# Verfahren und Werkzeuge moderner Softwareentwicklung

**Einsendeaufgabe 05:**

**Sammeln und protokollieren von Erfahrungen mit Buildmanagement-Tools**

Von:

Christian Obst

Version:

1.0 2014-12-12

# Inhaltsverzeichnis:

# Basis - Ant

Aufgabe: Bringen sie ANT für ihr eigenes Projekt zum laufen. Schreiben sie ein eigenes Ant Skript, welches die üblichen Aufgaben erbringt (Test, Compile, Run, Clean, etc.). Beweisen Sie mit Screenshots und ggf. Code, dass es Ihr eigenes Werk ist.

## Ant und Netbeans

Da das vorliegende Java-Projekt () bisher in NetBeans entwickelt wurde, soll zunächst betrachtet werden ob und wie Ant in dieser IDE integriert werden kann oder ist.

Ant ist das default Build-Tool von NetBeans [NBIDE14_BT] somit stellt die IDE hier alle notwendigen Konfigurationsmöglichkeiten von Ant über das GUI bereit.

## Eigene Erfahrungen mit Ant

Um der Aufgabenstellung dennoch gerecht zu werden “Beweisen Sie… dass es Ihr eigenes Werk ist” soll nun also eine von NetBeans losgelöste Ant-Umgebung geschaffen und genutzt werden. Zudem soll das automatisch generierte Ant-Skript durch selbst geschriebene Skripte ersetzt werden.

### Download und Installation von Ant

Da auf dem aktuell genutzten Rechner (Apple Mac OS X Version 10.9.5) noch keine Ant-Installation zur Verfügung stand wurde das Programm “Homebrew” genutzt um hierüber Ant zu installieren.
Homebrew ist ein Paket-Verwaltungstool, welches genutzt werden kann um auf OS X fehlende Binaries zu laden und zu installieren [BRW14]

1. Zunächst muss Homebrew heruntergeladen und installiert werden

```
cobst$ ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"

==> This script will install:
/usr/local/bin/brew
/usr/local/Library/...
/usr/local/share/man/man1/brew.1

==> The following directories will be made group writable:
/usr/local/.
/usr/local/bin
/usr/local/etc
/usr/local/include
/usr/local/lib/pkgconfig
/usr/local/sbin
/usr/local/share
/usr/local/share/locale
/usr/local/share/man
/usr/local/share/man/man1
/usr/local/share/man/man3
```

```
/usr/local/share/man/man5
/usr/local/share/man/man7
/usr/local/share/man/man8
/usr/local/share/doc

==> The following directories will have their group set to admin:
/usr/local/.
/usr/local/bin
/usr/local/etc
/usr/local/include
/usr/local/lib/pkgconfig
/usr/local/sbin
/usr/local/share
/usr/local/share/locale
/usr/local/share/man
/usr/local/share/man/man1
/usr/local/share/man/man3
/usr/local/share/man/man5
/usr/local/share/man/man7
/usr/local/share/man/man8
/usr/local/share/doc

Press RETURN to continue or any other key to abort

==> /usr/bin/sudo /bin/chmod g+rwx /usr/local/. /usr/local/bin /usr/local/etc
/usr/local/include /usr/local/lib/pkgconfig /usr/local/sbin /usr/local/share
/usr/local/share/locale /usr/local/share/man /usr/local/share/man/man1
/usr/local/share/man/man3 /usr/local/share/man/man5 /usr/local/share/man/man7
/usr/local/share/man/man8 /usr/local/share/doc
Password:

==> /usr/bin/sudo /usr/bin/chgrp admin /usr/local/. /usr/local/bin
/usr/local/etc /usr/local/include /usr/local/lib/pkgconfig /usr/local/sbin
/usr/local/share /usr/local/share/locale /usr/local/share/man
/usr/local/share/man/man1 /usr/local/share/man/man3 /usr/local/share/man/man5
/usr/local/share/man/man7 /usr/local/share/man/man8 /usr/local/share/doc

==> /usr/bin/sudo /bin/mkdir /Library/Caches/Homebrew

==> /usr/bin/sudo /bin/chmod g+rwx /Library/Caches/Homebrew

==> Downloading and installing Homebrew…
remote: Counting objects: 217836, done.
remote: Compressing objects: 100% (57496/57496), done.
remote: Total 217836 (delta 159129), reused 217808 (delta 159109)
Receiving objects: 100% (217836/217836), 48.21 MiB | 189.00 KiB/s, done.
Resolving deltas: 100% (159129/159129), done.
From https://github.com/Homebrew/homebrew
 * [new branch]      master     -> origin/master
HEAD is now at 3f1f44b deis: add 1.1.0 bottle.

==> Installation successful!

==> Next steps
Run `brew doctor` before you install anything
Run `brew help` to get started
```

2. Folgend kann Homebrew genutzt werden um Ant zu installieren

```
cobst$ brew install ant
==> Downloading
https://downloads.sf.net/project/machomebrew/Bottles/ant-1.9.4.mavericks.bottl
e.tar.gz
######################################################################
100,0%
==> Pouring ant-1.9.4.mavericks.bottle.tar.gz
  /usr/local/Cellar/ant/1.9.4: 1597 files, 39M
```

3. Nun steht Ant zur Nutzung bereit

```
cobst$ ant -version
Apache Ant(TM) version 1.9.4 compiled on April 29 2014
```

## Erstellen der Build.xml

Da nun nicht länger die automatisch generierte build.xml von NetBeans genutzt werden soll, wird im Folgenden beschrieben wie diese manuell erstellt wurde:

### Genereller Rahmen

Die build.xml besteht aus einem XML-Rahmen, welcher u. a. die auszuführenden Tasks enthält. Ein Beispiel [AAORG14_MTWT]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="jar">

    <target name="clean" description="Delete all generated files">
        <delete dir="classes"/>
        <delete file="MyTasks.jar"/>
    </target>

    <target name="compile" description="Compiles the Task">
        <javac srcdir="src" destdir="classes"/>
    </target>

    <target name="jar" description="JARs the Task">
        <jar destfile="MyTask.jar" basedir="classes"/>
    </target>

</project>
```

In unserem Fall sieht der Rahmen folgendermaßen aus

```
<?xml version="1.0"?>
<project name="AgeGuessingGame" basedir="." default="execute">

	<!-- Set properties for the targets -->
	<property name="src" value="src"/><!-- Source directory -->
	<property name="build" value="build"/><!-- Build directory -->
	<property name="lib" value="lib"/><!-- Directory for external libraries
-->
	<property name="reports" value="log"/><!-- Directory for reports -->
	<property name="classname" value="AgeGuessingGame" /><!-- Specific
classname of this project -->
	<property name="time.stamp.pattern" value="yyyy-MM-dd_HH:mm:ss" /><!--
Specifiy timestamp pattern for filenames etc. -->
	<property name="time.stamp.mills.pattern" value="HH:mm:ss:SSS"/><!--
Specify timestamp pattern with milliseconds for logging/debugging -->

	<!-- Save the actual Timestamp for logging -->
	<tstamp>
		<format property="time.stamp" pattern="${time.stamp.pattern}"/>
	</tstamp>

	<property name="logfile" value="${reports}/${time.stamp}.log"/><!--
Specify the name of the current logfile -->

	<!-- Set classpath for this program and needed libraries -->
	<path id="java">
		<pathelement location="${build}"/>
		<fileset dir="${lib}">
			<include name="*.jar"/>
		</fileset>
```

```
        </path>

<!-- TARGETS HERE -->
<!-- TARGETS HERE -->
<!-- TARGETS HERE -->

</project>
```

Für die vorliegende Aufgabe sollen folgende Targets erstellt werden:

1. Test: Prüfe ob Vorbedingungen erfüllt sind
2. Clean: Löschen ggf. bisher generierter Dateien
3. Create: Erstelle Verzeichnis für zu generierende Daten
4. Compile: Kompiliere das Projekt
5. Run: Führe das Projekt aus
6. Report: Abschließender Report

## Test

Test-Targets um zu überprüfen ob bspw. die benötigten Verzeichnisse und Dateien vorhanden sind. Hierbei wird ein Target genutzt, welches die auszuführenden Tests als Abhängigkeiten sammelt. Somit muss im späteren Verlauf, wann immer auf die Tests bestanden werden soll, nur dieses “Sammel-Target” als Abhängigkeit aufgeführt werden und ggf. hinzukommende Tests können einfach erweitert werden.

```
<!-- Test if report directory is available -->
<target name="test1">
       <available file="${reports}" property="${reports}.present"/>
       <fail message="Reports path '${reports}' not present!"
      unless="${reports}.present"/>
</target>



<!-- Test if source directory is available -->
<target name="test2">
       <available file="${src}" property="${src}.present"/>
       <fail message="Source path '${src}' not present!"
              unless="${src}.present"/>
</target>



<!-- Test if source file is available -->
<target name="test3">
       <available file="${classname}.java" filepath="${src}"
property="${classname}.present"/>
       <fail message="File '${src}/${classname}.java' not present!"
              unless="${classname}.present"/>
</target>


<!-- Just a test compilation. Extend dependencies for further tests. -->
<target name="tests" depends="test1,test2,test3"/>
```

Die Ausgaben der Tests sind im Idealfall leer. Im Fehlerfall entahlten Sie die angegebene Fehlernachrich; diese könnte bspw. auch in einen Errorlog umgelenkt werden, analog zum weiter unten noch vorgestellten Log.

Beispielausgabe im Fehlerfall:

```
test2:

BUILD FAILED
/Users/cobst/Documents/_studium/Master/S09/VWmodSE/Einsendeaufgaben/EA05-BUI/EA05_BUI_cobst/build.xml:43: Source path 'src' not present!
```

## Clean

Das Clean-Target wird ggf. im Verlauf immer erweitert werden müssen um auf neue Umgebungsvariablen wie Verzeichnisse/Dateien einzugehen.

Generell soll immer mitgeloggt werden, bspw. für spätere debuggings. Hierzu wird initial eine Log-Datei mit aktuellem Datum erstellt. Vor jedem Schritt (analog in allen witeren Targets) wird in diese Logdatei geschrieben. Die Einträge der Logdatei sollen auch einen Timestamp mit Millisekunden enthalten um eventuelle Debuggings zu erleichtern.

Final sieht sie im vorliegenden Beispiel so aus:

```
<!-- Clean old builds, if tests ran correctly -->
<target name="clean" depends="tests">

	<!-- Create a new log file for this run -->
	<touch file="${logfile}" datetime="${time.stamp}"
pattern="${time.stamp.pattern}"/>


	<!-- Protocol of current milliseconds in log output for debugging
purposes -->
	<tstamp>
		<format property="clean.stamp"
pattern="${time.stamp.mills.pattern}"/>
	</tstamp>
	<property name="cleanOut" value="Deleting directory: ${build}"/>
	<echo file="${logfile}" append="true" message="${clean.stamp}:
${cleanOut}${line.separator}"/>

	<delete dir="${build}"/>

</target>
```

Beispielausgabe von ant clean

```
clean:
    [touch] Creating
/Users/cobst/Documents/_studium/Master/S09/VWmodSE/Einsendeaufgaben/EA05-BUI/EA05_BUI_cobst/log/2014-12-14_11:06:38.log
   [delete] Deleting directory
/Users/cobst/Documents/_studium/Master/S09/VWmodSE/Einsendeaufgaben/EA05-BUI/EA05_BUI_cobst/build
```

## Create/Compile

Das Compile Target soll die src kompilieren und unter build/classes ablegen. Create sorgt im Vorfeld dafür, dass das Zielverzeichnis vorhanden ist.
Hinweis die Option `includeantruntime` scheint eine seit Ant 1.8 hinzugekommene Variable zu sein, welche gesetzt sein muss.

Zitat aus dem Ant Manual (http://ant.apache.org/manual/Tasks/javac.html)
*"Whether to include the Ant run-time libraries in the classpath; defaults to yes, unless build.sysclasspath is set. It is usually best to set this to false so the script's behavior is not sensitive to the environment in which it is run."*

```
<!-- create the build path if clean ran correctly -->
<target name="create" depends="clean">

	<!-- Protocol of current milliseconds in log output for debugging
purposes -->
	<tstamp>
		<format property="create.stamp"
pattern="${time.stamp.mills.pattern}"/>
	</tstamp>
	<property name="createOut" value="Creating output directory: ${build}"/>
	<echo file="${logfile}" append="true" message="${create.stamp}:
${createOut}${line.separator}"/>

	<mkdir dir="${build}"/>

</target>



<!-- compile source files if create ran correctly -->
<target name="compile" depends="create">

	<!-- Protocol of current milliseconds in log output for debugging
purposes -->
	<tstamp>
		<format property="compile.stamp"
pattern="${time.stamp.mills.pattern}"/>
	</tstamp>
	<property name="compileOut" value="Compiling files from: ${src} to:
${build}"/>
	<echo file="${logfile}" append="true" message="${compile.stamp}:
${compileOut}${line.separator}"/>

	<javac destdir="${build}" includeantruntime="false">
		<src path="${src}"/>
		<classpath refid="java"/>
	</javac>

</target>
```

Beispielausgabe von ant create

```
create:
    [mkdir] Created dir:
/Users/cobst/Documents/_studium/Master/S09/VWmodSE/Einsendeaufgaben/EA05-BUI/E
A05_BUI_cobst/build
```

Beispielausgabe von ant compile

```
compile:
    [javac] Compiling 1 source file to
/Users/cobst/Documents/_studium/Master/S09/VWmodSE/Einsendeaufgaben/EA05-BUI/E
A05_BUI_cobst/build
run:
     [java] Ich errate Ihr Alter!
     [java] Bitte geben Sie Ihren Namen an und bestätigen m
```

## Run

Das von Compile abhängige Target Run führt nun die unter “Classname” angegebene Applikation/Klasse mit dem unter “java” angegebenen Classpath aus.

```
<!-- run the program if compile ran correctly -->
<target name="run" depends="compile">

       <!-- Protocol of current milliseconds in log output for debugging
purposes -->
       <tstamp>
              <format property="run.stamp"
pattern="${time.stamp.mills.pattern}"/>
       </tstamp>
       <property name="runOut" value="Running: ${classname}"/>
       <echo file="${logfile}" append="true" message="${run.stamp}:
${runOut}${line.separator}"/>

       <java classname="AgeGuessingGame">
              <classpath refid="java"/>
       </java>
</target>
```

Beispielausgabe von ant run

```
run:
     [java] Ich errate Ihr Alter!
     [java] Bitte geben Sie Ihren Namen an und bestätigen m …
```

## Report

Das Report Target schreibt nun noch abschließend einen Eintrag in das Logfile um zu protokollieren, wann der Programmablauf abgeschlossen wurde.

Die Abhängigkeit von Run wurde gesetzt, da ein Report am Ende der Ausführung andernfalls sinnols wäre.

```
<!-- Report when run done -->
<target name="report" depends="run">

       <!-- Protocol of current milliseconds in log output for debugging
purposes -->
       <tstamp>
              <format property="report.stamp"
pattern="${time.stamp.mills.pattern}"/>
       </tstamp>
       <property name="reportOut" value="Successfully finished running:
${classname}"/>
       <echo file="${logfile}" append="true" message="${report.stamp}:
${reportOut}${line.separator}"/>
```

```
</target>
```

## Die resultierende Log-Datei

Zu letzt hier nun ein Beispiel der geschriebenen Log-Datei

```
2014-12-14_11/06/38.log

11:06:38:947: Deleting directory: build
11:06:38:999: Creating output directory: build
11:06:39:004: Compiling files from: src to: build
11:06:40:030: Running: AgeGuessingGame
11:27:48:638: Successfully finished running: AgeGuessingGame
```

# Advanced 1 - Maven

Aufgabe: Bringen sie Maven oder Gradle ebenfalls - mit einigen üblichen Tasks - zum laufen.

Beweisen sie auch hier, dass es ihr Werk ist.

Hinweis:

- Wie starte ich den Code in Maven? Bauen Sie >= 1 Testklasse für Maven.

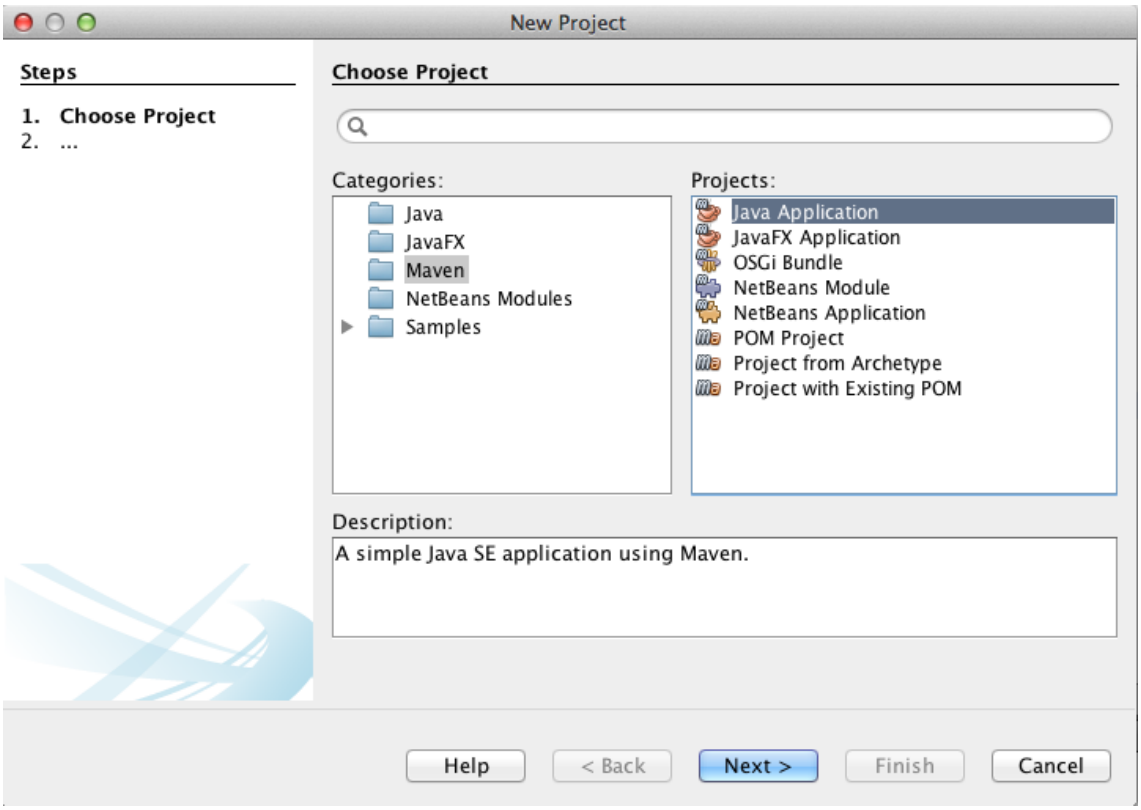
## Installation von Maven als NetBeans Plugin

In Aktuellen NetBeans-Versionen (seit Version 6.7 bzw. Maven 3 ab Version 7.0 [MAORG14_NM]). Da NetBeans Version 8.0.1 genutzt wird, ist keine Installation erforderlich.

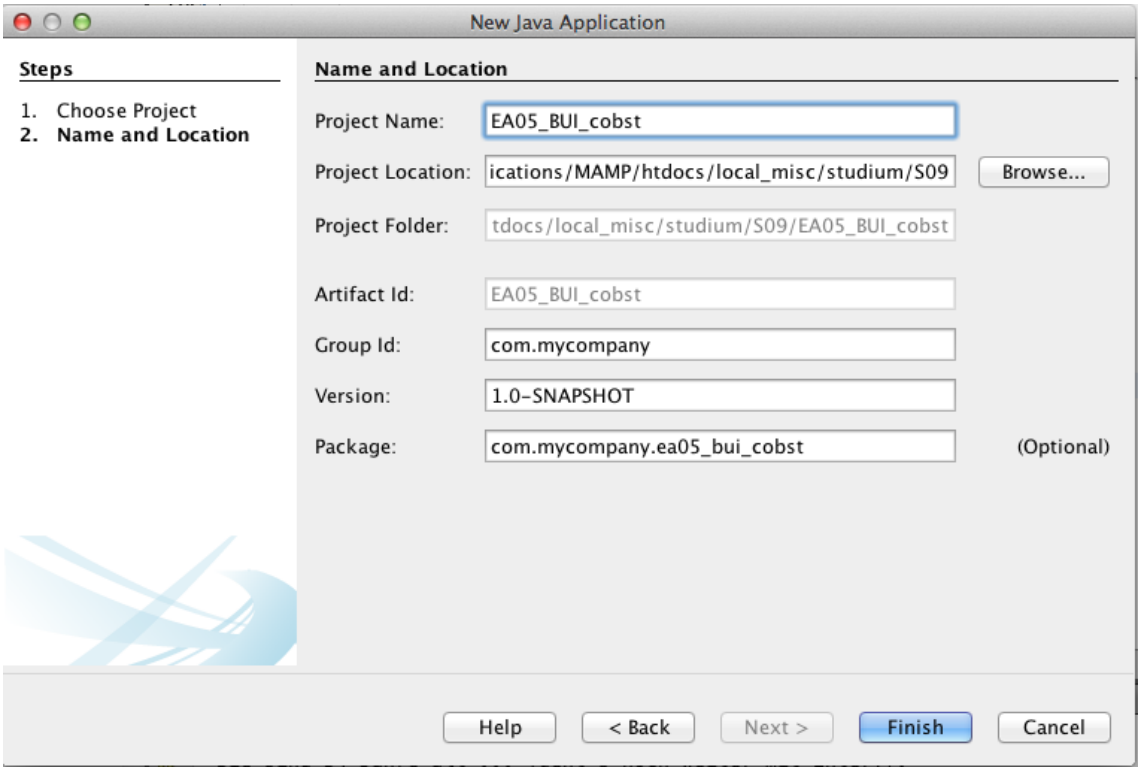
## Nutzung von Maven

### Ein neues Projekt

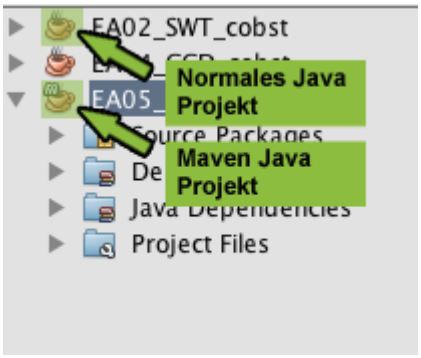
Zunächst wird ein neues Maven-Projekt erstellt:



und mit einem Namen versehen:




Ansicht des neuen, leeren Projektes in der Projektansicht. Hier wird durch ein etwas anders gestaltetes Icon bereits angezeigt, dass es sich um ein Maven-Projekt handelt.




Zu Testzwecken wurde nun das bestehende CCD-Projekt in das Maven-Projekt kopiert.

## Erste Ausführung

Wenn man nun die Main-Class erstmalig ausführt muss NetBeans noch einige Maven-Pakete herunterladen:

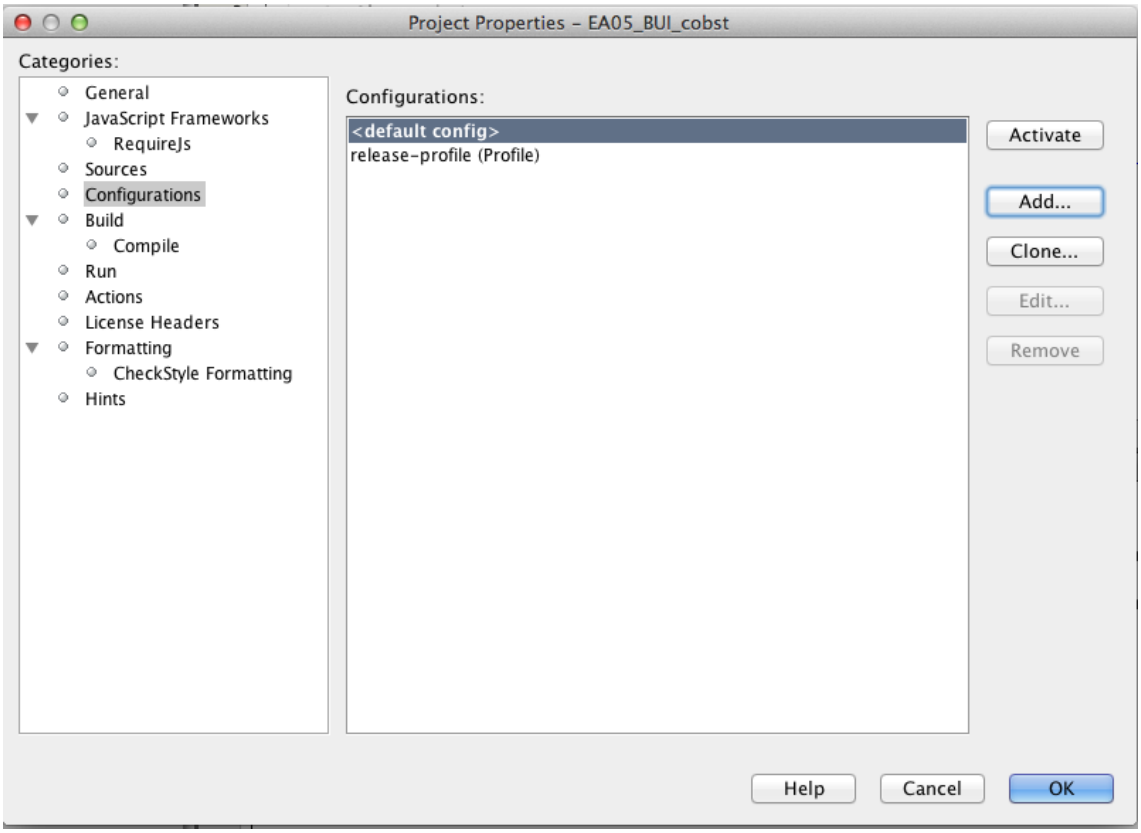
```
cd /Applications/MAMP/htdocs/local_misc/studium/S09/EA05_BUI_cobst; JAVA_HOME=/Library/Java/JavaVirtualMachines/
Scanning for projects...
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/1.2.1/exec-maven-plugin-1.2

Downloaded: http://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/1.2.1/exec-maven-plugin-1.2.
Downloading: http://repo.maven.apache.org/maven2/org/codehaus/mojo/mojo-parent/28/mojo-parent-28.pom

------------------------------------------------------------------------
Building EA05_BUI_cobst 1.0-SNAPSHOT
------------------------------------------------------------------------
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.5/maven-resou

Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.5/maven-resour
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/19/maven-plugins-19.pom

--- maven-resources-plugin:2.5:resources (default-resources) @ EA05_BUI_cobst ---

--- maven-compiler-plugin:2.3.2:compile (default-compile) @ EA05_BUI_cobst ---

--- exec-maven-plugin:1.2.1:exec (default-cli) @ EA05_BUI_cobst ---
Downloading: http://repo.maven.apache.org/maven2/junit/junit/3.8.2/junit-3.8.2.pom

Downloading: http://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.jar

Downloaded: http://repo.maven.apache.org/maven2/org/apache/commons/commons-exec/1.1/commons-exec-1.1.jar (52 KB
Ich errate Ihr Alter!
Bitte geben Sie Ihren Namen an und bestätigen mit Enter:
```

Folgend läuft das Projekt jedoch zunächst einmal ohne ersichtliche Unterschiede zu einem Build ohne Maven:

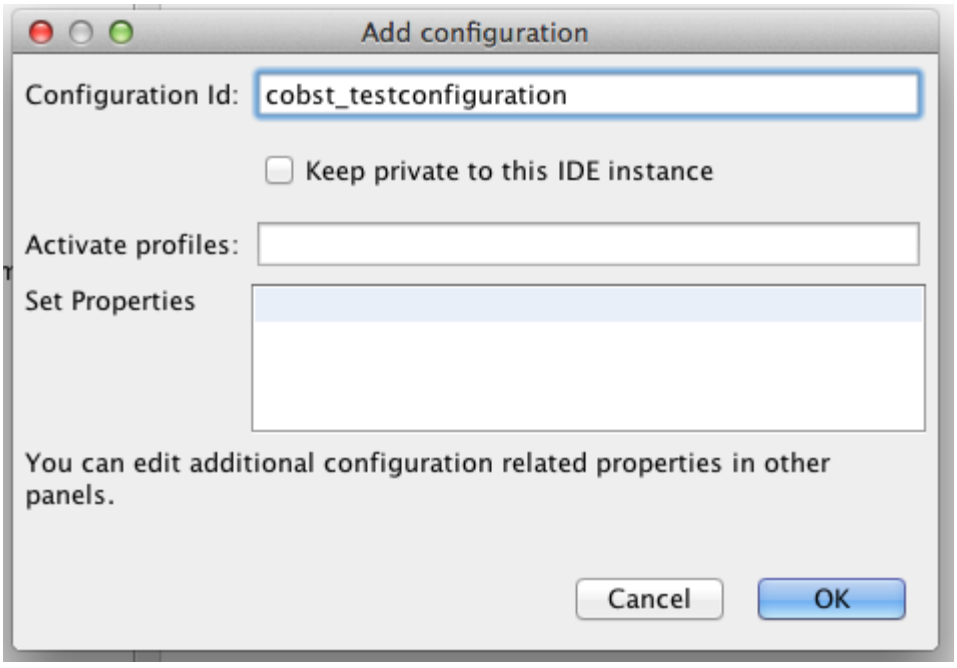
```
Ich errate Ihr Alter!
Bitte geben Sie Ihren Namen an und bestätigen mit Enter:
Christian Obst
Vielen Dank Christian Obst

Bitte fo1gen Sie nun den folgenden Anweisungen (jeweils weiter mit Enter)

Liebe/r Christian Obst, bitte denken Sie sich eine Geheimzahl (n ∈ N) zwischen 1 und 9 aus (weiter mit Enter).
   Hinweis: das Programm liest die kommenden Eingaben NICHT mit (es sei denn, es ist explizit angegeben).
Sie können also die Eingabezeile als Platz um "Zwischenergebnisse" zu notieren nutzen.

Multiplizieren Sie Ihre Geheimzahl mit 2 (weiter mit Enter).

Addieren Sie 2 zum Ergebnis (weiter mit Enter).

Multiplizieren Sie das Ergebnis mit 100 (weiter mit Enter).

Dividieren Sie nun durch 2 (weiter mit Enter).
```

## Konfiguration von Maven

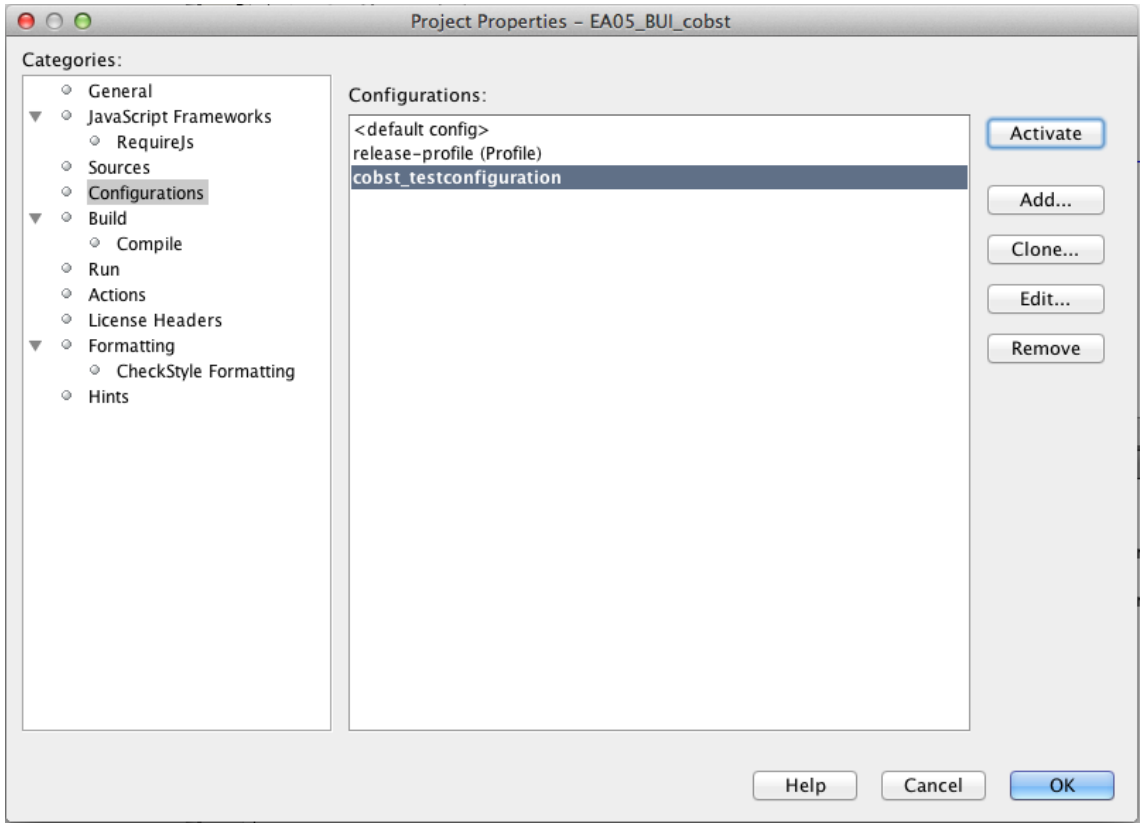
Nun kann das Maven-Projekt konfiguriert werden. Hierzu wurde zunächst eine eigene Konfiguration angelegt. Dies geschieht über einen Rechtsklick auf das Projekt und folgend die Auswahl “Configurations”:



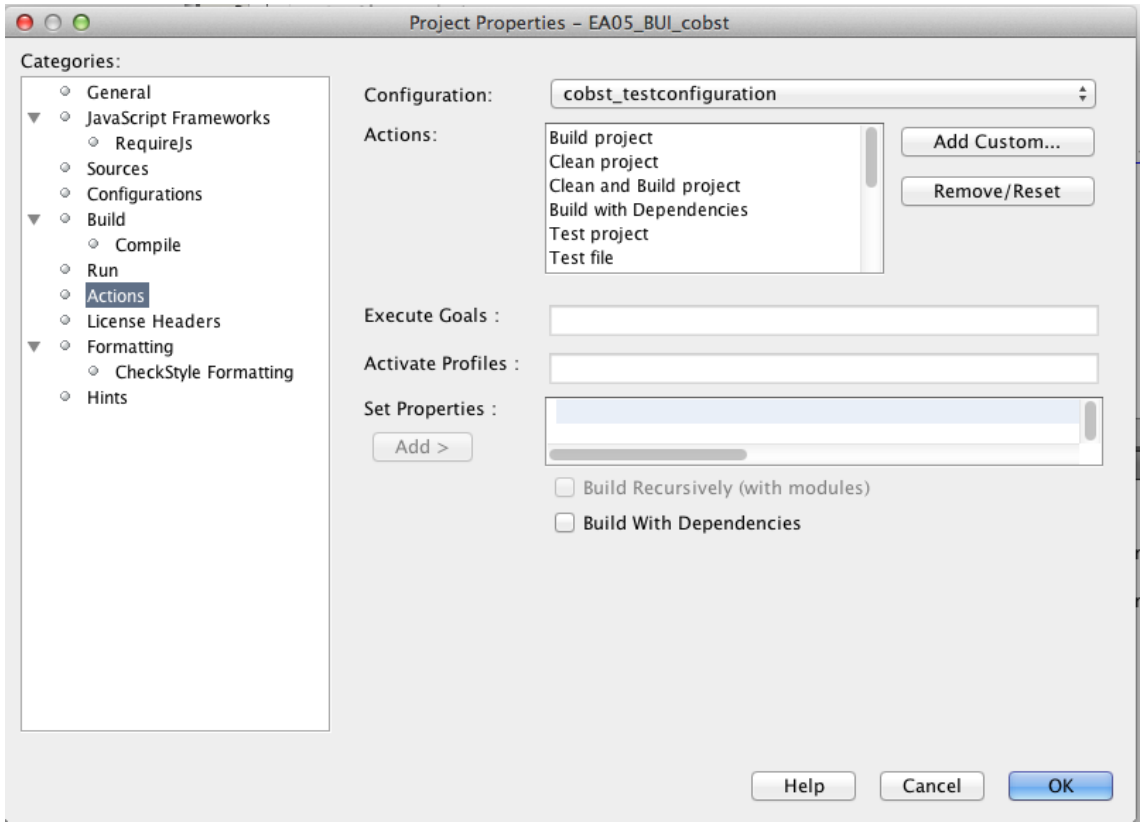

Hier kann nun eine eigene Konfiguration hinzugefügt werden:



Folgend muss diese noch aktiviert werden, damit sie für das Projekt genutzt wird:

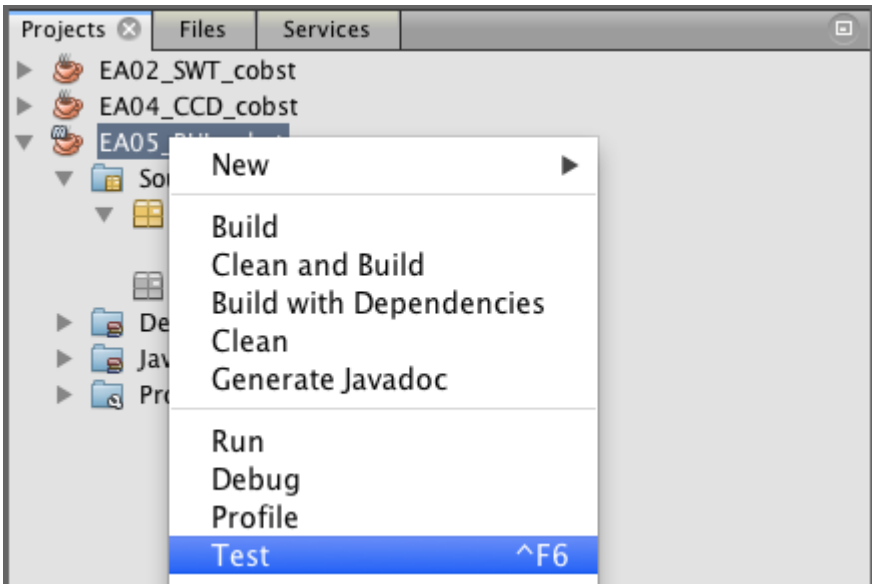



Unter Actions können nun die durchzuführenden Actions der entsprechenden Konfiguration gewählt, konfiguriert, gelöscht und/oder hinzugefügt werden:



## Testklassen für Maven

Zunächst kann das aktuelle Projekt, wie ein normales NetBeans-Projekt mittels Rechtsklick auf das Projekt und der Auswahl “Test” getestet werden:




Hier muss NetBeans wieder einige Maven-Pakete nachinstallieren:

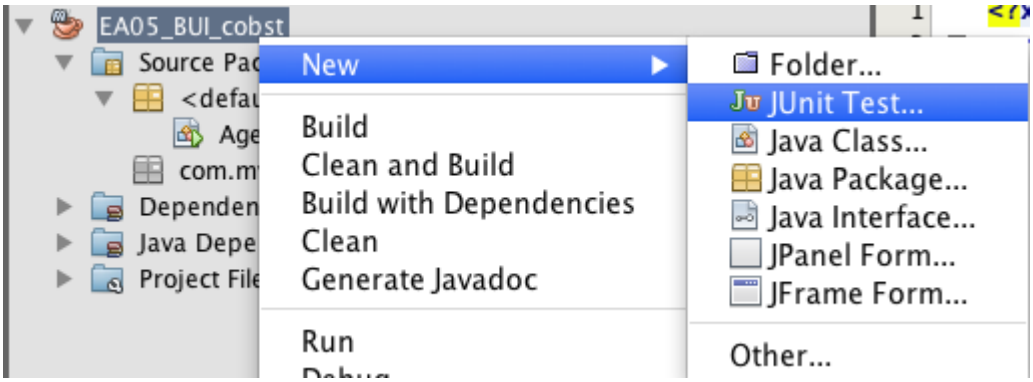
```
cd /Applications/MAMP/htdocs/local_misc/studium/S09/EA05_BUI_cobst; JAVA_HOME=/Library/Java/JavaVirtualMachines/
Scanning for projects...

------------------------------------------------------------------------
Building EA05_BUI_cobst 1.0-SNAPSHOT
------------------------------------------------------------------------
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-surefire-plugin/2.10/maven-suref

Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-surefire-plugin/2.10/maven-surefi
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire/2.10/surefire-2.10.pom

Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire/2.10/surefire-2.10.pom (12 KB
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/20/maven-parent-20.pom

Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/20/maven-parent-20.pom (25 KB at 1

--- maven-resources-plugin:2.5:resources (default-resources) @ EA05_BUI_cobst ---

--- maven-compiler-plugin:2.3.2:compile (default-compile) @ EA05_BUI_cobst ---

--- maven-resources-plugin:2.5:testResources (default-testResources) @ EA05_BUI_cobst ---

--- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ EA05_BUI_cobst ---

--- maven-surefire-plugin:2.10:test (default-test) @ EA05_BUI_cobst ---
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-api/2.0.9/maven-plugin-api-2.0.9.

Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit3/2.10/surefire-junit3-2
```

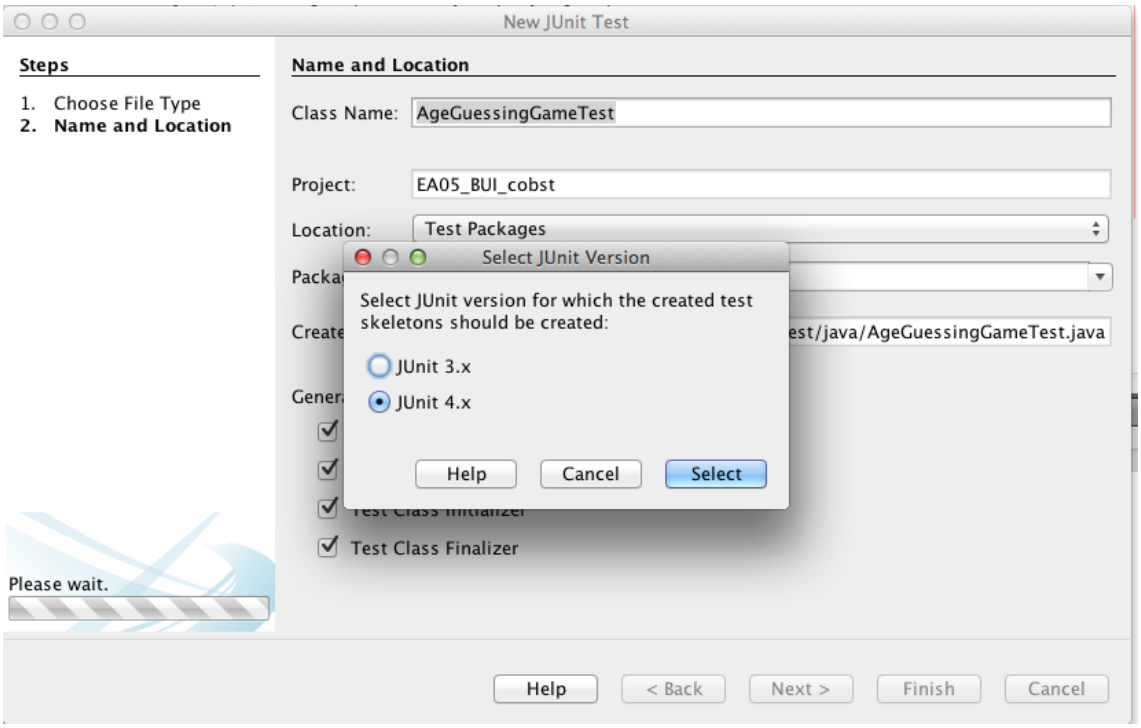
Das Ergebnis ist noch sehr überschaubar, da noch keine Tests definiert wurden:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
```

```
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 9.344s
Finished at: Sun Dec 14 13:57:21 CET 2014
Final Memory: 9M/220M
------------------------------------------------------------------------
```

Nun wird ein neuer jUnit-Test erzeugt:

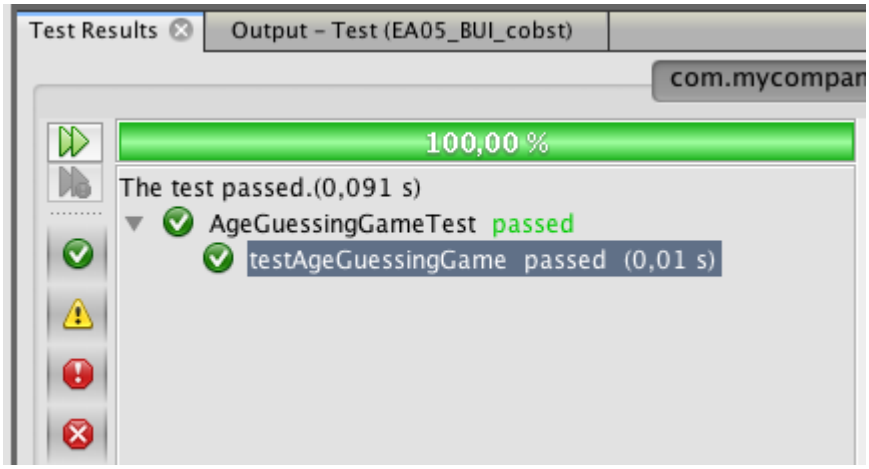


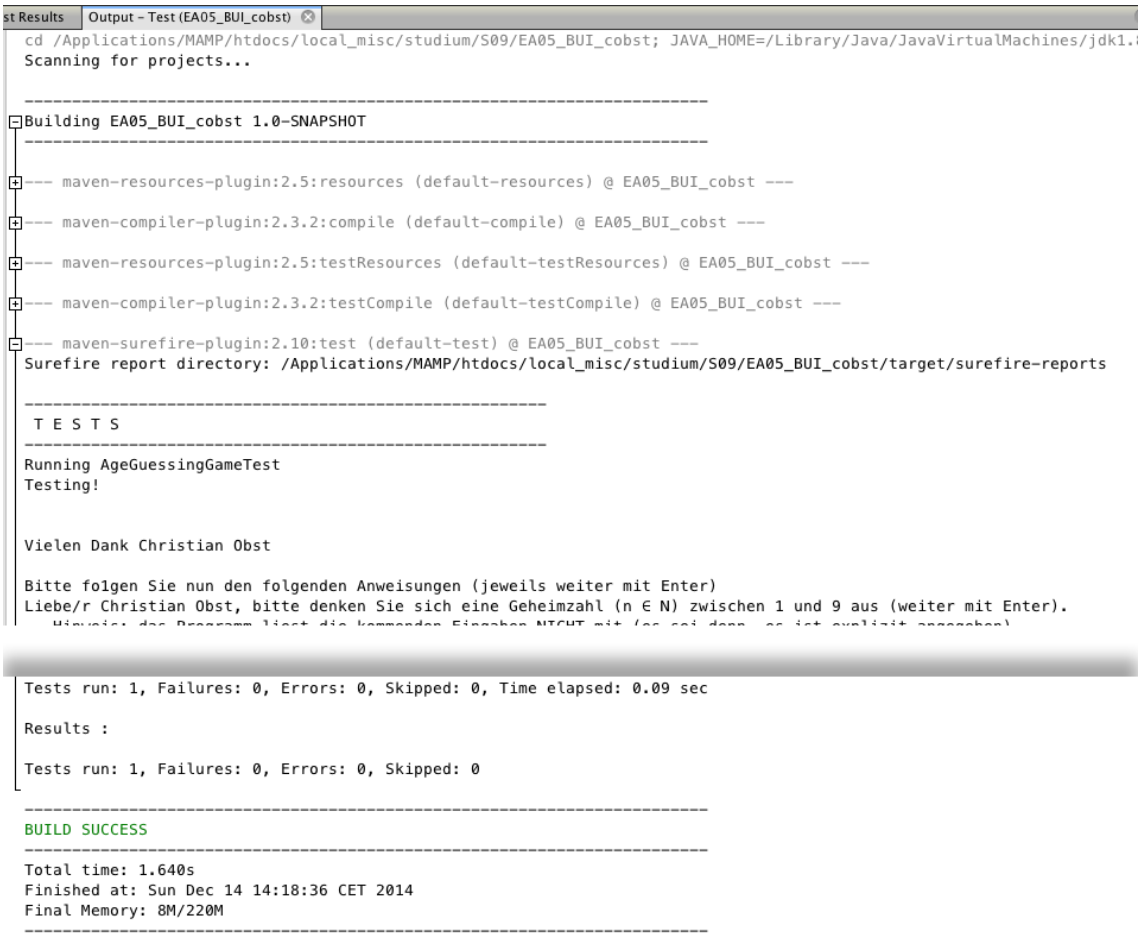
und ein einfacher Testaufruf hinzugefügt:

```
@Test
public void testAgeGuessingGame() {
    AgeGuessingGame agg = new AgeGuessingGame();
    agg.setTestrun();
    agg.setName("Christian Obst");
    agg.setBirthdayInActYear("Y");
    agg.setResult("345");
    try {
        agg.run();
    } catch (IOException e) {

    }
}
```

Mittels Maven kann nun der jUnit-Test ausgeführt werden und folgend entweder das Ergebnis unter “Test Results” eingesehen werden:



Oder die Ausgabe des Testruns unter Output eingesehen werden:

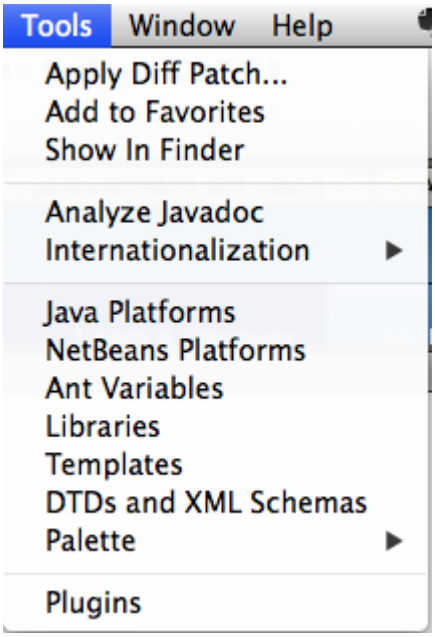


# Advanced 2 - Gradle

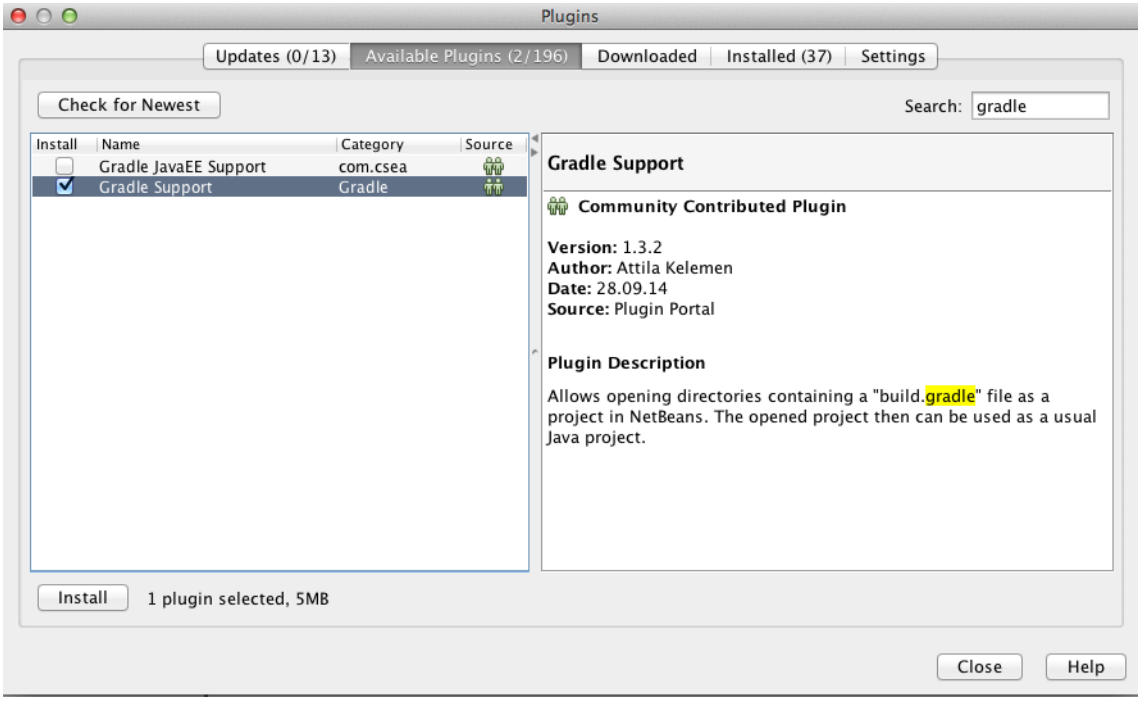
Aufgabe: Bringen sie Maven oder Gradle ebenfalls - mit einigen üblichen Tasks - zum laufen. Beweisen sie auch hier, dass es ihr Werk ist.

## Installation von Gradle als NetBeans Plugin

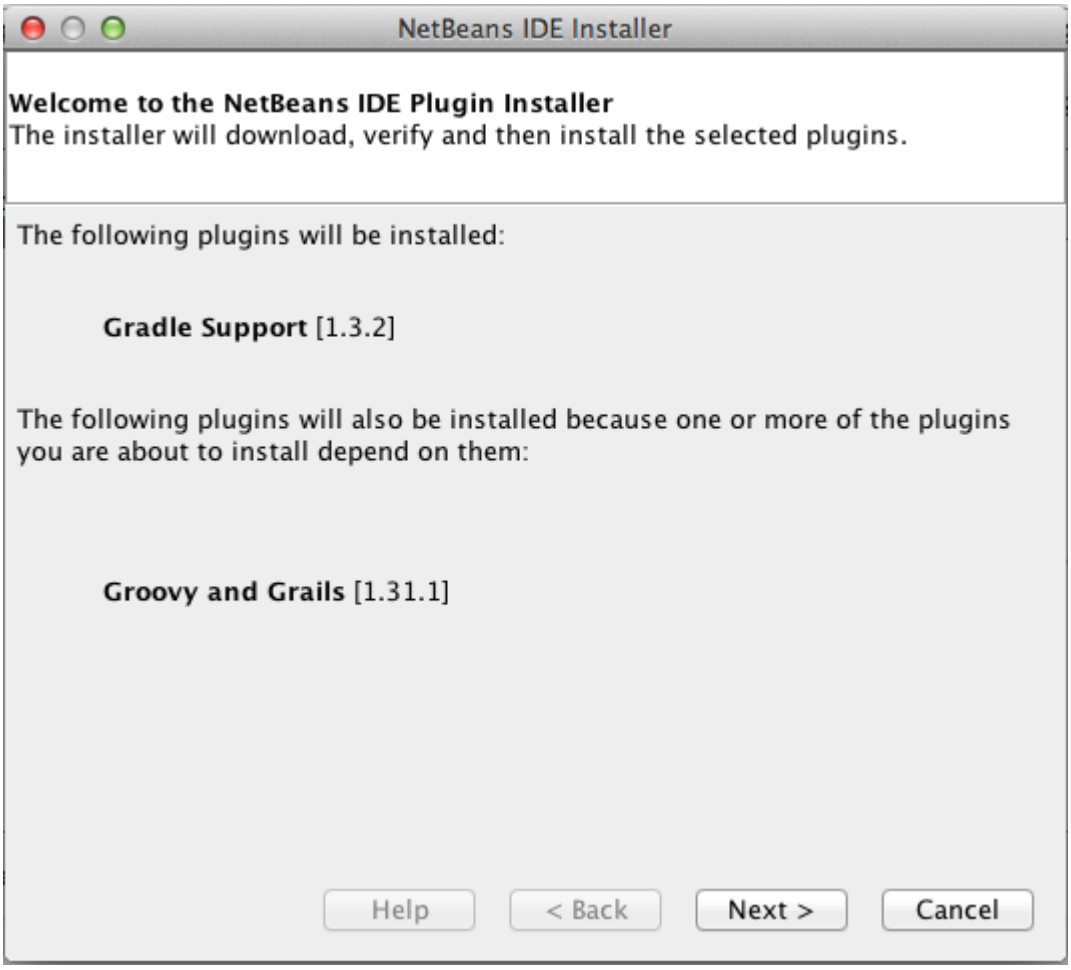
Gradle kann einfach als Plugin installiert werden. Hierzu findet man unter Tools das Untermenü Plugins:



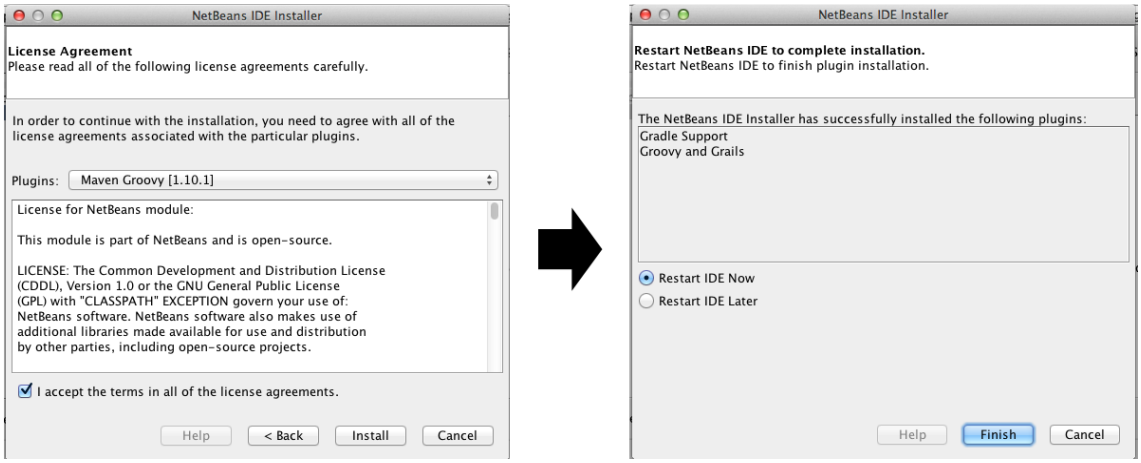

Hier kann dann unter Available Plugins nach Gradle gefiltert und das entsprechende Plugin zur Installation gewählt werden:



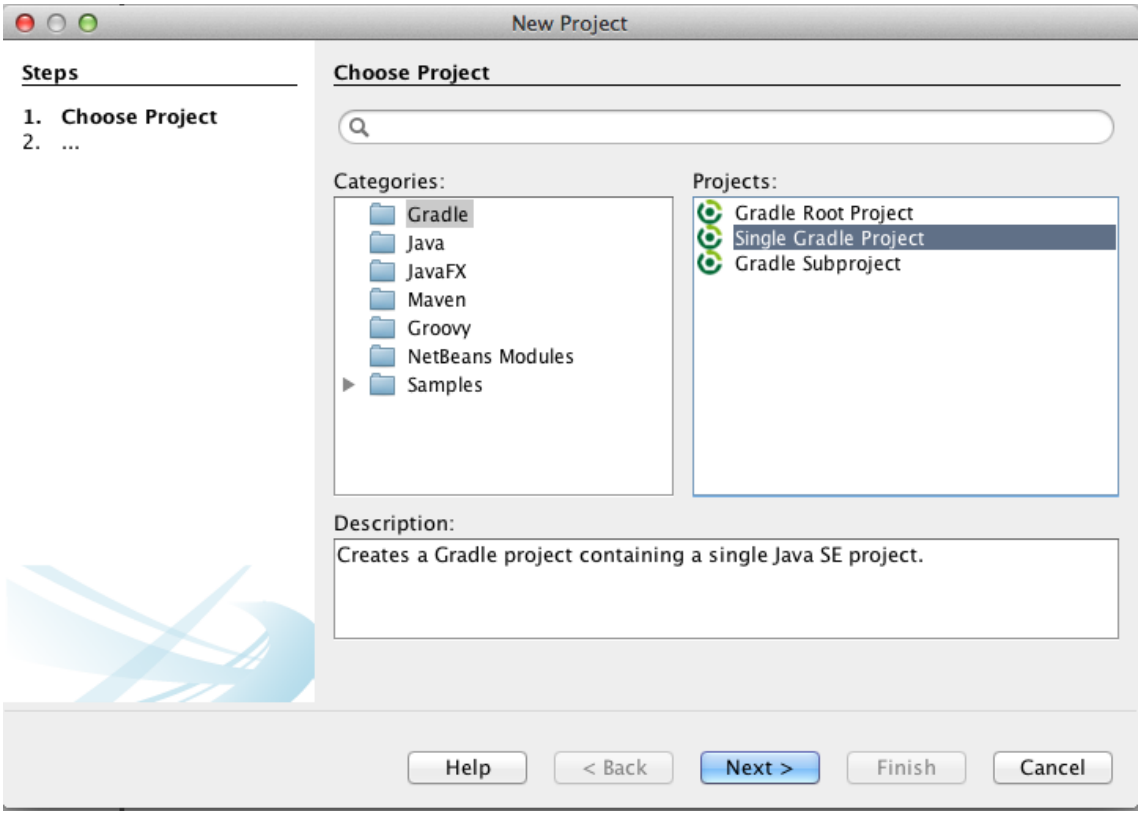
Von Gradle ggf. zusätzlich benötigte, aber nicht vorhandene Plugins können direkt mitinstalliert werden:



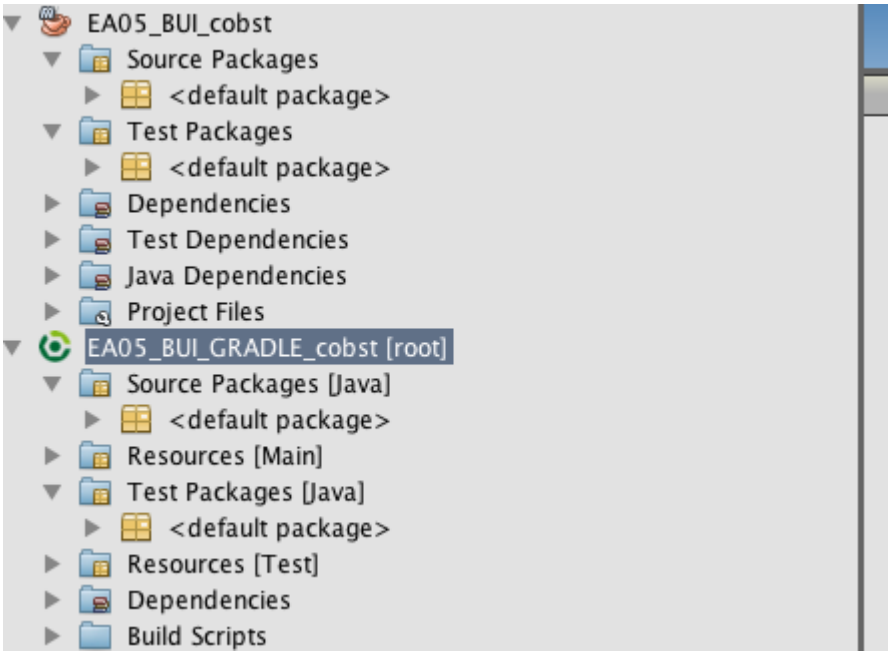

Nun kann die Installation erfolgen und folgend die IDE neu gestartet werden:



Nach dem Neustart steht nun der Projekttyp Single Gradle Project zur Verfügung:

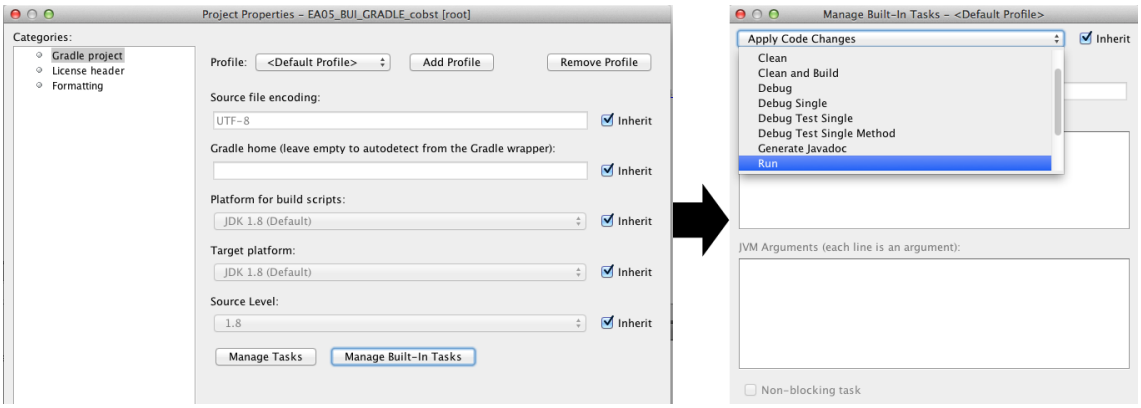



Nachdem ein neues Project mit Namen erstellt wurde findet sich eine ähnliche Ansicht im Inspector, wie dies auch schon bei normalen Java oder Maven-Java-Projekten der Fall ist:



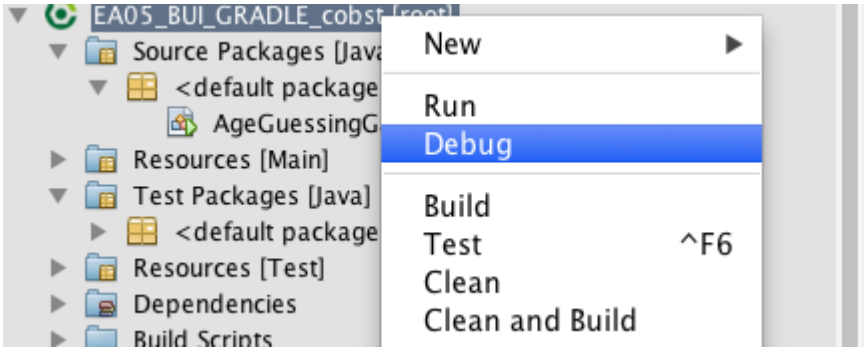
# Konfiguration von Gradle

Auch unter Gradle können nun unter “Properties” Tasks bearbeitet oder neue definiert werden:



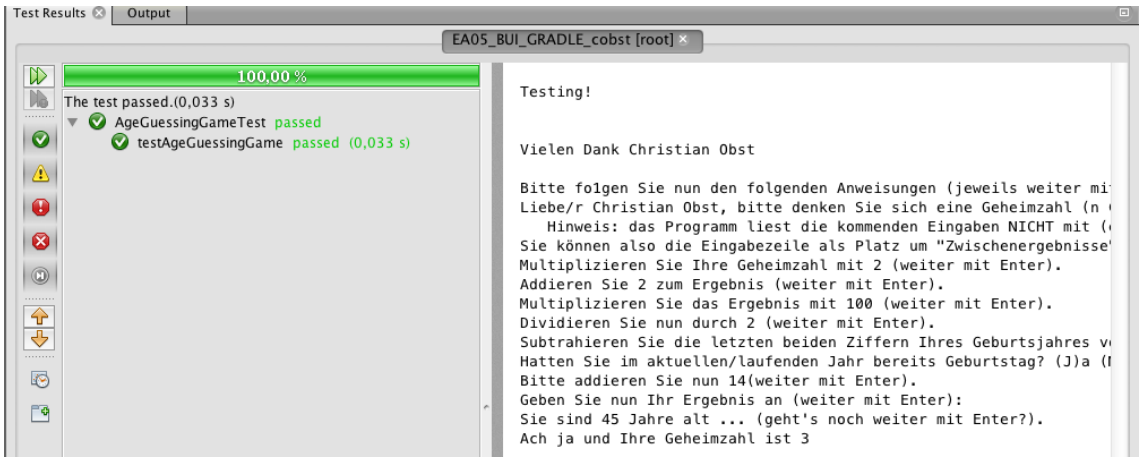

# Nutzung von Gradle

In Gradle können nun, genau so wie in Maven, direkt aus dem Submenü Tasks wie Build oder Test ausgeführt werden:




```
Executing: gradle build

:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar
:assemble
:compileTestJava
:processTestResources UP-TO-DATE
:testClasses
:test
:check
:build

BUILD SUCCESSFUL

Total time: 3.557 secs
```

```
Executing: gradle :cleanTest :test

:cleanTest
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test

BUILD SUCCESSFUL

Total time: 1.606 secs
```



# Fazit

## Vergleich von Maven und Gradle mit Ant

Vergleicht man den Aufwand eine oder mehrere XML-Konfigurationsdateien für Ant schreiben zu müssen inkl. dessen, dass diese dann wohlgeformt sein müssen etc., so erleichtert ein Build-Management-Tool wie Maven oder Gradle die Arbeit schon erheblich.

## Vergleich von Maven mit Gradle

Bei der Nutzung von Maven oder Gradle konnten keine großen Unterschiede festgestellt werden, wenn auch die Maven Nutzung leichter von der Hand zu gehen schien. Um hier jedoch final Resümieren zu können müssten die Möglichkeiten der beiden Tools mehr en Detail betrachtet und ausgelotet werden.

## Resümé

Die Nutzung von Build-Management-Tools erleichtert wiederkehrende Aufgaben/Abläufe enorm. Hat man sich in die Arbeitsweise mit diesen erst einmal eingefunden überwiegen die Vorteile wie Zeitersparnis, standardisierte Tests, Verfahrenssicherheit etc. schnell den anfänglichen Aufwand.

# Quellen


[NBIDE14_BT] 2014-12-12
https://netbeans.org/features/java/build-tools.html

[BRW14] 2014-12-12
http://brew.sh/

[AAORG14_MTWT] 2014-12-12
http://ant.apache.org/manual/tutorial-writing-tasks.html

[MAORG14_NM] 2014-12-13
http://maven.apache.org/netbeans-module.html