Modelling State and Behavior

Dan Geabunea
SENIOR SOFTWARE DEVELOPER

@romaniancoder

Overview

Variables in Java

- Types
- Naming conventions

Methods in Java

- Declaration & signature
- Method overloading
- Naming conventions

The “this” keyword

Demo: Implement state and behavior

Variables

Variable

Container/piece of memory that stores data values

String message = “Hello”

Variable Definition

Data type

Name

Value

Variable Data Type

Primitive

int, boolean, double

Reference

String, objects, arrays

Variable Categories

Instance variables

Class variables

Local variables

Parameters

```
class JetEngine {
    String model;                   // instance variable
    int maxThrust;          // instance variable
        private fanSpeed;               // instance variable
}
JetEngine engine1 = new JetEngine(“Trent 800”, 60000);
JetEngine engine2 = new JetEngine(“GM”, 60000);
```

Instance Variables

Objects store their individual state in non static fields

These fields are also called instance variables because they are unique to every instance of a class

They can have access modifiers

```
class JetEngine {
    String model;           // instance variable
    int maxThrust; // instance variable
        void changeThrust(int value){
                if(value > this.maxThrust) {…}
        }
}
```

Instance Variables

Instance variables are used only within their declaring class, usually by instance methods

```
class JetEngine {
    String model;
        static int nbInstances = 0;                                  // class variable
        static final int MAX_THEORETICAL_SPEED = 990; // constant class variable
}
int enginesCreated = JetEngine.nbInstances;
```

Class Variables

Fields defined with the “static” modifier

One copy of this variable, regardless of how many instances we create

We can also add the “final” modifier to indicate that the value of the field will not change

Instance variables and class variables can also be called fields

```
void land(){
    boolean isLandingGearDeployed = this.systems.landingGear.status;
    if(!isLandingGearDeployed){
        this.systems.displayAlert("Enable Landing Gear");
    }
}
```

Local Variables

Methods store their state in local variables, defined in the body of the method

Only visible to the methods that declare them

```
void moveFlaps(double degrees){
   // Implementation
}


aircraft.moveFlaps(25.0);
```

Parameters

Are variables declared in the signature of methods

Their value is provided by the calling code

Variable Naming Conventions

- Case sensitive (maxSpeed ≠ maxspeed)
- Must start with a letter, $ or _
- The rest of the characters can be letters, digits, $ or _
- If the name is made up of more words use camel case (currentThrust)
- Use nouns

Always try to give your variables meaningful names that can be easily understood by other humans

Abbreviations / cryptic names

```
public class Aircraft {

    int alt;     // altitude


    void moveFlaps(double d){


    }
}
```

Methods

Method

A block of code that runs when it is invoked

```
class Aircraft {
    void land(){
        if(this.systems.landingGear.deployed){
            this.systems.alerts.show("Drop landing gear");
        }
                // Implementation details
    }
}


// Method is executed when called
Aircraft a = new Aircraft();
a.land();
```

Method Declaration

Access modifier	Return type	Name
Parameter list	Exception list	Body

Method Declaration

Access modifier	**Return type**	**Name**
Parameter list	Exception list	**Body**

Aircraft.java

```
// Log aircraft remaining fuel to flight data recorder
public void logFuelCapacity(String file) throws IOException {
        Path fileName = Path.of(file);
        String content  = "Remaining fuel: " + this.remainingFuel;
        Files.writeString(fileName, content);
}
```

Return Type

```
String readFlightDataRecorder(String file){
        // returns a String value
}

void land(){
    // return nothing
}
```

Methods can return a value

Methods can return void (nothing)

Method Signature

Combination of method name and parameter list

Aircraft.java

```
// Signature = name + parameter list
public void logFuelCapacity(String file) throws IOException {
        Path fileName = Path.of(file);
        String content  = "Remaining fuel: " + this.remainingFuel;
        Files.writeString(fileName, content);
}

a1.logFuelCapacity(“remaining-fuel-log.txt”);
```

Method Overloading

A class can have multiple methods with the same name if their parameter lists are different. The differentiation is made by the number and type of arguments.

Method Overloading

```
public double calculateRange() {
    // Assume current aircraft speed and fuel consumption
}

public double calculateRange(double speed) {
    // Provide speed, calculate new fuel consumption and return result
}

public double calculateRange(int speed) {
    // Provide speed as integer
}
```

Incorrect Method Overloading

```
public double calculateRange(double speed) {
    // Provide speed, calculate new fuel consumption and return result
}

public double calculateRange(double desiredConsumption) {
    // Provide consumption, calculate range a
}
```

Compile time error: Same number of arguments, same type => the compiler can’t tell them apart

Method Naming Conventions

Can contain any legal identifier

Use camel case (getData(), calculateDistance())

First word should be a verb

Passing Information to a Method

Parameter List

```
void logFuelCapacity(String file, int remainingFuel) {
    Path fileName = Path.of(file);
    String content = "Remaining fuel: " + remainingFuel;
    try { Files.writeString(fileName, content); }
    catch (IOException e) { e.printStackTrace(); }
}

logFuelCapacity(“fuel.log”, 13575);
```

This method has 2 parameters

Parameters have names

They are used in the method body

The values are passed into the method when we call it

```
public void logFuelCapacity(String file, int remainingFuel) {
    int remainingFuel = 1200;
}

public void logFuelCapacity(String file, File file) {
}
```

Names Must Be Unique

We can not declare a local variable with the same name as a method parameter

We can not reuse the same name for multiple parameters

```
class JetEngine {
    String model;


    JetEngine(String model){
        this.model = model;
    }
}
```

Field Shadowing

A parameter can have the same name as the field of a class

In this case, the parameter is said to shadow the field

Remove ambiguity by using the “this” keyword

Suitable for constructors

```
public void printRoute(String... points){
    for (String point : points) {
        System.out.println(point);
    }
}
printRoute("BUC", "YVAN", "HERAN");
printRoute("BUC", "YVAN", "HERAN", "CDG");
```

Arbitrary Number of Arguments

Construct called varargs

Use it when you don’t know how many arguments of a particular type will pe used

A shorthand for using an array

Passing Parameters

Pass by value

Pass by reference

Pass Primitive Types

Pass by Value

```
private void swap(int a, int b) {
    int aux = a; a = b; b = aux;
}

int a = 1; int b = 2;
swap(a,b);
System.out.println(a + ", " + b);
// Result is 1, 2
```

Primitive types are passed by value

A copy of that parameter value is made in memory

The caller and callee have independent variables with same value

Changes inside the method do not affect the outer variables

Pass Reference Types

Pass by Reference

```
class Coordinate {
    double x; double y;
}
void offset(Coordinate c){
    c.x += 0.1; c.y += 0.1;
}
Coordinate c = new Coordinate(0.0, 0.0);
offset(c);
System.out.println(c.x + ", " + c.y); // 0.1, 0.1
```

Reference data types are passed to methods by value

A copy of the memory location is passed into the method

Changes to the state of the object are persisted

You use the words “parameter” and “argument”. Is there a difference?

You

Parameter Vs. Argument

```
// a and b are parameters, a variable defined in
// the method declaration
private void swap(int a, int b) {
    int aux = a; a = b; b = aux;
}

// 1 and 2 are arguments, they are the value passed
// into the method
swap(1, 2);
```

The “this” keyword

this

A reference to the current object

When to Use “this”?

Access instance field

Call a constructor

Access Instance Field / Avoid Shadowing

```
class JetEngine {

    private int thrust;


    JetEngine(int thrust) {

        this.thrust = thrust;

    }

}
```

Explicit Constructor Call

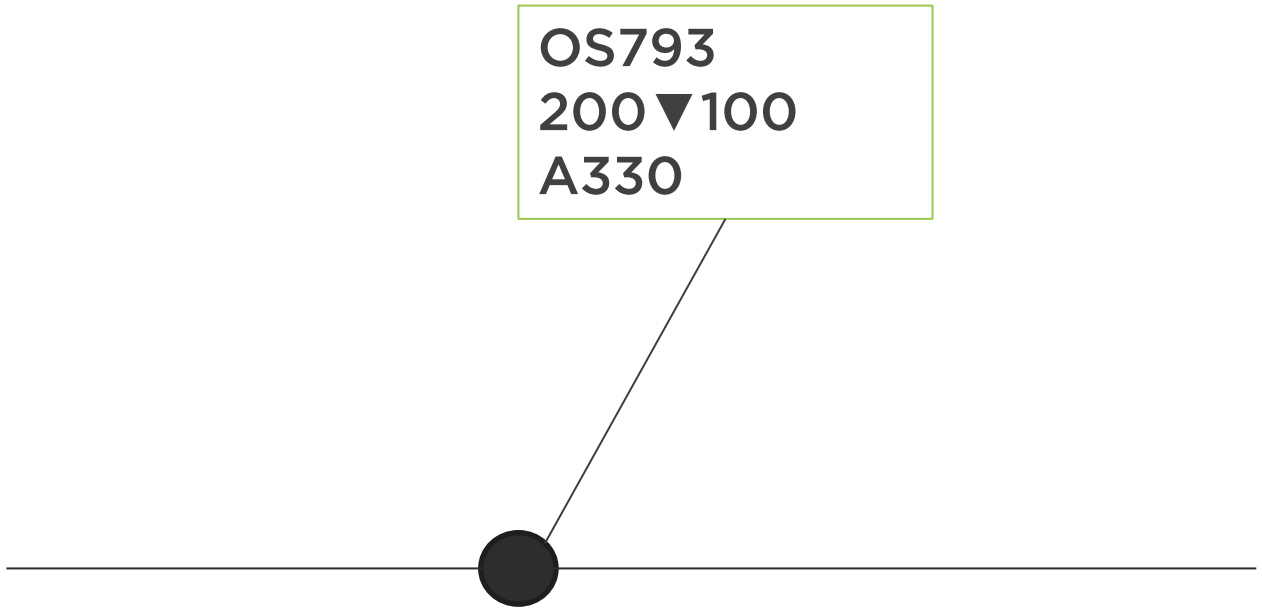
```
class JetEngine {
    private String id;
    private String model;

    public JetEngine(){
        this.id = UUID.randomUUID().toString();
    }

    public JetEngine(String model){
        this();
        this.model = model;
    }
}
```

Demo

Implement state and behavior

Model the target label

Implement commands



Summary

- Declare and use variables in Java
- Declare and use methods
- Method overloading
- Passing arguments to methods
- Understand the “this” keyword

Up Next:
Understanding Static Fields, Methods and Classes