Overview

- Administrative
- Discussion on the lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
Topics

- Introduction to Programming
  - Object Oriented Programming (OOP), Modeling, Formalization, Abstraction

- Algorithms in Java
  - Search, Recursion, Backtracking
  - Complexity Analysis

- Data Structures
  - Trees, Heaps, Lists

- Others: Simulation, Testing/Debugging, Parallel Programming
Administrative

- Place: HG F 26.3
- Time: Wednesdays at 13:15 – 14:00
  13:00 – 14:00

- Weekly submission deadline: Tuesday at 23:59
  - ETH Code Expert

- Individual submissions!
  - NOT in groups!

No photos of theoretical exercises. Use scanner.
Try to submit a good quality photo!
Submissions

- Late submissions are not accepted

- Plagiarism = ZERO Points for the Exercise Sheet

- Discussion with Prof. Mattern!
Overview

- Administrative
- Discussion on the lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
package u0a1;

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

Hello World!
Java vs. C++

- No `#define`, `typedef`
- No `structures` or `unions` (classes)
- No `enum` types (class with constants)
- No “functions”. Java uses classes and methods
- No `multiple inheritance`
- No `goto` statements
- No pointers
- No manual memory management needed (no `malloc`, `free`, `delete`)
JAVA files

- .java
  - source code files

- .class
  - (Bytecode) files generated by compiling .java files

- .jar
  - Package file (aggregate)
  - Applications or libraries
A Java program runs in principle on all major computers and operating systems (PC, Server, phone, Linux, Windows, etc..)

Bytecode is interpreted by a virtual machine (VM). All VMs understand the same language, the Bytecode.
Call by reference vs. call by value

- Object manipulation is done by reference and all object variables are references

- In methods, Java references are passed by value
Example

```java
public void foo(Dog someDog) {
    someDog.setName("Max"); // AAA
    someDog = new Dog("Fifi"); // BBB
    someDog.setName("Rowlf"); // CCC
}

Dog myDog = new Dog("Rover");
foo(myDog);

What would happen in the following case?

System.out.println(myDog.getName());
Max
```
Overview

- Administrative
- Discussion on the lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
Eclipse IDE

- New Project from external resource
- Exercise data Import/Export
- Code Execution + Run Configurations
- Debugging
JUnit 4

- Library for automatic verification and validation of software
- Supplied with Eclipse as Plugin

Integrated into Eclipse Project:
1. Right Click on Project -> Build Path -> Configure Build Path
2. Add Libraries...
3. «JUnit»
4. «JUnit 4»
Demo
Overview

- Administrative
- Discussion on the lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
Exercise 0

- Ex 1: HelloWorld.java

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

- Ex 2: Signum.java + Main.java

```java
public static void main(String[] args) {
    System.out.println("signum(-3) = "+ Signum.signum(-3));
    System.out.println("signum(0) = "+ Signum.signum(0));
    System.out.println("signum(7) = "+ Signum.signum(7));
}
```

- Ex 3: Automatic testing
- Ex 4: Small theory exercise, model building
Solution U0.A4a and U0.A4b
Solution U0.A4c

- Maximum number of transitions
  Sol: 6 (see Graph)

- Average number of transitions:
  Sol:
  \[
  \frac{\sum f(x)x}{\sum f(x)} = \frac{1\times2 + 2\times3 + 3\times2 + 4\times2 + 5\times2 + 6\times2}{2 + 3 + 2 + 2 + 2 + 2} = \frac{44}{13} \approx 3.38
  \]
  
  x = Path length
  f(x) = Occurrences of paths with length x
Overview

- Administrative
- Discussion on the lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
Exercise 1

- 1. Termination & correctness proofs (Egyptian Multiplication)
- 2. Recursion, Cost estimation of algorithms
- 3. Exceptions, Unit-Testing (explained before), Documenting Programms
Exercise 1 – Termination and Correctness

- Egyptian Multiplication

\[ f(a, b) = \begin{cases} a, & \text{falls } b = 1 \\ f(2a, b/2), & \text{falls } b \text{ gerade} \\ a + f \left( 2a, \frac{b-1}{2} \right), & \text{sonst} \end{cases} \]

- Proof by induction

  - The basic step (Induktionsanfang): showing that the statement holds for lowest value of \( n \) (\( n = 0 \) or \( 1 \))
  - The inductive step (Induktionsschritt): Showing that statement holds for \( n+1 \) assuming it holds for \( n \)
  - The assumption in the inductive step that statement holds for some \( n \) is called the induction hypothesis (Induktionsannahme)

a) Can you prove the correctness over \( a \)? Give reasons for this.

b) Does the algorithm terminate? Prove it!

c) Now we change the algorithm! Base Case to \( (b = 0) \) instead of \( (b = 1) \)!

  New proof requirements...
Exercise 1 – Algorithm cost estimation

- How many method calls produced by ...

  gerade(int x)
  verdopple(int x)
  halbiere(int x)

  f(int a, int b)   Without considering recursion!
  f(int a, int b)   With recursion!
Exercise 1 – Exceptions

- “An exception is an event that occurs during the execution of a program that 
  disrupts the normal flow of instructions.”

- Exceptions can be thrown (throw) and caught (catch)

- Tutorial: http://docs.oracle.com/javase/tutorial/essential/exceptions/
Exercise 1 – Javadoc

- Semantic Annotation
  - @author name
  - @version version
  - @since JDK-Version
  - @param name description
  - @return description
  - @exception classname description

- Generate Javadoc for the whole project:
  
  Project -> Generate Javadoc
Have Fun!
Eclipse

www.eclipse.org
Eclipse IDE for Java Developers
New Java Project

1. Extract u0.zip to D:\projects\u0
JUnit 4 should be specified!
New Java Project

Run as Java Application

Java „perspective“ for Java Development

Editor
CTRL+SHIFT+F
For autoformat

Output Console
Debug as Java Application

Debug "perspective" for Debugging

Breakpoint

Output Console
Junit Tests

Run Tests.java as Junit test

All tests passed

JUnit test methods

import org.junit.Test;

public class Tests {
    @Test public void negative() {
        Assert.assertEquals(-1, Signum.signum(-2));
    }
    @Test public void zero() {
        Assert.assertEquals(0, Signum.signum(0));
    }
    @Test public void positive() {
        Assert.assertEquals(1, Signum.signum(7));
    }
}
Javadoc

ALT+SHIFT+J to add Javadoc