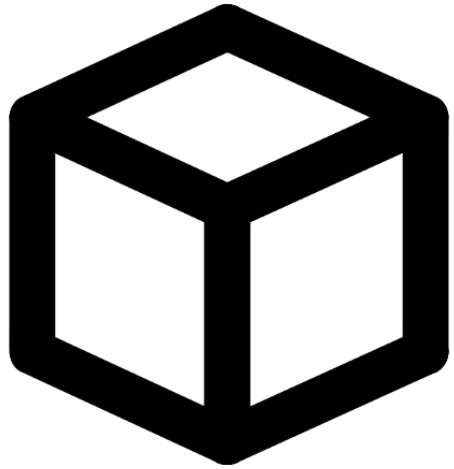
Class name	**Cube**
Data fields	**color: String** **length: double**
Constructors and methods	**Cube()** **Cube(color: String, length: double)** **toString(): String** **getVolume(): double**



Objects of Cube

UML Class Diagram

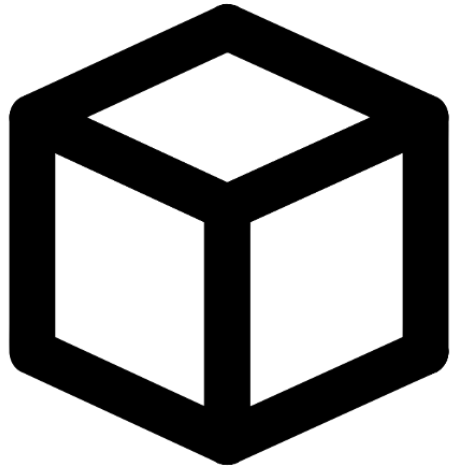
Cube
color: String **length: double**
Cube() **Cube(color: String, length: double)** **toString(): String** **getVolume(): double**



Objects of Cube

UML Class Diagram

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



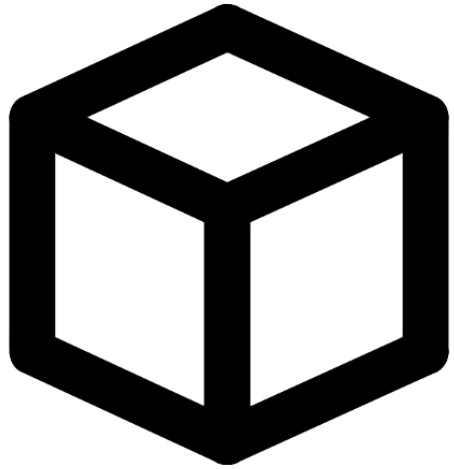
UML Objects

cube1: Cube
color = "Blue" **length = 1.0**

Objects of Cube

UML Class Diagram

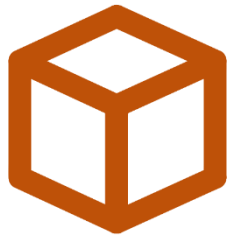
Cube
color: String length: double
Cube() Cube(color: String, length: double) toString(): String getVolume(): double



UML Objects

cube1: Cube
color = "Blue" length = 1.0

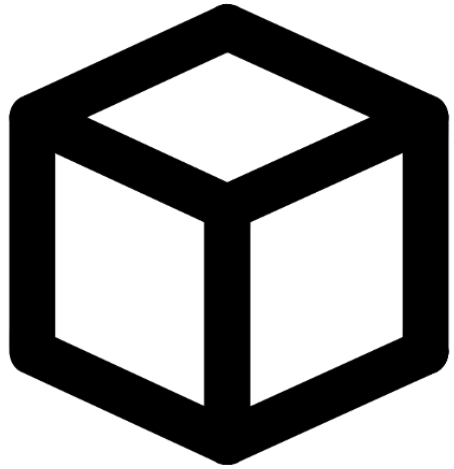
cube2: Cube
color = "Red" length = 2.0



Objects of Cube

UML Class Diagram

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



UML Objects

cube1: Cube
color = "Blue" length = 1.0

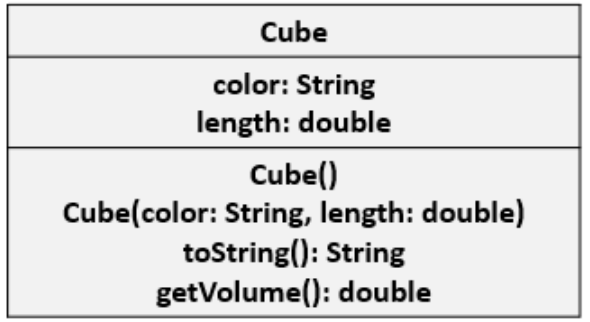
cube2: Cube
color = "Red" length = 2.0

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

..or cookie cutters

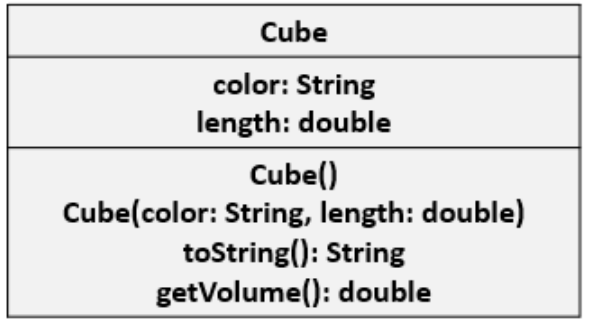
Let's code the Cube class!

UML Class Diagram



Let's code the Cube class!

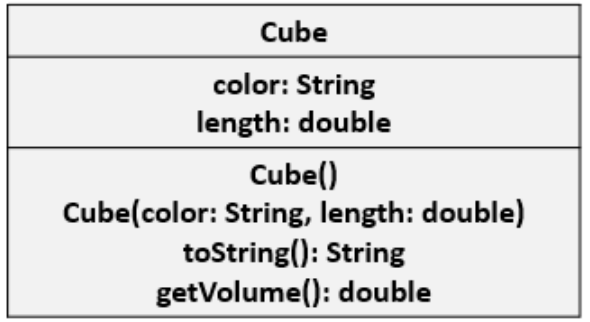
UML Class Diagram




```
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;
	} // code continues..
```

Let's code the Cube class!

UML Class Diagram

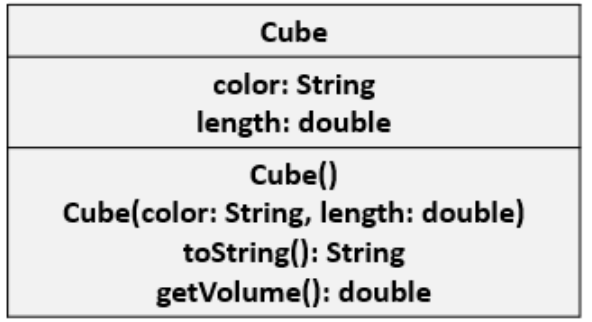



```
public class Cube {

      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;
      } // code continues..
```

These two guys are called: Data fields

Let's code the Cube class!

UML Class Diagram




```
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;
      } // code continues..
```

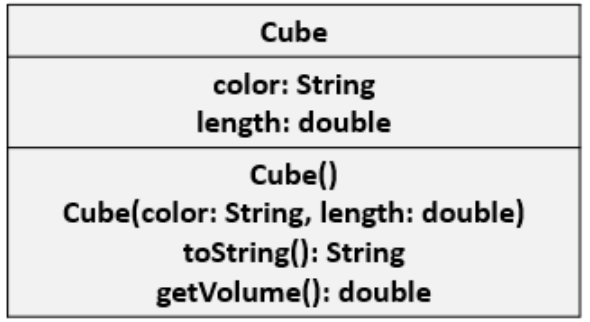
These two guys are called: Data fields

And these two guys are: Constructors

Let's code the Cube class!

```
	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);
	}

}
```

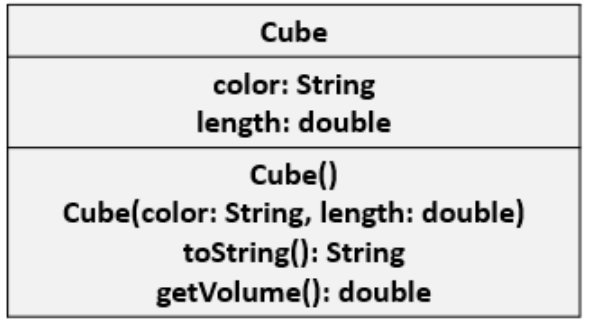
UML Class Diagram



Let's code the Cube class!

```
	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);
	}

}
```

UML Class Diagram



These are: Methods

Quiz time: Let's code the Sphere class!

Quiz time: Let's code the Sphere class!

```
// .. continuation
      public String toString() {
            return String.format("Sphere with radius = %.2f and
color = %s",this.radius,this.color);
      }

      public double getVolume() {
            return 4.0/3 * Math.PI * Math.pow(this.radius, 3.0);
      }

}
```

Quiz time: Is this illegal?

```
public class A { }
```

Quiz time: Is this illegal?

```
public class A { }
```

No, this is not illegal (= not not legal = legal :-).

It's the simplest class definition you can have.

**When no constructors are defined,
default constructors are used to
instantiate the objects of the class.**

Recall the Cube class, let's instantiate it!

```
Cube cube1 = new Cube();
Cube cube2 = new Cube("Blue", 4.0);
```

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

Recall the Cube class, let's instantiate it!

```
Cube cube1 = new Cube();
Cube cube2 = new Cube("Blue", 4.0);
```

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

code inside Cube.java

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

Recall the Cube class, let's instantiate it!

```
Cube cube1 = new Cube();
Cube cube2 = new Cube("Blue", 4.0);
```

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

code inside Cube.java

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

Now, let's access the data fields, and call the methods.

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

Now, let's access the data fields, and call the methods.

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

Output:

```
White
1.0
4.0
64.0
```

Special method: toString()

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

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

Special method: toString()

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

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

Printing objects calls the toString() method!

Special method: toString()

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

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

Printing objects calls the toString() method!

Output:

```
Cube with length = 1.00 and color = White
Cube with length = 4.00 and color = Blue
```

Variables store references of objects

```
Cube cube;
cube = new Cube("Red", 2.0);
Cube cubeCopy = cube;
cubeCopy.color = "Blue";
System.out.println(cube.color);
System.out.println(cubeCopy.color);
```

Variables store references of objects

```
Cube cube;
cube = new Cube("Red", 2.0);
Cube cubeCopy = cube;
cubeCopy.color = "Blue";
System.out.println(cube.color);
System.out.println(cubeCopy.color);
```

Output:

```
Blue
Blue
```

Variables store references of objects

```
Cube cube1 = new Cube("Red", 2.0);
Cube cube2 = new Cube("Red", 2.0);
cube2.color = "Blue";
System.out.println(cube1.color);
System.out.println(cube2.color);
```

Output:

```
Red
Blue
```

null

It's a special value, meaning "no 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());
```

null

It's a special value, meaning "no 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

Real-world classes: Set

```
import java.util.HashSet;
```

```
HashSet<Integer> s1 = new HashSet<Integer>();
HashSet<Integer> s2 = new HashSet<Integer>();

s1.add(3); s1.add(3); s1.add(1); s1.add(2);
s2.add(2); s2.add(0); s2.add(7); s2.add(3);

s1.retainAll(s2);
System.out.println(s1);
```

Real-world classes: Set

```
import java.util.HashSet;
```

```
HashSet<Integer> s1 = new HashSet<Integer>();
HashSet<Integer> s2 = new HashSet<Integer>();

s1.add(3); s1.add(3); s1.add(1); s1.add(2);
s2.add(2); s2.add(0); s2.add(7); s2.add(3);

s1.retainAll(s2);
System.out.println(s1);
```

Output:

```
[2, 3]
```

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());
```

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());
```

Output:

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

Real-world classes: LocalDate

```
import java.time.LocalDate; // introduced in java 8
```

```
LocalDate dateNow = LocalDate.now();
System.out.println(dateNow);
System.out.println(dateNow.getDayOfWeek());
System.out.println(dateNow.plusDays(1)); // besok
System.out.println(dateNow.plusDays(2)); // lusa
System.out.println(dateNow.plusDays(3)); // tulat
System.out.println(dateNow.plusDays(4)); // tubin
```

Real-world classes: Random

```
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);
	}
}
```

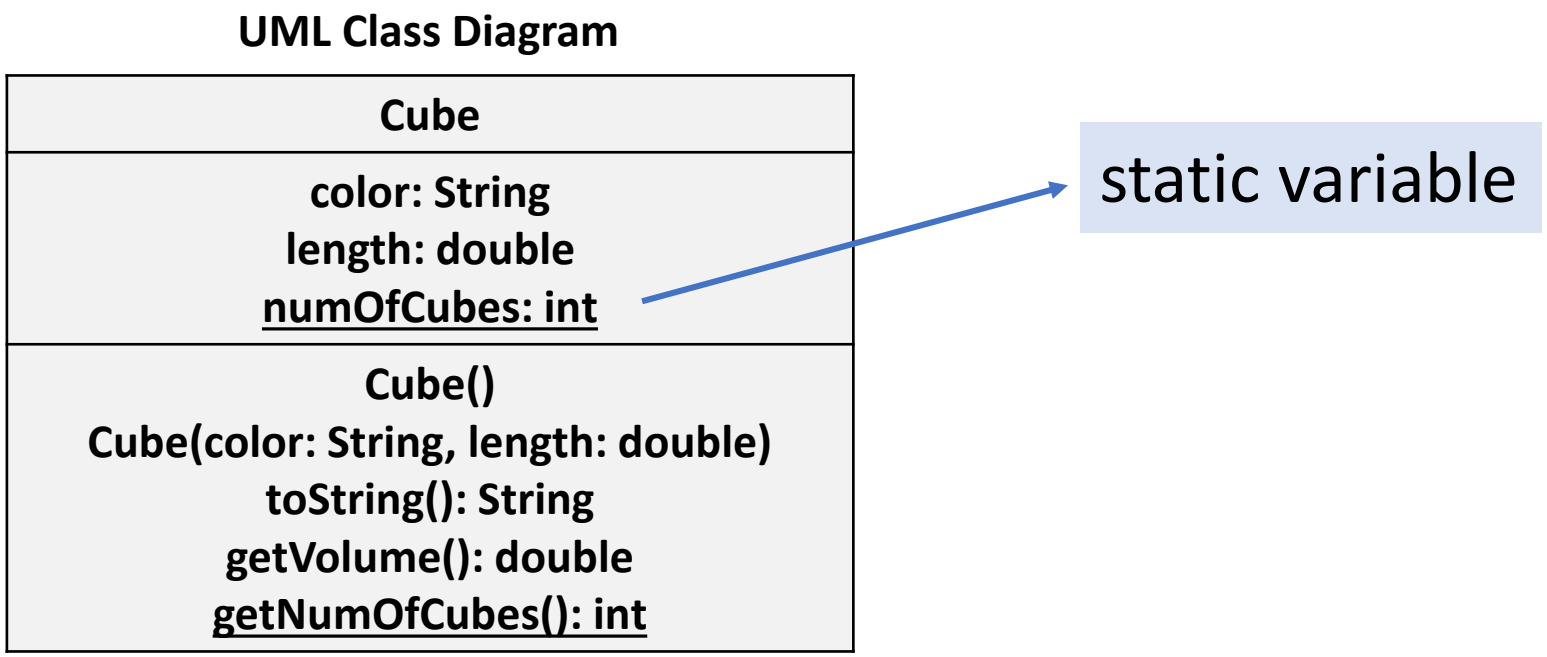
Quiz time: Create a method to initialize an ArrayList of Integers with *n* random Integers from 11 to 20!

Quiz time: Create a method to initialize an ArrayList of Integers with *n* random Integers from 11 to 20!

```
public static ArrayList<Integer> initRandom(int n) {

	ArrayList<Integer> lstInt = new ArrayList<Integer>();
	Random rnd = new Random();
	while(n > 0) {
		lstInt.add(rnd.nextInt(10)+11);
		n--;
	}
	return lstInt;

}
```

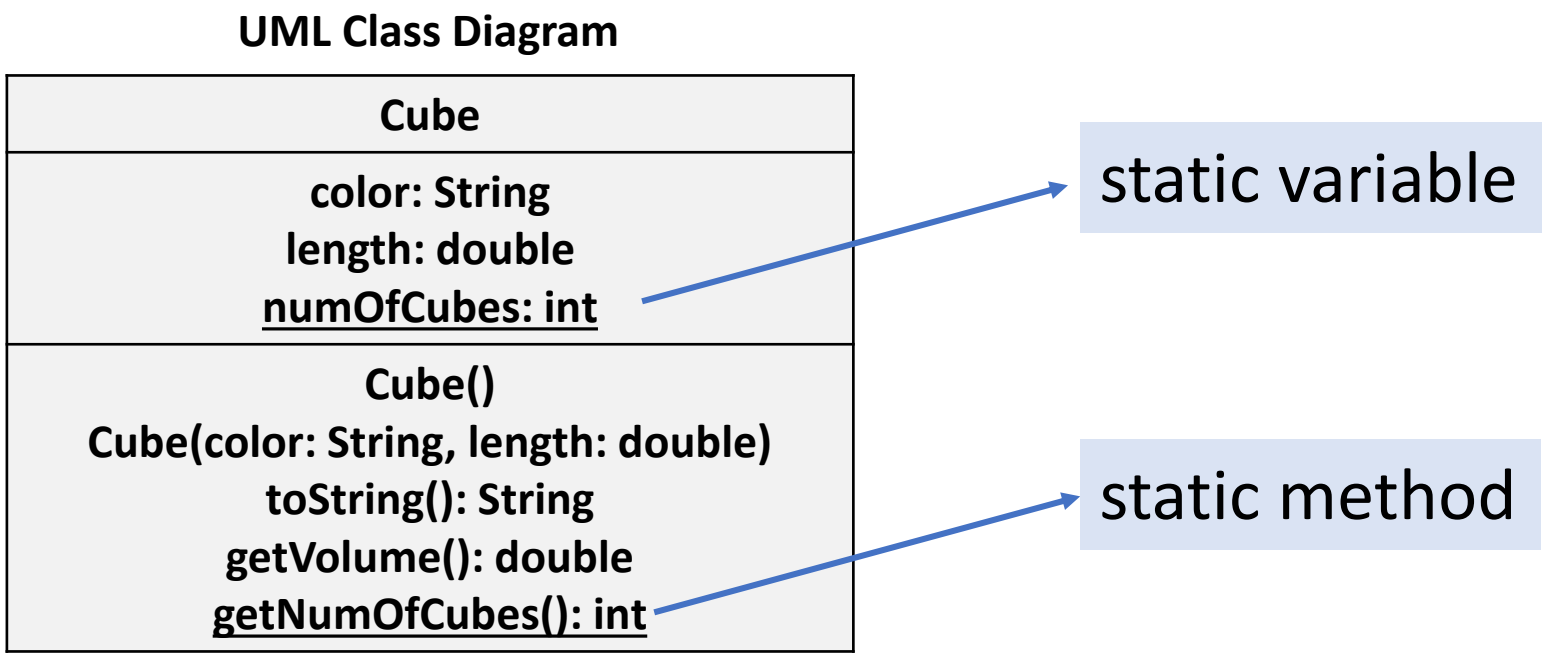
Static variables, constants, and methods

Static means shared by all objects of the class.



Static variables, constants, and methods

Static means shared by all objects of the class.



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

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());
```

Output:

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());
```

Output:

```
0
2
2
2
2
```

Cube.java with static constant

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

Edit the previous Cube.java accordingly

```
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);
```

Output:

```
Silver
Silver
Silver
```

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

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

```
public class Time {

	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 declare the data fields

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

```
public class Time {

	int hour;
	int minute;
	double second;

	public Time() {
		this.hour = 0;
		this.minute = 0;
		this.second = 0.0;
	}

}
```

We create a constructor method

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

```
public class Time {

	int hour;
	int minute;
	double second;

	public Time() {
		this.hour = 0;
		this.minute = 0;
		this.second = 0.0;
	}

}
```

We create a constructor method

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 Time(int hour, int minute, double second) {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
    }

}
```

We create another constructor: this time you can provide your own values for hour, minute, and 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);
```

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);
```

Output:

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

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

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);
      }

}
```

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);
    }

}
```

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.

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);
```

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);
```

Output:

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

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

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

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

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

Output:

```
01:03:02.1
```

`equals` method

We have seen two ways to check whether values are equal: the == operator and the `equals` method. With objects you can use either one, but they are not the same.

- 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.

`equals` method

We have seen two ways to check whether values are equal: the == operator and the `equals` method. With objects you can use either one, but they are not the same.

- 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.

What does it mean by "same" in the same value?

`equals` method

We have seen two ways to check whether values are equal: the == operator and the `equals` method. With objects you can use either one, but they are not the same.

- 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.

What does it mean by "same" in the same value?

We define the **equals** method for our objects.

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;
      }

}
```

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;
      }

}
```

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(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));
```

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));
```

Output:

```
false
true
false
false
```

Another use of **this**

```
public class Time {

	// ...
	public Time() {
		this(0, 0, 0.0); // this can be used to call another constructor
	}

	public Time(int hour, int minute, double second) {
		this.hour = hour;
		this.minute = minute;
		this.second = second;
	}
// ...
```

Visibility modifiers

```
public class Time {

	int hour;
	int minute;
	double second;

}
```

- Data fields inside a class are too "open".
- Other classes can directly access the values of those data fields.
- We need **information hiding**: a way to control what can be accessed from outside, and what cannot.

Visibility modifiers

```
public class Time {

	private int hour;
	private int minute;
	private double second;

}
```

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

Visibility modifiers

```
public class Time {

	private int hour;
	private int minute;
	private double second;

}
```

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

Can you now access those variables outside the Time class? Try it out.

Visibility modifiers

```
public class Time {

	private int hour;
	private int minute;
	private double second;

}
```

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

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); // compiler error
  }
}
```

Getters and setters

```
// inside Time.java

        public int getHour() {
                return this.hour;
        }

        public int getMinute() {
                return this.minute;
        }

        public double getSecond() {
                return this.second;
        }

}
```

Getters and **setters**

```
// 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;
        }

}
```

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

```
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());
```

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

```
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());
```

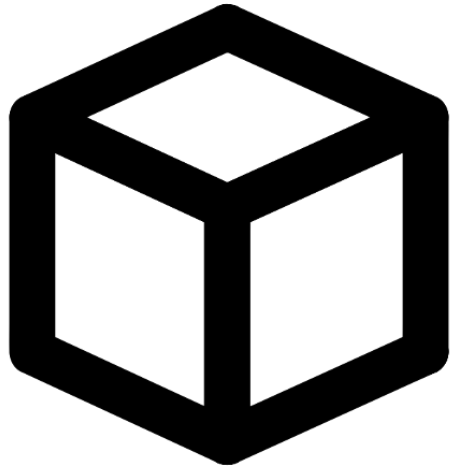
Output:

```
10.0
11
50
```

Recall our Cube.java

UML Class Diagram

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

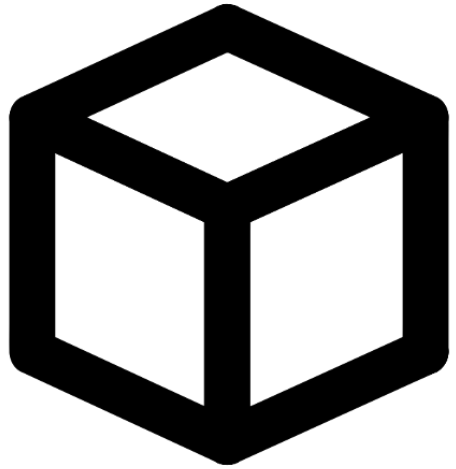


This is the original UML,
now suppose we want the data fields to be private,
and all the methods to be public, what would change?

Recall our Cube.java

UML Class Diagram

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



- sign indicates private, while

+ sign indicates not private

Quiz time:

Can a Java object have a field of another object?

Quiz time:

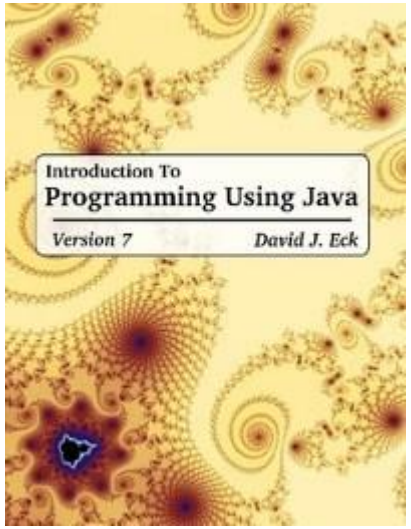
Can a Java object have a field of another object?

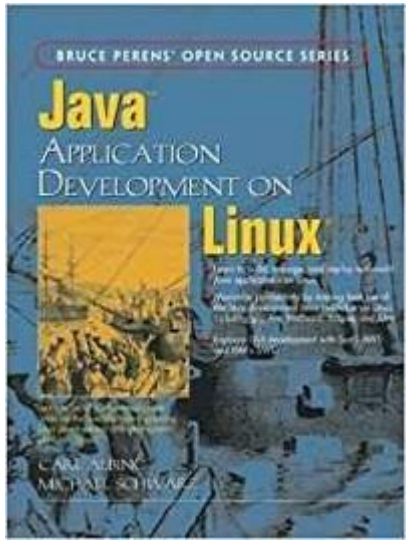
Yes, why not. We have seen an example of this in Cube.java.

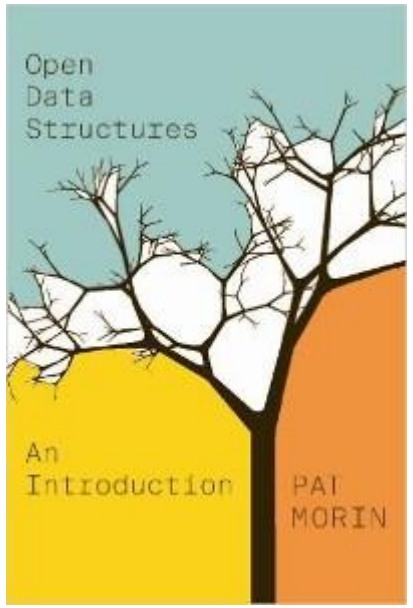
Take-away messages

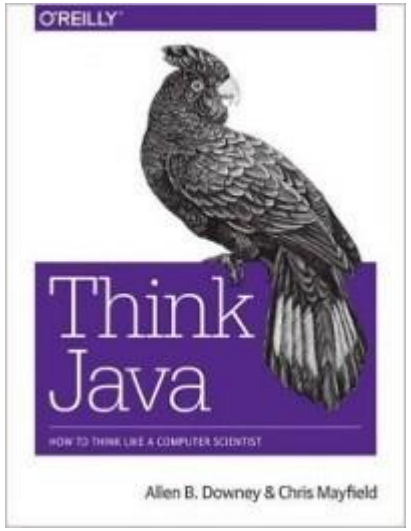
- Defining a class creates a new object type.
- Every object belongs to some object type.
- A class definition is like a template for objects, it specifies:
  - what **attributes** the objects have; and
  - what **methods** can operate on them.
- The **new** operator creates new instances of a class.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

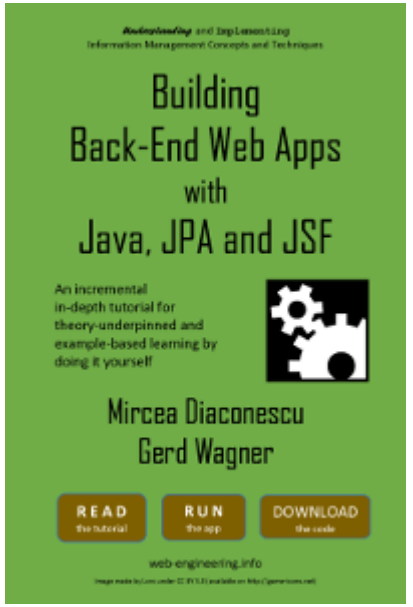
Btw, open Java books:



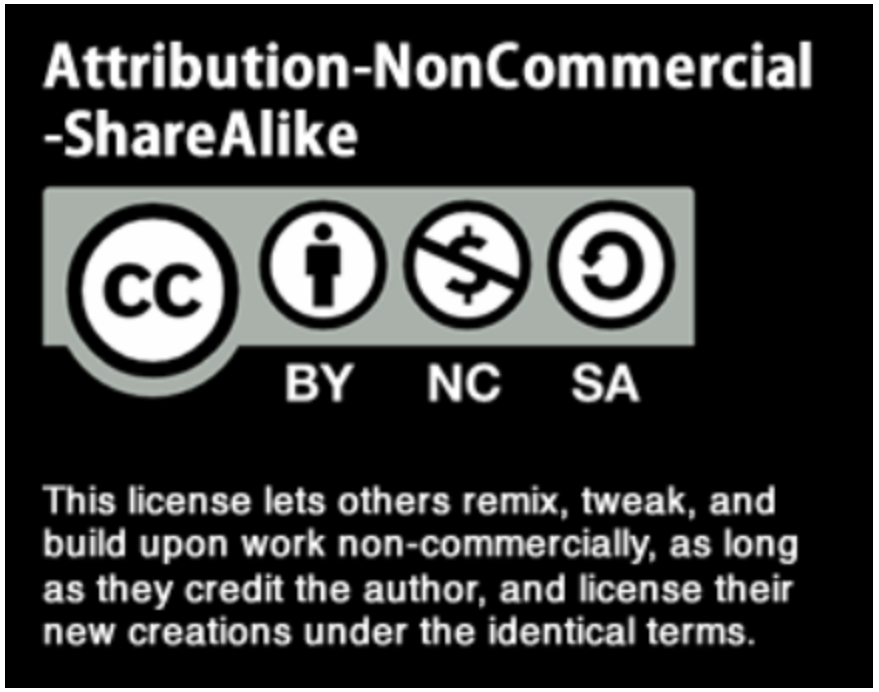








This slideset is open, and free to use and share!



Inspired by:
Liang. Introduction to Java Programming. Tenth Edition. Pearson 2015.
Chapter 10 & 11 of Think Java book by Allen Downey and Chris Mayfield.