Java und Ant
Datamining und Sequenzanalyse

Florian Rasche, Sascha Winter

24.10.2010
Java

(http://www.java.com/)
Object-Oriented Programming (OOP)

- **OOP Principles**
  - **Abstraction**: think of interfaces and functions instead of implementation details
  - **Encapsulation**: hiding data or information by restricting access to methods (getter/setters)
  - **Inheritance**: Objects inherit attributes and functionality
  - **Subtype Polymorphism**: one function name for all subclasses, many different executions
Object-Oriented Programming (OOP)

- **OOP Principles**
  - **Abstraction**: think of interfaces and functions instead of implementation details
  - **Encapsulation**: hiding data or information by restricting access to methods (getter/setters)
  - **Inheritance**: Objects inherit attributes and functionality
  - **Subtype Polymorphism**: one function name for all subclasses, many different executions

- **The OOP World**
  - **Class**: defines the abstract characteristics of a thing (object)
  - **Object**: a particular instance of a class
  - **Attribute**: an object’s properties.
  - **Method**: an object’s abilities
Lets build a house
A house

- ... has a room number
- ... has a size (maximal number of persons)
- ... contains a specific number of persons
Let's build a house

A house

- ... has a room number
- ... has a size (maximal number of persons)
- ... contains a specific number of persons

Properties

- roomNumber
- size
- persons
A house

- ... has a roomnumber
- ... has a size (maximal number of persons)
- ... contains a specific number of persons

Properties

- roomNumber
- size
- persons

Methods

- person_enters
- person_exits
Inheritance and Subtype Polymorphism

- A Lab is a Room
- A BioLab is a Lab is a Room
- An Office is a Room, but is NOT a Lab

Objects inherit both, methods and properties.
Inheritance and Subtype Polymorphism

- A Lab is a Room
- A BioLab is a Lab is a Room
- An Office is a Room, but is NOT a Lab

Objects inherit both, methods and properties.

- Java has the `instanceof` keyword to check is a conditions

- Static vs. Instance classes/methods/attributes
Encapsulation and Abstraction

- **Encapsulation:**
  - Access can be restricted using access modifiers

<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>no modifier</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>
Encapsulation and Abstraction

**Encapsulation:**
- Access can be restricted using access modifiers

<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>no modifier</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>

**Abstraction:**
- Specify what to do, not how to do it.
- Specify what a method does, not how to do it.
- Specifications do not indicate how to implement a method.
Mored Design

Room
- roomNumber
- size
- persons
- person_enters
- person_exits

Contains Persons
- can enter
- can leave

Person
- name
- activity
- location
- enter_room
- exit_room

Consists of rooms

House
- rooms
- persons

Contains Persons

Java Benchmarking JavaDoc Ant
Java und Ant
Benchmarking
How to Measure

- via command line **time**
  
  ```
  #$ time sleep 10
  real 0m10.116s
  user 0m0.001s
  sys 0m0.007s
  ```
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
  (resolution is about 16 ms at Windows XP)

  System.nanoTime() - time in nano seconds (resolution is about 2-3 ms at Windows XP)

  Example:
  ```java
  long start = System.currentTimeMillis();
  long duration = System.currentTimeMillis() - start;
  ```
What to Measure

- Measure the right thing
- ... at the right location
- ... more than one time
Describe the Environment

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

(http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html)
JavaDoc

- Document your work
- Create a browsable HTML based documentation
- Documentation is embedded into your source code
- ... and your IDE can help you!
What can be documented

Classes

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

}</code>

Variables

```java
/**
 * The Room number
 */
private int roomNumber;
```

Constructors

```java
/**
 * Creates a new room of given size.
 * @param size
 */
public Room(int size){
    super();
    setSize(size);
    this.persons = new ArrayList<Person>();
}
```

Methods

```java
/**
 * Adds a [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);
    }
```
How to create the documentation

- Use the command line tool javadoc
- Use your IDE
- Use Ant
Ant

(http://ant.apache.org)
"Apache Ant is a Java-based build tool. In theory, it is kind of like Make, but without Make’s wrinkles."

**Pros:**
- Java Class like extensions
- Uses XML
- Standardized
- Platform independent
- Compiles fast

**Cons:**
- Only for Java
- Property value assignment very simple and inflexible

**Get Ant:**
- Often already included in your IDE
- [http://ant.apache.org](http://ant.apache.org)
Ant uses a simple XML file to
- define Projects that
- contain Targets that
- use Tasks
Do it by hand vs. Ant

What we do... ... can be done with Ant:

- Call the compiler (javac -cp...)
- Run the Program (java -cp...)
- Remove .class files (rm -R *.class)
- Build a Jar File (jar -f foobar.jar...)

Write a file that defines:

- Where the source files are located
- Where the class files should be located
- Where are libraries located
- How to call the compiler
- How to call java
- How to do a cleanup
Do it by hand vs. Ant

What we do... ... can be done with Ant:

- Call the compiler (javac -cp ...)
- Run the Program (java -cp ...)
- Remove .class files (rm -R *.class)
- Build a Jar File (jar -f foobar.jar ...)

What we do... can be done with Ant:
Do it by hand vs. Ant

What we do...

- Call the compiler (javac -cp ...)
- Run the Program (java -cp ...)
- Remove .class files (rm -R *.class)
- Build a Jar File (jar -f foobar.jar ...)

... can be done with Ant:

Write a file that defines:

- Where the source files are located
- Where the class files should be located
- Where are libraries located
- How to call the compiler
- How to call java
- How to do a cleanup
The Build File: build.xml

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

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

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

</project>
```
Get Prepared

```xml
<project name="foobar" default="run" basedir=".">
  ...
  <property name="target.dir" location="target" />
  <target name="init">
    <mkdir dir="${target.dir}" />
  </target>
</project>
```
Get Compiled

```xml
<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="foobart" default="run" basedir=".">

  ...

  <target name="clean"
        description="Deletes all .class files">
    <delete dir="${target.dir}" />
  </target>

</project>
What's there, what's where

#$> ant -p
Buildfile: build.xml

Main targets:

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