

# Advanced Programming

## Lecture 1: Introduction, elementary concepts in OOP

Tommi Tervonen

Econometric Institute, Erasmus School of Economics

# 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

- 7 lectures
  - Theory
  - Provide background for the exercises
- 4 exercise sessions
  - 4 large exercises done alone or in pairs
  - Come to exercises to ask questions and get help with your code

- 4 ECTS = 112h
- 7 lectures = 14h
- 4 exercise sessions = 8h
- Exam = 3h
- $\Rightarrow$  Independent programming 87h  $\approx$  22h/w (for 4 weeks)

Tommi Tervonen	Lectures (& exercises)	H11-26
Ruben Janssen	Exercises	-

- Also: you! Participate in course discussion forums in BB to get and provide help with the exercises

- Exercises: 50% (each 12.5%)
  - Alone or in pairs
  - Published in BB, latest after Tuesday's lecture
  - **Strict** deadline on following week's Sunday @ 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 10 points
- Written exam: 50%
  - Open questions (like in 2nd year programming)

- 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

## 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)



## L1 Introduction, elementary concepts in OOP

- Practicalities
- Objects and classes
- Memory allocation and garbage collection
- Packages

## L2 Errors, exceptions and streams

- Arrays, ArrayList
- Error handling
- Exception hierarchy
- Streams

## L3 Programming by contract

- Data hiding
- Contract documentation
- Unit testing
- Class invariants
- Static variables and methods

## 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

- Lectures = main exam material
- Horstmann: Java Concepts (6th ed.), Wiley (or almost any other decent Java book)
- All course material is posted in <http://smaa.fi/tommi/courses/prog3/>
- If you don't know how computers work: LN-TT-22012-3 (<http://smaa.fi/static/prog2/ln-tt-22012-3.pdf>)

- 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](http://eclipse.org))

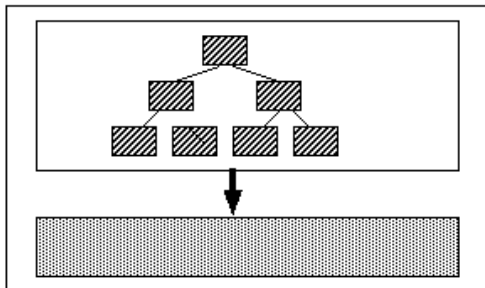
Q?

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

## Procedural Languages

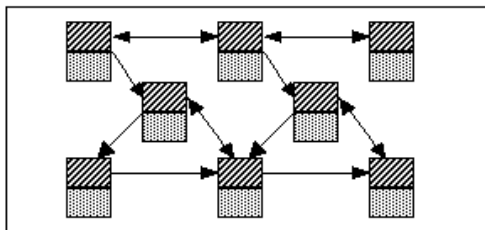


Computation involves code operating on Data

 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
```

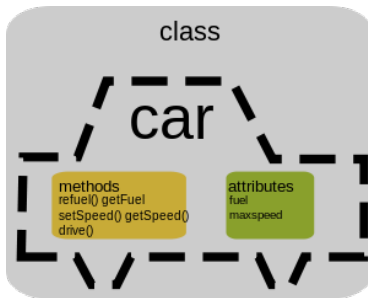
```
public class MyString {  
    private char[] data;  
  
    public MyString(char[] contents) {  
        data = contents;  
    }  
    public MyString subString(int start, int end) {  
        char[] carr = new char[end-start];  
        for (int i=start; i<end; i++) {  
            carr[i-start]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

Forget everything you know about  
programming

- **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 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;  
    }  
}
```



- 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 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 () { ... }
```

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

# Method documentation

```
/**
 * 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;
}
```

- Computer memory is linear (c.f. LN-TT-22012-3)
- 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

- 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;  
// now: name2 -> "tommi" <- 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`)

- Download, install Eclipse
- Start thinking about exercise #1 (posted online later on today)
- Questions? Use the BB forums

