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
OOP characteristics

- A: Abstraction
- P: Polymorphism
- I: Inheritance
- E: Encapsulation
Administrative

- Place: HG D 3.1
- Time: Wednesdays at 13:00 – 14:00
- Submissions in groups of two

Weekly submission deadline: Wednesday at 13:00
- By e-mail
- In person to the TA at the beginning of the tutorial
- In the mailbox, CNB H

No photos of theoretical exercises. Use scanner.
<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biancardi</td>
<td>Aristotele</td>
<td></td>
</tr>
<tr>
<td>Bronz</td>
<td>Omar</td>
<td></td>
</tr>
<tr>
<td>Cacciatore</td>
<td>Marco</td>
<td></td>
</tr>
<tr>
<td>Chakraborty</td>
<td>Ritika</td>
<td></td>
</tr>
<tr>
<td>Della Briotta</td>
<td>Sebastian</td>
<td></td>
</tr>
<tr>
<td>Ferrazzini</td>
<td>Luca</td>
<td></td>
</tr>
<tr>
<td>Freiburghaus</td>
<td>Jonas</td>
<td></td>
</tr>
<tr>
<td>Hodel</td>
<td>Carlo</td>
<td></td>
</tr>
<tr>
<td>Huber</td>
<td>Fabian</td>
<td></td>
</tr>
<tr>
<td>Inauen</td>
<td>Gregory</td>
<td></td>
</tr>
<tr>
<td>Kellermann</td>
<td>Andreas</td>
<td></td>
</tr>
<tr>
<td>Rossli</td>
<td>Felix</td>
<td></td>
</tr>
<tr>
<td>Salzmann</td>
<td>Tiago</td>
<td></td>
</tr>
<tr>
<td>Schefer</td>
<td>Philip</td>
<td></td>
</tr>
<tr>
<td>Scheidegger</td>
<td>Robin</td>
<td></td>
</tr>
<tr>
<td>Schmid</td>
<td>Marco</td>
<td></td>
</tr>
<tr>
<td>Stulz</td>
<td>Micha</td>
<td></td>
</tr>
<tr>
<td>Waupotitsch</td>
<td>Christoph</td>
<td></td>
</tr>
<tr>
<td>Yogarajah</td>
<td>Nivethan</td>
<td></td>
</tr>
<tr>
<td>Zellweger</td>
<td>Alick</td>
<td></td>
</tr>
<tr>
<td>Muller</td>
<td>Gabriel</td>
<td></td>
</tr>
</tbody>
</table>
Submissions

- Exercise submission
  - By e-mail to: subho.basu@inf.ethz.ch
  - Subject: Info2_<Assignment no>_<Group no> e.g. “Info2_U1_G2”
  - Attachments:
    - Theoretical answers: Uxx_Gyy.pdf
    - Source Code: Uxx_Gyy.zip (Right click on the project from the IDE, Export > General > Archive File)

- Questions (Exercises/Lectures)
  - Slides online: http://www.vs.inf.ethz.ch/edu/FS2016/I2/
  - E-mail

Group Name should exist in every document and be consistent over the semester!
Exercises

- Exercises aligned to end semester tests
- Late submissions are not accepted
- No plagiarism
Overview

- Administrative
- Discussion on the Lecture
- Introduction to: Eclipse, JUnit, Debugging
- Debriefing Exercise 0
- Briefing Exercise 1
Hello World!

```java
package u0a1;

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

- Compiled in C++ vs Interpreted in Java
- C++ is platform dependent, Java is platform independent
- No `#define`, `typedef` or `preprocessor directives`
- No `structures` or `unions` (Only classes)
- No `enum` types (class with constants)
- No “functions”. Java uses classes and methods
- No `multiple inheritance`
- No `goto` statements (Keyword but not used)
- No pointers.
- No operator overloading in Java
- No manual memory management needed (no `malloc`, `free`, `delete`). Java has its own garbage collector that cleans up memory automatically by removing unreferenced objects.
JAVA files

- .java
  - source code files

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

- .jar
  - Libraries
Java

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 always 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?

```java
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:

\[
\frac{\sum f(x)x}{\sum f(x)} = \frac{1\cdot2+2\cdot3+3\cdot2+4\cdot2+5\cdot2+6\cdot2}{2+3+2+2+2+2+2} = \frac{44}{13} \approx 3.38
\]

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 basis (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 – Algorithms Cost Estimation

- How many method calls produced by ...

  \begin{align*}
  \text{gerade}(\text{int } x) \\
  \text{verdopple}(\text{int } x) \\
  \text{halbiere}(\text{int } x) \\
  \text{f}(\text{int } a, \text{int } b) & \quad \text{Without recursion!} \\
  \text{f}(\text{int } a, \text{int } b) & \quad \text{With recursion!}
  \end{align*}
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)

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

Debug as Java Application

Debug „perspective” for Debugging

Breakpoint

Output Console

Informatik II – Tutorial 1

02.03.2016
JUnit Tests

Run Tests.java as Junit test

All tests passed

JUnit test methods

02.03.2016

Informatik II – Tutorial 1
Javadoc

**Javadoc description**

**Javadoc tab**

ALT+SHIFT+J to add Javadoc