Maven

- Compiles your java files.
- Packs your class files into java archive files.
- Maintains dependencies your project has.
- Creates documentation.
- Runs tests.
- Basically all steps requiered in the life cycle of a java software project.

Maven

- Not specific to an IDE
- Works with netbeans, idea and eclipse
- Commandline maven program: `mvn`.
- Download from `maven.apache.org`
- Needs java to run.

Convention over configuration

- Maven provides a default behaviour for the build process.
- Default is mostly what you need. Most details can be adjusted.
- Default project structure:

Source code	<project>/src/main/java
Resources	<project>/src/main/resources
Tests	<project>/src/test/java
Jar file(s)	<project>/target
Classes	<project>/target/classes
Test resources	<project>/src/test/resources

Projects pom.xml

- Contains the description and configuration details of the project.
- Needs `groupId`, `artifactID` and `version`
- Used to distinguish different maven projects in a maven repository. Take care when you are selecting names.
- Convention: Name of group id: reverse domain name of organization.

  `de.uni.tuebingen.sfs`
- Overwrites entries in the maven "Super POM"

Minimal pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Sets

`modelVersion` to 4.0.0
`groupID` to com.mycompany.app
`artifactId` to my-app
`version` to 1

Create project

- Done with your IDE

  or

```
mvn archetype:generate
-DgroupId=com.mycompany.app
-DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-
quickstart -DinteractiveMode=false
```

- Creates needed directory structure and `pom.xml`

Project life cycles

- Specify what maven should do.
- Very specific targets. Only a subset is needed

Clean up project	mvn clean
Compile source code	mvn compile
Build application	mvn package
Run tests	mvn test
Install to local repository	mvn install

Dependencies

- Are kept in repositories
- Local repository: `<home>/.m2`
- Global repository. Specified by maven configuration.
- Network repositories.
- Specify dependencies in a `<dependency>` section and let maven fetch and install the depency for you.

Depencies in pom.xml

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        …
    </dependency>
 </dependencies>
```

Plugins

- Do all the work. Compile code, create jar files, run tests.
- Are responsible for a build phase.
- Defaults are usually ok. You can change the behaviour of the default plugins by adding parameters.
- Look at page of plugin on `https://maven.apache.org/plugins/index.html` to see what can be adjusted. Also google is your friend.

Plugins

- You can overwrite/configure the default plugin for a build phase.
- Examples on next pages.

  First example configures a plugin.

  Second example configures a plugin and binds the plugin to a build phase.

Customize jar plugin: Add main class to jar so `java -jar name.jar` will find a main class

```
.
.
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.0.2</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>de.uni.tuebingen.sfs.mavenproject1.App</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
.
.
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>2.6</version>
      <configuration>
        <!-- get all project dependencies -->
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <!-- MainClass in mainfest make a executable jar -->
        <archive>
          <manifest>
            <mainClass>de.uni.tuebingen.sfs.Whatever</mainClass>
          </manifest>
        </archive>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <!-- bind to the packaging phase -->
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```