Defining Your Own Classes

Paolo Perrotta

Developer, Author

@nusco | www.paoloperrotta.com

```
Alarm alarm = new Alarm();
alarm.active    → false
alarm.turnOn();
alarm.active    → true
```

```
class Alarm {
}
```

Class Names in Camel Case

Alarm

MyAlarm

MyBeautifulAlarm

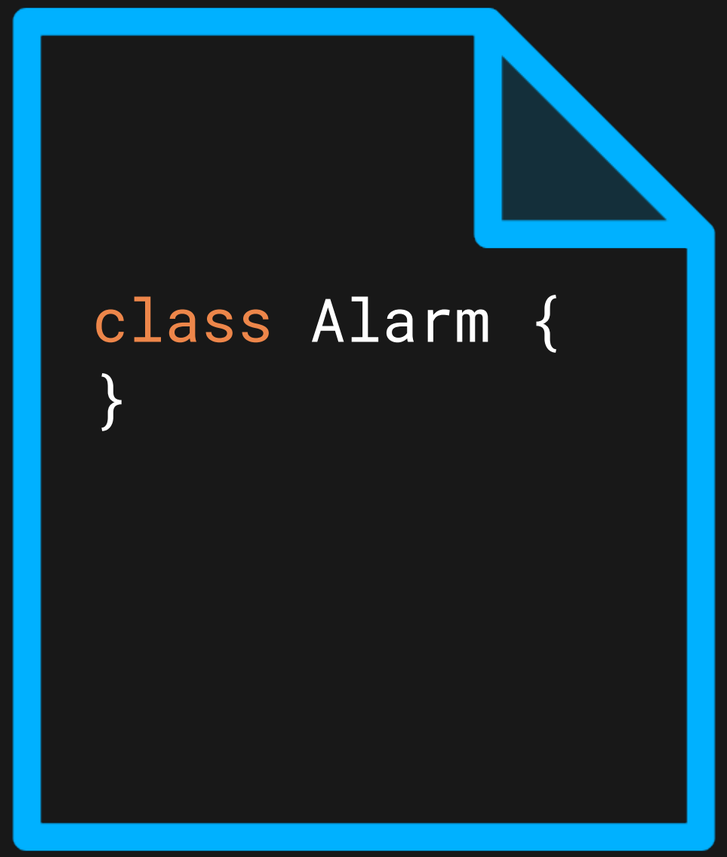
USBPort

SomeLongClassName

URL

```
Alarm alarm = new Alarm();
alarm.active    → false
alarm.turnOn();
alarm.active    → true
```

Alarm.java



Field, Method and Variable Names

active

myVariable

doSomething()

turnOn()

jumpHigh()

x

```
✓ Alarm alarm = new Alarm();
✓ alarm.active    → false
⊗ alarm.turnOn();
⊗ alarm.active    → true
```

```
class Alarm {
    boolean active;
}
```

```
Alarm alarm = new Alarm();
alarm.active    → false
alarm.turnOn();
alarm.active    → true
```

```
class Alarm {
    boolean active;

    void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
    }
}
```

A Request for New Features

Add a message to alarms

- The message describes what an alarm is about
- Example: “Temperature too high”

Add a method to get a report

- The report describes the state of the alarm
- If the alarm is active, the report is the alarm’s message
- If the alarm is inactive, the report is empty
- Add an option to get the report all uppercase

Add a notification method

- For now, just a placeholder that prints the report
- It prints the uppercase version of the report

Add a message to alarms

- The message describes what an alarm is about
- Example: “Temperature too high”

```
class Alarm {
    boolean active;
    String message;

    void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
    }
}
```

Overloading

```
String getReport() {
    return getReport(false);
}

String getReport(boolean uppercase) {
   if (active) {
       if (uppercase)
           return message.toUpperCase();
       else
           return message;
   } else
       return "";
}
```

Add a method to get a report

- The report describes the state of the alarm
- If the alarm is active, the report is the alarm's message
- If the alarm is inactive, the report is empty
- Add an option to get the report all uppercase

```
    void turnOn() {
        active = true;
    }

    void turnOff() {
        active = false;
    }


    String getReport() {
        return getReport(false);
    }


    String getReport(boolean uppercase) {
        if (active) {
            if (uppercase)
                return message.toUpperCase();
            else
                return message;
        } else
            return "";
    }
}
```

Add a notification method

- For now, just a placeholder that prints the report
- It prints the uppercase version of the report

```
    void turnOff() {
        active = false;
    }

    String getReport() {
        return getReport(false);
    }

    String getReport(boolean uppercase) {
        if (active) {
            if (uppercase)
                return message.toUpperCase();
            else
                return message;
        } else
            return "";
    }

    void sendReport() {
       System.out.println(getReport(true));
    }
}
```

All Done!

~~Add a message to alarms~~

- ~~The message describes what an alarm is about~~
- ~~Example: “Temperature too high”~~

~~Add a method to get a report~~

- ~~The report describes the state of the alarm~~
- ~~If the alarm is active, the report is the alarm’s message~~
- ~~If the alarm is inactive, the report is empty~~
- ~~Add an option to get the report all uppercase~~

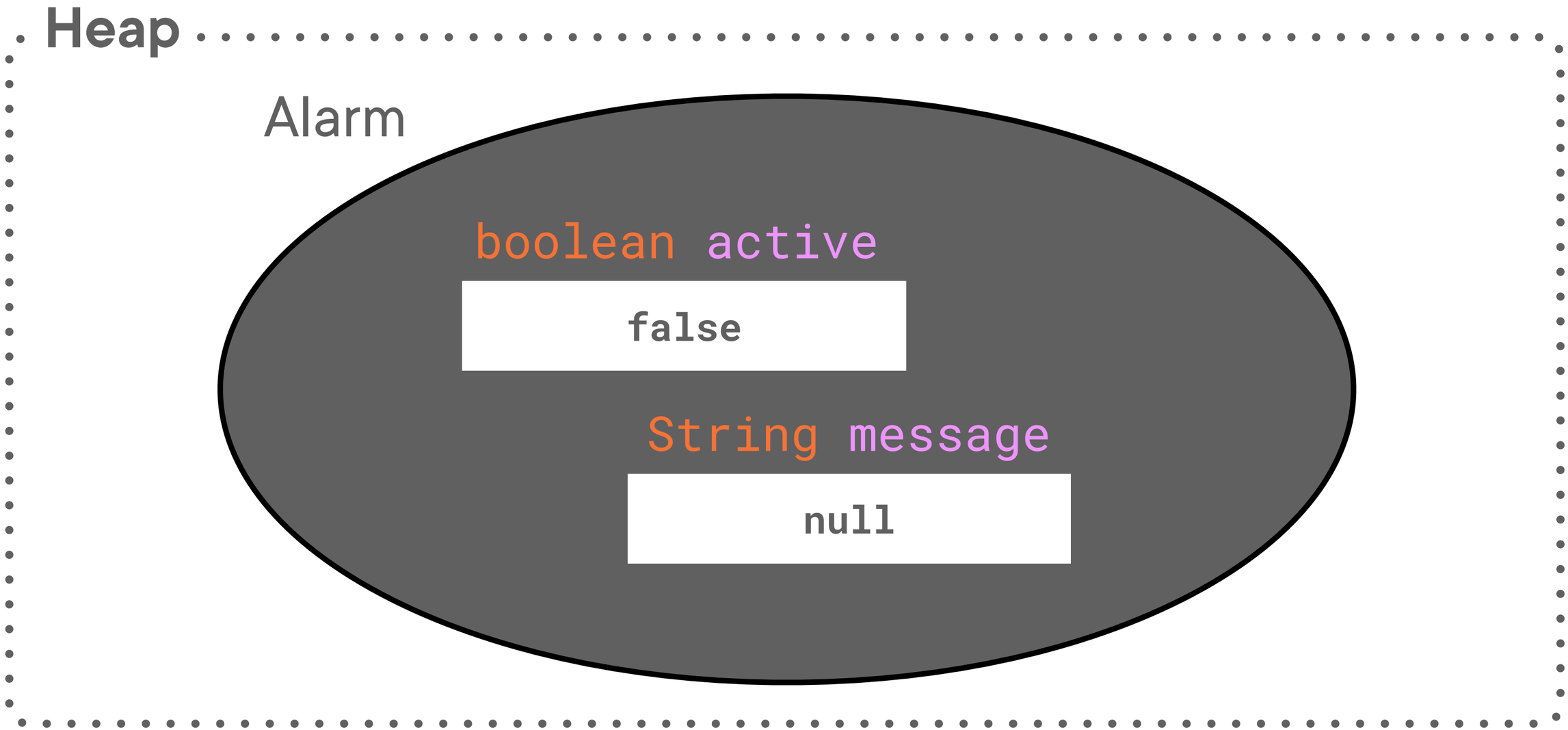
~~Add a notification method~~

- ~~For now, just a placeholder that prints the report~~
- ~~It prints the uppercase version of the report~~

```java
class Alarm {
    boolean active;
    String message;

    void turnOn() {…}

    void turnOff() {…}

    String getReport() {
        return getReport(false);
    }

    String getReport(boolean uppercase) {
        if (active) {
            if (uppercase)
                return message.toUpperCase();
            else
                return message;
        } else
            return "";
    }

    void sendReport() {
        System.out.println(getReport(true));
    }
}
```

```java
Alarm alarm = new Alarm();
alarm.turnOn();
alarm.sendReport();
```

Object Initialization



```
class Alarm {
    boolean active;
    String message;

    void turnOn() {…}

    void turnOff() {…}

    String getReport() {
        return getReport(false);
    }

    String getReport(boolean upperc
        if (active) {
            if (uppercase)
                return message.toUpperCase();
            else
                return message;
        } else
            return "";
    }

    void sendReport() {
        System.out.println(getReport(true));
    }
}
```

null

```
Alarm alarm = new Alarm();
alarm.turnOn();
alarm.sendReport();
```

NullPointerException

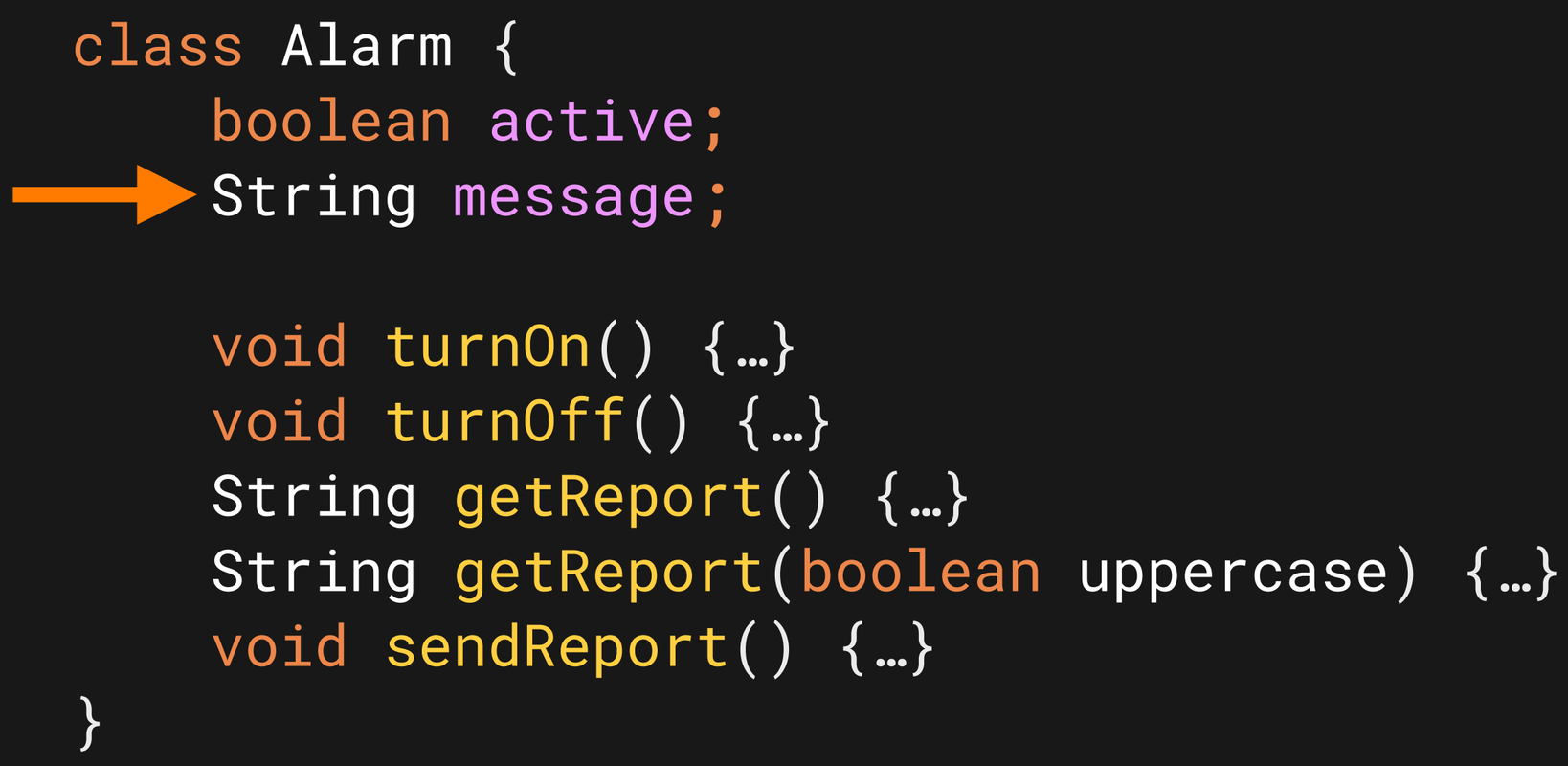
Initializing a Field

```
class Alarm {
    boolean active;
    String message;

    void turnOn() {…}
    void turnOff() {…}
    String getReport() {…}
    String getReport(boolean uppercase) {…}
    void sendReport() {…}
}
```

Leaving It to the Client

```
Alarm alarm = new Alarm();
alarm.message = "Temperature too high";
alarm.turnOn();
alarm.sendReport();
```

Initializing a Field at the Point of Definition



Constructor Arguments

```
Alarm alarm = new Alarm("Temperature too high");
```

```
class Alarm {
    boolean active;
    String message;

    Alarm(String message) {
        this.message = message;
    }

    void turnOn() { ... }
```

Changing a Field After Construction

```
Alarm alarm = new Alarm("Temperature too high");
alarm.message = "Actually, the temperature is OK";
```

Final Fields

```
class Alarm {
    boolean active;
    String message;

    Alarm(String message) {
        this.message = message;
    }

    void turnOn() {…}
    void turnOff() {…}
    String getReport() {…}
    String getReport(boolean uppercase) {…}
    void sendReport() {…}
}
```

Final Fields

```
Alarm alarm = new Alarm("Temperature too high");
alarm.message = "Actually, the temperature is OK";
```

Final Fields

```
class Alarm {
    boolean active;
    final String message;

    Alarm(String message) {

    }

    void turnOn() {…}
    void turnOff() {…}
    String getReport() {…}
    String getReport(boolean uppercase) {…}
    void sendReport() {…}
}
```

“Constant” vs. “Immutable”

Constant (Final)

Applies to variables, including object references. It means that you cannot reassign the variable.

Immutable

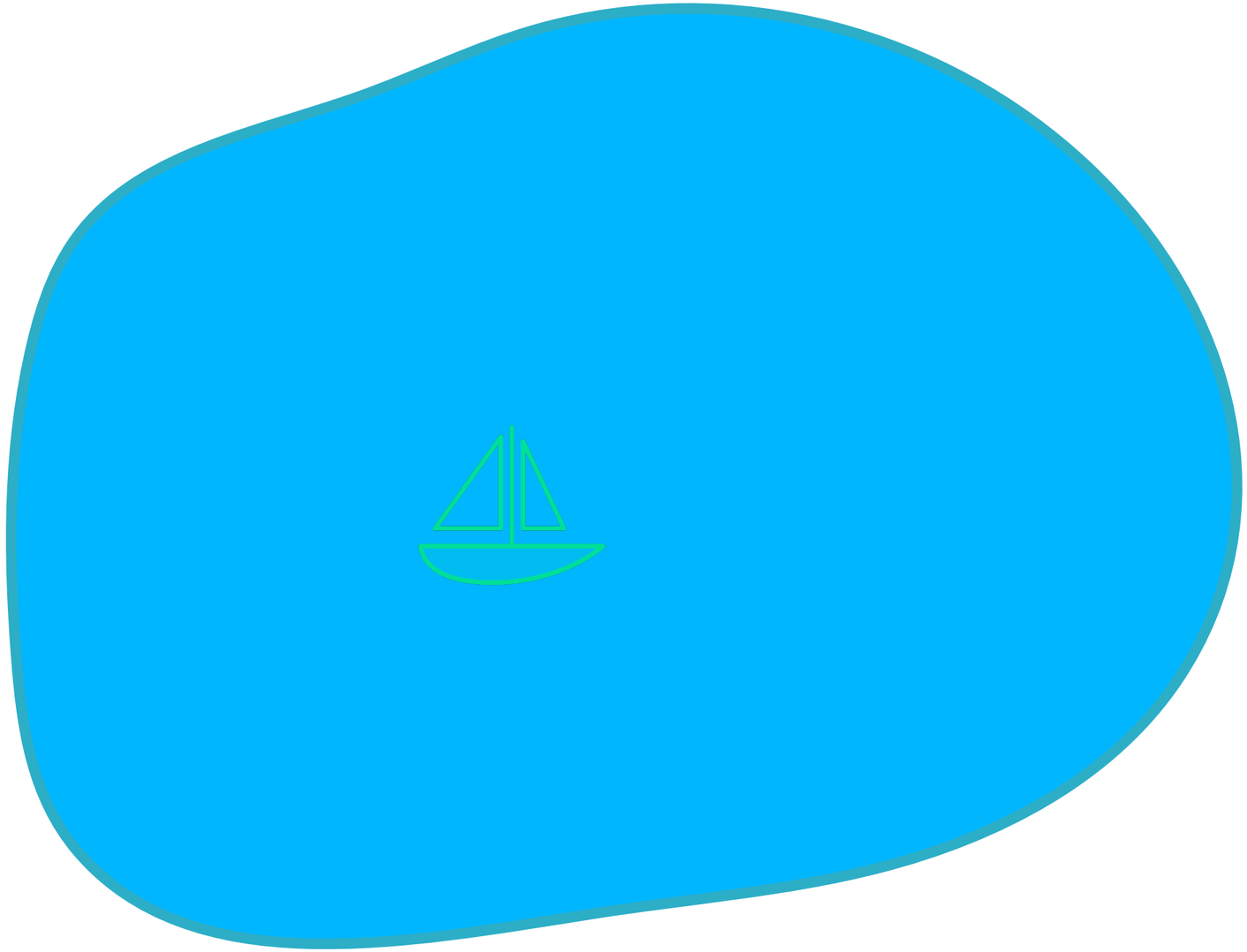
Applies to objects (for example, strings). It means that you cannot change the state of the object.

To make an object immutable, all its fields should be immutable *and* final.

More About Constructors

```
class Alarm {
    boolean active;
    final String message;

    Alarm() {
        this("Some default message");
    }

    Alarm(String message) {
        this.message = message;
    }

    void turnOn() {…}
    void turnOff() {…}
```

Unreachable Objects



The garbage collector is unpredictable.
(Also, it’s great.)

Constructors and Nothing Else

Constructors

Initialize the object

Garbage collection is only about memory, not other resources.

A Type Tells You...

What something looks like

What you can do with it

Primitive Types

int

float

boolean

char

byte

...

Classes Are Types

```
class Alarm {
    boolean active;
    String message;

    void turnOn() {…}
    void turnOff() {…}
    String getReport() {…}
    String getReport(boolean uppercase) {…}
    void sendReport() {…}
}
```

As Many Types as You Want

String

Money

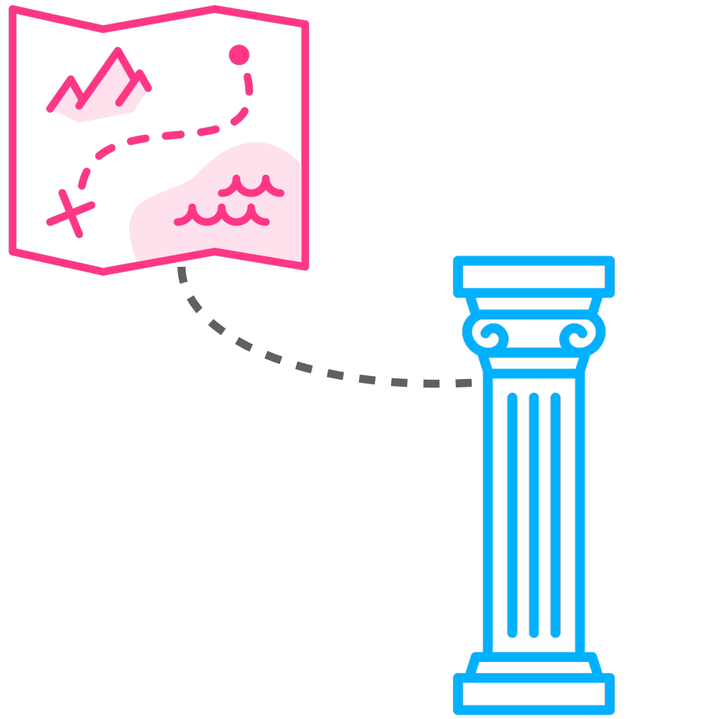
Budget

Alarm

Banana

...

The First Pillar of OOP



Abstraction

Summary

- Defining classes
- Fields, methods and constructors
- Overloaded methods
- Default constructors
- Garbage collection
- Abstraction

Up Next:

Hiding Information