

# Erste Schritte mit Maven

Hier werden euch ein paar Aufgaben gestellt, die mit Vorkenntnissen binnen Minuten zu erledigen sind. Sie sollen das Ausprobieren und Herumspielen fördern. Hast du eine Idee, dann probiere sie ruhig aus! Die kursiv gesetzten Fragen sollen dich zum Denken anregen. Du brauchst die Antworten nicht niederschreiben, aber denke drüber nach. Sie werden das Verständnis verbessern. Bei Fragen meldet euch einfach bei uns.

## Kickstart

Für die folgenden Aufgaben müsst ihr euch auf die Kommandozeile begeben. Falls ihr in eurem Home-Directory noch kein Verzeichnis für Entwicklungsprojekte habt, erstellt euch eines.

### Begeht euch in euer persönliches Entwicklungsverzeichnis, bspw.:

```
$ cd ~/development
```

### Prüft, ob die Voraussetzungen für Maven auf eurem System gegeben sind

```
$ java -version
```

```
$ mvn -v
```

### Setzt ein eigenes Projekt mit Maven und dem Archetype-Plugin auf

```
$ mvn archetype:generate
```

Auswahl:

Choose a number: einfach mit *Return* den default-Wert (maven-archetype-quickstart) auswählen

Choose a version: 1

Define value for property 'groupId': de.fhkoeln.ass

Define value for property 'artifactId': ff

Restliche Einstellungen: einfach mit *Return* die default-Werte bestätigen

Wechselt in das Verzeichnis und schauen wir uns die Inhalte an:

```
$ cd ff
```

```
$ ls -lR
```

### Analysiert das Projekt:

*Wo liegt der Source-Code?*

*Was für Sourcen oder Arten von Sourcen gibt es?  
Was müsste beim ausführen der Sourcen geschehen?*

### **Kompiliert das Projekt:**

```
$ mvn compile
```

*Was wird alles beim Kompilieren gebaut / generiert?  
Wo ist das Kompilat? Was ist das target Verzeichnis? Braucht ihr noch ein bin Verzeichnis?*

### **Führt die Java-Klasse auf der Kommandozeile aus.**

```
$ cd target/classes  
$ java de.fhkoeln.ass.App  
$ cd ../../
```

*Hat sich deine Erwartung von oben bestätigt?*

### **Führt den Test aus**

```
$ mvn test
```

### **Eclipse auf die Maven-Verwendung vorbereiten**

```
$ mvn eclipse:configure-workspace -Declipse.workspace=<deinWorkspacePfad>  
<deinWorkspacePfad> Bspw.: /Users/ass/development
```

### **Erstellt ein Eclipse Projekt mit Maven**

```
$ mvn eclipse:eclipse
```

### **Importiert das Projekt in Eclipse**

Startet Eclipse!

Falls Eclipse beim Start fragt, wo der Workspace liegen soll, gebt euer Entwicklungs-Verzeichnis an. Falls Eclipse nicht fragt, ändert dies von Hand: File -> Switch Workspace -> Others...

Dann:

File -> Import -> Existing Project into Workspace

### **Fügt eine eigene Klasse hinzu (Fast.java)**

Fast.java:

```
package de.fhkoeln.ass;  
  
public class Fast {
```

```
public String getFast() {  
    return "Fast!!";  
}  
  
public String getFurious() {  
    return "Furious!";  
}  
}
```

### Fügt eine Testklasse hinzu (FastTest.java)

FastTest.java:

```
package de.fhkoeln.ass;  
  
import junit.framework.TestCase;  
  
public class FastTest extends TestCase {  
  
    public void testGetFast() {  
        Fast fast = new Fast();  
        assertEquals("Fast!!", fast.getFast());  
    }  
  
    public void testGetFurious() {  
        Fast fast = new Fast();  
        assertEquals("Furious!", fast.getFurious());  
    }  
  
}
```

### Führt den Test aus (s.o.)

*Was ist passiert?*

*Formuliere in deinen Worten, was wir in den Schritten 8-10 gemacht haben.*

*Was kann man außerdem noch testen?*

*Siehst du Verbesserungsmöglichkeiten?*

*Führe diese Verbesserungen durch!*

### Stelle eine auslieferungsfertige Version der Software her

```
$ mvn package
```

*Wo ist das Auslieferungspaket?*

*Was beinhaltet es?*

### Lass Maven das Projekt wieder aufräumen.

```
$ mvn clean
```

# Praktische Infos für Maven

## Maven Installation & Test

Maven 2 herunterladen und in ein beliebiges Verzeichnis entpacken. Dann die Umgebungsvariable M2\_HOME auf dieses Verzeichnis setzen. Dann muss das \$M2\_HOME/bin Verzeichnis in den Pfad aufgenommen werden.

Test mit:

```
$ mvn --version
```

## Maven funktioniert nicht mehr...

Ein häufiger Fehler: der mvn-Befehl muss im Projekt-Hauptverzeichnis aufgerufen werden. Das ist das Verzeichnis, wo die pom.xml Datei liegt!

## Woher bekomme ich Dependencies und Plugins?

Hier muss man erstmal unterteilen: Du brauchst zwei Dinge für das Einbinden von Abhängigkeiten und Plugins: 1. die XML-Definition und 2. die Binaries.

Es gibt Quellen, da bekommt man direkt beides:  
die Webseite des Plugins / Dependency, welches du suchst

<http://repository.apache.org>

<http://www.artifact-repository.org>

<http://mvnrepository.com> (unser Favorit)

<http://www.mvnbrowser.com>

<http://www.jarvana.com>

<http://mavensearch.net>

und wie immer: Google

Andernfalls kannst du auch Binaries runterladen und sie von Hand ins lokale Repository installieren. Hierzu benutzt man das maven-install-plugin.

## Maven Dependencies und Eclipse

Du hast eine Dependency eingetragen und Eclipse meckert, dass es die Klassen nicht finden kann?

Wenn man eine Dependency in das Maven-POM hinzugefügt hat, so ist diese im Classpath während der Maven-Verarbeitung verfügbar.

Eclipse hat jedoch zum Zeitpunkt des Code-Schreibens einen eigenen Classpath, weswegen man nach dem Hinzufügen einer Dependency in Maven dies auch Eclipse mitteilen muss.

Dies geschieht über den Befehl

```
$ mvn eclipse:eclipse
```

## Was kann man alles ins POM schreiben?

Eine vollständige Übersicht bietet die Referenz des POMs:

<http://maven.apache.org/ref/2.2.0/maven-model/maven.html>

Dokumentationen der im POM zu konfigurierenden Plugins:

<http://maven.apache.org/plugins/index.html>

Praktisch im Umgang mit Eclipse ist das Code Completion Feature, welches auch für XML Dateien funktioniert. Außerdem gibts ein Plugin für Eclipse, welches den Umgang mit Maven erleichtert: m2eclipse.

# Maven im Einsatz

Die folgenden Aufgaben bauen auf dem Kickstart auf. Du benötigst sowohl das pom.xml sowie die Java Klasse und die Testklasse. Auch hier solltest du wieder eine Kommandozeile benutzen, um verschiedene Kommandos auszuführen.

Die Aufgaben hier stellen übliche Tasks dar, die man in Projekten ausführt: per Dependency Bibliotheken (JARs) einfügen, Plug-Ins integrieren, die selbständig Aufgaben erledigen usw. Auch hier gilt wieder: probiere ruhig deine Ideen aus und frag nach, wenn etwas unklar ist.

## Dependencies

### **Integriere Logging durch einen Dependency-Eintrag ins POM!**

*Wie bist du an den Dependency-Eintrag gekommen?*

*Für welche Version hast du dich entschieden?*

*Gibt es Alternativen?*

### **Setz einen Logeintrag in die getFast-Methode.**

Eine Logger-Instanz kann man so erstellen:

```
static final Logger log = Logger.getLogger(MeineKlasse.class);
```

*Prüfe deinen Code, indem du ihn mit Maven kompilierst!*

*Was passiert beim Ausführen?*

*Siehst du die Logausgabe? Wo sollte sie hingehen?*

*Funktioniert der Test weiterhin?*

*Funktioniert das Eclipse-Projekt weiterhin?*

*Falls nein: was funktioniert nicht? Wie kannst du das beheben?*

### **Erweitere die Maven Projektstruktur um ein Resources-Verzeichnis**

Maven-Konvention: src/main/resources

Kopiere die log4j.properties Datei in das resources-Verzeichnis

### **Prüfe das Logging erneut**

*Lassen sich die Sourcen nun kompilieren?*

*Meldet das Projekt in Eclipse immer noch Fehler? Oder wieder?*

*Was ist der Output des Tests?*

*Gibt es irgendwo eine Log-Ausgabe?*

*Wenn ja: woher weiß Log4j nun von der Properties-Datei?*

## Plug-Ins

Plugins werden in den `<build />` Abschnitt des POMs eingetragen. Hier wird mit 2 verschiedenen Plugins gearbeitet!

### **Integriere ein Plugin, welches die Testklasse(n) in ein eigenes JAR packt!**

Das Jar soll immer dann erstellt werden, wenn auch für den Rest der Applikation ein JAR gebaut wird. Das JAR soll nicht erstellt werden, wenn nur kompiliert oder getestet wird.

### **Integriere ein Plugin, welches die Sourcen der Testklasse(n) in ein eigenes JAR packt!**

Das Jar soll immer dann erstellt werden, wenn auch für den Rest der Applikation ein JAR gebaut wird. Das JAR soll nicht erstellt werden, wenn nur kompiliert oder getestet wird.

## Web-Projekt (web-layer)

**Erstelle ein neues Web-Projekt (maven-archetype-webapp) und gebe ihm die ArtifactID *web-layer*. Dieses Projekt wird später wiederverwendet.**

**Integriere das jetty-maven-plugin**

**Führe das `jetty:run` Goal aus und besuche `http://localhost:8080`**

*Gelangst du direkt auf die Startseite des Projekts? Wenn nicht, wie sieht der direkte Link aus?*

*Kannst du einen anderen Link im Plugin konfigurieren?*

*Welche Möglichkeiten gibt es das Plugin zu stoppen? Welche Vorteile/Nachteile haben diese?*

Extra-Aufgabe, falls alle anderen Aufgaben erledigt wurden:

## Projekt-Management

**Fülle das POM mit sinnvollen Informationen über das Projekt**

Die Maven-Webseite ([maven.apache.org](http://maven.apache.org)) bietet eine Übersicht über gängige Plugins.

# Aufgaben im Team

## Multi-Modul Projekt (ff-mm)

Die nächsten Aufgaben basieren auf einem Gerüst, welches wir für euch gebaut haben. Die Applikation liest einen Teil der Daten aus einer Datenbank aus und ein weiterer Teil der Daten stammt aus einem Webservice. Diese Datenschicht ist ein eigenes Subprojekt im Multi-Modul Projekt.

### **Probiere aus, was die Applikation macht!**

Führe im ff-mm Projekt die Befehle `mvn install` und `mvn eclipse:eclipse` aus.

Führe die `main()`-Methode in der App Klasse aus.

### **Automatisiere den Vorgang der Stub-Generierung aus der WSDL**

Im Beispielprojekt wurde im Modul *data-layer* mit einem WSDL2Java Tool die Java-Sourcen einmalig aus der WSDL generiert. Dies soll durch eine Integration in Maven automatisiert werden! Informiere dich darüber, wo nach Maven Konvention generierte Dateien üblicherweise abgelegt werden.

Um die Aufgabe zu lösen, muss unter `src/main/java` das „net“ Verzeichnis gelöscht werden. Die WSDL für den Service findest du hier:

<http://www.websvcex.net/globalweather.asmx?wsdl>

### **Überarbeite das Haupt-POM und füge Properties und Variablen ein, wo es deiner Meinung nach sinnvoll ist.**

### **Erweitere das Projekt um ein Modul *service-layer* für die Geschäftslogik**

Nutze dabei die bestehenden DAOs als Basis. Binde dich an das Interface, mehr braucht man nicht.

Du könntest bspw folgende Services in der Geschäftslogik-Ebene einbauen: `listAllCities()`, `listAllWeatherInformation()` usw.

Bitte achte darauf, dass nicht die Programmieraufgaben im Vordergrund stehen sollen. Falls also Probleme auftreten sollten, erledige lieber erst die restlichen Aufgaben dieses Handouts.

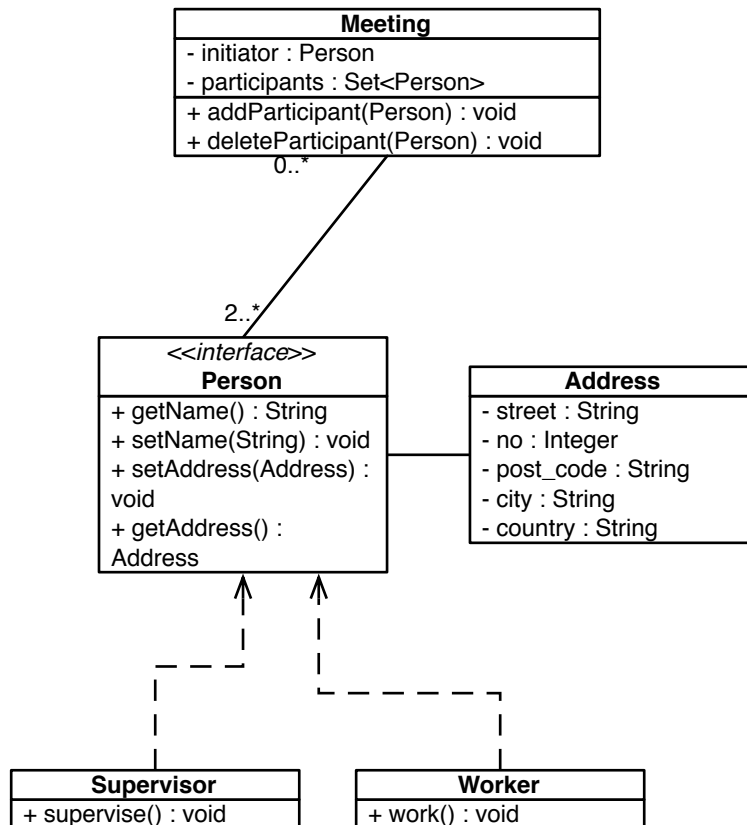
### **Erweitere das Haupt-POM derart, dass durch das Goal „package“ ein WAR entsteht, welches alle Module, auch das Projekt/Modul *web-layer*, beinhaltet.**

# Erste Schritte für Spring

## Aufbau eines Objektnetzes

Wir haben für diesen Abschnitt eine Klassenstruktur entworfen, die ihr nachbauen sollt. Diese Klassenstruktur hätten wir auch anders nennen können, es kommt hier einzig und allein auf die Beziehungen zwischen den Klassen an. Die Aufgabe soll eine Problemstellung vermitteln, die euch möglicherweise noch nicht bewusst und nur schwer zu erkennen ist. Es ist wichtig, dass ihr über eure Probleme mit der Aufgabenstellung nachdenkt.

### Erstelle eine Klassenstruktur nach Vorlage der Abbildung.



### Erstelle eine Instanz der Klasse Meeting mit mehreren Personen als Teilnehmer.

Das Meeting wird mit mehreren teilnehmenden **Person-Instanzen** besetzt. Es sollen sowohl Supervisor als auch Worker dabei sein. Für das Meeting-Objekt ist es unerheblich, ob der Initiator ein Supervisor oder Worker ist; bei den übrigen Teilnehmern ist es genauso.

Hinweis: Um eine Instanz eines Meetings zu erstellen, brauchst du natürlich wieder eine Klasse, der Einfachheit halber kann diese Klasse eine `main()`-Methode haben, welche du ausführst.

**Formuliere, wo / wie welche Klassen untereinander gekoppelt sind!**

Bspw: Klasse X hat eine direkte Kopplung zu Klasse Y in Methode zXY.

*Wie würdest du die Kopplungen beschreiben?*

*Welche Kopplungen sind notwendig, welche sind es nicht?*

**Lasse dir das Meeting als Text ausgeben, mit allen Teilnehmern und Adressen.**

**Falls du es noch nicht getan hast: entwirf eine weitere Klasse, die dir ein Person-Objekt zurückgibt, wenn du einen „Worker“ oder „Supervisor“ anforderst.**

*Was ist der Vorteil dieses Verfahrens? Wo liegt nun die Kopplung zwischen den Klassen im Gegensatz zu vorher?*

Tipp: Für Infos zu diesem Verfahren suche nach dem Factory-Method Design Pattern.

Weiterer Tipp: Diese Factory-Klasse sollte man ungefähr so benutzen können:

```
Person p1 = Factory.getPerson(„Worker“, „Name1“);  
Person p2 = Factory.getPerson(„Supervisor“, „Name2“);
```

# Konfigurationen mit Spring

Das letzte Handout sollte zeigen, dass man gezwungen ist, irgendwo Klassen zu instantiieren. Dabei ist es egal, ob man Interfaces verwendet oder nicht, irgendwo werden Abhängigkeiten geschaffen und sei es zu / in einer Factory.

Nun versuchen wir, diese Probleme mit Hilfe von Spring zu umgehen. Wir werden eine andere Klassenstruktur verwenden, die ein Beispiel für die Geschäftslogik- und Daten-Ebene ist. Wir möchten die beiden Ebenen soweit wie möglich entkoppeln.

## Aufbau eines Objektnetzes

Für diese Aufgaben stellen wir ein Projekt-Grundgerüst zur Verfügung.

**Füge in dein Maven-Projekt eine Dependency für Spring in Version 2.5.6 ein**

**Erstelle die Klassen OrderService, CustomerService und ProductService. Die Klassen sollen getter/setter für die Order / Customer / Product DAO Klassen besitzen.**

Diese Klassen stellen deine Service-Ebene dar. Sie implementieren mit Hilfe der DAO Klassen die Geschäftslogik.

**Erweitere die DAO Klassen um sinnvolle Methoden.**

Beispielsweise Methoden wie save, getAllXXX, update, getXXXByName, usw.

**Nutze Springs Dependency Injection, um die Klassen und deren Abhängigkeiten zu strukturieren.**

Hierfür muss die von uns vorbereitete Spring-Konfiguration angepasst werden: Konfiguriere das Objektnetz in Spring, strukturiert in Service/Manager-Ebene, DAO-Ebene, Infrastruktur-Beans. Sorge dafür, dass jede deiner Service-Klassen ein entsprechendes DAO und jedes DAO eine SessionFactory durch Spring injiziert bekommt.

## Spring Resources

Resources erlauben den Input in Klassen über Dateien im Filesystem, wie .properties Dateien, Textdateien, etc.

**Füge in eine deiner Beans ein Feld vom Typ Resource ein und benutze die Properties im Code**

### **Injiziere Daten über folgende Konfiguration**

```
<bean id="myBean" class="...">  
  <property name="template" value="some/resource/path/myTemplate.txt"/>  
</bean>
```

Das template Feld in der Bean muss vom Typ Resource sein. Gib die Daten als Text in der Konsole oder dem Logfile aus.

## **Spring Validation**

**Validiere die Felder deiner Klassen mit Springs Validation-Interface und der ValidationUtils Klasse**

# Advanced Spring Features

## Spring JDBC

**Erstelle in der Spring Konfiguration eine DataSource, welche auf die Datenbank zugreifen kann.**

**Erstelle eine Klasse, welche über ein Feld JdbcTemplate jdbcTemplate verfügt und injiziere eine DataSource in deine eigene Spring-Bean (Vgl. Beispiel):**

```
private JdbcTemplate jdbcTemplate;  
public void setDataSource(DataSource dataSource) {  
    this.jdbcTemplate = new JdbcTemplate(dataSource);  
}
```

**Erstelle in deiner Bean ein JdbcTemplate-Objekt (Alternativ: Leite deine Klasse von JdbcDaoSupport ab (Vgl. Beispiel oben)).**

**Benutze die query/execute/update Methoden des JdbcTemplates, um Daten in der Datenbank abzurufen und zu verändern.**

## Transaktionen

**Sorge dafür, dass alle Methoden in der Service-Ebene als Transaktionen durchgeführt werden.**

Zu beachten ist, dass außer der deklarativen oder programmatischen Integration auch die Konfiguration verändert werden muss. Alle Infos sind in Abschnitt 9.5.6 der Spring Spezifikation zu finden.

**Probiere verschiedene Einstellungen für die @Transactional Annotation aus.**

Wie kann man auf unterschiedliche Exceptions reagieren? Was sind Propagations und welche Propagations sind für unsere Anwendungsfälle sinnvoll?

**Schreibe JUnit Tests für deine Klassen, die transaktionsbasiert sind.**

Teste auch, ob die Exceptions im Fehlerfall korrekt geworfen werden. Außerdem überprüfe, ob nicht doch irgendwie Werte in der Datenbank verändert wurden, die nicht hätten geändert werden dürfen.

In den Ressourcen findest du die Beschreibung vieler hilfreicher Annotationen, die beim Testen helfen. Allerdings muss JUnit hierfür geupdated werden, falls du sie benutzen willst.

## Web Services

### **Veröffentliche eine deiner Service-Klassen als Webservice.**

Wie kann man seinen Web Service testen?

Wo kommt die WSDL her? Wofür braucht man eigentlich die WSDL?

Was musst du zuerst erstellen: den Service oder die WSDL?

### **Baue einen Client für deinen Web Service.**