Voortgezet Programmeren

Lecture 1: Introduction, elementary concepts in OOP

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

Course learning objectives

- Understand core concepts of the object-oriented paradigm (e.g. inheritance, interfaces, abstract classes, polymorphism, generics) and be able to program with these in Java
- Be able to use unit testing frameworks (such as JUnit) and develop software in a test-first manner
- Understand and be able to efficiently use classes from the Java Collections Framework

Course organization

- 7 lectures
  - Theory
  - Provide background for the exercises
- 6 exercise sessions
  - 6 large exercises done alone or in pairs
  - Come to exercises to ask questions and get help with your code
- 6 question hours
  - Office hours for the TAs to give answers to grading

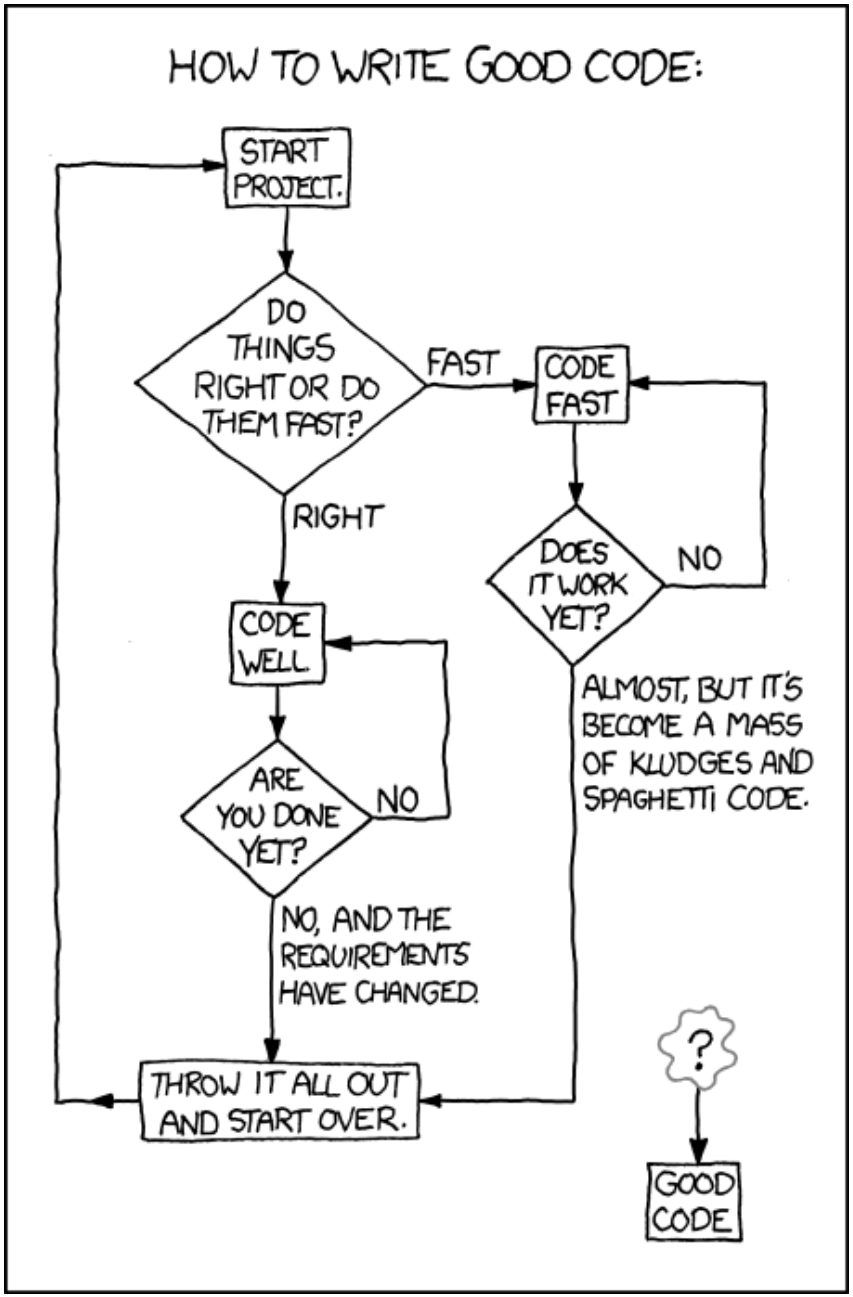
Study load

- 4 ECTS = 112h
- 7 lectures = 14h
- 6 exercise sessions = 12h
- 6 question hours = 6h
- Exam = 4h
- $\Rightarrow$ Independent programming 76h $\approx$ 13h/w

Grading

- Exercises: 50% (each $8.\bar{3}\%$)
  - Alone or in pairs
  - Published in BB after Monday's lecture
  - Strict deadline on Sundays @ 23.59
  - Submission via BB: *only* the source files in a ZIP. Include a comment in all files with your names and student numbers
  - Incorrect submission format = 0 points
  - Non-compiling code = 0 points
  - Crashing code = 0 points
  - Not adhering to good programming practices = max 6 points
- Written exam: 50%
  - Essay questions

Making the exercises



- Don't underestimate the importance of theory

```
if(stuck()) {
   askHelp() || fail();
}
```

- BB forums for discussion on the exercises (collaborate!)

ERASMUS UNIVERSITEIT ROTTERDAM

Plagiarism

- Do not submit anything you haven't written yourself
- Do not submit anything that is not your idea
- Co-operation is allowed
- "But I could've solved this problem myself, it was just faster to google the solution"
- All suspected plagiarism will be reported to the examination board

Course staff

Tommi Tervonen	Lectures & exercises	H11-26
Carlijn Liqui Lung	Exercises	H10-13
Weisheng Li	Exercises	H10-13
Zi-Zhao Chang	Exercises	H10-13
Ruben Janssen	Exercises	H10-13

- Also: you! Participate in course discussion forums in BB to get and provide help with the exercises
- TAs grade exercises and give feedback during question hours

Required knowledge

Inleiding programmeren:

- Variables and methods
- Program flow
- Decisions and branching
- Control structures
- Bitwise operators
- Arithmetic operators
- Scoping

Programmeren:

- Programming paradigms
- Typing
- Procedures/functions
- Memory organization
- Computational complexity
- Pre- and post conditions
- Side effects
- Unit testing (a bit)

Lectures

L1 Introduction, elementary concepts in OOP
- Practicalities
- Objects and classes
- Memory allocation and garbage collection
- Packages, arrays, ArrayList

L2 Errors, exceptions and streams
- Error handling
- Exception hierarchy
- Streams

L3 Programming by contract
- Data hiding
- Contract documentation
- Unit testing
- Class invariants
- Static variables and methods

Lectures

L4 Interfaces and polymorphism

- Interfaces
- Casting
- Polymorphism
- Inner classes

L5 Inheritance

- Inheritance hierarchies
- Overriding
- Subclass construction
- Polymorphism and inheritance

L6 Java Collections Framework

- Object identity
- Generics
- Collections, Lists, Sets, Maps
- Iterators

L7 Overview

Literature

- Lectures = main exam material
- Horstmann: Java Concepts (6th ed.), Wiley
- All course material is posted in `http://smaa.fi/tommi/courses/prog3/`
- If you don't know how computers work: LN-TT-22012-2 (`http://smaa.fi/static/prog2/ln-tt-22012-2.pdf`)

Software

- JDK v6+
- Exercises must compile & run with Sun JDK with JRE 1.6.0_26-b03 (default in Ubuntu with sun-java6-jdk package)
- The exercise sessions will be guided with Eclipse (`eclipse.org`)

Q?

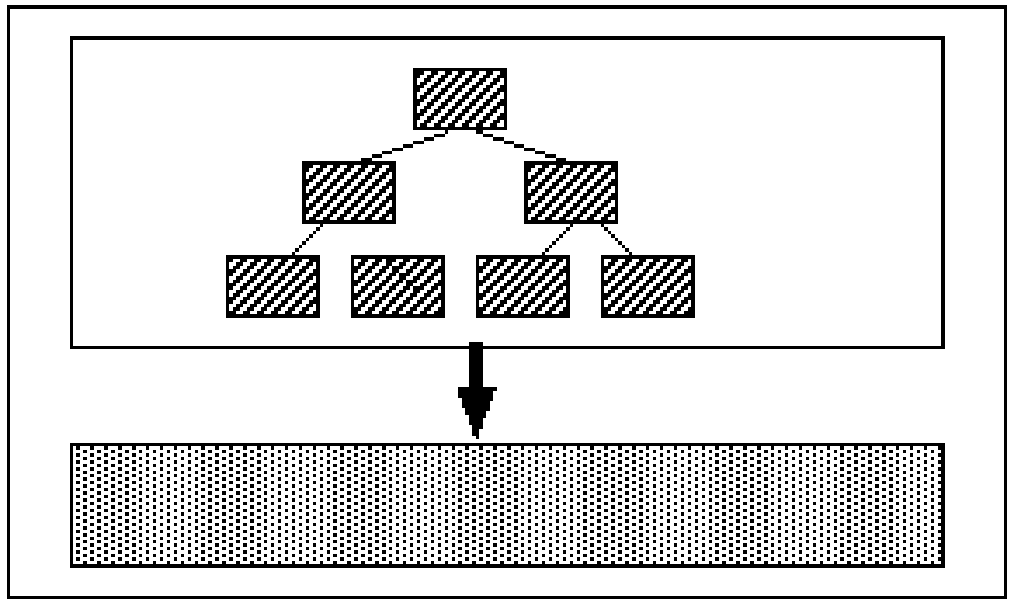
"The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer."

E.W. Dijkstra

Procedural vs OOP

- Procedural programming: data structures and methods to operate on them
- Object oriented paradigm: data and related methods are coupled on the language level

Procedural Languages

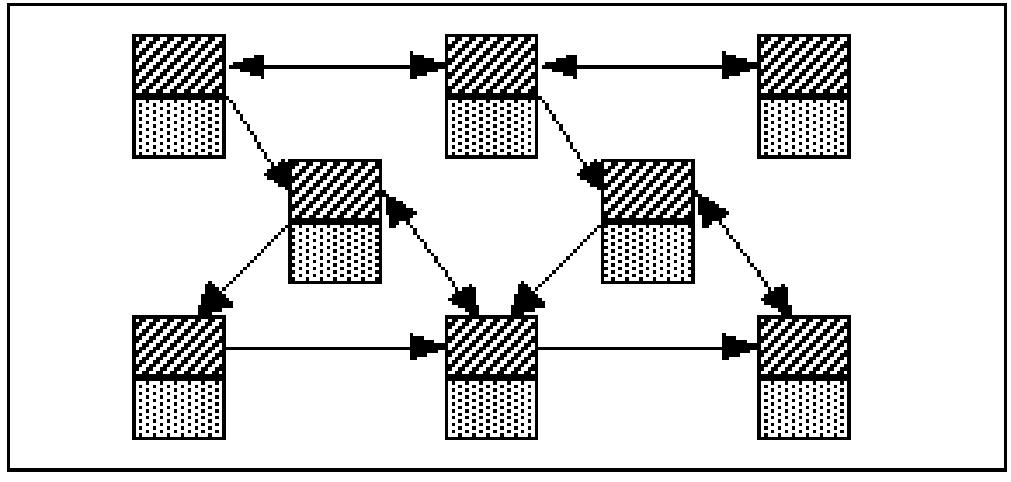


Computation involves code operating on Data

Code

Data

Object-Oriented Languages



An object encapsulates both code and data

Code

Data

Computation involves objects interacting with each other

```
function [ret] = subString(str, startIdx, endIdx)
  ret = '';
  for i=startIdx:(endIdx-1)
    ret = concat(ret, str[i]);
  end
end
```

```
public class MyString {
  private char[] data;

  public MyString(char[] contents) {
    data = contents;
  }
  public MyString subString(int start, int end) {
    char[] carr = new char[end-start];
    for (int i=start;i<end;i++) {
      carr[i-start] = data[i];
    }
    return new MyString(carr);
  }
  public String toString() {
    return new String(data);
  }
}
```

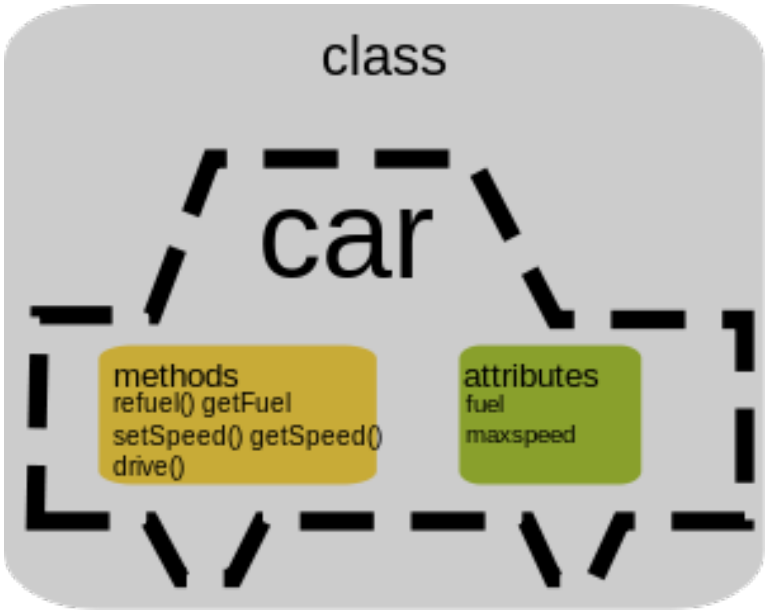
Forget everything you know about programming

Objects and classes

- **Classes** are blueprints for generating classes, the "design"
- **Objects** are instantiations of the classes
- Emphasis in OOP is on class design
- In program execution, objects communicate with each other through method calls
- In Java: 1 source file = 1 class

Class contents

- Attributes for data contents (variant between objects of the same class)
- Methods for behaviour (e.g. attribute access and manipulation)



- Java code convention: classes begin with an uppercase letter, methods and variables with lowercase ones. Multiple words = camelCasing.

Class declaration: instance variables (attributes)

```
public class Car {

    // maximum speed in km/h
    private int maxSpeed;

    // current fuel in percentages
    private double fuel;

    ...
}
```

Methods

- Methods are separated to accessor- (functions) and mutator (procedures) methods
  - Accessor methods return a value but do not the change state of the object
  - Mutator methods change the state of the object, but do not return a value
- Not enforced on language level!

Example: accessor- and mutator methods

```
public class Car {
   ...
   public void drive(double perc) {
      fuel -= perc;
   }
   public void refuel() {
      this.fuel = 100.0;
   }
   public double getFuel() {
      return fuel;
   }
   public void setSpeed(int newSpeed) {
      maxSpeed = newSpeed;
   }
   public int getSpeed() {
      return maxSpeed;
   }
}
```

Constructor

- Classes have a special method with a name of the class, that is called when a new instance is generated

```
public class Car {

    /**
     * Constructs a new car with given max speed and
     * a full tank of fuel.
     *
     * @param maxSpeed maximum speed in km/h
     */
    public Car(int maxSpeed) {
        this.maxSpeed = maxSpeed;
        fuel = 100.0;
    }

    ...
}
```

```
Car mySeat = new Car(189);

// I'm driving to university, take away fuel
mySeat.drive(1.0);
// Tank
mySeat.refuel();

System.out.println("My seat has currently "
+ mySeat.getFuel() + "% fuel");
```

```
public class Car {
  // maximum speed in km/h
  private int maxSpeed;
  // current fuel in percentages
  private double fuel;
  /**
   * Constructs a new car with given max speed and
   * a full tank of fuel.
   *
   * @param maxSpeed maximum speed in km/h
   */
  public Car(int maxSpeed) {
    this.maxSpeed = maxSpeed;
    fuel = 100.0;
  }
  public void refuel() {
    this.fuel = 100.0;
  }
  ...
}
```

Code documentation

- Code is not complete without documentation
- Javadoc is a standard way that can be used to automatically generate documentation in e.g. html
- What you should document:
  - methods (always)
  - instance variables (if unclear)
  - classes (always, to include @author)
  - in-line comments (if unclear)
- Method **signature** describes **how** to call it, not **what** it does

```
public int getSpeed() { ... }
```

Class documentation

```
/**
 * Models a single car with top speed and fuel.
 *
 * @author Tommi Tervonen <tervonen@ese.eur.nl>
 */
public class Car {
   ...
}
```

Method documentation

```java
/**
 * Sets the top speed.
 *
 * @param newSpeed new top speed in km/h
 */
public void setSpeed(int newSpeed) {
    maxSpeed = newSpeed;
}
/**
 * Gives the top speed.
 *
 * @return top speed in km/h
 */
public int getSpeed() {
    return maxSpeed;
}
```

Object references

- Computer memory is linear (c.f. LN-TT-22012-2)
- Primitive type variables (int, double, char) are references to contents: always copied when reassigned
- Object type variables are references to the actual objects: when copied, only the reference is reassigned

On immutability

- String is a standard class in java although it has non-standard implicit constructor “contents”
- Strings are immutable: once constructed, their contents cannot change
- Our Car was mutable (setSpeed, drive)

Memory allocation and garbage collection

```
String name1 = new String(``tommi'');
String name2 = new String(``alex'');
name2 = name1;
name1 = null;
```

Packages in Java

- Multiple classes can have same name, as long as they are in different packages
- Same package classes are automatically in the same namespace
- Others you need to import or refer to them explicitly (java.util.ArrayList)
- Some standard library packages:
  - `java.lang` (core classes, always in the namespace)
  - `java.util`
  - `java.io`
  - `java.math`
- Convention: name package according to domain in inverse order (`fi.smaa.jsmaa`)

Arrays in Java

- Special objects with publicly visible (immutable) field `length`
- Fixed length
- Example: `int[] myNumbers = new int[10];`
- Object arrays do **not** construct the objects to be stored (example: `String[] myNames = new String[3];`)
- `String[] myNames = new String[] {"tommi", "alex", "fred"};`
- Multidimensional arrays do not exist within computers (c.f. LN-TT-22012-2)

ArrayList<T>

- A collections framework class that implements a dynamically allocated list (array that can grow/shrink) to store objects of a certain type `T`

```
ArrayList<String> myStrs = new ArrayList<String>();
myStrs.add(``s1'');
myStrs.add(``s2'');

for (String s : myStrs) {
  System.out.println(s);
}
```

Now

- Download, install Eclipse
- Make exercise #0 (not graded)
- Start with exercise #1 (graded)
- Questions? Use the BB forums