Voortgezet Programmeren

Lecture 1: Elementary concepts in OOP

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

Erasmus
ERASMUS UNIVERSITEIT ROTTERDAM

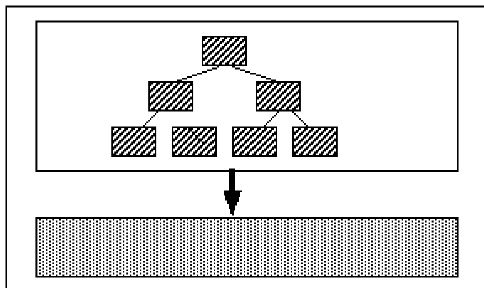
Procedural vs OOP

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

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

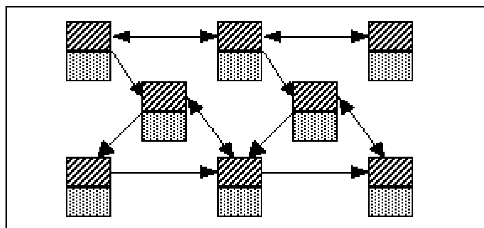
```
public class MyString {
  private char[] data;

  public MyString(char[] contents) {
    data = contents;
  }
  public MyString subString(start, 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);
  }
}
```

Procedural Languages

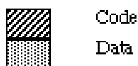


Computation involves code operating on Data

Object-Oriented Languages



An object encapsulates both code and data



Computation involves objects interacting with each other

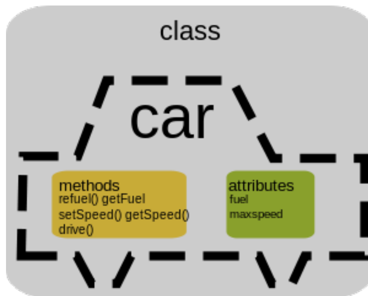
Forget everything you know about programming

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- and mutator 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;
  }
}
```

Erasmus Universiteit Rotterdam

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;
  }

  ...
}
```

Erasmus Universiteit Rotterdam

```java
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");
```

```java
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 {
    ...
}
```

Erasmus
ERASMUS UNIVERSITEIT ROTTERDAM

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-22012-1)
- 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

```java
public class Course {
  private String name;
  public Course(String name) {
    this.name = name;
  }
}
public class Student {
  private String name;
  private int id;
  private Course major;

  public Student (String name, int id, Course m) {
    this.name = name;
    this.id = id;
    this.major = m;
  }
}
```

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)

ERASMUS UNIVERSITEIT ROTTERDAM

Memory allocation and garbage collection

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

Introduction to arrays in Java

- Arrays are special type of objects (with public final attribute length)
- When allocated, null objects are included
- Arrays are indexed starting from 0 (until length-1)
- Example: array allocation and traversal with for-loop