# Introduction to Java programming

## Getting to know Java


Michiel Noback
Institute for Life Sciences and Technology
Hanze University of Applied Sciences

# Today: The Java platform and its syntax

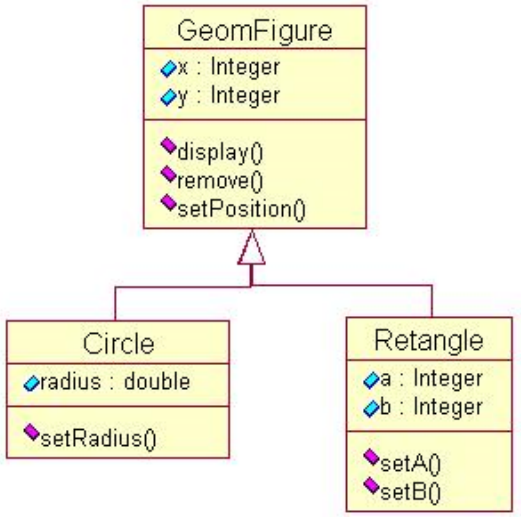
- Intro on the Java language
- What OO (pronounce OhOh) is
- Java syntax
  - data types
  - flow control
- Working with an Integrated Development Environment (IDE)

*Note: This course is based on the latest version of Java, Java8*

# The Java language and platform

- Java is
  - portable, platform independent
  - strongly typed
  - multithreaded
  - object-oriented
  - embeddable (electrical appliances, telephones)
  - the basis of Android

# What is OO(P)




Object-oriented programming (OOP) is a programming paradigm using "objects" – usually instances of a class – consisting of data fields and methods together with their interactions to design applications and computer programs.
Programming techniques may include (...) abstraction, encapsulation, (...), polymorphism, and inheritance.
*(wikipedia.org)*

# Hello, World, old school

- Create an text file called HelloWorld.java
- Open this file with a simple code editor
- Type in the following code

```
class HelloWorld{
  public static void main(String[] args){
    System.out.println("Hello, World!");
  }
}
```

# “Hello World”

- Save the file and start a terminal.
- Go to the folder where HelloWorld.java lives.
- Type the command
```
$ javac HelloWorld.java
```
- Type the command “ls”. You will see a new file has been created:

  “HelloWorld.class”

```
$ ls
HelloWorld.class
HelloWorld.java
```
- Type the command
```
$ java HelloWorld
```
- *Et voila*! Your first Java program has been run!

```
Hello, World!
```

# Java world: Who does what?

- Java programs run in the JVM (Java Virtual Machine). This is an OS emulator. The JVM runs your byte code from HelloWorld.class. That's why Java programs are *Platform independent*

```
$ java HelloWorld
```

Command `java` starts the java virtual machine

`HelloWorld` is the argument passed to the JVM. It says: take the bytecode from HelloWorld.class and execute it.

# Java world: Who does what?

- **javac**: is the java compiler which translates your source code (what you type) to byte code
- **HelloWorld.java** contains the source code
- **HelloWorld.class** contains the byte code
- Thus, Java is a compiled language!

```
$ javac HelloWorld.java
```

# HelloWorld.java code dissected

`class` signals the start of class code. A class is the blueprint for making objects. **No java code can exist outside classes!**

A Java application must have (at least) one **main()** method. It is the starting point of your application

```
class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello, world!");
    }
}
```

These are access and scope modifiers

System is the … system, and *out* is the stream to the console

**void** is the return type of main (here: nothing)

String[] args is the array of command-line arguments to **main()**

# Java is Strongly Typed

- In Java, you need to specify the type of everything that you work with
  - Variables have a type
  - Methods have typed arguments and a return type
- This is called **strongly typed**

```
String name = "FUBAR";
Duck getDuck(){}
void setAge(int ageToSet){}
```

# Where are the objects in this picture?

```
class HelloWorld{
   public static void main(String[] args){
      System.out.println("Hello, World!");
   }
}
```

# A first look at Objects

- **Objects** are pieces of your application that describe a **data type/entity** with its properties and functionality.
- **Modelling** in object-oriented programming is about recognizing what objects are present in your system and how they relate to and interact with each other

# A simple object: a Duck

- What IS a duck and what DOES a duck?
- **IS** = properties
  - speciesName
  - size
  - swimSpeed
- **DOES** = behaviour
  - swim()
  - fly()
  - quack()

- Only select properties and functionality that are relevant to your project!

https://commons.wikimedia.org/wiki/File:Male_mallard_duck_2.jpg

# Classes and objects

- A class contains the specifications how to create an object: A class is a **blueprint** for creating the actual stuff
- A class is defined once, Objects that are instantiated can number from 0 to a gazillion
- Objects are **instances** of a class

# A simple class: Duck.java

```
class Duck {
  int size;
  String name;

  void swim() {
    //swimming logic code goes here
    System.out.println("swimming like a " + name)
  }
}
```

These are the object properties or **instance variables**

the **`swim()`** method specifies some **behaviour**

# Creating Duck instances

```
class DuckTester {
  public static void main(String[] args) {
    Duck d = new Duck();
    d.size = 42;
    d.name = "mallard";
    d.swim();
  }
```

Create a Duck instance as specified in the Duck class

Give the Duck instance some property values and make it swim

```
$ javac Duck.java
$ javac DuckTester.java
$ java DuckTester
swimming like a mallard
```

# Being a good OO programmer

- The key to being a good OO programmer is being able to translate the problem at hand to a good object model
  - what are the objects
  - what properties and behaviour should they have (and make available to the outside world!)
  - what are the relationships between the objects

# Another class: a point in space

- What are the properties?
  - Coordinates: x, y and z
- What methods should such an object have?
  - Calculate distance to another point

```
class PointInSpace {
    int x, y, z;
    int calculateDistance(PointInSpace otherPoint) {
       //calculate distance
       return distance;
   }
}
```

Yes, classes can be this simple, even in real life!

# A composite object: a Square in space

- What are the properties?
  - Lower left and upper right corners: PointInSpace ll, PointInSpace ur
- What methods can you think of?
  - Calculate surface area

```
class SquareInSpace{
    PointInSpace ll, ur;
    int calculateSurface( ){
       //calculate surface
       return surface;
   }
}
```

This is an extremely important aspect of OO: ***build complex systems from simple building blocks***!

# Naming conventions

- Classes & source code files: use a noun, with first letter in capital. A flight simulator class will be named FlightSimulator, and its corresponding source file MUST be named FlightSimulator.java

# Naming conventions

- Method and variable names: start with lower case letter
- Methods should be verbs, like simulate()
- Variables should be nouns: numberOfKills or killCount
- Never start names with a number and do not use special characters! These are wrong: 1forTheMoney, k#ller

- **Be as descriptive as possible!**

# Three ways for commenting

```
1
2  /**
3   * Class HelloWorld, but with comments o
4   * This comment is generally used for cl
5   */
6  public class HelloWorldCommented {
7      /**
8       * This is the main method.
9       */
10     public static void main(String[] arg
11         /*Inside the main method. This is the second type of comment*/
12
13         //About to start printing. This is the third comment type
14         System.out.println( "Hello World!" );
15     }
16 }
```

Multiline comment for classes and methods
Will be used for JavaDoc so ALWAYS create these

General purpose multiline comment

Single-line comment

# More classes and objects

- Another example: Modeling the growth of cells in a test tube
- We will start off with two classes: class `Cell` and class `TestTube` and build on this in the next presentation

# A test tube for growing cells

```
/**
 * This is a TestTube. It will contain cells that can
 * grow and divide.
 */
class TestTube {
    /** here is the main method, as ordered.*/
    public static void main( String[] args ){
        TestTube testTube = new TestTube();
    }
}
```

Within the main method, an instance is created of the containing class
This construct is very often used and the reason for this will be addressed later

# Object construction

The new keyword combined with the class name followed by parentheses calls the constructor method of a class that instantiates and returns an object of that class:

**TestTube testTube = new TestTube();**

The parentheses () enclose the argument list for the constructor method, which is empty in this case.

# Object construction

- The three steps of object construction are
- **declaration**, **creation** and **assignment**:

```
TestTube tt = new TestTube();
```

(1) declare a variable of type TestTube (a reference)

```
TestTube tt = new TestTube();
```

(2) create an object of type TestTube and put it on the **heap**

```
TestTube tt = new TestTube();
```

(3) link the object and the reference

# Object construction

- So…where is this constructor method and what does it do?
- It is created automagically by the Java compiler, if you don't specify it yourself

- *More on constructors in a later presentation*

# Class Cell

```
/**
* Class cell simulates a growing cell
*/
class Cell {
    /*makes the cell grow*/
    void grow() {
        System.out.println("Growing bigger");
    }
}
```

# Growing cells

```
class TestTube {
    public static void main(String[] args) {
        TestTube testTube = new TestTube();
        testTube.growCells();
    }

    void growCells() {
        System.out.println("cells are growing");
        Cell cell = new Cell();
        cell.grow();
    }
}
```

# Single Responsibility Principle

- An object is a place to hold methods and data that are closely related to each other.
- These methods and properties should form a coherent unity.
- This is called the **Single Responsibility Principle (SRP)**

# Summary

In this presentation, you have seen:

- Java code resides in .java files
- .java files need to be compiled to .class files
- A Java application needs a main() method as starting point
- The essence of OOP is building complex systems from simple building blocks
- Everything you code should adhere to the SRP principle