OOP, Java und Ant
Datamining und Sequenzanalyse

Marvin Meusel, Kai Dührkop

21.10.2013
OOP in Java
Object-Oriented Programming (OOP)

**OOP Principles**
- Abstraction
- Encapsulation
- Inheritance
- Subtype Polymorphism

**The OOP World**
- Class
- Object
- Attribute
- Method
A house
- ... has a roomnumber
- ... has a size (maximal number of persons)
- ... contains a specific number of persons

Properties
- rooms
- persons

Methods
- personEnters
- personExits
Inheritance and Subtype Polymorphism

- Subclasses inherit both, methods and properties.
- Java has the `instanceof` keyword to check is a conditions
- Static vs. Instance
- Interfaces instead of multiple inheritance.
- Methods can be overridden
- **Abstract** keyword
# Encapsulation and Abstraction

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class</th>
<th>Package</th>
<th>Subclass</th>
<th>World</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>protected</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>default</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>private</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Benchmarking
How to Measure

- via command line **time**
  
  ```
  #$ time sleep 10
  real 0m10.116s
  user 0m0.001s
  sys 0m0.007s
  ```

- via Java
  - `System.currentTimeMillis()` - time in milli seconds
  - `System.nanoTime()` - time in nano seconds
  - Example:
    ```java
    long start = System.nanoTime();
    <long time job>
    long duration = System.nanoTime() - start;
    ```
Describe the Environment

- Operating System
- Memory
- CPU(s)
- Java VM version
- Heapspace
JavaDoc
JavaDoc

- Document your work
- Documentation is embedded into your source code
- Create a browsable HTML based documentation
What can be documented

Classes

```java
/**
 * Models a room.
 * 
 * @author Thasso Griebel (thasso@minet.uni-jena.de)
 * 
 */

public class Room {

}
```

Variables

```java
/**
 * The Room number
 */

private int roomNumber;
```

Constructors

```java
/**
 * Creates a new room of given size.
 * <p>
 * Size has to be > 0
 * @param size
 */

public Room(int size){
    super();
    setSize(size);
    this.persons = new ArrayList<Person>();
}
```

Methods

```java
/**
 * Adds a {link Person} to this room. If the person is located
 * somewhere else, the Person is removed from its old location and
 * transferred to this room.
 * 
 * @param person the Person to add
 * @throws exception if the room is full
 * @return true if person successfully entered the room
 */

public synchronized boolean addPerson(Person person) throws Exception{
    if(person == null) return false;
    if(person.getLocation() != null){
        ....
        return persons.add(person);
    }
```
Ant
Features:

- Java-based build tool
- Platform independent
- Uses XML
- Java Class like extensions
- Supports around 150 operations.
- Compiles fast
- Optimized for Java
The Build File: build.xml

**Structure**
- Projects
- Targets
- Tasks
- Properties
### Common tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>javac</td>
<td>move</td>
</tr>
<tr>
<td>copy</td>
<td>exec</td>
</tr>
<tr>
<td>delete</td>
<td>zip</td>
</tr>
<tr>
<td>mkdir</td>
<td>replace</td>
</tr>
<tr>
<td>junit</td>
<td>mail</td>
</tr>
</tbody>
</table>
<project name="foobar" default="run" basedir="."/>

...
Properties and Locations

<project name="foobart" default="run" basedir=".">
  <property name="src.dir" location="src" />
  <property name="target.dir" location="target" />
  <property name="lib.dir" location="lib" />
</project>
(Class)paths

<project name="foobar" default="run" basedir=".">
  ...
  <property name="lib.dir" location="lib" />

  <path id="classpath">
    <fileset dir="${lib.dir}" includes="**/*.jar" />
  </path>

</project>
<project name="foobar" default="run" basedir=".">
  ...
  <property name="target.dir" location="target" />

  <target name="init">
    <mkdir dir="${target.dir}" />
  </target>

</project>
<project name="foobar" default="run" basedir=".">

...<target name="compile" depends="init"
    description="Compile all source files">

    <javac srcdir="${src.dir}"
        destdir="${target.dir}"
        classpathref="classpath" />

</target>

</project>
<project name="foobar" default="run" basedir=".">
  <target name="run" depends="compile"
    description="Run FooBar">
    <java classname="foo.Bar" fork="true">
      <arg value="-h" />
      <classpath>
        <pathelement location="${target.dir}" />
        <pathelement path="${java.class.path}" />
        <path refid="classpath" />
      </classpath>
    </java>
  </target>
</project>
<project name="foobar" default="run" basedir=".">
  ...
  <target name="clean"
    description="Deletes all .class files">
    <delete dir="${target.dir}" />
  </target>
</project>
Whats there, whats where

#$>ant -p
Buildfile: build.xml

Main targets:

  clean   Deletes all .class files
  compile Compile all source files
  run     Run FooBar
Default target: run
#$>_