W Überblick über die 6. Übung

- Lösung der 4. Aufgabe
- ORB
- ANT

W.1 Lösung der 4. Aufgabe

- Teilaufgabe (a) [1 Punkt]
  - Aufteilung der Klassen auf Pakete
  - - 0.5 Punkte: alte Dateien nicht entfernt

W.1 Lösung der 4. Aufgabe (2)

- Teilaufgabe (b) [4 Punkte]
  - 2 Punkte:
    Entfernen: **`addShape(String)`** in Klasse **`ShapeContainer`**
    Hinzufügen: **`createShape(String)`** in Klasse **`Shape`**
  - 2 Punkte: eindeutige ID erzeugen
    - wirklich eindeutig wird es nur, wenn der Server die ID erzeugt:
      - Jedes Shape-Objekt bekommt vom Server seine ID zugeteilt (Spezielles Request-Objekt; Shape mit ID 0; ...)
      - Eindeutige Client-ID (bei Verbindungsaufbau) + laufende Nummer
    - "fast immer" funktionierende Lösungen: (-0.5 Punkte)
      - Zufallszahlen, System.currentTimeMillis() / Date.getTime(), Hash
    - ID nicht eindeutig: -2 Punkte

W.1 Lösung der 4. Aufgabe (3)

- Teilaufgabe (c) [3 Punkte]
  - Interface **`ShapeContainer`** anpassen: 1 Punkt: Ergänzen von
    `void addShape(Shape);` Keine **`FormatException`** !!! (-0.5 Pkt)
    `Vector getShapes();`
    `void updateShape(Shape);`
  - **`Vector`** richtig verwenden: 2 Punkte (**nicht** einfach!)

W.1 Lösung der 4. Aufgabe (4)

- Shapecontainer.java

```
package whiteboard;

import java.io.*;
import java.util.Vector
import java.awt.Graphics;
import whiteboard.shapes.Shape;

public interface ShapeContainer {

        public void addShape(Shape s);
        public void updateShape(Shape s);
        public Vector getShapes();

        public Shape getShape(int x, int y);
        public void deleteShapes(int x, int y);
        public void load(String filename)
                              throws IOException, FormatException;
        public void save(String filename) throws IOException;
        public void paintDrawing(Graphics g);

}// ShapeContainer
```


W.1 Lösung der 4. Aufgabe (5)

- Wie verwende ich einen **`java.util.Vector`**?
  - **`Vector`** kennt seine eigene Grösse (**`size()`**)!
    (*keine eigene Variable notwendig!*)
  - Dynamisch Unterstützung beliebig vieler Shapes:
    Konstruktor von **`ShapeContainerImpl`** mit Grössenangabe macht keinen Sinn mehr! (-0.5 Pkt)
  - Iterieren über Elemente explizit (**`for`**, **`elementAt`**)
    oder **`Iterator`**/**`Enumeration`** verwenden

W.1 Lösung der 4. Aufgabe (6)

- **`Vector`**: Vorsicht bei Modifikationen während Iteration! (-1 Pkt)
  - **`Enumeration`**: Verhalten bei parallelen Modifikationen nicht spezifiziert!!!
  - explizite **`for`**-Schleife: Ohne spezielle Massnahme werden Elemente werden übersprungen!

❑ **`void deleteShapes(int x, int y)`**, 1. Lösung (umständlich)

```
void deleteShapes(int x, int y) {
    Vector delete=new Vector();
    Enumeration en=shapes.elements();
    while(en.hasMoreElements()) {
        Shape s = (Shape)en.nextElement();
        if(s.isInside(x,y)) delete.addElement();
    }
    en=delete.elements();
    while(en.hasMoreElements()) {
        shapes.remove( en.nextElement() )
    }
}
```


W.1 Lösung der 4. Aufgabe (7)

❑ **`void deleteShapes(int x, int y)`**, 2. Lösung

```
void deleteShapes(int x, int y) {
    for(int i=0; i<shapes.size(); i++) {
        if(shapes.elementAt(i).isInside(x,y)) {
            shapes.remove(i);
            i--;
        }
    }
}
```

❑ **`void deleteShapes(int x, int y)`**, 3. Lösung (am einfachsten!)

```
void deleteShapes(int x, int y) {
    Iterator it = shapes.iterator();
    while(it.hasNext()) {
        if((Shapes)it.next()).isInside(x,y)) {
            it.remove();
        }
    }
}
```

W.1 Lösung der 4. Aufgabe (8)

- Teilaufgabe (d): Whiteboard-Server [10 Punkte]
  - Socket erzeugen, **`accept:`** Wichtig: Fehlerbehandlung!!! [1 Punkt]
    - nicht schön: **`static void main() throws Exception`**
    - noch schlechter: **`catch(Exception e){System.exit(0);}`** [-0.5Pkt]
    - ganz übel: Meldung ausgeben, und einfach weiter im Programm! [-1Pkt]
      `catch(Exception e) { System.out.println("..."); }`
  - Eigener Thread für die Ausgabe zu den Clients [2 Punkte]
    Richtige Verwendung von **`wait/notify`**!
    Aktives Warten: -1 Pkt; Race Condition, keine InterruptedException: - 0.5Pkt

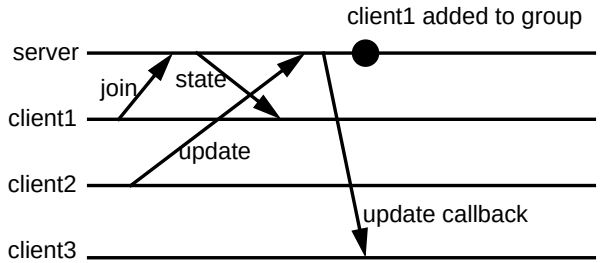
```
synchronized(reqlist) {
    while(reqlist.isEmpty()) {
        try { reqlist.wait(); }
        catch( InterruptedException ex ) {}
    }
}
```


W.1 Lösung der 4. Aufgabe (9)

- Initialen Zustand bei neuem Client übertragen [2 Punkte]
  Synchronisation mit Aufname der Zustellung "normaler" Requests! (-1Pkt)
  (siehe nächste Folie)
- Einen Thread für jeden Client erzeugen [1 Punkt]
- Anforderungen von Clients korrekt behandeln [2 Punkte]
  - **`Request`**-Objekte
  - Synchronisierte Reihenfolge zwischen mehreren Clients (- 1 Pkt])
- Nicht mehr exisiterende Clients entfernen [2 Punkte]
  - Clients werden nicht entfernt (-1 Pkt)
  - Keine sinnvolle Exception-Behandlung bei Objekt/EA (-1 Pkt)

W.1 Lösung der 4. Aufgabe (10)

- Aufname neuer Clients
  - Zustandsübertragung (des Whiteboard) und Aufname in Gruppe der Clients muss atomar erfolgen
  - Inkonsistenter Nachrichtenaustausch:

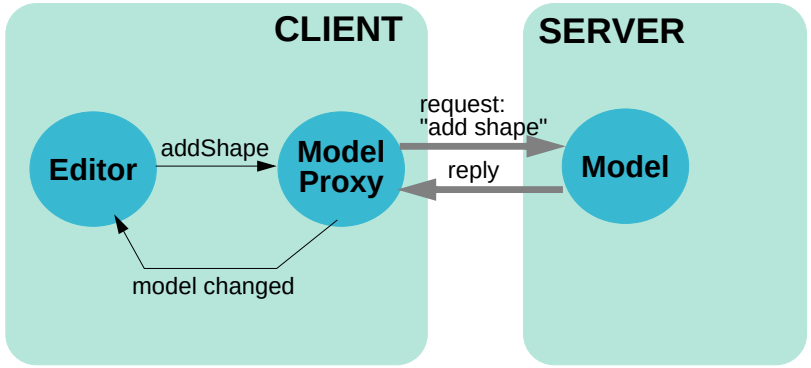


W.1 Lösung der 4. Aufgabe (11)

- Teilaufgabe (e): Whiteboard-Client [7 Punkte]
  - WhiteBoard verwendet ShapeContainerProxy [1 Punkt]
  - updateShape im mouseReleased-Event [1 Punkt]
  - Socket erzeugen, **`socket`**: Wichtig: Fehlerbehandlung!!! [1 Punkt]
  - unschön: Server fest auf "localhost" (-0.5)
  - eigener Thread für Request-Bearbeitung [1 Punkt]
  - sinnvolle Implementierung von Load und Save [2 Punkte]
    - kein "clear" vor dem Laden (-1 Pkt)
  - Request-Objekte [1 Punkt]

W.1 Repaint handling (local)

- Proxy wartet auf Antwort und sendet Modellveränderung an das Whiteboard

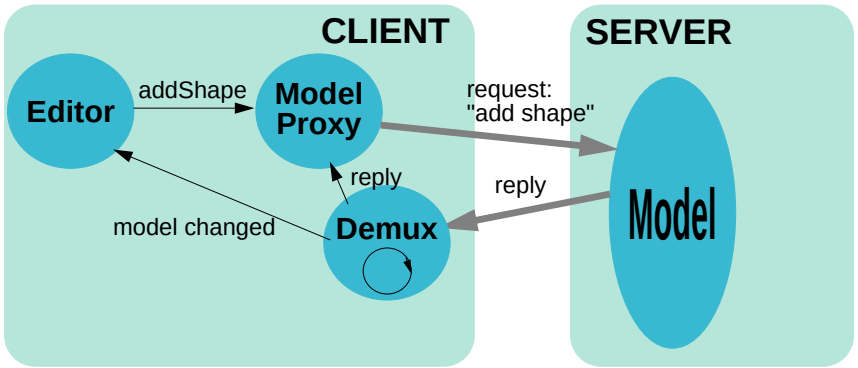


X Object Request Brokers

- ermöglichen Methodenaufrufe an entfernten Objekten (Objekte in anderen JVM)
- Beispiel-ORBs: RMI, JavaIDL

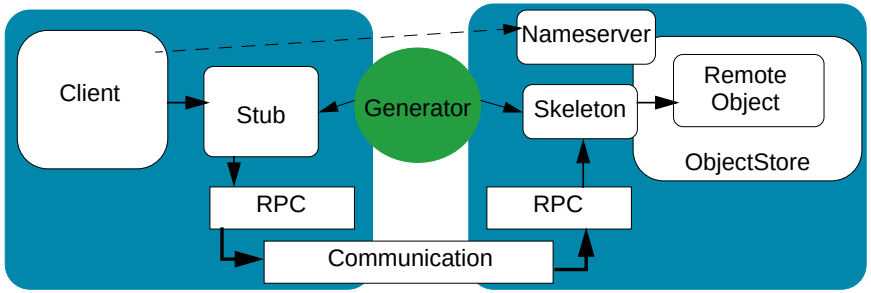
W.2 Repaint handling (Remote Observable)

- Besser: Observer Design-Pattern
- Observer nimmt Antworten entgegen und informiert das Whiteboard über die Modelveränderung



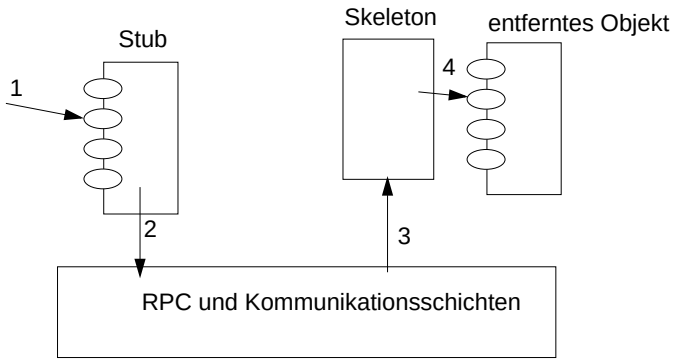
X.1 Komponenten eines ORBs

- *Kommunikationsschicht*: tauscht Daten zwischen zwei Rechnern aus
- *RPC Schicht*: definiert die Aufrufsemantik und das Marshalling
- *Object Store*: verwaltet den Lebenszyklus der Objekte
- *Stub / Skeleton Generator*: erzeugt Code für die Stubs und die Skeletons
- *Nameserver*: findet Objekte anhand deren Namen



X.2 Stubs und Skeletons

- Stub: Stellvertreter (Proxy) des entfernten Objekts.
- Skeleton: Ruft die Methoden am entfernten Objekt auf



1 Stub

- implementiert den gleichen Typen wie das entfernte Objekt (gleiches Interface)
- verpackt einen Methodenaufruf in ein Request-Objekt:
  - Objekt ID, Methoden ID, Parameter
- verwendet die RPC-Schicht um eine Anforderung zu versenden
- transformiert das Rückgabeobjekt in den entsprechenden Type
- Beispiel: erzeugte Methode (ohne Ausnahmebehandlung)

```
public int deposit(int param0) {
   Object[] parameters = new Object[1];
   parameters[0] = new Integer(param0);
   Request req = new Request(ctx, oid, 9, parameters);
   Object ret = req.send();
   return ((Integer)ret).intValue();
 }
```


1 Stub (2)

- Beispiel: erzeugte Stub-Klasse

```
package test;
import orb.*;
public class AccountImpl_Stub implements test.Account {
  int oid;
  ClientContext ctx;
  public AccountImpl_Stub(int oid, ClientContext ctx) {
    this.oid = oid;
    this.ctx = ctx;
  }
  // erzeugte Methoden
  // ...
}
```


2 Skeleton

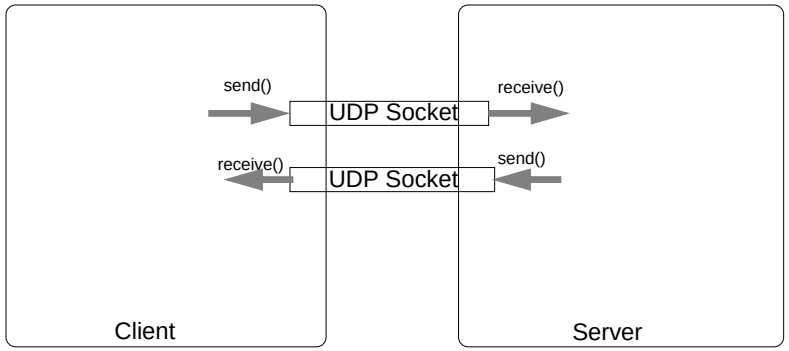
- ruft Methoden am "echten" Objekt auf
- notwendige Informationen:
  - Objektreferenz
  - Methoden ID
  - Parameter

2 Skeleton Beispiel

```
package test;
import orb.*;
public class AccountImpl_Skel  implements Skeleton {
  AccountImpl  object;
  public AccountImpl_Skel() { /* ... */ }
  public void init(int oid, Object obj) {
    this.oid = oid;
    this.object = (AccountImpl) object;
  }
  public Object invoke(int mid, Object[] parameters)
                                        throws Exception {
    switch(mid) {
       ...
      case 9: {
         return new Integer(
    object.deposit( ((Integer)parameters[0]).intValue() ));
      }
    }
  }
}
```


X.3 Kommunikationsschicht

- zuständig für den Datenaustausch zwischen zwei Rechnern
- verwendet **`DatagramSocket`** (UDP)

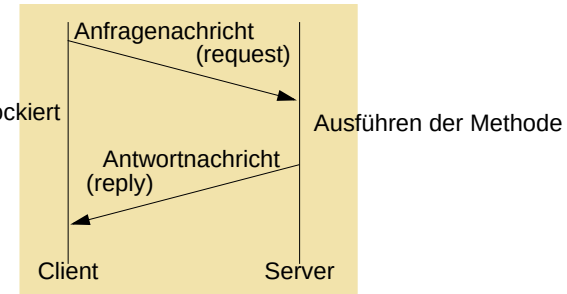


X.4 RPC-Schicht

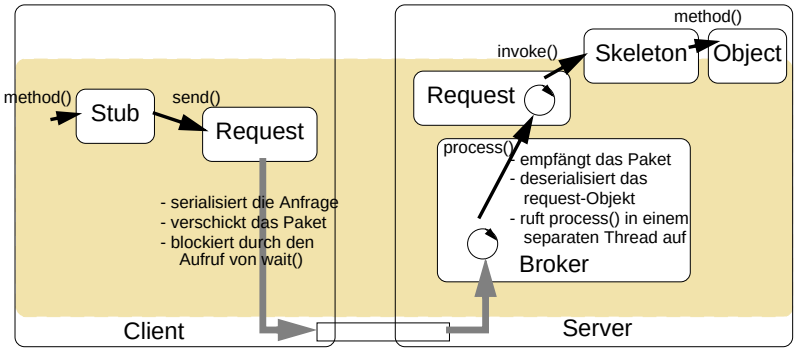
- erledigt die Weiterleitung von Methodenaufrufen
- wird von den Stubs und den Skeletons verwendet
- verwendet die Kommunikationsschicht um Bytes zu versenden

1 RPC

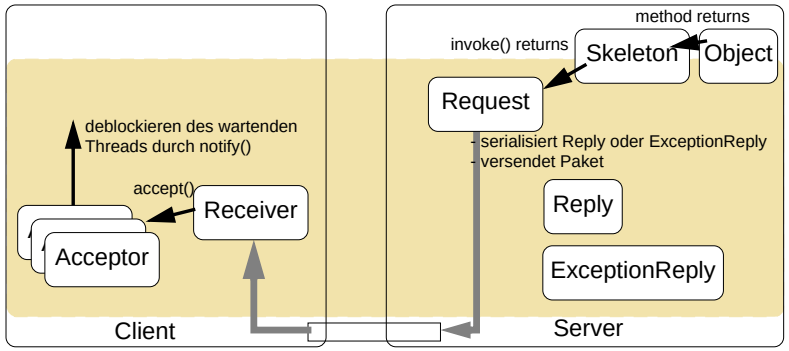
- einfachster RPC ist ein primitives request/reply-Protokoll:



2 Request



3 Reply



4 Marshalling

- RPC-Schicht verpackt die Parameter in ein Anfragepaket und den Rückgabewert in ein Antwortpaket == Marshalling
- kopiere alle Parameter zum Server
  (besser wäre es, eine Referenz zu verschicken wenn ein Parameter ein *remote Interface* implementiert)
- verwende ObjectStreams/ByteArrayStreams/Datagrams um Objekte zu verschicken:

```
ByteArrayOutputStream stream = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(stream);
out.writeObject(...);
byte[] buf = stream.toByteArray();
DatagramPacket packet = new DatagramPacket(buf, ... );
```

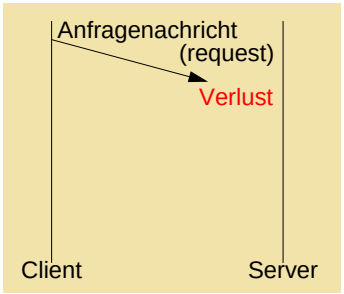
- nimm an, dass alle Parameter in ein **`DatagramPacket`** passen

5 Fehlerbehandlung

- Implementierung einer bestimmten Aufrufsemantik
  - exactly once
  - at least once
  - at most once
  - last of many
  - ...
- abhängig von der Servicequalität der Kommunikationsschicht
- muss mit Kommunikationsfehlern umgehen können:
  - verlorene Pakete
  - veränderte Reihenfolge (non-FIFO)
  - duplizierte Pakete
  - veränderte Pakete

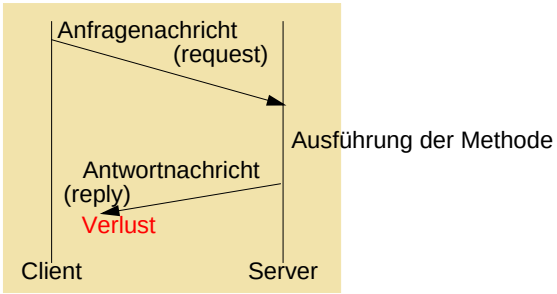
5 Fehlerbehandlung (2)

- Kommunikationsfehler können zu folgenden Problemen führen:
  - Verlust einer Anfragenachricht



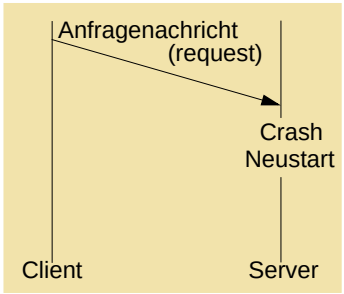
5 Fehlerbehandlung (3)

- Kommunikationsfehler können zu folgenden Problemen führen:
  - Verlust einer Antwortnachricht



5 Fehlerbehandlung (4)

- Kommunikationsfehler können zu folgenden Problemen führen:
  - Fehler während des Ausführens der Methode



5 Fehlerbehandlung (5)

- Annäherung an *exactly once*:
  - Um verlorene Anfragenachrichten entgegenzuwirken werden die Pakete nach einem Timeout erneut gesendet.
  - Zum vermeiden von mehrfach Ausführungen bei doppelt gesendeten Paketen werden IDs verwendet, welche bei einem erneuten versenden der gleichen Anfrage nicht verändert werden.
  - Der Server soll abgesendete Antworten puffern, für den Fall, dass eine Antwort verloren geht und um eine erneute Ausführung der Methode zu vermeiden.
    Der Puffer kann mithilfe einer **`Hashtable`** implementiert werden, die Anfrage IDs auf Antworten abbildet.
  - Den Absturz eines Servers zu verkraften ist etwas schwieriger ..... daher ignorieren wir das Problem.

X.5 Object Store

- Interface:
  - **`int registerObject(Object obj)`**
    gibt eine eindeutige Objekt-ID zurück.
  - **`Object lookupByID(int oid)`**
    liefert das Objekt zur gegebenen Objekt-ID
- Implementationstechnik:
  - **`java.util.Hashtable`** kann verwendet werden um die Abbildung zwischen Objekt-IDs und Objekten vorzunehmen (Hinweis: die Hashtabelle kann nur Objekte speichern, die *OID* sollte daher in ein **`Integer`**-Objekt gekapselt werden.)

X.6 Nameserver

- Stub-Erzeuger
  - sucht die ID (und den Klassennamen); erzeugt daraufhin ein Stub-Objekt
    - **`Object lookup(String name)`**
  - der Nameserver lädt die Klasse und erzeugt eine Instanz der Klasse

X.6 Nameserver

- ist aufgeteilt in einen ID-Finder und einen Proxy-Erzeuger
- ID Finder
  - bildet Namen auf OIDs ab:
    - **`void bind(String name, int oid)`**
    - **`int lookupID(String name)`**
  - verwendet **`Hashtable`**
  - ist selbst als entferntes Objekt mit der OID 1 implementiert
- ein realistischer Nameserver würde Referenzen auf Stub-Objekte zurückliefern anstatt OIDs
- das erfordert, dass zusammen mit der OID der Klassenname des Stubs zurückgegeben wird.

Y Bauen von Projekten mit ANT

- Was ist ANT?
- ANT im Detail
  - Projekte
  - Properties
  - Targets
  - Core Tasks
  - Eigene Tasks
- Zusätzliche Informationen

Y.1 Was ist ANT?

- ANT ist ein Werkzeug zum Übersetzen von größeren Java Projekten.
- Arbeitsschritte werden nur angestossen, wenn die Abhängigkeiten sich verändert habe.
- Wie bei “make” können Ziele (targets) und Abhängigkeiten (dependences) angegeben werden. Die Beschreibung erfolgt in einer XML-Datei.
- ANT ist eine Java-Anwendung und kann durch Klassen flexibel erweitert werden.

Y.2 Warum nicht das gute alte “make”?

- Make berücksichtigt die Eigenheiten von Java nicht
  - Verzeichnisstruktur von Paketen müssen explizit angegeben werden
  - Die teilweise automatische Übersetzung der Klassen durch den Compiler wird nicht berücksichtigt
- Beim Aufruf von javac wird jedesmal eine neue JVM gestartet (ist langsam)
- Make ist nicht durch Java-Klassen flexible erweiterbar

Y.3 Grundlagen

- Jedes Projekt besitzt ein *Buildfile* (build.xml)
- Ein Buildfile beschreibt mehrere *Targets*
  - Targets definieren typische Aufgaben des Übersetzungsprozesses
  - Targets können Abhängigkeiten zu anderen Targets besitzen
- Jedes Taget kann sich aus verschiedenen *Tasks* zusammensetzen
  - Tasks werden sequentiell hintereinander abgearbeitet
- Konfiguration des Übersetzungsprozesses durch *Properties*

Y.4 Beispiel: Buildfile <project>

- Das Buildfile beschreibt den Generierungsprozess in XML
- Das <project>-Tag beschreibt das Projekt und enthält beliebig viele Properties und Targets.

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default=”compile" name=”example">
<!--
        [ property definitions ]
        [ path and patternset ]
        [ targets ]
-->
</project>
```

Y.5 Beispiel: Buildfile <property>

- Properties dienen im Buildfile zur einfachen Konfiguration
- Properties können mit Hilfe des <property>-Tags definiert werden
- Zusätzlich gibt es Properties für bestimmte Aufgaben zum Beispiel für Verzeichnispfade (<path>-Tag)

Y.6 Beispiel: Buildfile <target>

- Die Targets dienen zum Beschreiben einzelner Abschnitte des Übersetzungsprozesses
- Sie besitzen immer einen Name
- Können weitere Targets als Abhängigkeit benennen
- Sollten eine Beschreibung besitzen, welche während der Übersetzung angezeigt wird
- Umfasst mehrere Arbeitsschritte (Tasks)
- Ausgehend vom im <project>-Tag benannten default-Target werden alle Abhängigkeiten überprüft. Ist eine Abhängigkeit neuer als das Ziel, so werden die Tasks abgearbeitet.

Y.5 Beispiel: Buildfile <property>

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default=”all" name=”example">
  <property name="srcDir" value="src"/>
  <property name="buildDir" value="classes"/>

  <path id="classpath">
     <pathelement location="lib/"/>
     <pathelement path="${java.class.path}/"/>
     <pathelement path="${junit}"/>
  </path>

  <!--
        [ targets ]
   -->
</project>
```


Y.6 Beispiel: Buildfile <target>

```
<target name="all" depends="test, doc, dist"
        description="runs test, javadoc and dist"/>

<target name="clean" depends="cleanTestLog"
   description="deletes all generated files">
    <delete dir="${buildDir}"/>
    <delete dir="${docDir}"/>
    <delete file="${distFile}"/>
</target>

<target name="cleanTestLog"
   description="deletes generated JUnit log files">
    <delete>
      <fileset dir=".">
        <include name="TEST-*.txt"/>
      </fileset>
    </delete>
</target>
```

Y.6 Beispiel: Buildfile <target>

```
<target name="prepare"
  description="creates output directories">
    <mkdir dir="${buildDir}"/>
    <mkdir dir="${docDir}"/>
</target>

<target name="compile" depends="prepare"
  description="compile source files">
    <javac srcdir="${srcDir}" destdir="${buildDir}"
       classpathref="classpath"/>
</target>
```


Y.6 Beispiel: Buildfile <target>

```
<target name="test" depends="compile, cleanTestLog"
  description="runs all JUnit tests">
    <junit printsummary="yes">
      <classpath refid="classpath"/>
      <batchtest>
        <fileset dir="${srcDir}">
             <include name="**/*Test.java"/>
        </fileset>
      </batchtest>
    </junit>
</target>

<target name="doc" depends="compile"
  description="generates javadoc">
  <javadoc sourcepath="${srcDir}" destdir="${docDir}"
                        packagenames="org.blub.*"
                           classpathref="classpath" />
</target>
```


Y.6 Beispiel: Ausgabe von ANT

- ANT für Aufgabe 6 könnte so aussehen:

```
faui40f: 13:39 felser/afg6 > ant
Buildfile: build.xml

prepare:
    [mkdir] Created dir: /proj/i4oovs/felser/afg6/classes
    [mkdir] Created dir: /proj/i4oovs/felser/afg6/docs

comm:
    [javac] Compiling 3 source files to /proj/.../afg6/classes

stubgen:
    [javac] Compiling 1 source file to /proj/.../afg6/classes

orb:
    [javac] Compiling 15 source files to /proj/.../afg6/classes

all:

BUILD SUCCESSFUL
Total time: 10 seconds
faui40f: 13:39 felser/afg6 >
```


Y.7 Core Tasks

- Ant
  Ant, AntCall, AntStructure
- Basics
  Condition, Chmod, Copy, Delete, Echo, Exec, Fail, Java, MkDir, Move, PathConvert, Property, Sleep, Touch, Tstamp
- Dokumentation und Test
  Javadoc, Mail, Record,
- Package
  Jar, SignJar, Gzip, Zip, GUzip, War, Tar
- Versionskontrolle
  CVS, CVSPass, Patch

Y.8 Optionale Tasks

- Tools
  JavaCC, Javah, JDepend, JJTree, JProbe
- Dokumentation und Test
  JUnit, JUnitReport, Mime Mail, Test
- Package
  Cab, RPM
- Versionskontrolle
  Clearcase, PVCS, SourceSave

Y.9 Entwicklung eigener Tasks (1)

- Vorgehensweise bei der Entwicklung eigener Tasks

(1) Ableiten der Klasse **`org.apache.tools.ant.Task`**

(2) Implementieren einer **`set`**-Methode für jedes Attribut des Tasks

```
public void setAttrName(type attrName);
```

(3) Implementieren einer **`create`** oder **`add`**-Methode für jedes untergeordnete Element

```
public ChildTask creatChildTask();
```

oder

```
public void addChildTask(ChildTask childTask);
```

(4) Implementieren der eigentlichen Task-Methode

```
public void execute();
```


Y.9 Entwicklung eigener Tasks (2)

- Programmcode eines Beispieltasks:

```
public class ZakTask extends Task {
 private String msg;

 // The method executing the task
 public void execute() throws BuildException {
   System.out.println(msg);
 }

 // The setter for the "message" attribute
 public void setMessage(String msg) {
    this.msg = msg;
 }
}
```


Y.9 Entwicklung eigener Tasks (3)

- Einbindung des Tasks in ein Buildfile:

```
<project name="TestZakTask" default="main" basedir=".">

  <taskdef name="mytask" classname="ZakTask"/>

  <target name="main">
    <mytask message="Hello World! ZakTask works!"/>
  </target>

</project>
```

Y.10Zusätzliche Informationen

- Mehr Informationen unter
  - http://jakarta.apache.org/ant/
  - Ant in Anger - http://jakarta.apache.org/resources.html
- Ant im CIP-Pool:

```
#java
setenv JAVA_HOME /local/java-1.3.1_01

# ant
setenv ANT_HOME /local/jakarta-ant
set path=( /local/jakarta-ant/bin $path)
```