Document History

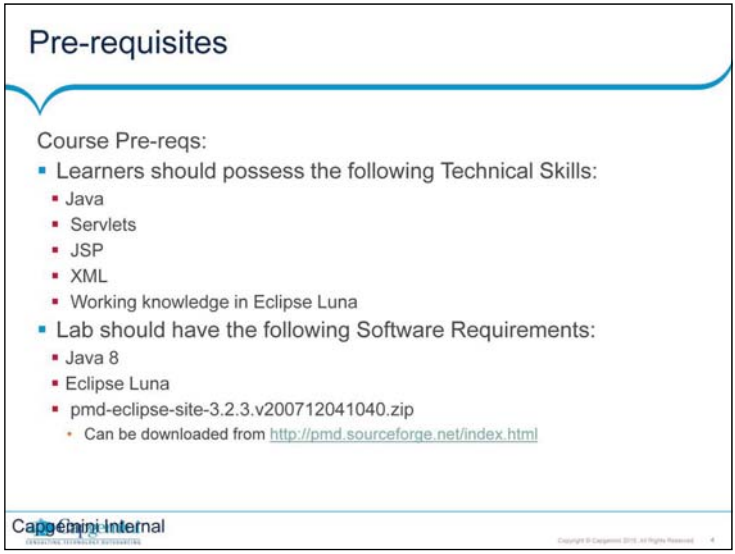
Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
21/11/2011	1.0	Maven-3.0	Rathnajothi Perumalsamy	Content Creation
16/05/2016	1.1	Maven-3.3.9	Zainab	Modified as per TOC for ELTP

Capgemini

Course Goals and Non Goals

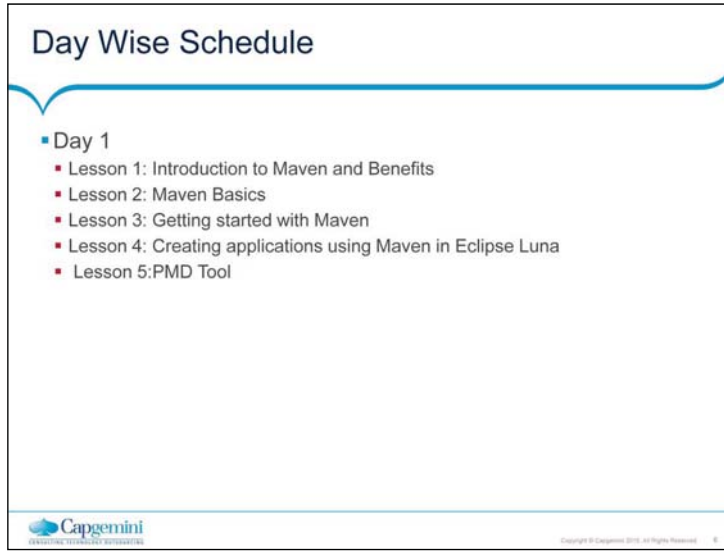
- Course Goals
  - Enabling Java/JEE projects to be build efficiently
  - To understand PMD Tool with Eclipse
- Course Non Goals
  - Generating reports and continuous integration of projects using Maven

Capgemini


Pre-requisites
Course Pre-reqs:
Learners should possess the following Technical Skills:
Java
Servlets
JSP
XML
Working knowledge in Eclipse Luna
Lab should have the following Software Requirements:
Java 8
Eclipse Luna
pmd-eclipse-site-3.2.3.v200712041040.zip
Can be downloaded from http://pmd.sourceforge.net/index.html
Capgemini Internal

Intended Audience

- Developers.


Day Wise Schedule
Day 1
Lesson 1: Introduction to Maven and Benefits
Lesson 2: Maven Basics
Lesson 3: Getting started with Maven
Lesson 4: Creating applications using Maven in Eclipse Luna
Lesson 5:PMD Tool
Capgemini

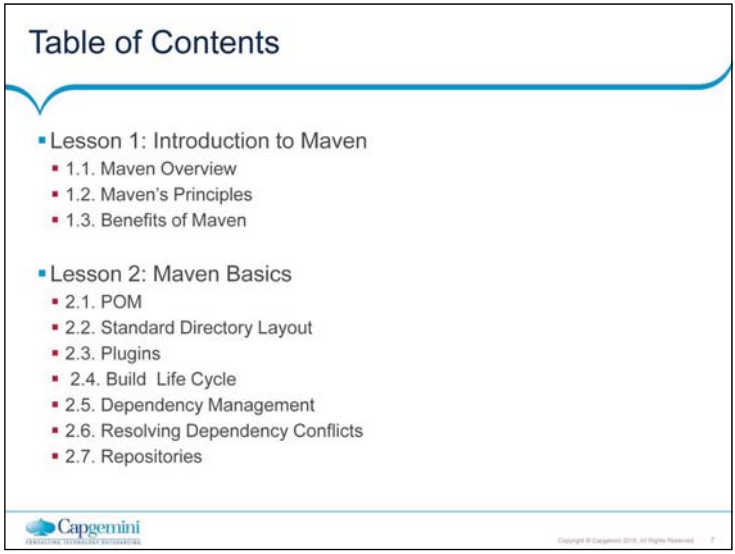


Table of Contents
Lesson 1: Introduction to Maven
1.1. Maven Overview
1.2. Maven's Principles
1.3. Benefits of Maven
Lesson 2: Maven Basics
2.1. POM
2.2. Standard Directory Layout
2.3. Plugins
2.4. Build Life Cycle
2.5. Dependency Management
2.6. Resolving Dependency Conflicts
2.7. Repositories
Capgemini

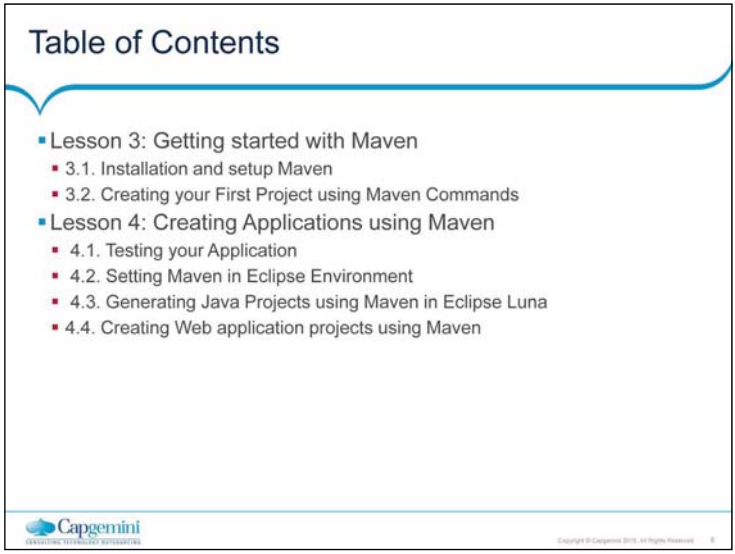


Table of Contents
Lesson 3: Getting started with Maven
3.1. Installation and setup Maven
3.2. Creating your First Project using Maven Commands
Lesson 4: Creating Applications using Maven
4.1. Testing your Application
4.2. Setting Maven in Eclipse Environment
4.3. Generating Java Projects using Maven in Eclipse Luna
4.4. Creating Web application projects using Maven
Capgemini

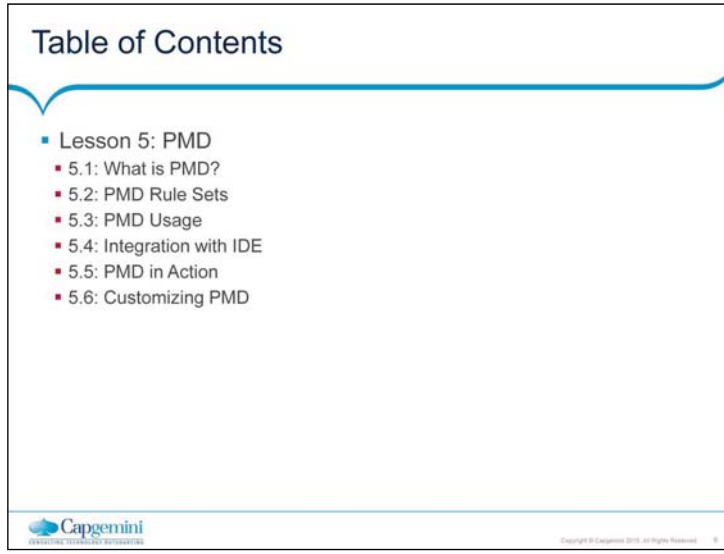


Table of Contents
Lesson 5: PMD
5.1: What is PMD?
5.2: PMD Rule Sets
5.3: PMD Usage
5.4: Integration with IDE
5.5: PMD in Action
5.6: Customizing PMD
Capgemini

References

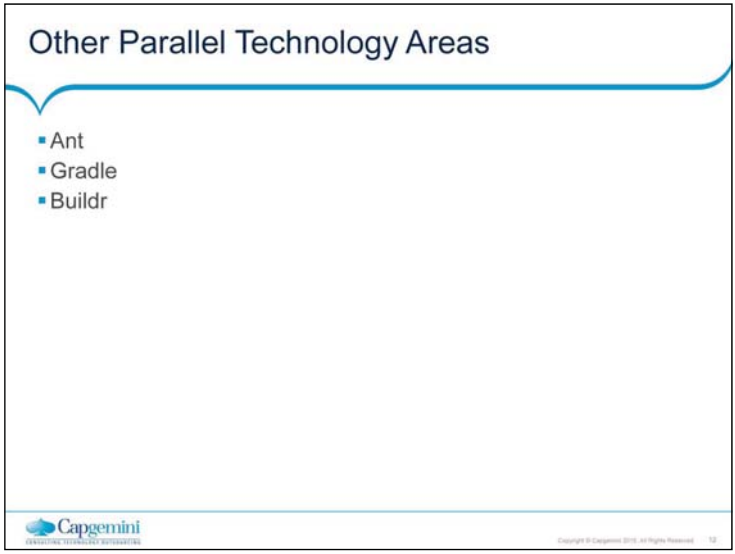
- Books
  - Apache Maven2 Effective Implementation by Maria Odea Ching and Brett Porter.
  - Better Builds with Maven by Vincent Massol and Jason van Zyl
  - The Maven Handbook by Tim O'Brien, stuart McCulloch and Brian Demers.(Sonatype)
  - Apache Maven 3.0 Cookbook by Srirangan

Capgemini

References

- Websites
  - http://www.maven.apache.org
  - Maven 2 Plugins Project
  - http://mojo.codehaus.org/
  - http://maven.apache.org/plugins/index.html
  - http://pmd.sourceforge.net/
- Popular Repository
  - http://repo1.maven.org/maven2/
- Repository for Jboss
  - http://repository.jboss.org/maven2

Capgemini

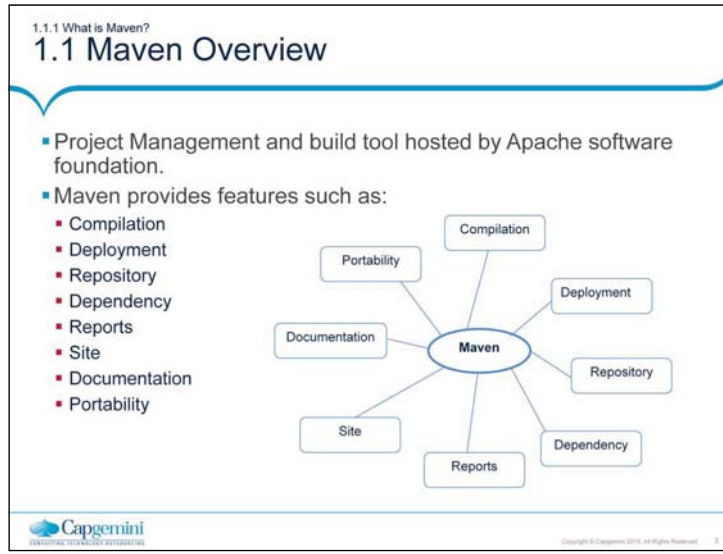

Other Parallel Technology Areas
Ant
Gradle
Buildr
Capgemini


Maven
Introduction to Maven

Lesson Objectives

- In this lesson, you will learn:
- Maven Overview
  - What is Maven?
  - Maven's Objective
- Maven's Principles
- Benefits of Maven

Capgemini

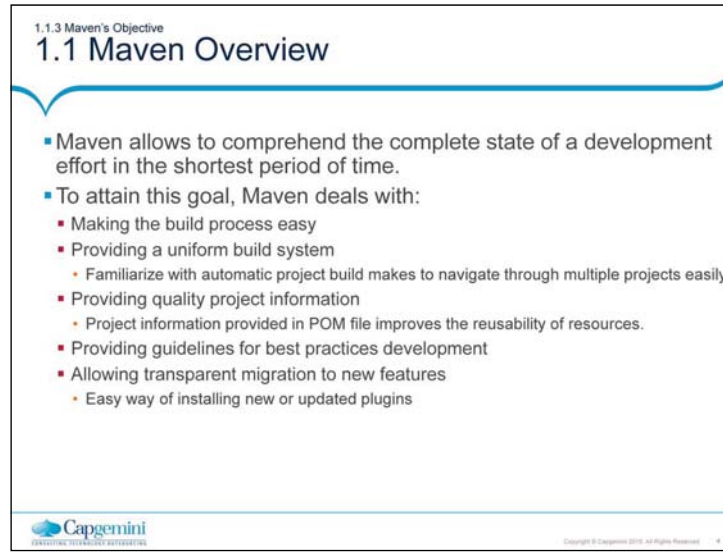


- **Maven** is a tool for project management and build automation.
- **Maven is not Ant++**
- Maven serves a similar purpose to the Apache Ant tool, but it is based on different concepts and works in a profoundly different manner.
- Maven is hosted by the Apache Software Foundation, where it was formerly part of the Jakarta Project**.**
- It is pronounced as “May-ven” and Maven is a Yiddish (Jewish Lang) word which means “**accumulator of knowledge**“.

Actually, there is a open source project named Jakarta Turbine which was facing complexity issues with module build processes and atlast the Apache team came up with a solution to simplify this build process which was than termed as Apache Maven.

Maven provides features such as:

- Build tool Capabilities
- Run Reports
- Generate a website
- Dependency Management
- Repositories (Reusable Plug-ins)
- Continuous Integration build systems
- Portable
- Building configuration using maven are portable to another machine without any effort



Maven allows to comprehend the complete state of a development effort in the shortest period of time. To attain this goal, Maven deals with:

Making the build process easy:

While using Maven doesn't eliminate the need to know about the underlying mechanisms, Maven does provide a lot of shielding from the details.

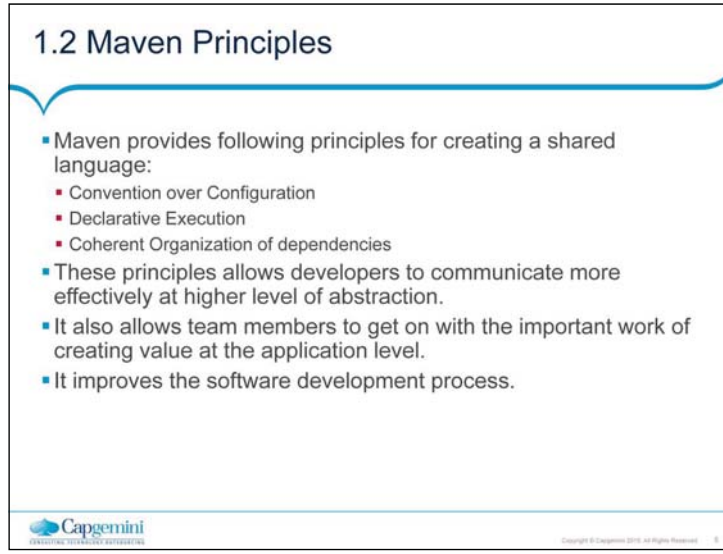
Providing a uniform build system:

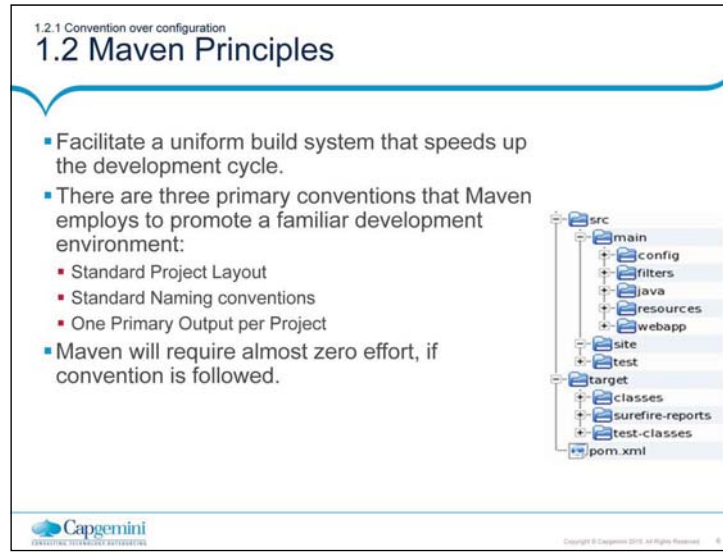
Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system. Once you familiarize yourself with how one Maven project builds you automatically know how all Maven projects build saving you immense amounts of time when trying to navigate many projects.

Providing quality project information:

Maven provides plenty of useful project information that is in part taken from your POM and in part generated from your project's sources. For example, Maven can provide:

- Change log document created directly from source control
- Cross referenced sources
- Mailing lists
- Dependency list
- Unit test reports including coverage


1.2 Maven Principles
Maven provides following principles for creating a shared language:
Convention over Configuration
Declarative Execution
Coherent Organization of dependencies
These principles allows developers to communicate more effectively at higher level of abstraction.
It also allows team members to get on with the important work of creating value at the application level.
It improves the software development process.
Capgemini



Conventions over configuration used to facilitate a uniform build system that speeds up the development cycle.

There are three primary conventions that Maven employs to promote a standardized development environment:

Standard Project Layout:

Having a common directory layout would allow for users familiar with one Maven project to immediately feel at home in another Maven project. The advantages are analogous to adopting a site-wide look-and-feel.

Standard Naming conventions

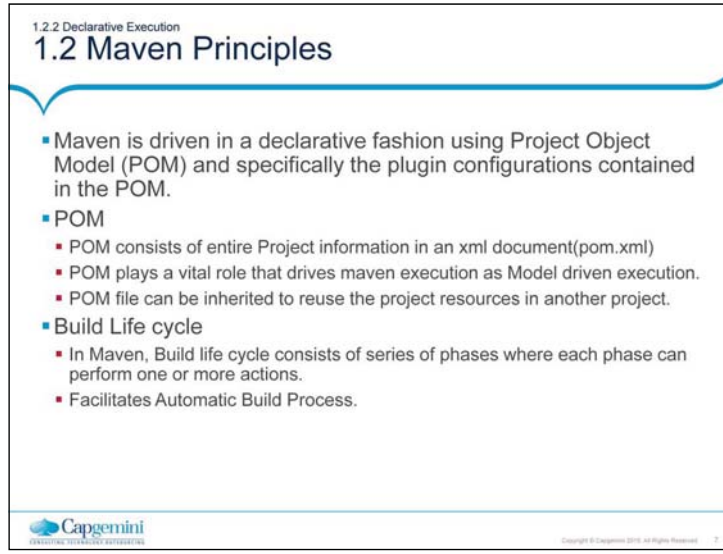
If multiple projects are involved, standard naming convention for directories provide clarity and immediate comprehension.

One Primary Output per Project:

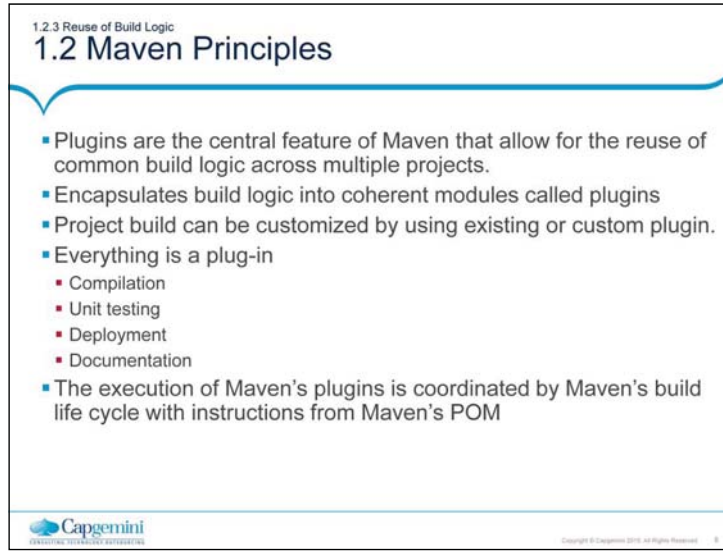
Instead of producing a single JAR file for entire project, Maven would encourage you to have three, separate projects: a project for the client portion of the application, a project for the server portion of the application, and a project for the shared utility code portion.

This separation of concerns (SoC) principle used to achieve the required engineering quality factors such as adaptability, maintainability, extendibility and reusability.

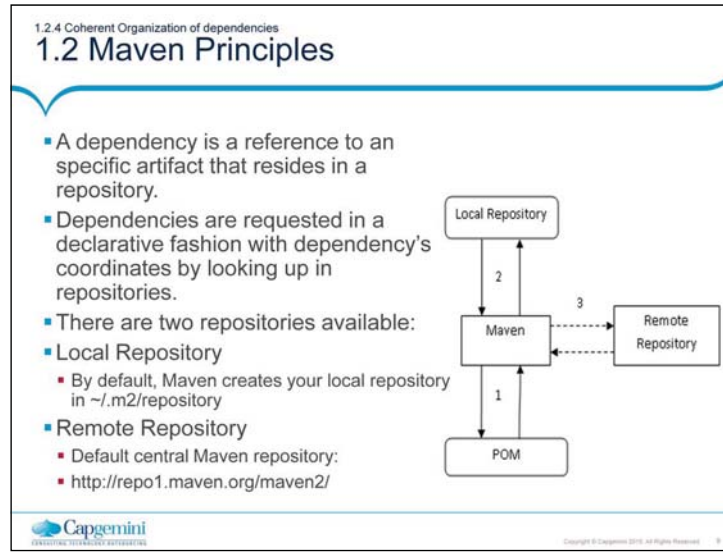
If you follow the convention, Maven will require almost zero effort - just put your source in the correct directory and Maven will take care of the rest.



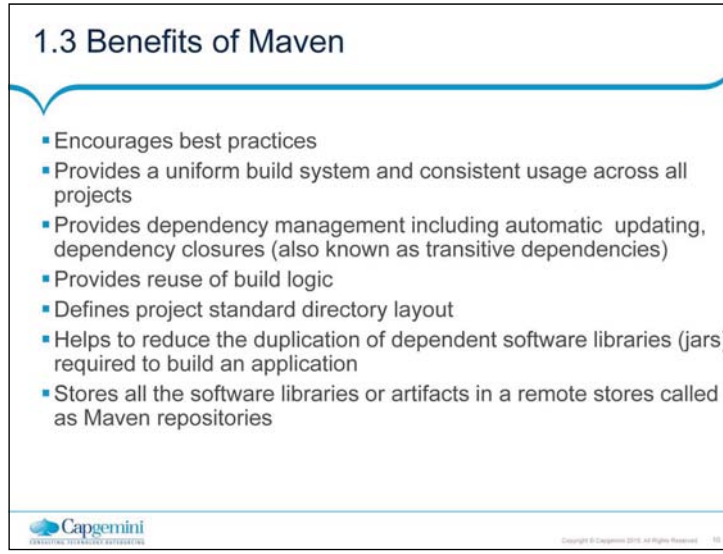
Maven is driven in a declarative fashion using Maven's Project Object Model (POM) and specifically the plug-in configurations contained in the POM.



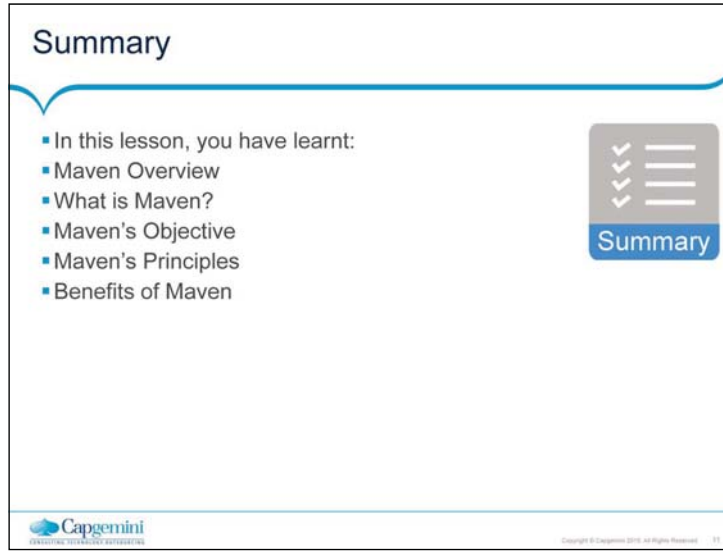
Maven uses SoC principle to promote reuse of build logic by encapsulating build logic into coherent modules called plugins. In Maven, there is a plug-in for compiling source code, a plug-in for running tests, a plug-in for creating JARs, a plug-in for creating Javadocs, and many other functions. Plugins are the key building blocks for everything in Maven.



When a dependency is declared within the context of your project, Maven tries to satisfy that dependency by looking in repositories. Maven uses a local repository to resolve its dependencies. If not found one or more remote repositories are consulted to find a dependency. If found, the dependency is downloaded to the local repository and used from the local repository.



Maven can provide benefits for your build process by employing standard conventions and practices to accelerate your development cycle while at the same time helping you achieve a higher rate of success.


Summary
In this lesson, you have learnt:
Maven Overview
What is Maven?
Maven's Objective
Maven's Principles
Benefits of Maven
Summary
Capgemini

Review Question

- Question1: To promote standardized development environment, Maven uses
  - Option 1: Standard directory Layout
  - Option 2: No Naming Conventions
  - Option 3: Build.xml file creation
  - Option 4: None of the above

- Question 2: Order of repository invocation:
  - Option 1: pom.xml Remote Repository, local repository
  - Option 2: pom.xml, local repository and remote repository
  - Option 3: local repository, settings.xml and remote repository

Capgemini


Maven
Lesson 02
Maven Basics

Lesson Objectives

- POM
- Standard Directory Structure
- Build Life Cycle
- Plug-in
- Dependency Management
- Resolving Dependency Conflicts
- Repositories

Capgemini

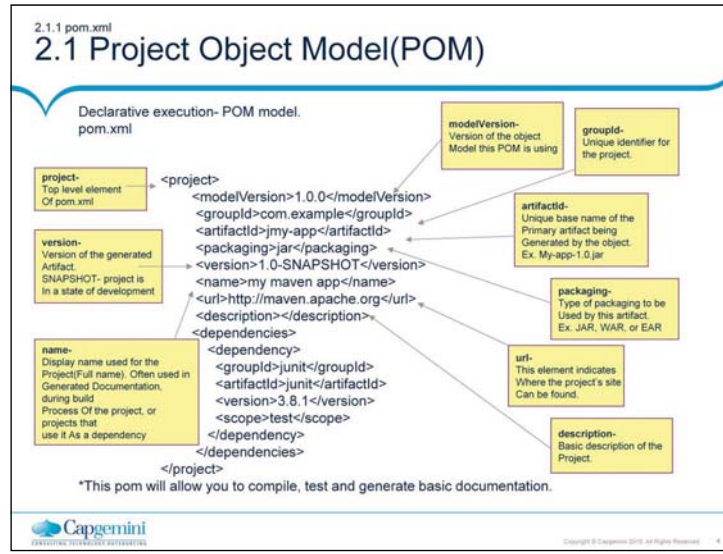
2.1 Project Object Model(POM)

- POM = Project Object Model = pom.xml
- Contains metadata about the Project
  - Location of directories, Developers/Contributors, Issue tracking system, Dependencies, Repositories to use, etc
- Example:

```
<project>
  <modelVersion>1.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <name>my maven app</name>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies/>
  <build/>
[…]
```

Minimal POM

Capgemini



The **<project>** element is the root of the project descriptor.

<modelVersion> : Declares to which version of project descriptor this POM conforms.

<groupId> :A universally unique identifier for a project. It is normal to use a fully-qualified package name to distinguish it from other projects with a similar name (eg. org.apache.maven).

<artifactId > : The identifier for this artifact that is unique within the group given by the group ID. An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.

<packaging> : The type of artifact this project produces, for example jar war ear pom. Plugins can create their own packaging, and therefore their own packaging types, so this list does not contain all possible types.
Default value is: jar.

<Name> : The full name of the project.

<url> : The URL to the project's homepage.

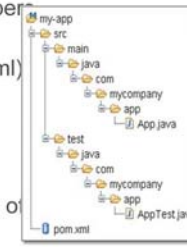
<Version> : The current version of the artifact produced by this project.

<description> : A detailed description of the project, used by Maven whenever it needs to describe the project, such as on the web site.

<dependencies>: This element describes all of the dependencies associated with a project. These dependencies are used to construct a classpath for your project during the build process. They are automatically downloaded from the repositories defined in this project.

2.2 Standard Directory Layout

- Having a common directory layout make the project easier to understand by other developer
- It makes it easier to integrate plugins.
- Project home directory consists of POM(pom.xml) and two subdirectories initially:
  - src : contains all source code and
  - test: contains test source codes
- Target directory generated after the compilation of sources.



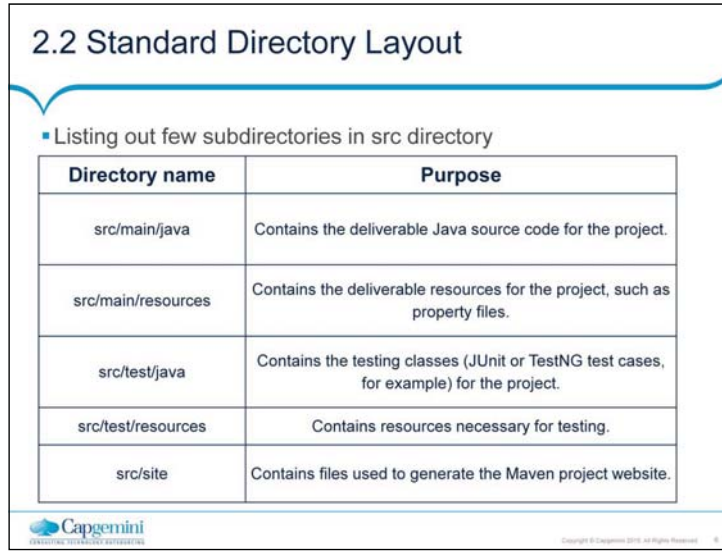
Directory structure before project execution

Capgemini

Advantages:

– A developer familiar with Maven will quickly get familiar with a new project

– No time wasted on re-inventing directory structures and conventions



The src directory has a number of subdirectories, each of which has a clearly defined purpose. Listing out few subdirectories in src directory:

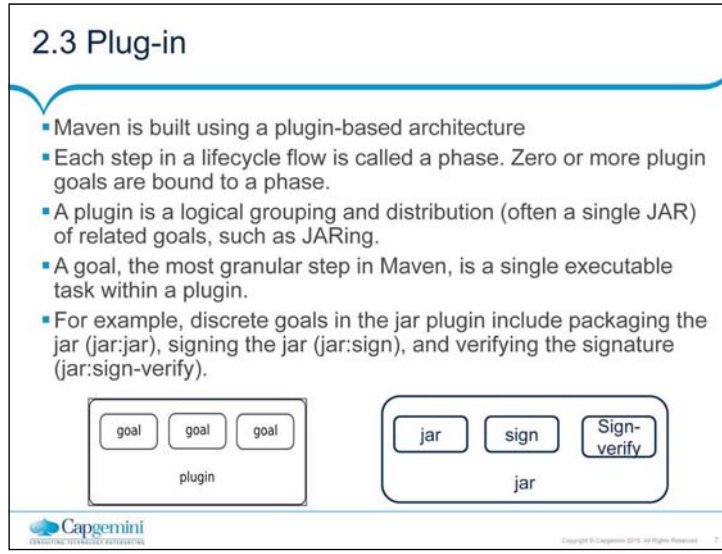
src/main/java - Contains the deliverable Java source code for the project.

src/main/resources - Contains the deliverable resources for the project, such as property files.

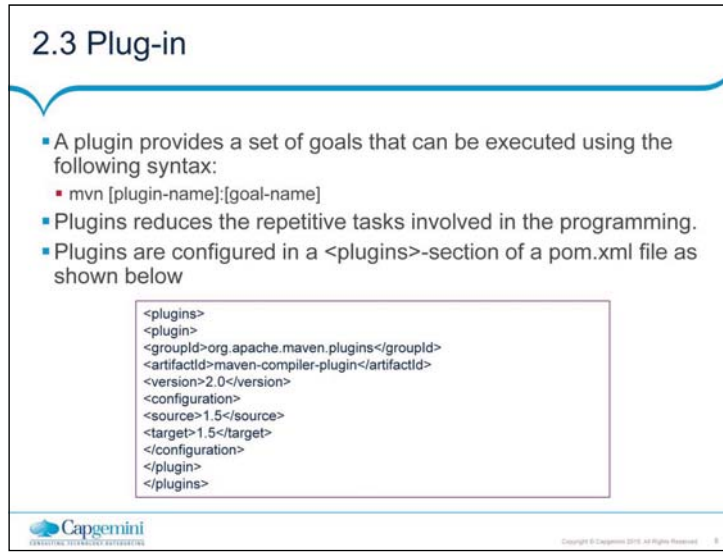
src/test/java - Contains the testing classes (JUnit or TestNG test cases, for example) for the project.

src/test/resources - Contains resources necessary for testing.

src/site - Contains files used to generate the Maven project website.



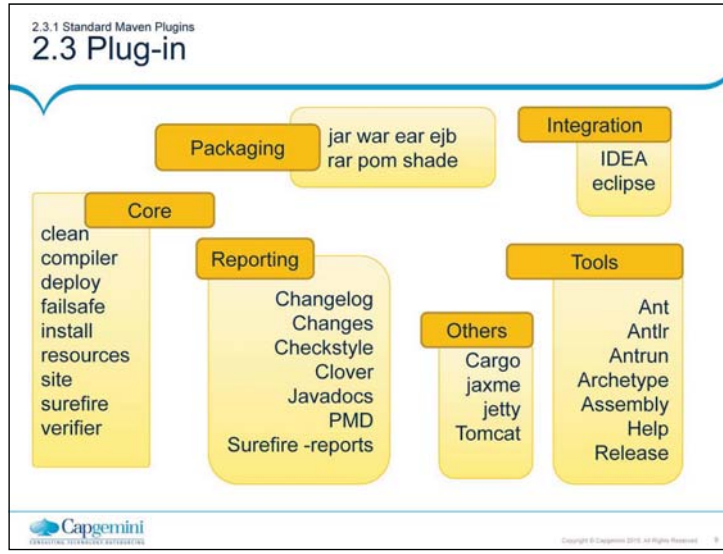
Plugin	Description
Core plugins	Plugins corresponding to default core phases (ie. clean, compile). They may have multiple goals as well.
clean	Clean up after the build.
compiler	Compiles Java sources.
deploy	Deploy the built artifact to the remote repository.
failsafe	Run the JUnit integration tests in an isolated classloader.
install	Install the built artifact into the local repository.
resources	Copy the resources to the output directory for including in the JAR.
site for Maven 2	Generate a site for the current project.
surefire	Run the JUnit unit tests in an isolated classloader.
verifier	Useful for integration tests - verifies the existence of certain conditions.



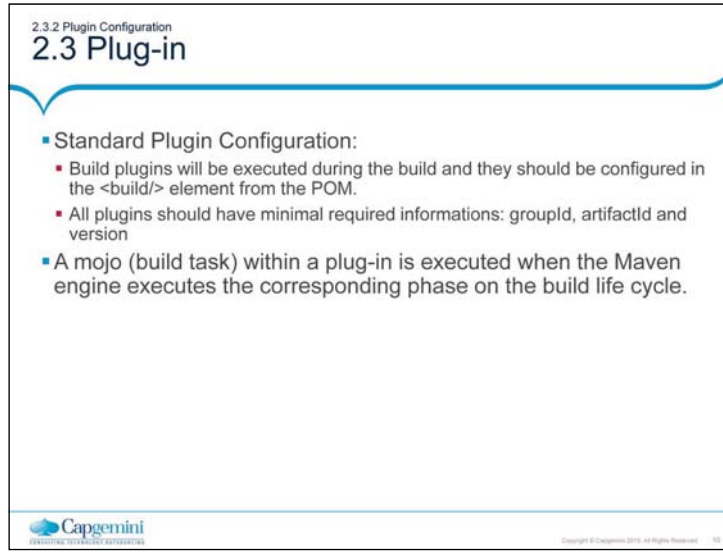
For example, configuring the Java compiler to allow JDK 5.0 sources in a project. This is as simple as adding this following to your POM:

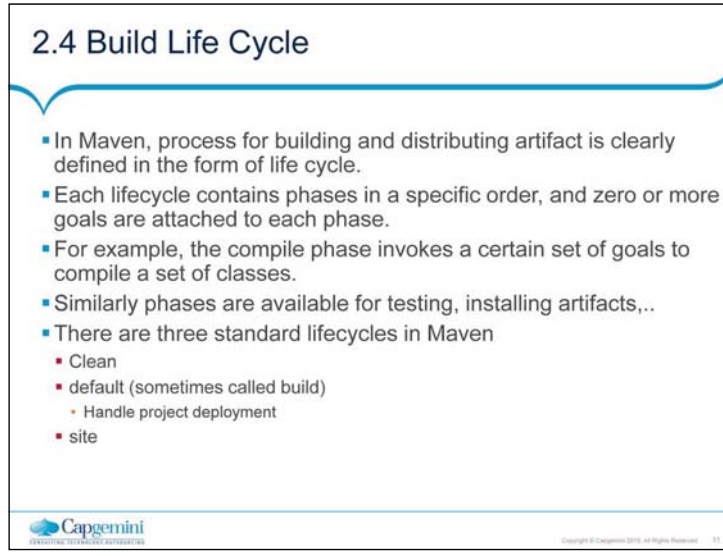
```
<project>
...
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0</version>
<configuration>
<source>1.5</source>
<target>1.5</target>
</configuration>
</plugin>
</plugins>
</build>
...
</project>
```

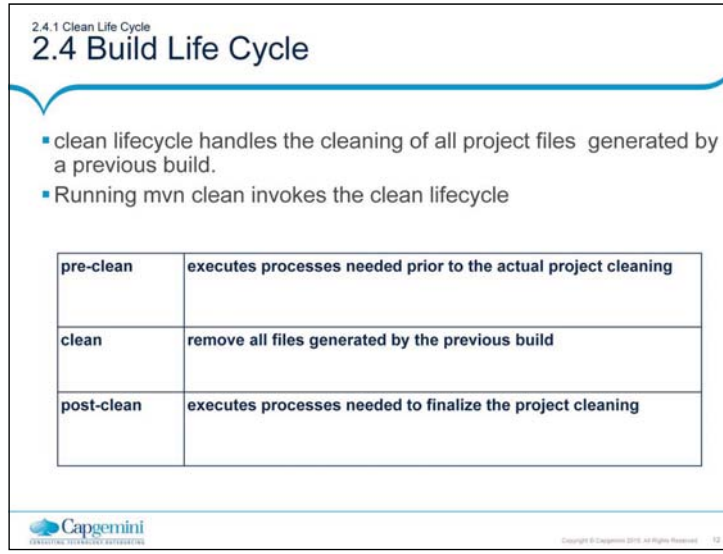
This plugin will be downloaded and installed automatically, if it is not present on your local system.

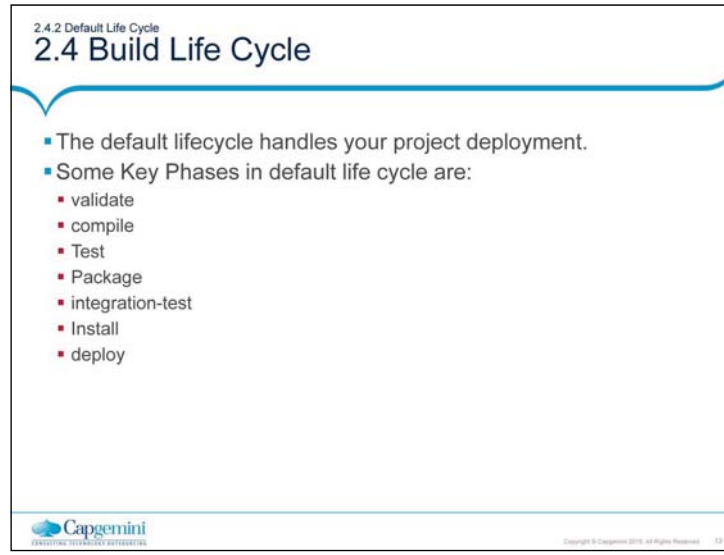


To see the most up-to-date list browse the Maven repository at http://repo1.maven.org/maven2/, specifically the org/apache/maven/plugins subfolder. *(Plugins are organized according to a directory structure that resembles the standard Java package naming convention)*


2.3.2 Plugin Configuration
2.3 Plug-in
Standard Plugin Configuration:
Build plugins will be executed during the build and they should be configured in the <build/> element from the POM.
All plugins should have minimal required informations: groupId, artifactId and version
A mojo (build task) within a plug-in is executed when the Maven engine executes the corresponding phase on the build life cycle.
Capgemini


2.4 Build Life Cycle
In Maven, process for building and distributing artifact is clearly defined in the form of life cycle.
Each lifecycle contains phases in a specific order, and zero or more goals are attached to each phase.
For example, the compile phase invokes a certain set of goals to compile a set of classes.
Similarly phases are available for testing, installing artifacts,..
There are three standard lifecycles in Maven
Clean
default (sometimes called build)
Handle project deployment
site
Capgemini


2.4.1 Clean Life Cycle
2.4 Build Life Cycle
clean lifecycle handles the cleaning of all project files generated by a previous build.
Running mvn clean invokes the clean lifecycle
pre-clean
executes processes needed prior to the actual project cleaning
clean
remove all files generated by the previous build
post-clean
executes processes needed to finalize the project cleaning
Capgemini



validate - validate the project is correct and all necessary information is available

compile - compile the source code of the project

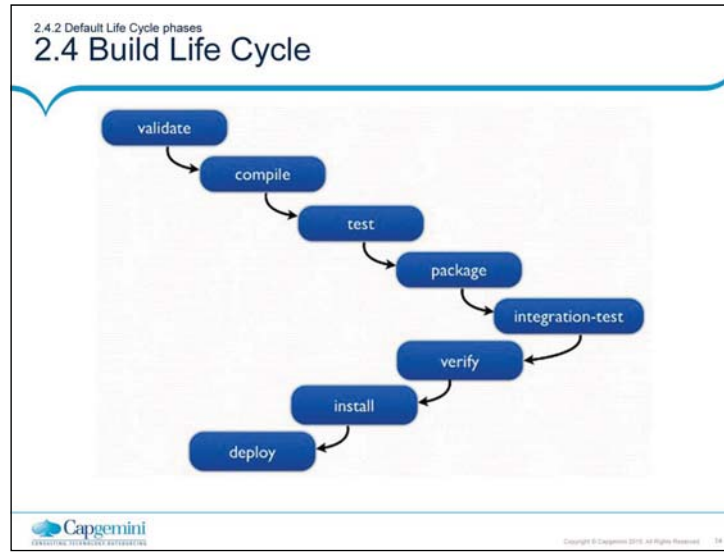
test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

package - take the compiled code and package it in its distributable format, such as a JAR

integration-test - process and deploy the package if necessary into an environment where integration tests can be run

install - install the package into the local repository, for use as a dependency in other projects locally

deploy - done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects



The default lifecycle of Maven first validates the project, like checking the consistency of pom.xml file, and then it tries to compile the code. If the compile is successful, it then tries to run the test against the compiled code, package the project in the specified format, run the integration tests against that package, verify the package by checking for the validity, install the verified package to the local repository, and finally deploy the package in the specified environment.

To execute phases/goals we can follow the notation below in the command line:

mvn *phase* mvn *phase:goal*

We can also invoke multiple phases/goals within one command line, like:

mvn *phase phase:goal* mvn *phase:goal phase:goal*

When we invoke the *mvn integration-test* command, Maven executes all the phases that are registered before that phase. So the validate, compile, and package phases get executed before the integration-test phase.

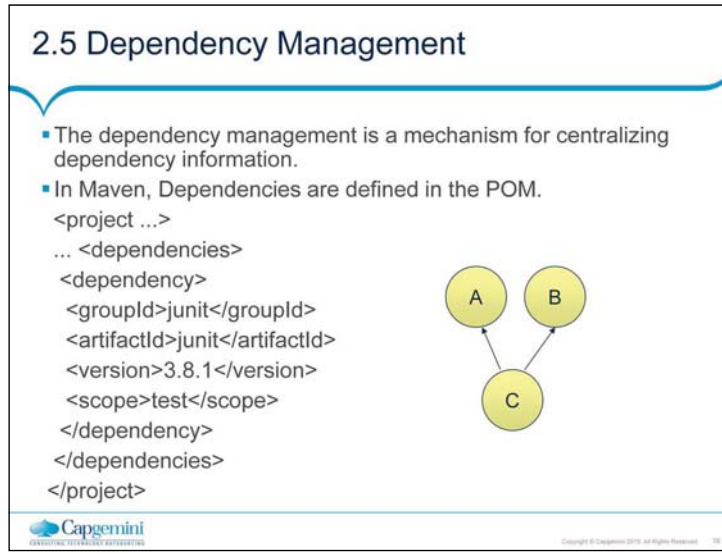
2.4.3 Site Life Cycle

2.4 Build Life Cycle

- Site lifecycle handles the creation of your project's site documentation.
- You can generate a site from a Maven project by running the following command:
-

pre-site	executes processes needed prior to the actual project site generation
site	generates the project's site documentation
post-site	executes processes needed to finalize the site generation, and to prepare for site deployment
site-deploy	deploys the generated site documentation to the specified web server

Capgemini



• Dependency: a third-party or project-local software library (JAR or WAR file) of one project will be reused in another projects.

• In Maven, Dependencies are defined in the POM.

Dependency scope is used to limit the transitivity of a dependency, and also to affect the classpath used for various build tasks.

There are 6 scopes available:

• **Compile** - This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.

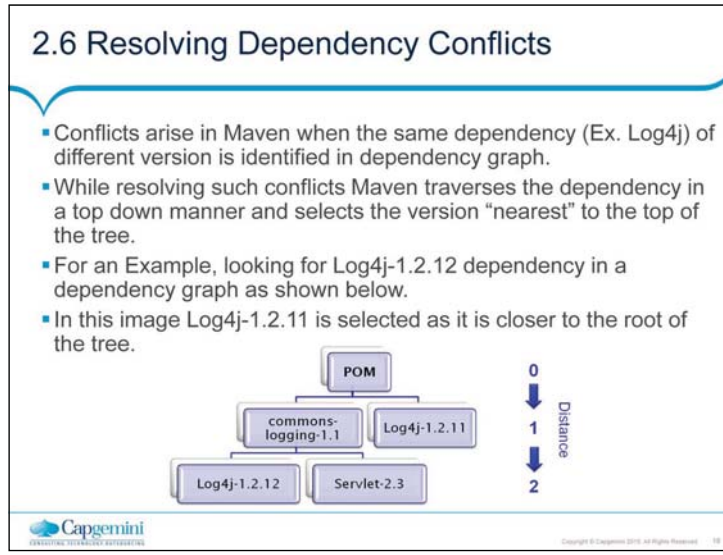
• **Provided** - This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime.

.For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes. This scope is only available on the compilation and test classpath, and is not transitive.

- **Runtime** - This scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.
- **Test** - This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.
- **System** - This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
- **import** - This scope is only used on a dependency of type pom in the <dependencyManagement> section. It indicates that the specified POM should be replaced with the dependencies in that POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

Transitive Dependencies:

- Resolves Dependencies of dependencies are called transitive dependencies, and they are made possible by the fact that the Maven repository stores more than just bytecode; it stores metadata about artifacts.
- Transitive dependencies allows you to avoid needing to discover and specify the libraries that your own dependencies require, and including them automatically.
- For example, if you define a dependency to commons-logging and commons-logging itself defines a dependency to log4j, the commons-logging.jar and log4j.jar will be added to your build process. More formally spoken, transitiveness means that if A->B and B->C then A->C



As the tree grows, it is inevitable that two or more artifacts will require different versions of a particular dependency.

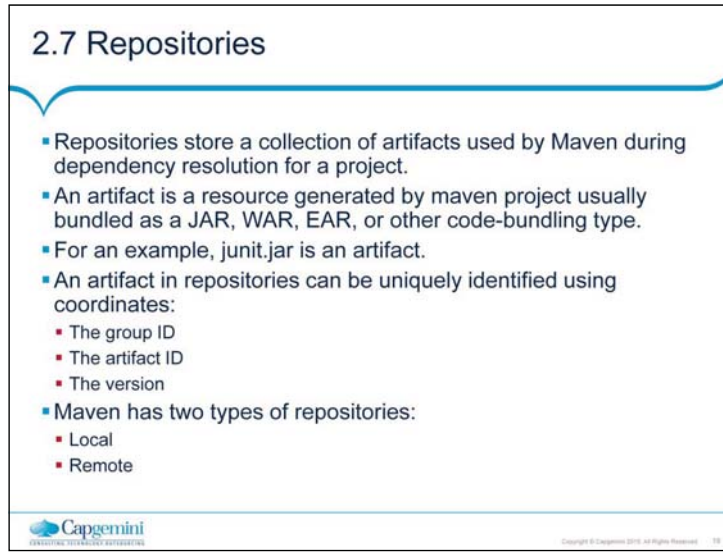
For an Example: The sample project defines two direct dependencies: One to commons-logging-1.1 and one to log4j-1.2.11. Now, because Maven transitively loads all dependencies that are defined for commons-logging-1.1, a second version of log4j (V1.2.12) pops up in the dependency tree.

Which dependency will be used by the build process?

Answer : log4j-1.2.11 is been loaded as it is the nearest to the top of the tree.

The dependency version to be used by the build process can be recommended as follows:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
     <version>1.2.12</version>
</dependency>
```



The poor approach:

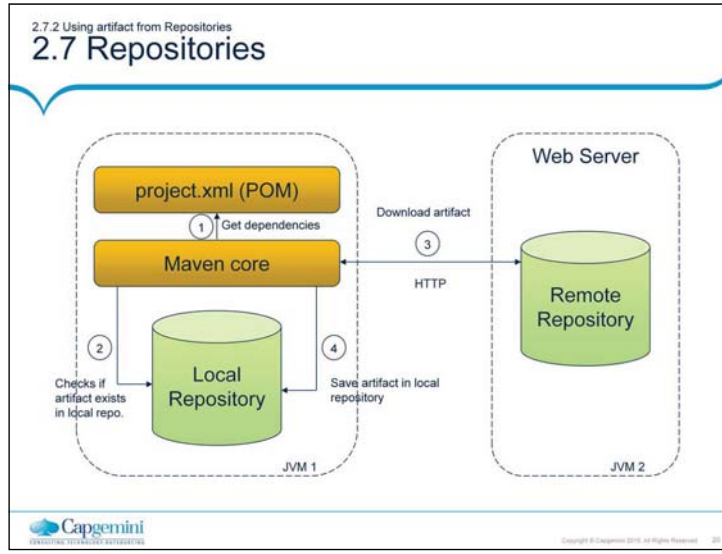
Replicating all dependencies for every project (put in /lib folder within the project)

– Dependencies are replicated and use more storage

– Checking out a project will be slow

– Difficult to keep track of versions

• The preferred solution: Use a repository


2.7.2 Using artifact from Repositories
2.7 Repositories
project.xml (POM)
1
Get dependencies
Maven core
2
Checks if artifact exists in local repo.
Local Repository
4
Save artifact in local repository
JVM 1
Download artifact
3
HTTP
Web Server
Remote Repository
JVM 2
Capgemini


Summary
POM
Standard Directory Structure
Build Life Cycle
Plug-in
Dependency Management
Resolving Dependency Conflicts
Repositories
Summary
Capgemini

Review Question

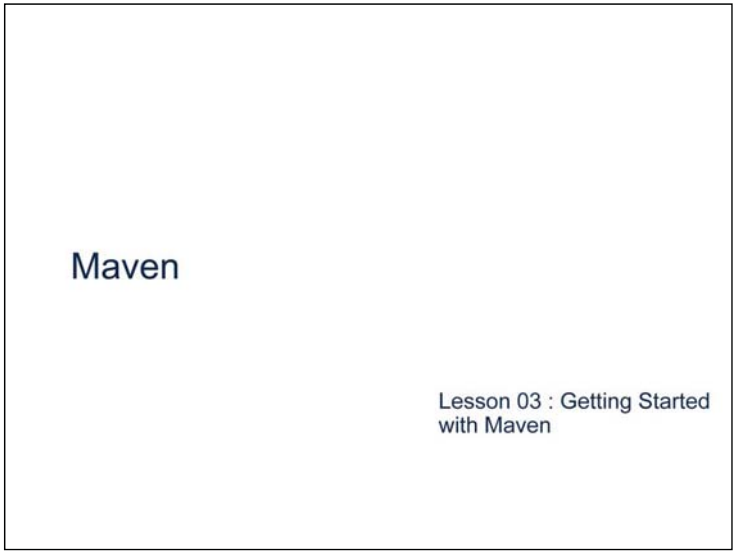
- Question1: Clean Life cycle phases are _____, _____, _____
- Question 2: Which command generates default Site for a Maven Project?
- Question 3: Plugin used for running JUnit tests _____.
- Question 4: For identifying artifact in repositories, coordinates required are _____, _____ and _____.

Capgemini

Review Question

- Question 5: Invoking the deploy phase deploys the application in which environment?
  - Option 1: Local repository
  - Option 2: Release environment
  - Option 3: External Repository

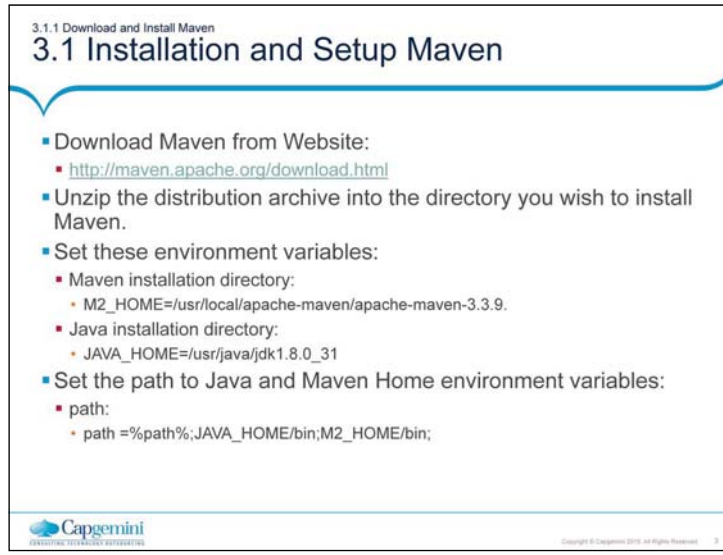
Capgemini

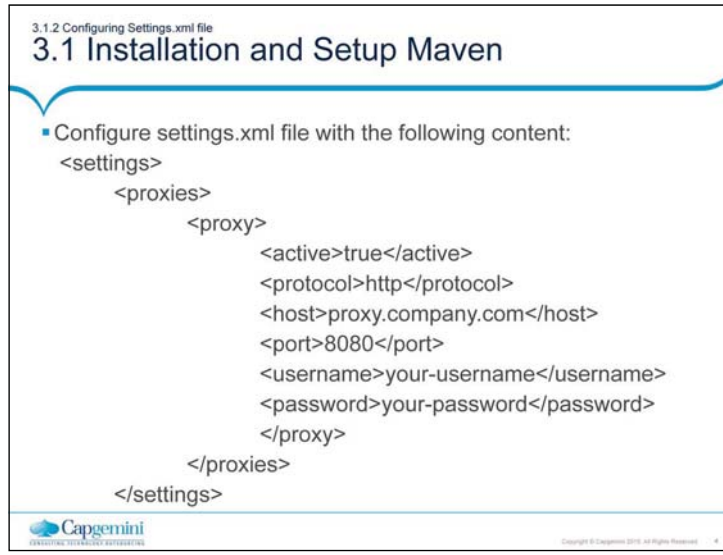

Maven
Lesson 03 : Getting Started with Maven

Lesson Objectives

- In this lesson, you will learn about:
  - Installation and setup Maven
    - Download and Install Maven
    - Configure settings.xml file
  - Creating your first project using Maven commands
    - Creating Project Template
    - Compiling project
    - Testing the sources

Capgemini

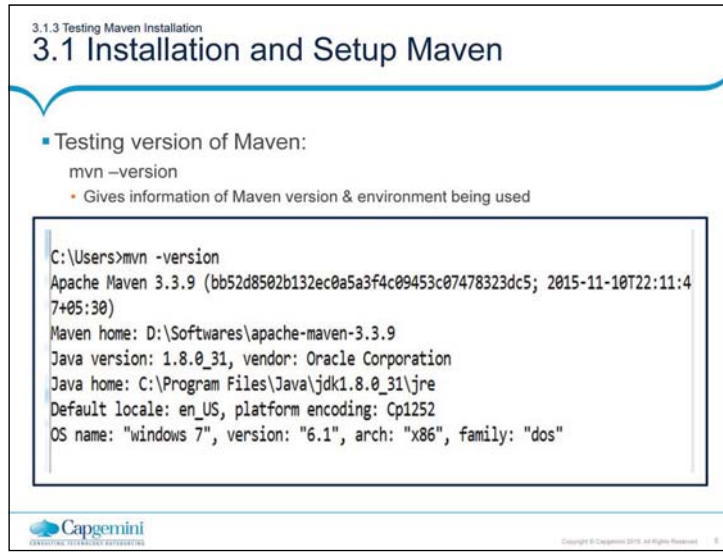

3.1.1 Download and Install Maven
3.1 Installation and Setup Maven
Download Maven from Website:
http://maven.apache.org/download.html
Unzip the distribution archive into the directory you wish to install Maven.
Set these environment variables:
Maven installation directory:
M2_HOME=/usr/local/apache-maven/apache-maven-3.3.9.
Java installation directory:
JAVA_HOME=/usr/java/jdk1.8.0_31
Set the path to Java and Maven Home environment variables:
path:
path =%path%;JAVA_HOME/bin;M2_HOME/bin;
Capgemini

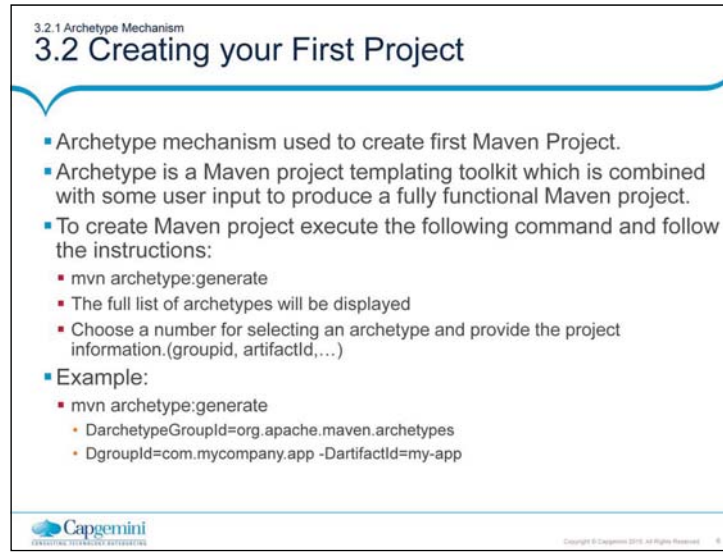


If you are behind a firewall, then you will have to set up Maven to understand and download required plugins from repository. To do this, create a <your-home directory>/.m2/settings.xml file with the following content:

For Example:

```
<settings>
<proxies>
    <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>username</username>
    <password>password</password>
    <host>192.168.104.40.org.com</host>
    <port>8080</port>
    <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
   </proxy>
  </proxies>
</settings>
```


3.1.3 Testing Maven Installation
3.1 Installation and Setup Maven
Testing version of Maven:
mvn –version
Gives information of Maven version & environment being used
C:\Users>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T22:11:47+05:30)
Maven home: D:\Softwares\apache-maven-3.3.9
Java version: 1.8.0_31, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_31\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "x86", family: "dos"
Capgemini

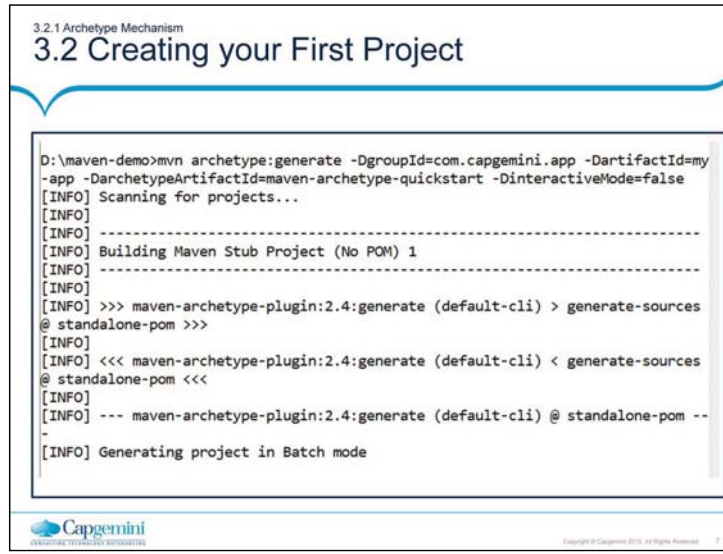


"An archetype is defined as an original pattern or model from which all other things of the same kind are made. "
You may want to standardize J2EE development within your organization so you may want to provide archetypes for EJBs, or WARs, or for your web services. Once these archetypes are created and deployed in your organization's repository they are available for use by all developers within your organization.
Archetypes are packaged up in a JAR and they consist of the archetype metadata which describes the contents of archetype and a set of Velocity templates which make up the prototype project.

To create Maven project execute the following command and follow the instructions:

```
mvn archetype:generate
```

If this is the first time you have run the archetype plugin, you will notice that Maven spends some time downloading the archetype functionality as well as the other Java libraries that it uses. These files are stored in your local repository for re-use in
the future.
Once the downloads finish from the earlier archetype command, you will be prompted (with a long list!) to select an archetype to generate the project from.



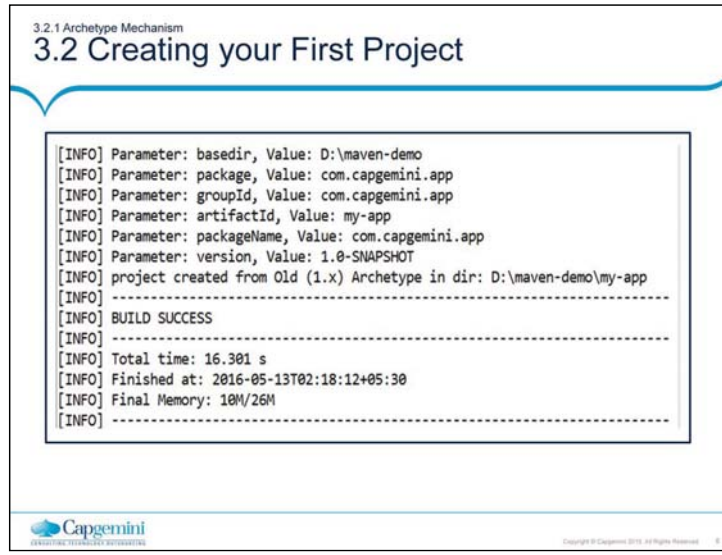
Among the many choices, the default is the quick start archetype that is used in Maven's getting started guide. This is a good choice for a minimal Maven project or an archetype from which new modules can be built (particularly for JAR projects).

Each Maven project needs some coordinates to allow it to be identified by other Maven projects. These are called the group ID, artifact ID, and version (sometimes referred to as the GAV).

Group ID: An identifier for a collection of related modules. This is usually a hierarchy that starts with the organization that produced the modules, and then possibly moves into the increasingly more specialized project or sub-projects that the artifacts are a part of. This can also be thought of as a namespace, and is structured much like the Java package system.

Artifact ID: The Artifact ID is a unique identifier for a given module within a group.

Version: The version is used to identify the release or build number of the project.



In Maven, there are two types of versions: releases and snapshots (those that end in -SNAPSHOT). A snapshot should always be used by the projects during development as it has a special meaning to Maven to indicate that development is still occurring and that the project may change. A release is assumed never to change, so release versions (such as 1.1, 2.0, or 3.0-beta-5) should only be used for a single state of the project when it is released, and then updated to the next snapshot.

The final prompt for a **package** is the Java package that will be used for source code. It is not the packaging type that to take by the project as that is provided by the archetype itself.

After entering the values above, prompt will be displayed to confirm that the values are as you intended, and then the project will be created. If the below highly desirable banner displayed, then congratulations—Maven project created successfully!

[INFO] --

[INFO] BUILD SUCCESSFUL

[INFO] --

3.2.2 Viewing Project strcture

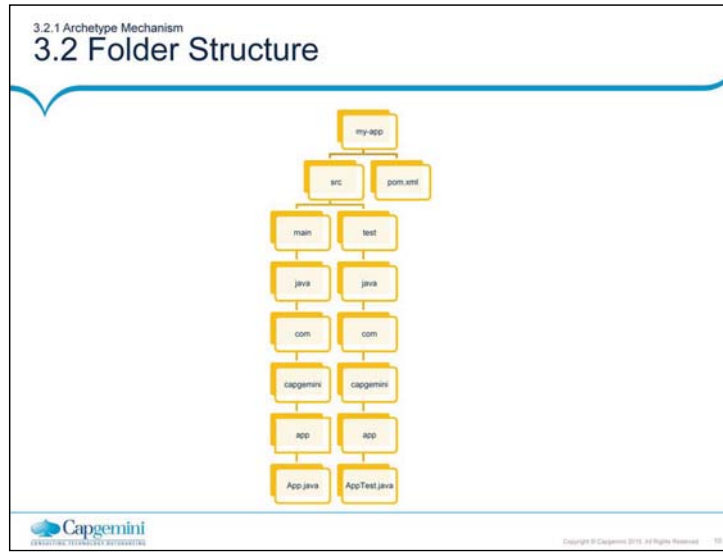
3.2 Creating your First Project

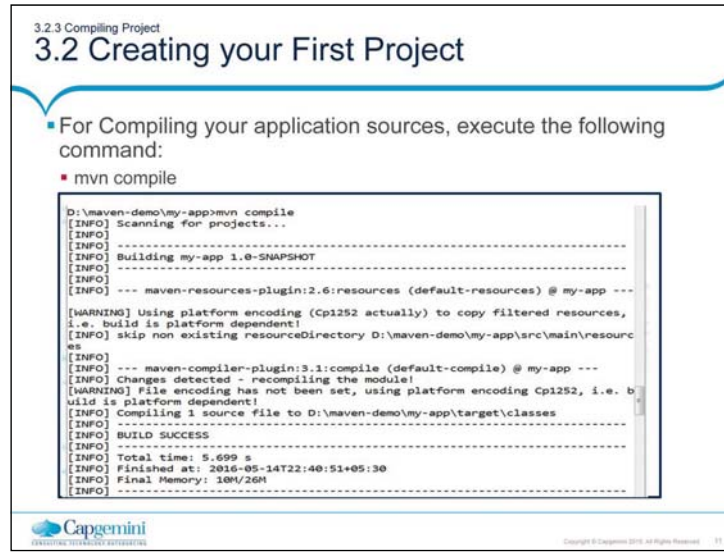
- To view the project structure which is created based on archetype, execute the following command:
  - tree application-name

```
D:\maven-demo>tree my-app
Folder PATH listing
Volume serial number is 00000002 8A01:5839
D:\MAVEN-DEMO\MY-APP
└─src
    ├─main
    │  └─java
    │      └─com
    │          └─capgemini
    │              └─app
    └─test
        └─java
            └─com
                └─capgemini
                    └─app
```

- The src/main/java directory contains the basic java application, while pom.xml specifies the information Maven needs to know.
- This directory structure has followed the Maven convention, which requires a minimal amount of configuration to construct a functional build.

Capgemini


3.2.1 Archetype Mechanism
3.2 Folder Structure
my-app
src
pom.xml
main
test
java
java
com
com
capgemini
capgemini
app
app
App.java
AppTest.java
Capgemini

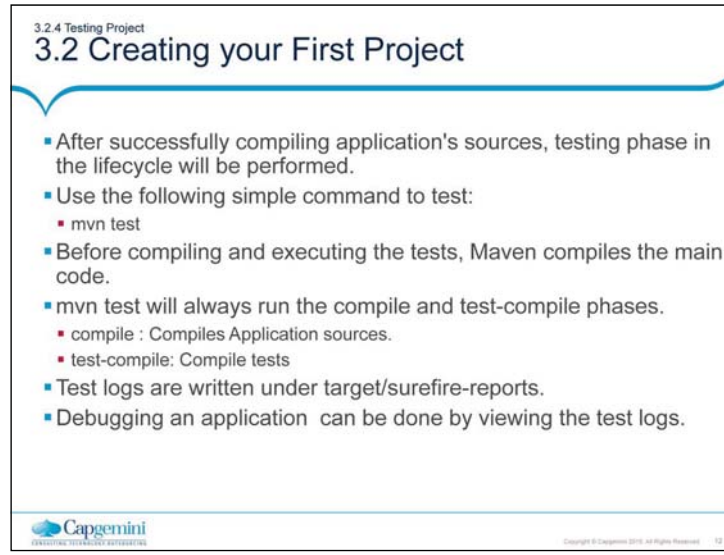


For Compiling your application sources, execute the following command:

- **mvn compile**

After compilation, the compiled classes generated in the default target directory i.e) target/classes.

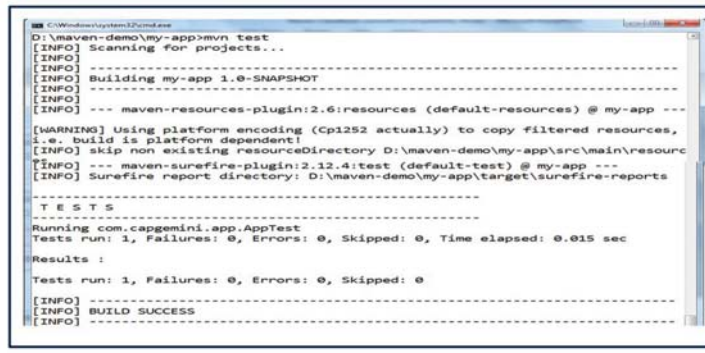
The compiler plugin was used to compile the application sources, that maps to the Maven's default build life cycle.



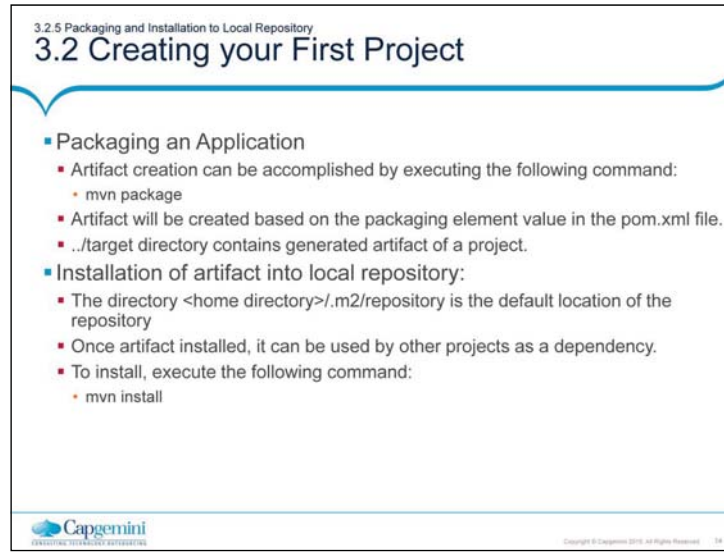
- After successfully compiling application's sources, testing phase in the lifecycle will be performed.
- Use the following simple command to test:
  - mvn test
- Maven downloads more dependencies this time. These are the dependencies and plugins necessary for executing the tests.
- Before compiling and executing the tests, Maven compiles the main code.
- mvn test will always run the compile and test-compile phases.
  - compile - Compiles the application source codes.
  - test-compile - compiles the test codes.

3.2.1 Archetype Mechanism

3.2 Creating your First Project



Capgemini

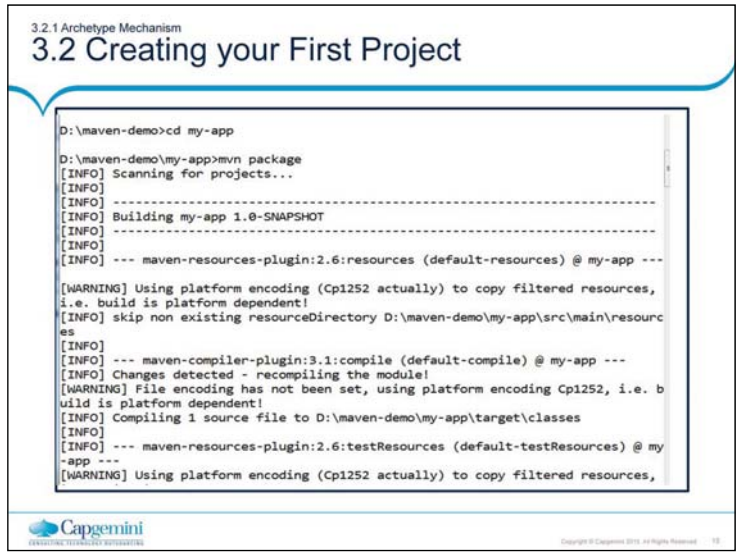


Packaging an Application:

- Artifact creation can be accomplished by executing the following command:
    - mvn package
- Artifact will be created based on the packaging element value in the pom.xml file.
- ../target directory contains generated artifact of a project.

Installation of artifact into local repository:

- The directory <home directory>/.m2/repository is the default location of the repository
- Once artifact installed, it can be used by other projects as a dependency.
- To install, execute the following command:
    - mvn install
- This command can be used instead most of the time, as it will process all of the previous phases of the build life cycle (generate sources, compile, compile tests, run tests, etc.).


3.2.1 Archetype Mechanism
3.2 Creating your First Project
D:\maven-demo>cd my-app
D:\maven-demo\my-app>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] --
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] --
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ my-app ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\maven-demo\my-app\src\main\resourc
es
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ my-app ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. b
uild is platform dependent!
[INFO] Compiling 1 source file to D:\maven-demo\my-app\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ my
-app ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
Capgemini

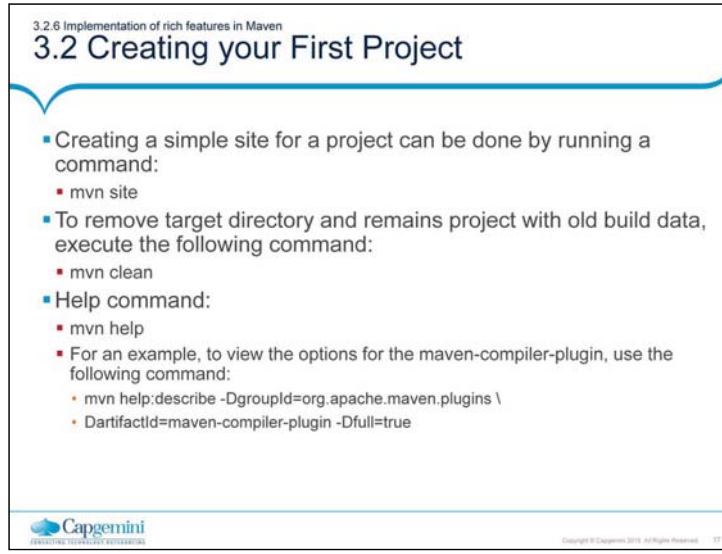
3.2.1 Archetype Mechanism

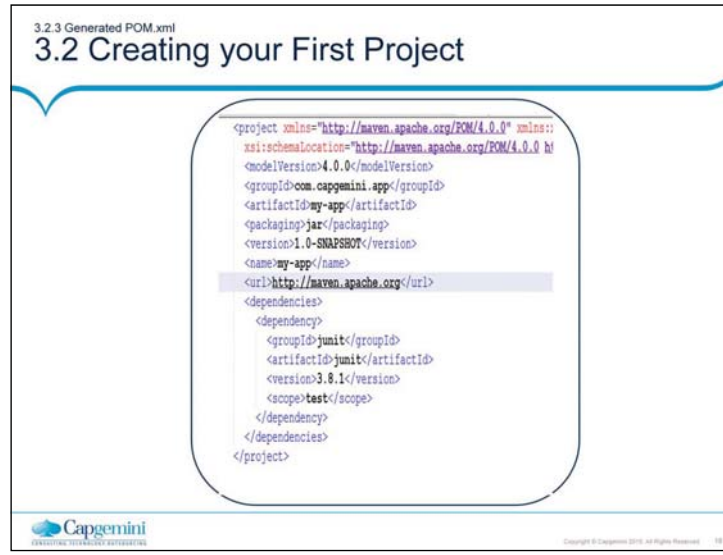
3.2 Creating your First Project

```
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ my-app ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.760 s
[INFO] Finished at: 2016-05-13T02:20:58+05:30
[INFO] Final Memory: 7M/19M
[INFO] ------------------------------------------------------------------------

D:\maven-demo\my-app>java -cp target/my-app-1.0-SNAPSHOT.jar com.capgemini.app.A
pp
Hello World!
```

Capgemini


3.2.6 Implementation of rich features in Maven
3.2 Creating your First Project
Creating a simple site for a project can be done by running a command:
mvn site
To remove target directory and remains project with old build data, execute the following command:
mvn clean
Help command:
mvn help
For an example, to view the options for the maven-compiler-plugin, use the following command:
mvn help:describe -DgroupId=org.apache.maven.plugins \
DartifactId=maven-compiler-plugin -Dfull=true
Capgemini



The archetype has substituted the coordinates that it prompted for earlier in the **pom.xml** file . It has also added the jar packaging type to indicate that the build structure for the project is a Java application. A default name has been added, which you would typically change to something which describes the project better.

Maven's declarative project model defines a number of well-known properties that the whole build can reuse for a number of different purposes. For Example, even javadoc can be produced if requested.

The POM contains important pieces of information about the project, which include:

- The Maven coordinate of the project for reuse by other Maven projects
- The project name, description and license
- Project resource information such as the location of source control, issue tracking, and continuous integration
- The developers, contributors and organizations participating in the project

The POM will also include information about how the project should be built, such as:

- The source directory layout
- Dependencies on other projects
- Build requirements (by means of Maven plugins) and configuration

3.2.7 Demo on Project creation

3.2 Creating your First Project

- Creating and executing the “Hello World!” program

Capgemini

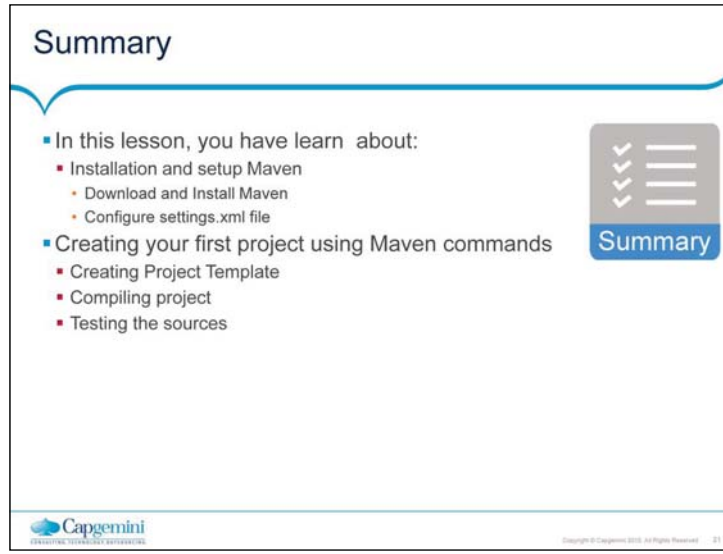
 19

3.2.8 Demo on Project creation

Lab: 1

- 1.1 : Getting Started With Maven
- 1.2 : Configuring Maven Settings
- 1.3 : Creating first standalone Maven application using archetype

Capgemini


Summary
In this lesson, you have learn about:
Installation and setup Maven
Download and Install Maven
Configure settings.xml file
Creating your first project using Maven commands
Creating Project Template
Compiling project
Testing the sources
Summary
Capgemini

Review Question

- Question1: _____ command execution will remove target directory and remains project with old build data.
  - mvn compile
  - mvn clean
  - mvn stage
  - mvn verify

- Question2: Maven stores all the software libraries or artifacts in stores called a repository.
  - True
  - False

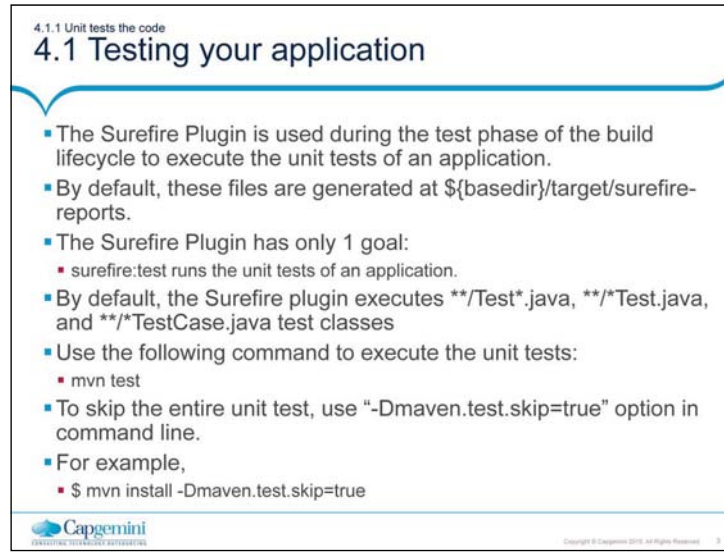
Capgemini


Maven
Creating Applications with Maven

Lesson Objectives

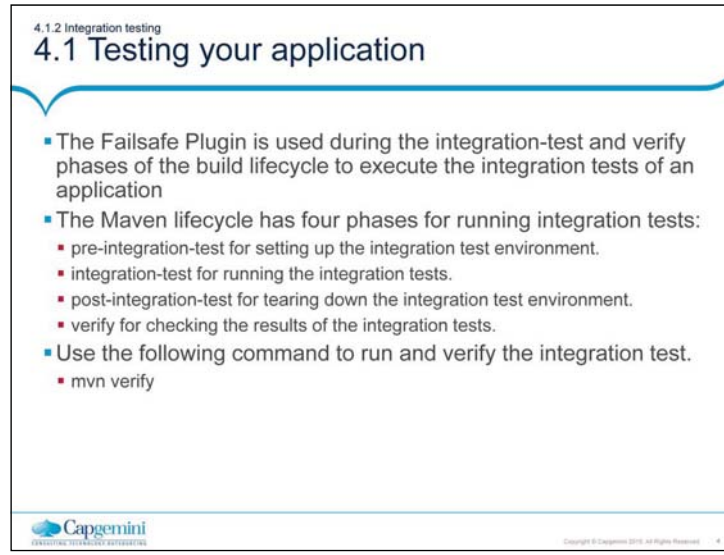
- In this lesson, you will learn about:
- Overview
- Testing your application
- Generating Java Projects using Maven in Eclipse Luna
  - Setting Maven in Eclipse Environment
  - Adding/Updating Dependencies
  - Importing Project
  - Creating Maven Module
- Creating Web application using Maven
  - Creating Maven web Module

Capgemini



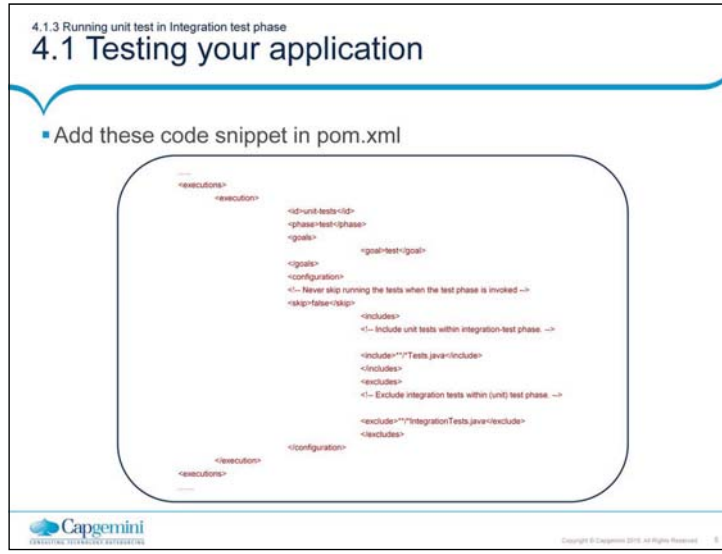
It generates reports in 2 different file formats:

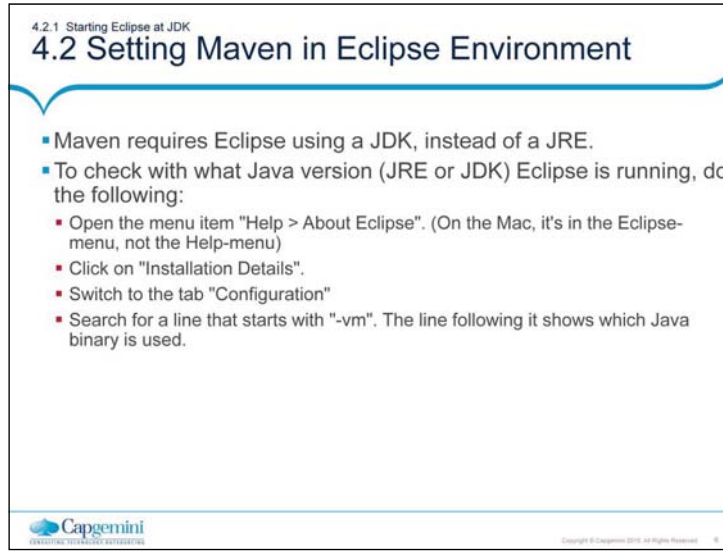
- Plain text files (*.txt)
- XML files (*.xml)

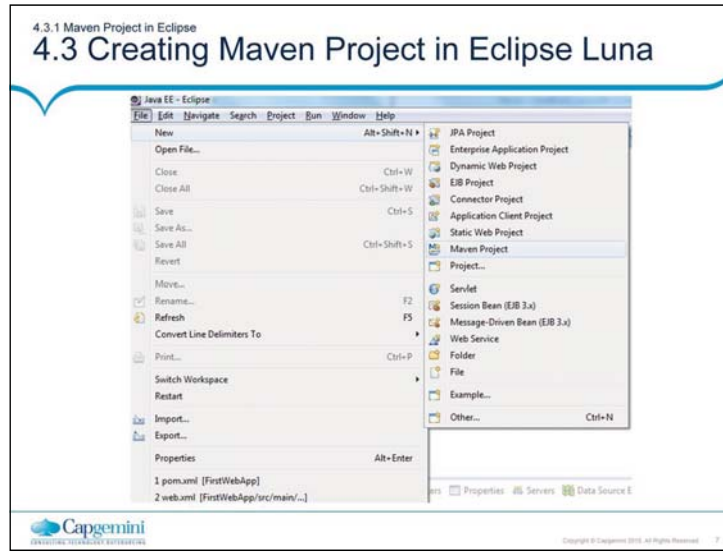


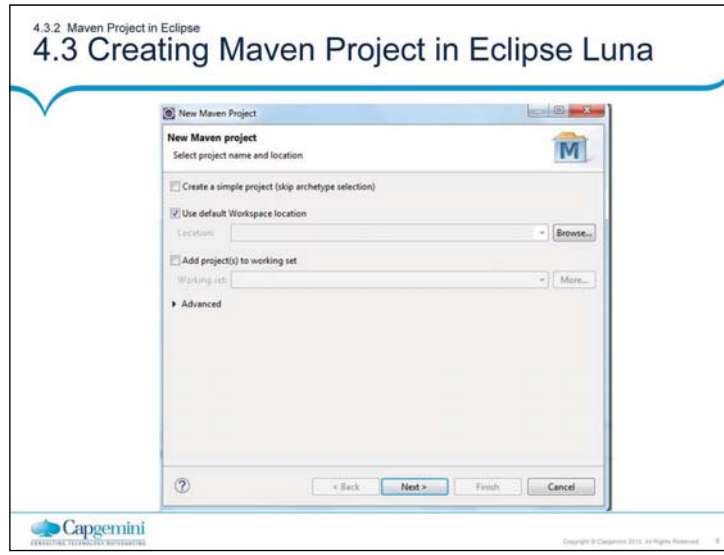
The Failsafe Plugin has only 2 goals:
failsafe:integration-test runs the integration tests of an application.
failsafe:verify verifies that the integration tests of an application passed.

The Failsafe plugin will look for **/IT*.java, **/*IT.java, and **/*ITCase.java.


4.1.3 Running unit test in Integration test phase
4.1 Testing your application
Add these code snippet in pom.xml
<executions>
<execution>
<id>unit-tests</id>
<phase>test</phase>
<goals>
<goal>test</goal>
</goals>
<configuration>
<!-- Never skip running the tests when the test phase is invoked -->
<skip>false</skip>
<includes>
<!-- Include unit tests within integration-test phase. -->
<include>**/*Tests.java</include>
</includes>
<excludes>
<!-- Exclude integration tests within (unit) test phase. -->
<exclude>**/*IntegrationTests.java</exclude>
</excludes>
</configuration>
</execution>
</executions>
Capgemini

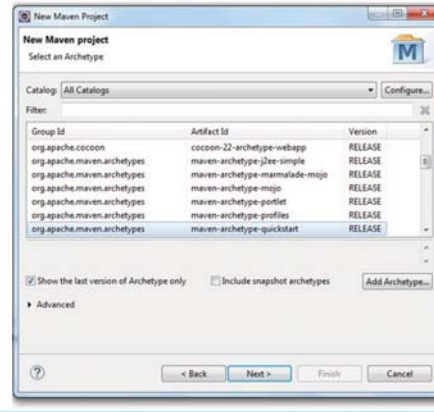

4.2.1 Starting Eclipse at JDK
4.2 Setting Maven in Eclipse Environment
Maven requires Eclipse using a JDK, instead of a JRE.
To check with what Java version (JRE or JDK) Eclipse is running, do the following:
Open the menu item "Help > About Eclipse". (On the Mac, it's in the Eclipse-menu, not the Help-menu)
Click on "Installation Details".
Switch to the tab "Configuration"
Search for a line that starts with "-vm". The line following it shows which Java binary is used.
Capgemini


4.3.1 Maven Project in Eclipse
4.3 Creating Maven Project in Eclipse Luna
Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 pom.xml [FirstWebApp]
2 web.xml [FirstWebApp/src/main/...]
JPA Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
Application Client Project
Static Web Project
Maven Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Web Service
Folder
File
Example...
Other... Ctrl+N
Capgemini


4.3.2 Maven Project in Eclipse
4.3 Creating Maven Project in Eclipse Luna
New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel
Capgemini

4.3.3 Maven Project in Eclipse

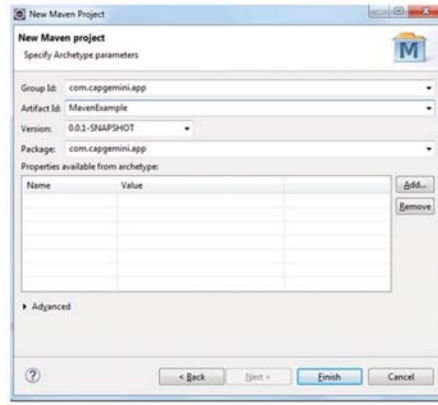
4.3 Creating Maven Project in Eclipse Luna



Capgemini

4.3.4 Maven Project in Eclipse

4.3 Creating Maven Project in Eclipse Luna



Capgemini

4.3.5 Maven Project in Eclipse

4.3 Creating Maven Project in Eclipse Luna

- Update the jUnit version to 4.11 in pom.xml

*MavenExample/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://w
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.a
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.capgemini.app</groupId>
  <artifactId>MavenExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>MavenExample</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

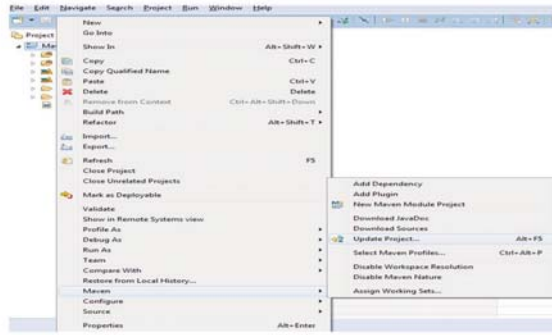
Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Capgemini

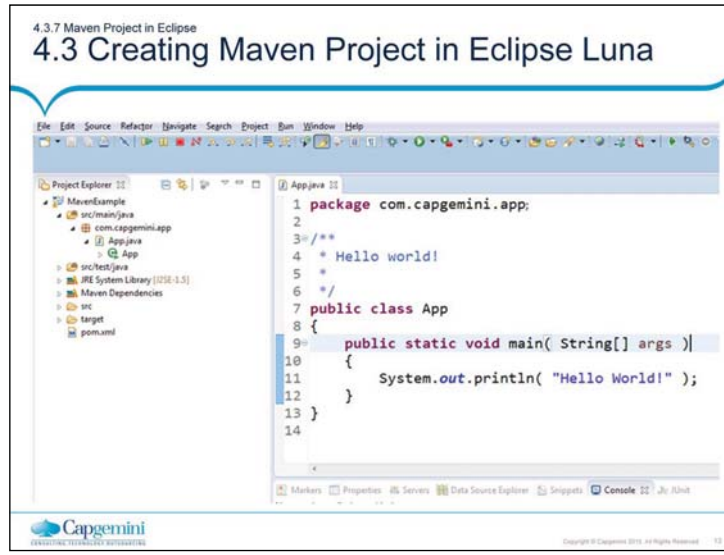
4.3.6 Maven Project in Eclipse

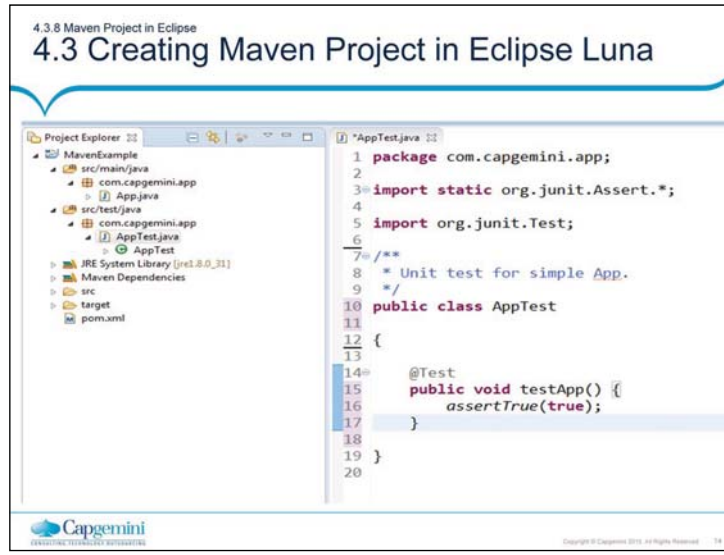
4.3 Creating Maven Project in Eclipse Luna

- Update the project as shown:



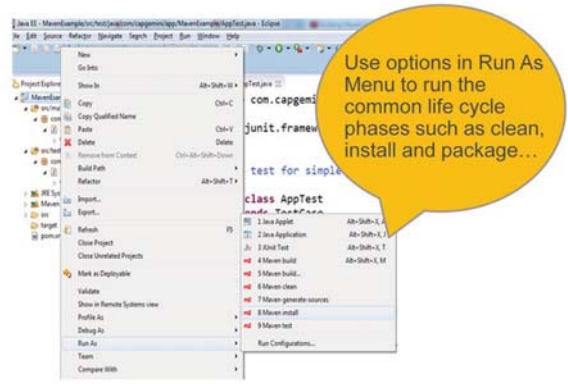
Capgemini

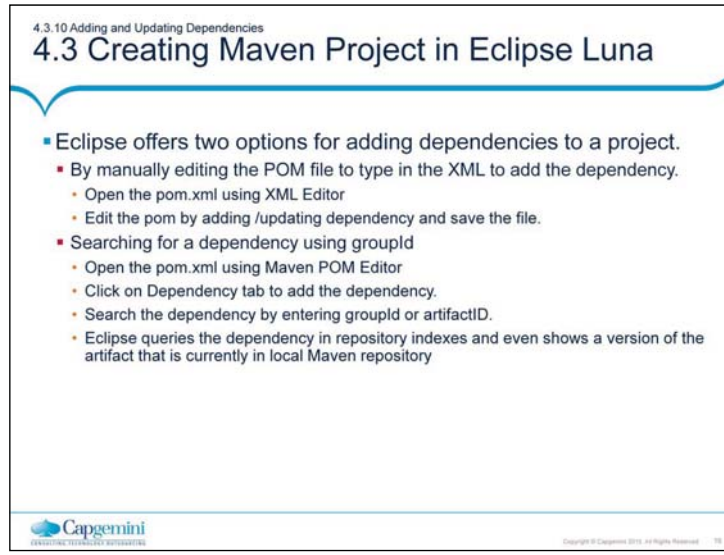

4.3.7 Maven Project in Eclipse
4.3 Creating Maven Project in Eclipse Luna
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
MavenExample
src/main/java
com.capgemini.app
App.java
App
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
App.java
1 package com.capgemini.app;
2
3 /**
4 * Hello world!
5 *
6 */
7 public class App
8 {
9 public static void main(String[] args)
10 {
11 System.out.println("Hello World!");
12 }
13 }
14
Markers Properties Servers Data Source Explorer Snippets Console JUnit
Capgemini


4.3.8 Maven Project in Eclipse
4.3 Creating Maven Project in Eclipse Luna
Project Explorer
MavenExample
src/main/java
com.capgemini.app
App.java
src/test/java
com.capgemini.app
AppTest.java
AppTest
JRE System Library [jre1.8.0_31]
Maven Dependencies
src
target
pom.xml
*AppTest.java
1 package com.capgemini.app;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.Test;
6
7 /**
8 * Unit test for simple App.
9 */
10 public class AppTest
11
12 {
13
14 @Test
15 public void testApp() {
16 assertTrue(true);
17 }
18
19 }
20
Capgemini

4.3.9 Maven Project in Eclipse

4.3 Creating Maven Project in Eclipse Luna




4.3.10 Adding and Updating Dependencies
4.3 Creating Maven Project in Eclipse Luna
Eclipse offers two options for adding dependencies to a project.
By manually editing the POM file to type in the XML to add the dependency.
Open the pom.xml using XML Editor
Edit the pom by adding /updating dependency and save the file.
Searching for a dependency using groupId
Open the pom.xml using Maven POM Editor
Click on Dependency tab to add the dependency.
Search the dependency by entering groupId or artifactID.
Eclipse queries the dependency in repository indexes and even shows a version of the artifact that is currently in local Maven repository
Capgemini

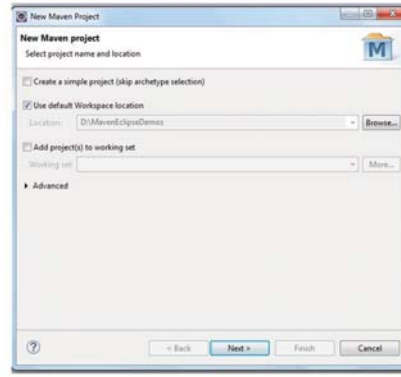
4.3 Creating Maven Projects in Eclipse Luna

- Creating and executing the “Maven Example!” program

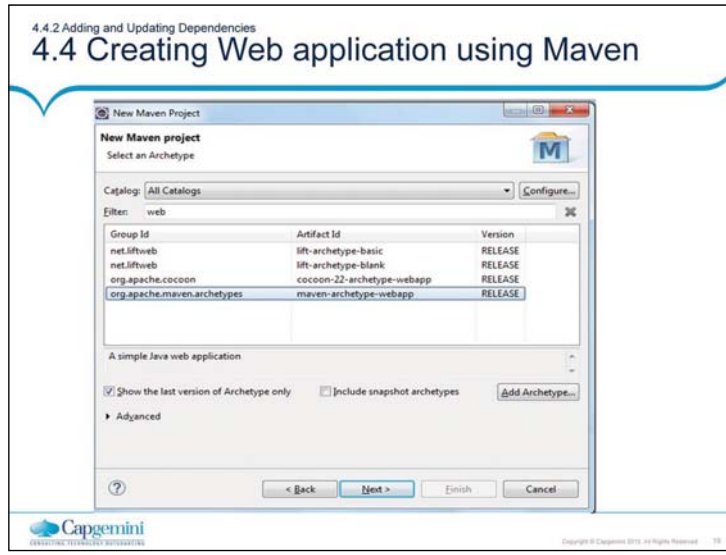
Capgemini

4.4.1 Adding and Updating Dependencies

4.4 Creating Web application using Maven

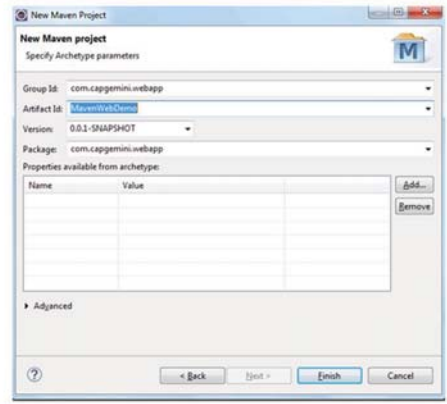


Capgemini


4.4.2 Adding and Updating Dependencies
4.4 Creating Web application using Maven
New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter: web
Group Id
Artifact Id
Version
net.liftweb
lift-archetype-basic
RELEASE
net.liftweb
lift-archetype-blank
RELEASE
org.apache.cocoon
cocoon-22-archetype-webapp
RELEASE
org.apache.maven.archetypes
maven-archetype-webapp
RELEASE
A simple Java web application
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Finish
Cancel
Capgemini

4.4.3 Adding and Updating Dependencies

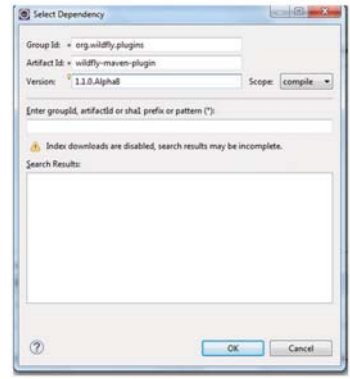
4.4 Creating Web application using Maven



Capgemini

4.4.4 Adding and Updating Dependencies

4.4 Creating Web application using Maven



Capgemini

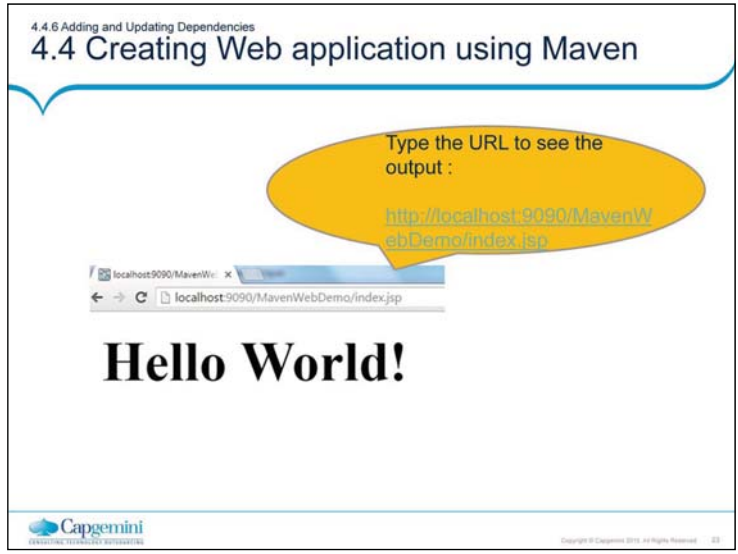
4.4.5 Adding and Updating Dependencies

4.4 Creating Web application using Maven

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version> 4.11 </version>
    <scope>test</scope>
  </dependency>
  <dependency>
      <groupId>org.wildfly.plugins</groupId>
      <artifactId>wildfly-maven-plugin</artifactId>
      <version>1.1.0.Alpha8</version>
  </dependency>
</dependencies>
<build>
  <finalName>MavenWebDemo</finalName>
</build>
</project>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Capgemini


4.4.6 Adding and Updating Dependencies
4.4 Creating Web application using Maven
Type the URL to see the output :
http://localhost:9090/MavenWebDemo/index.jsp
localhost:9090/MavenWe
localhost:9090/MavenWebDemo/index.jsp
Hello World!
Capgemini

4.4.7 Adding and Updating Dependencies

4.4 Creating Web application using Maven

Creating and executing the “Maven Example!” program

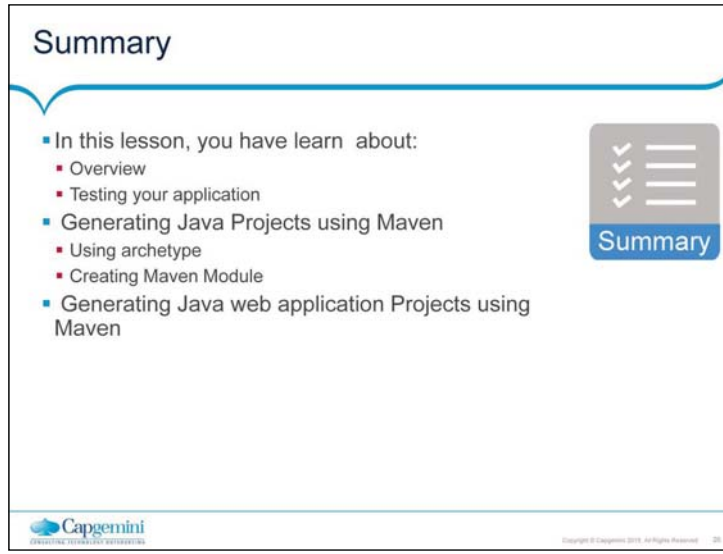
Capgemini

4.4.8 Integrating Maven with Eclipse

Lab: 2

- 2.1: Configure environment to run Maven project
- 2.2: Creating a sample Maven project

Capgemini


Summary
In this lesson, you have learn about:
Overview
Testing your application
Generating Java Projects using Maven
Using archetype
Creating Maven Module
Generating Java web application Projects using Maven
Summary
Capgemini

Review Question

- Question 1: mvn deploy command executes all the build phases in the default lifecycle
  - True
  - False
- Question 2:Configuration of dependencies, is specified in which of the following files?
  - pom.xml
  - setting.xml
  - .m2 folder
  - test files

Capgemini

Review Question

- Question 3: _____repository for containing released artifacts and _____ repository for storing deployed snapshots.
- Question 4: Which of the following statements is true?
  - groupId and artifactId must be same in a project.
  - groupId and artifactId may be same in a project.
  - groupId and artifactId must not be same in a project.

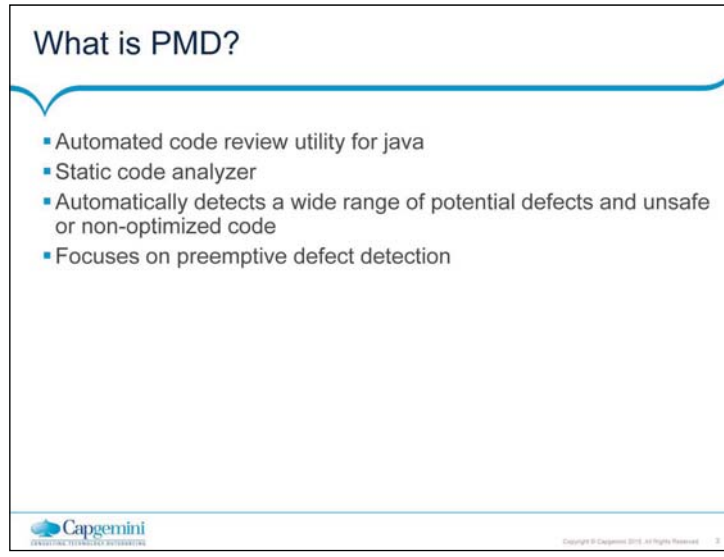
Capgemini


Developer WorkBench
PMD - Code Review Tool

Lesson Objectives

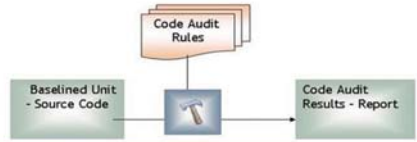
- In this lesson, you will learn:
  - What is PMD
  - PMD Rulesets
  - PMD usage
  - PMD in Action

Capgemini


What is PMD?
Automated code review utility for java
Static code analyzer
Automatically detects a wide range of potential defects and unsafe or non-optimized code
Focuses on preemptive defect detection
Capgemini

What is PMD? (Contd…)

- Possible bugs: Empty try/catch/finally/switch statements
- Dead code: Unused local variables, parameters and private methods
- Suboptimal code: Wasteful String/String Buffer usage
- Overcomplicated expressions - Unnecessary if statements, for loops that could be while loops
- Duplicate code: Copied/pasted code means copied/pasted bugs



Capgemini

PMD Rule Sets

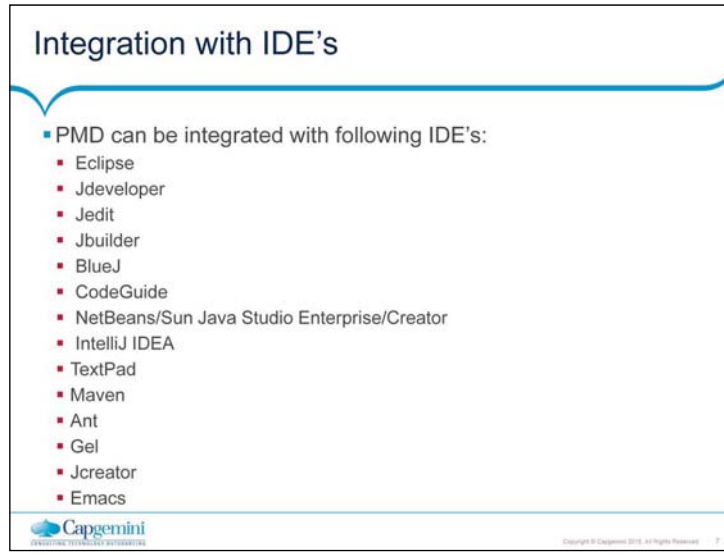
- Rules are grouped together and form a Rule Set
  E.g. Basic Rules, Code Size Rules, Coupling Rules, Design Rules, JavaBeans Rules, Junit Rules, Migration Rules, Naming Rules, Optimization Rules, Security Code Guidelines, Unused Code Rules, etc.
- For more information visit:
http://pmd.sourceforge.net/rules/index.html

Capgemini

PMD Usage

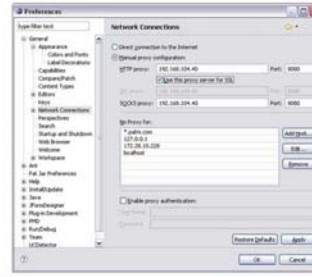
- Installation:
  - Windows and UNIX
- Prerequisites:
  - JDK 1.4 or higher
  - WinZip for Windows or "zip" utility InfoZip for UNIX
- Download the latest binary distribution - i.e., pmd-bin-x.xx.zip from http://pmd.sourceforge.net/index.html. Unzip it into any directory
  - *Note: Please contact ITIMD Helpdesk to download the plug-in

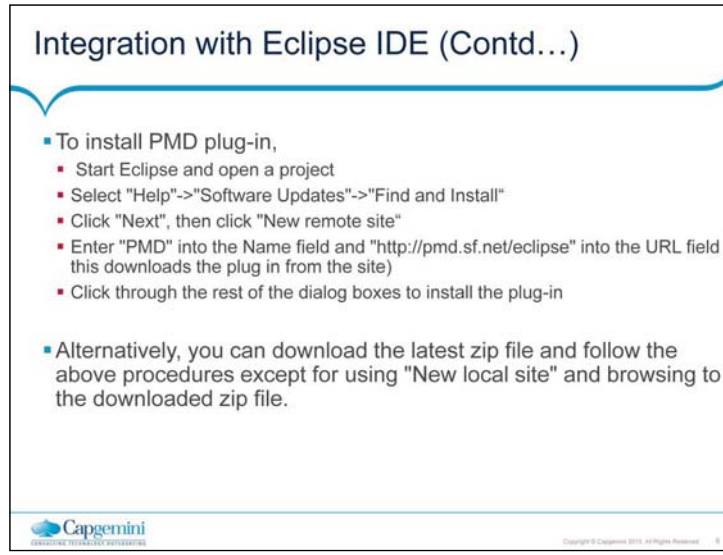
Capgemini


Integration with IDE's
PMD can be integrated with following IDE's:
Eclipse
Jdeveloper
Jedit
Jbuilder
BlueJ
CodeGuide
NetBeans/Sun Java Studio Enterprise/Creator
IntelliJ IDEA
TextPad
Maven
Ant
Gel
Jcreator
Emacs
Capgemini

Integration with Eclipse IDE

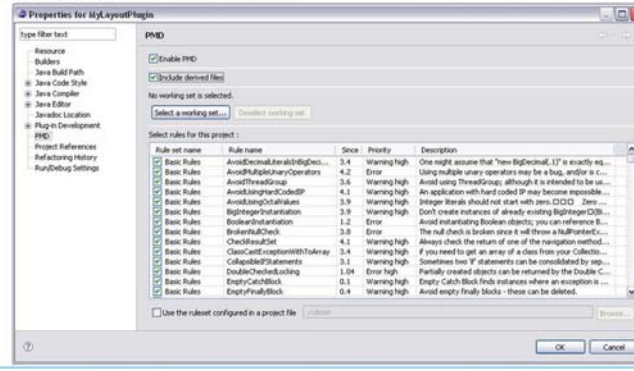
- Configure Eclipse:
  - Setup the network connection in Eclipse preferences. Go to Windows -> Preferences -> General -> Network Connection
  - Set your preferences as per your Internet Browser Connections Settings




Integration with Eclipse IDE (Contd…)
To install PMD plug-in,
Start Eclipse and open a project
Select "Help"->"Software Updates"->"Find and Install"
Click "Next", then click "New remote site"
Enter "PMD" into the Name field and "http://pmd.sf.net/eclipse" into the URL field (this downloads the plug in from the site)
Click through the rest of the dialog boxes to install the plug-in
Alternatively, you can download the latest zip file and follow the above procedures except for using "New local site" and browsing to the downloaded zip file.
Capgemini

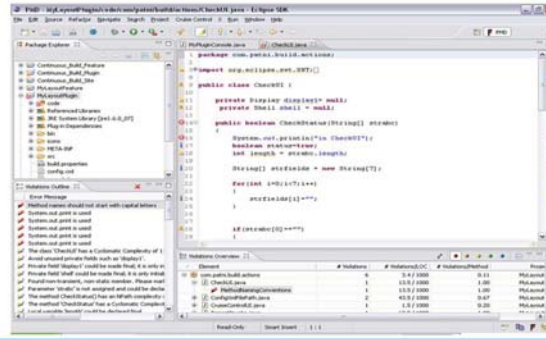
Integration with Eclipse IDE (Contd…)

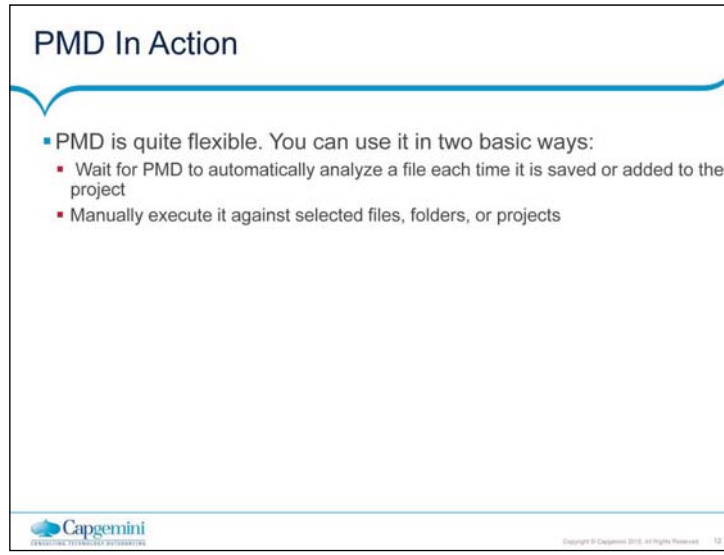
- Configuring PMD in a Project:



Integration with Eclipse IDE (Contd…)

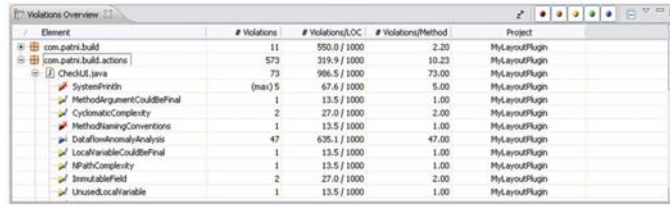
- To run PMD with Eclipse – Right click on the project, select PMD – Check Code with PMD




PMD In Action
PMD is quite flexible. You can use it in two basic ways:
Wait for PMD to automatically analyze a file each time it is saved or added to the project
Manually execute it against selected files, folders, or projects
Capgemini

PMD In Action (Contd…)

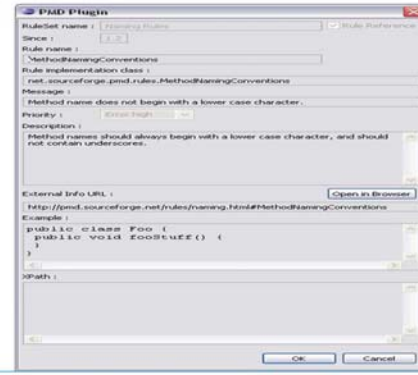
- PMD violations have the following five levels of severity:
  - Error (high)
  - Error
  - Warning (high)
  - Warning
  - Information



Capgemini

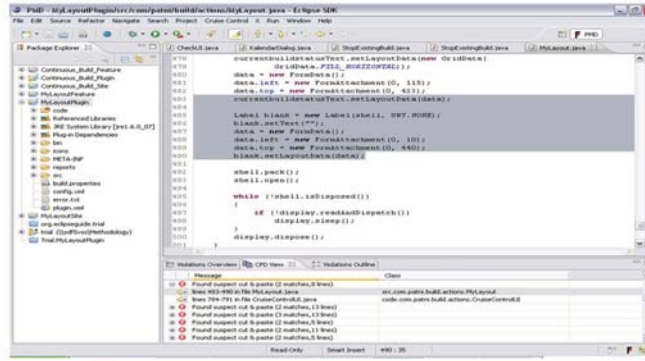
PMD In Action (Contd…)

- Details of the Rule



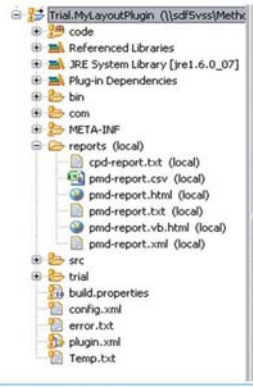
PMD In Action (Contd…)

- Detecting Cut Pasted Code



Capgemini

PMD In Action (Contd…)



- Reporting Feature: PMD generates reports in following formats:
  - .txt
  - .csv
  - .html
  - .vb.html
  - .xml

Capgemini

PMD In Action (Contd…)

- HTML Reporting Format:

PMD report

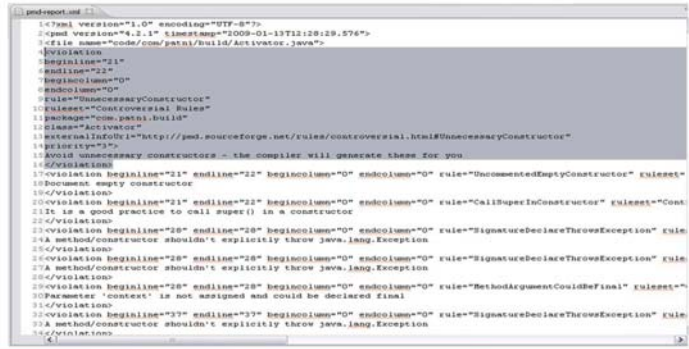
Problems found

#	File	Line	Problem
1	code/com/patni/build/Activator.java	21	Avoid unnecessary constructors - the compiler will generate these for you
2	code/com/patni/build/Activator.java	21	Document empty constructor
3	code/com/patni/build/Activator.java	21	It is a good practice to call super() in a constructor
4	code/com/patni/build/Activator.java	28	A method/constructor shouldn't explicitly throw java.lang.Exception
5	code/com/patni/build/Activator.java	28	A method/constructor shouldn't explicitly throw java.lang.Exception
6	code/com/patni/build/Activator.java	28	Parameter 'context' is not assigned and could be declared final
7	code/com/patni/build/Activator.java	37	A method/constructor shouldn't explicitly throw java.lang.Exception
8	code/com/patni/build/Activator.java	37	A method/constructor shouldn't explicitly throw java.lang.Exception
9	code/com/patni/build/Activator.java	37	Parameter 'context' is not assigned and could be declared final
10	code/com/patni/build/Activator.java	38	Assigning an Object to null is a code smell. Consider refactoring.
11	code/com/patni/build/Activator.java	58	Parameter 'path' is not assigned and could be declared final
12	code/com/patni/build/actions/CheckUI1.java	7	The class 'CheckUI1' has a Cyclomatic Complexity of 11 (Highest = 10).
13	code/com/patni/build/actions/CheckUI1.java	9	Avoid unused private fields such as 'display1'.
14	code/com/patni/build/actions/CheckUI1.java	9	This final field could be made static
15	code/com/patni/build/actions/CheckUI1.java	10	This final field could be made static
16	code/com/patni/build/actions/CheckUI1.java	12	The method 'checkStatus' has a Cyclomatic Complexity of 10.
17	code/com/patni/build/actions/CheckUI1.java	12	The method checkStatus() has an NPath complexity of 512

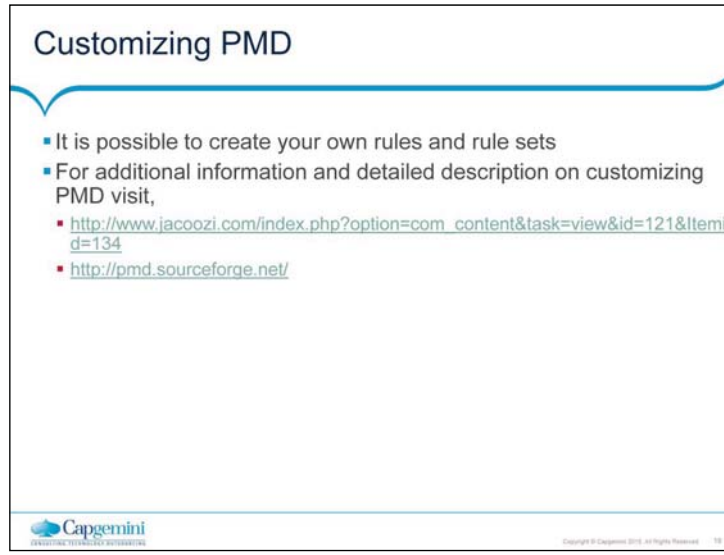
Capgemini

PMD In Action (Contd…)

- XML Reporting Format:



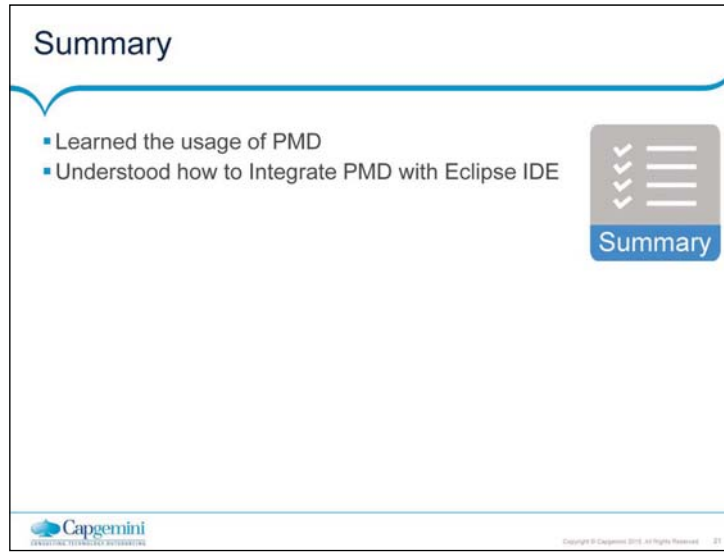
Capgemini


Customizing PMD
It is possible to create your own rules and rule sets
For additional information and detailed description on customizing PMD visit,
http://www.jacoozi.com/index.php?option=com_content&task=view&id=121&Itemid=134
http://pmd.sourceforge.net/
Capgemini

Lab: 3

- 2.1: Installing PMD
- 2.2 Using PMD with Eclipse

Capgemini

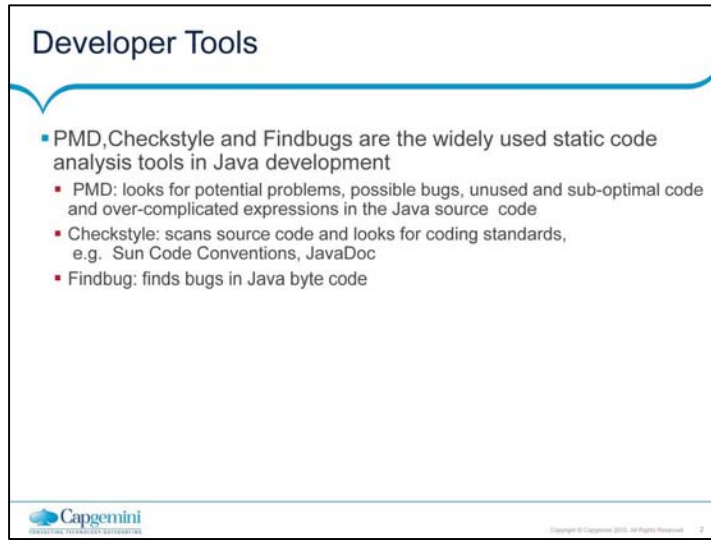

Summary
Learned the usage of PMD
Understood how to Integrate PMD with Eclipse IDE
Summary
Capgemini

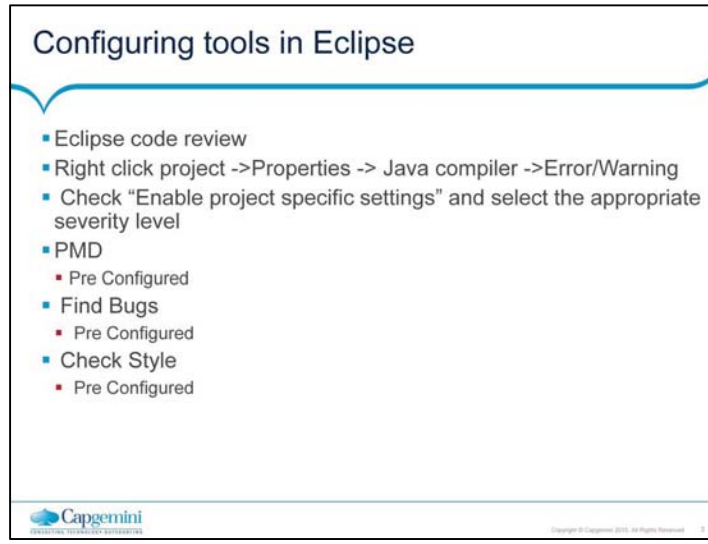
Review Question

- Question 1: PMD is a
  - Review tool
  - Testing Tool
  - Design Tool
- Question 2: PMD violations has _____ level of severity
  - Error (high)
  - Fatal
  - Warning (high)
  - Information

Capgemini


Developer WorkBench
Appendices


Developer Tools
PMD,Checkstyle and Findbugs are the widely used static code analysis tools in Java development
PMD: looks for potential problems, possible bugs, unused and sub-optimal code and over-complicated expressions in the Java source code
Checkstyle: scans source code and looks for coding standards, e.g. Sun Code Conventions, JavaDoc
Findbug: finds bugs in Java byte code
Capgemini

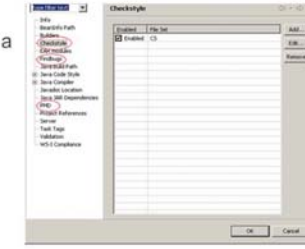

Configuring tools in Eclipse
Eclipse code review
Right click project ->Properties -> Java compiler ->Error/Warning
Check "Enable project specific settings" and select the appropriate severity level
PMD
Pre Configured
Find Bugs
Pre Configured
Check Style
Pre Configured
Capgemini

Tools Selection

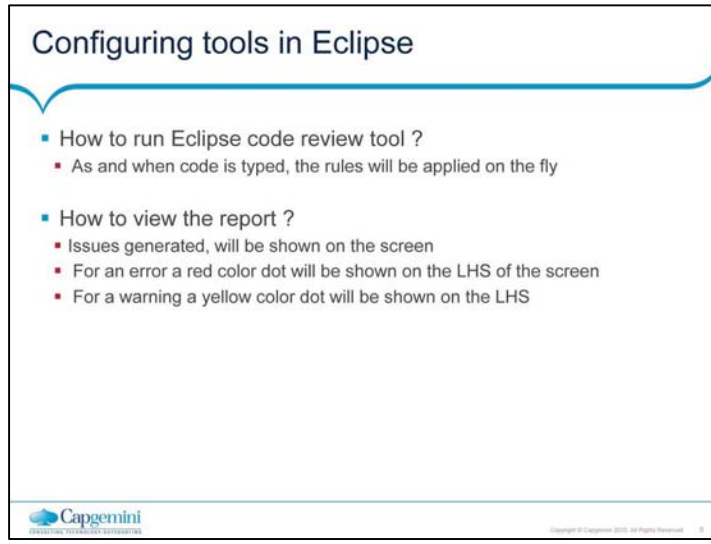
- Eclipse can be configured to run a particular or all code review tools
  - In the “Problem” panel, click Filter
  - Tools can be selected / de-selected from “Show Items of Type” section
  - Select PMD, Find Bugs or Check style accordingly

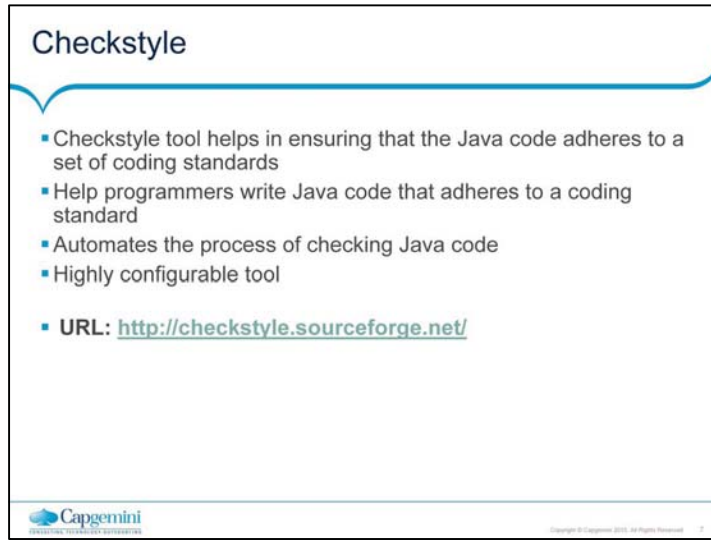
Configuring tools in Eclipse

- Rules can be configured according to the specific needs of a project
  - Select the project and right click-> go to properties-> go to Window -> Preference
  - Select the tool that needs to be configured



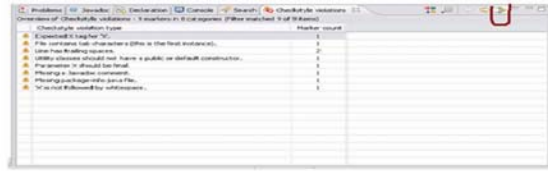
Capgemini


Configuring tools in Eclipse
How to run Eclipse code review tool ?
As and when code is typed, the rules will be applied on the fly
How to view the report ?
Issues generated, will be shown on the screen
For an error a red color dot will be shown on the LHS of the screen
For a warning a yellow color dot will be shown on the LHS
Capgemini

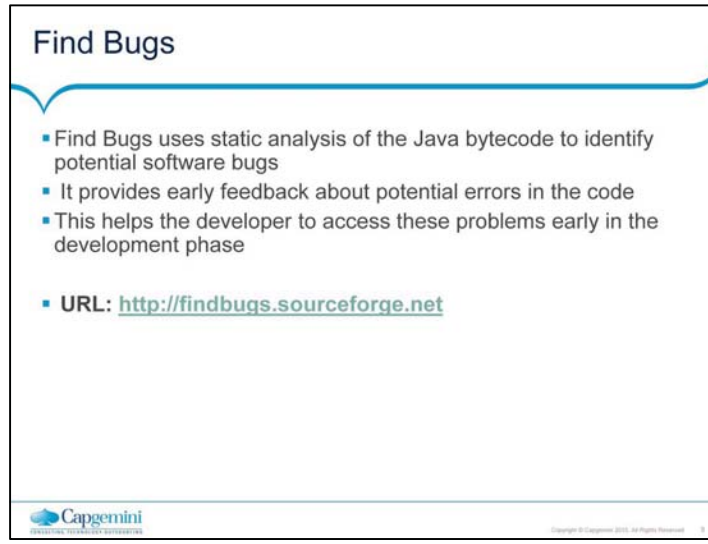

Checkstyle
Checkstyle tool helps in ensuring that the Java code adheres to a set of coding standards
Help programmers write Java code that adheres to a coding standard
Automates the process of checking Java code
Highly configurable tool
URL: http://checkstyle.sourceforge.net/
Capgemini

Configuring Checkstyle tool in Eclipse

- Steps to run Check Style code review tool
  - Select the project, right click -> Properties -> Check Style, and click Add
  - Provide a “File set name” and select “Sun Checks” in Check configuration
  - Click OK
- Steps to view the report
  - Click “Filters” icon in the View panel, and Check “Checkstyle Marker” and click OK



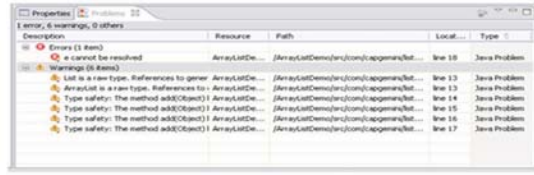
Capgemini



Appendix B. Code Examples

Configuring Find Bugs tool in Eclipse

- Steps to run the FindBugs review tool
  - Select the Project->Right click and select Find Bugs -> Find Bugs
- Steps to view the report
  - Go to Window ->Show View -> Problems, and Click OK
  - Error report is generated, under “Problem” View



#	Source (Book / Site url)	Author / Site details	Specify Courseware Lesson and Slide Number/ Page Number where Content is reproduced from the Source
1			
2			
3			

Developer WorkBench for JEE
Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
01-Nov -11	1.0	Rathnajothi Perumalsamy	
5-6-2016	1.1	Zainab Kulkarni	ANT was replaced with MAVEN

Table of Contents

Document Revision History 2
Getting Started 4
Overview 4
Setup Checklist for Maven and PMD 4
Instructions 4
Lab 1. Getting Started With Maven 5
1.1: Setting environmental variables 5
1.2 Configuring Maven Settings: 8
1.3 Creating first standalone Maven application using archetype: 9
Figure 8: Creating first standalone Maven application using archetype 10
<<TODO>> 12
Lab 2. Integrating Maven with Eclipse 13
2.1: Configure environment to run Maven project 13
2.2: Creating a sample Maven project 13
<<TODO>> 18
Lab 3. PMD Tool 20
2.1: Installing PMD 20
2.2 Using PMD with Eclipse 20
<< TO DO>> 24
Appendices 24
Appendix A: Table of Figures 24

Getting Started

Overview

This lab book is a guided tour for learning Maven tool and PMD tool. It comprises step by step guidance and 'To Do' assignments. Follow the steps provided in the solved examples and work out the 'To Do' assignments given which will expose you to working with Java applications.

Setup Checklist for Maven and PMD

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher
- MS-Access/Connectivity to Oracle database
- Apache Tomcat Version 5.0.
- Apache Maven 3.0

Please ensure that the following is done:

- Eclipse Luna is installed.
- JDK 1.8 is installed. (This path is henceforth referred as <java_install_dir>)
- Wildfly 8.1

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory java_assgn. For each lab exercise create a directory as lab <lab number>.

Lab 1. Getting Started With Maven

Goals	• Learn and Understand the process of
	  • Setting environment variables
	  • Configuring Maven settings
	  • Creating a simple Maven Project using commands

Time	60 minutes

1.1: Setting environmental variables

Step1: set M2_Home to Maven Installation Directory using the following command:

- **set** M2**_HOME= C:\apache-maven-version.**

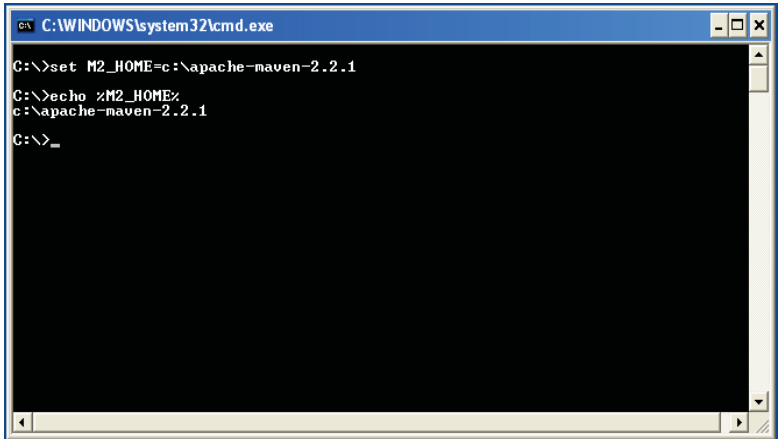


Figure 1: Environmental Variable

Step 2: Set **JAVA_HOME** to Jdk1.8.0_31 using the following command:

- **set JAVA_HOME= C:**\Program Files\Java\jdk1.8.0_31

Step 3: Set PATH environment variable:

- **Set PATH=%PATH%;%JAVA_HOME%\bin;%M2_HOME%\bin;**

Alternatively follow the following steps for setting the environment variables

Alternate approach:

Step 1: Right click **My Computers**, and select **Properties→Environment Variables**.

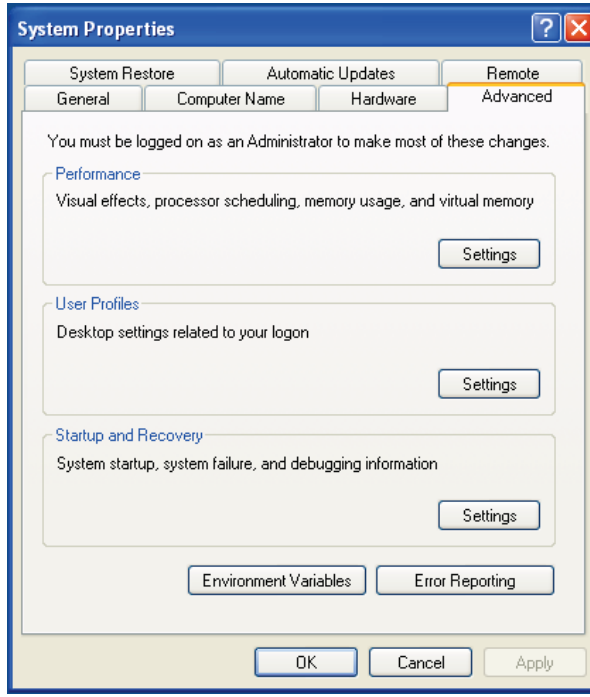


Figure 2: System Properties

Step 2: Click **Environment Variables**. The Environment Variables window will be displayed.

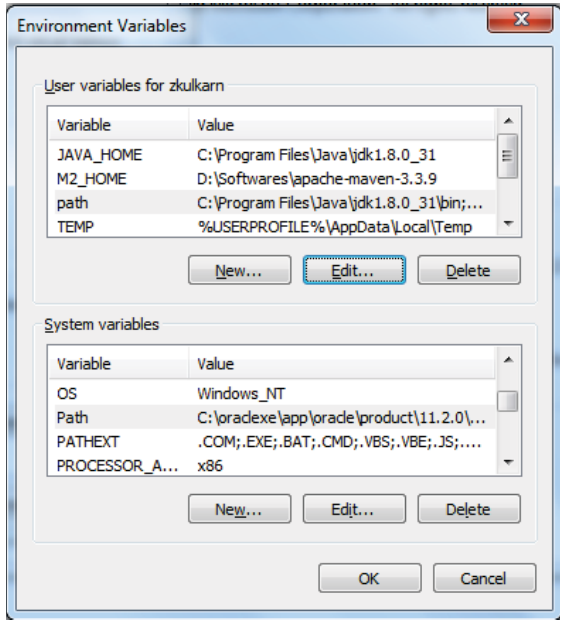


Figure 3: Environment Variables

Step 3: Create a new user variable M2_HOME by clicking on edit and set the path of Apache Maven installation path as shown in the figure.

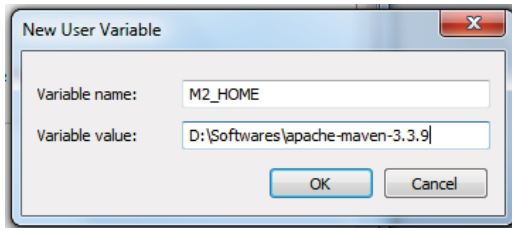


Figure 4: Edit User Variable

Step 4: Click **JAVA_HOME** System Variable if it already exists, or create a new one and set the path of JDK1.8.0_31 as shown in the figure.

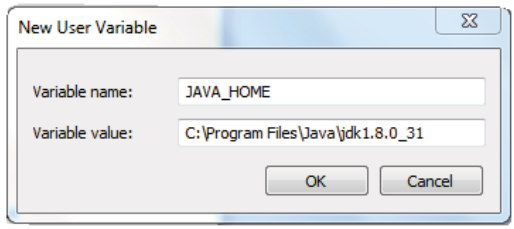


Figure 5: Edit User Variable

Step 5: Click **PATH** System Variable and set it as ***%JAVA_HOME%\bin*;%M2_HOME%\bin;**

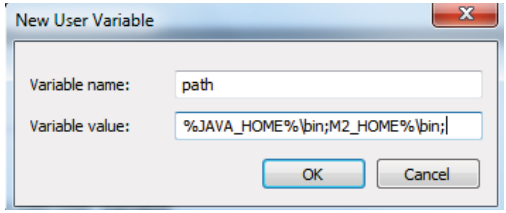


Figure 6: Edit User Variable

1.2 Configuring Maven Settings:

Step 1: Edit settings.xml to configure the proxy settings for downloading artifacts from remote repository.

```
<settings>
….
        <proxies>
                <proxy>
                        <active>true</active>
                        <protocol>http</protocol>
                        <host>proxy.mycompany.com</host>
                        <port>8080</port>
                        <username>your-username</username>
                        <password>your-password</password>
                </proxy>
        </proxies>
```

Figure 7: settings.xml

1.3 Creating first standalone Maven application using archetype:

Step 1: Execute the following command to create Maven Project

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

The execution of the above command should result into the display of archetypes as shown below:


```
D:\maven-demo>mvn archetype:generate -DgroupId=com.capgemini.app -DartifactId=my
-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources
@ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources
@ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom --
-
[INFO] Generating project in Batch mode

[INFO] Parameter: basedir, Value: D:\maven-demo
[INFO] Parameter: package, Value: com.capgemini.app
[INFO] Parameter: groupId, Value: com.capgemini.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Parameter: packageName, Value: com.capgemini.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: D:\maven-demo\my-app
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.301 s
[INFO] Finished at: 2016-05-13T02:18:12+05:30
[INFO] Final Memory: 10M/26M
[INFO] ------------------------------------------------------------------------
```

Figure 8: Creating first standalone Maven application using archetype

Step 2: After successful execution of the command, Maven will create the project directory my-app having pom.xml with the contents shown below.

Keep the application specific files in ${basedir}/src/main/java and test sources reside in ${basedir}/src/test/java, where ${basedir} represents the directory containing pom.xml.

```
<project>
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.capgemini.app</groupId>
	<artifactId>my-app</artifactId>
	<packaging>jar</packaging>
	<version>1.0</version>
	<name>my-app</name>
	<url>http://maven.apache.org</url>
	<dependencies>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
```

Figure 9: Sample pom.xml

Step 3: Execute the "mvn compile" command to compile your application sources as shown below:

```
D:\maven-demo\my-app>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ my-app ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory D:\maven-demo\my-app\src\main\resourc
es
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ my-app ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. b
uild is platform dependent!
[INFO] Compiling 1 source file to D:\maven-demo\my-app\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.699 s
[INFO] Finished at: 2016-05-14T22:40:51+05:30
[INFO] Final Memory: 10M/26M
[INFO] ------------------------------------------------------------------------
```

Figure 10: Compiling Sample Project

Step 4: Unit testing of the project can be performed by executing the following command:

mvn test

After successful execution of the command, the test result will be displayed which comprises the details such as test run,failures, errors...

Step 5: Artifact can be packaged and installed into local repository using commands such as
mvn package and mvn install.

- mvn package packages the artifact based on the packaging type specified in the pom.xml.
- mvn install installs the packaged artifact into the local repository.

Step 6: Basic Standard site for project can also be created using command mvn site.

<<TODO>>

Assignment-1: Create a Banking System project in maven which maintains two kinds of accounts for customers, one called savings account and the other as current account. The savings account provides compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book but no interest. Current account holders should have a minimum balance else they should pay service charges. Build, test and deploy the project into local repository.

Lab 2. Integrating Maven with Eclipse

Goals	• Learn and Understand the process of
	  • Installing m2eclipse plugin
	  • Configure environment to run Maven Project
	  • Creating a simple Maven Project in eclipse

Time	60 minutes

2.1: Configure environment to run Maven project

Step1: To execute Maven project, eclipse needs to be run using JDK instead of JRE.

Step2: Start the eclipse now, tocreate a Maven Project and run successfully.

2.2: Creating a sample Maven project

Create a simple java project named 'myproject'.

Solution:

Step 1: Open **eclipse3.3**.

Step 2: Select **File→New→Project →Maven project**.

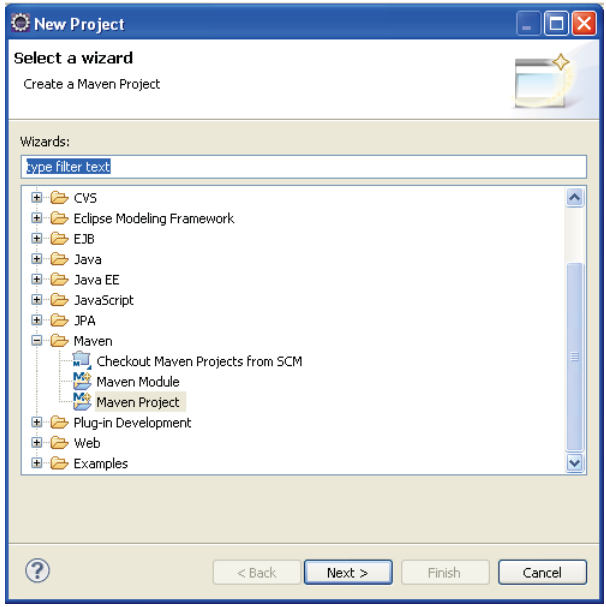


Figure 9: Select Wizard in Eclipse

Step 3: Choose Maven Project and use the default Workspace location or specify the location if necessary.

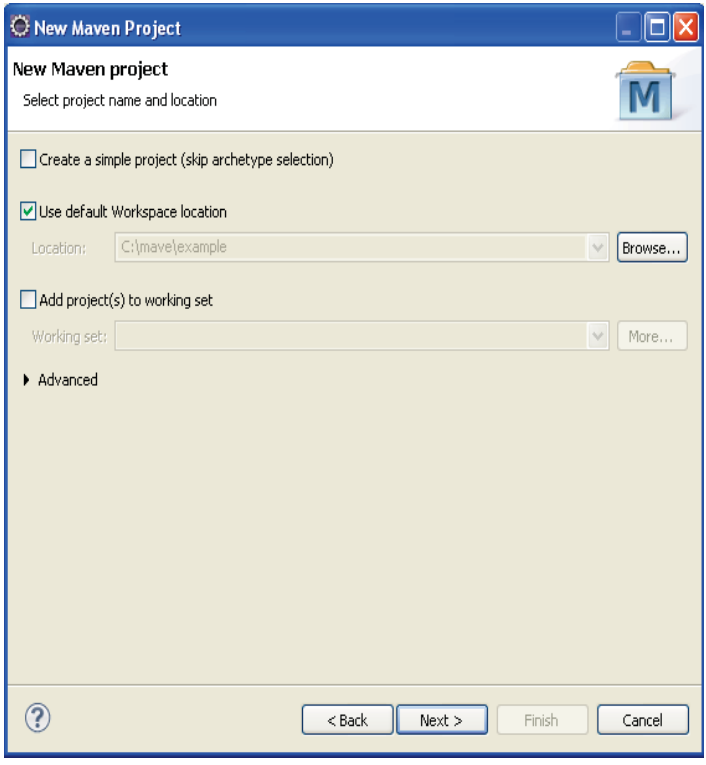


Figure 10: New Maven Project

Step 4: Select the maven-archetype-quickstart archetype from the list.

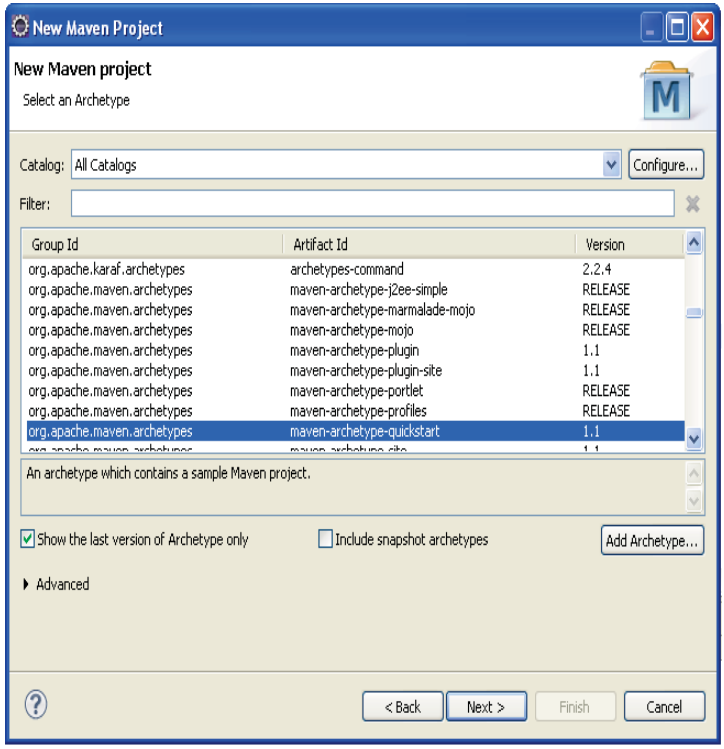


Figure 11: Selecting an archetype

Step 5: Enter the project coordinate details such as Group Id, Artifact Id and click 'Finish'

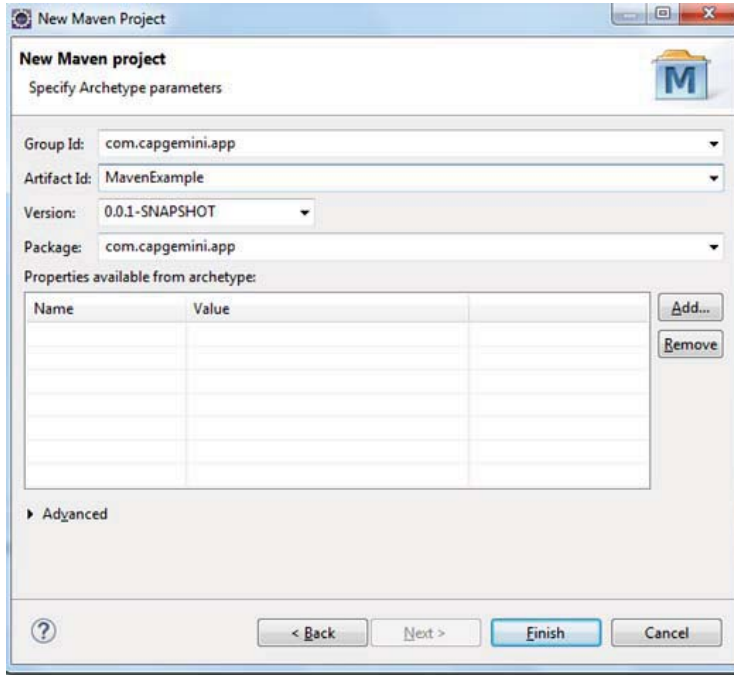


Figure 12: Specifying Archetype Parameters

Step 6: To build the project, right click on project named “examples” and choose Maven Build under Run As option.

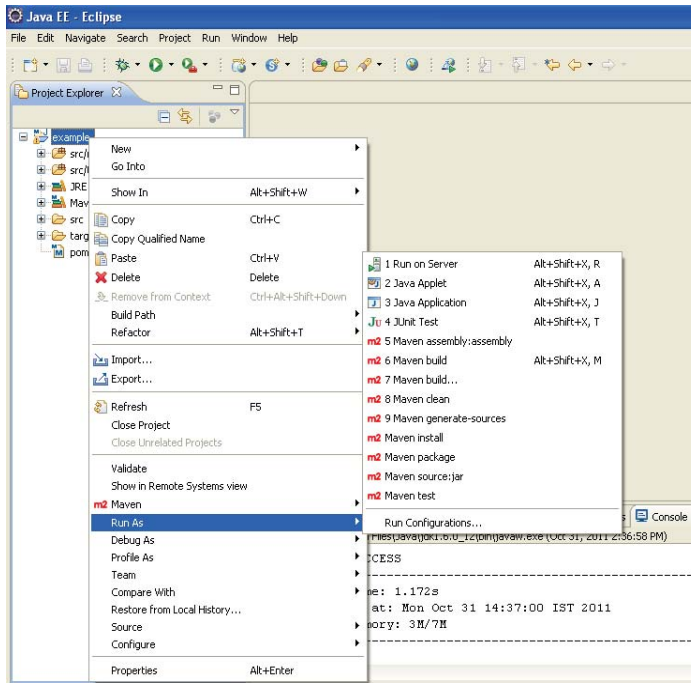


Figure 13: Building an Application

<<TODO>>

Assignment 1: Create aweb application which displays product details such as Product Name, Product description, and its price. Users can place orders specifying the quantity of each product. Once the order is placed by customer, the invoice for the current products transaction showing the product name,

quantity ordered, price and total amount should be displayed. Build and execute the common life cycle phases for the web application in eclipse.

Assignment 2: Use the Assignment 1 in Lab 1, import the Banking system project into eclipse and build the project in eclipse environment.

Lab 3. PMD Tool

Goals	At the end of this lab session, you will be able to: ○ Install PMD. ○ Use PMD with Eclipse.
Time	90 minutes

2.1: Installing PMD

Solution:

Step 1: Download the latest binary distribution - i.e., pmd-bin-x.xx.zip from Http://pmd.sourceforge.net/index.html or obtain it from the faculty.

Step 2: Extract the contents of the downloaded zip file.

2.2 Using PMD with Eclipse

Solution:

Step 1: import the Project named **PMDExample**.

Step 2: select the project and click properties. Select PMD and check Enable PMD

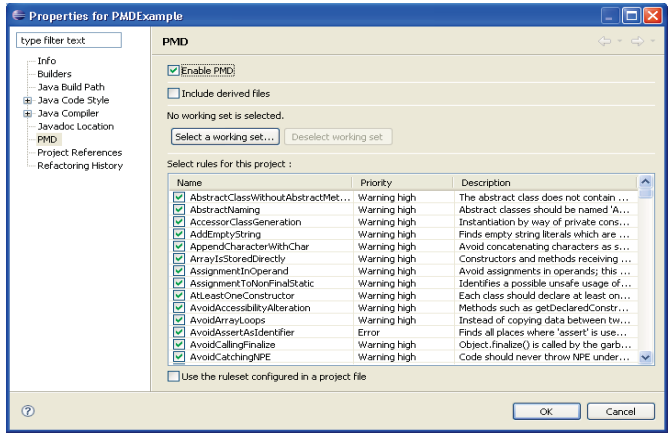


Figure 114: Enable PMD

Step 3: To run PMD with Eclipse right click on the project, select PMD, check code with PMD.

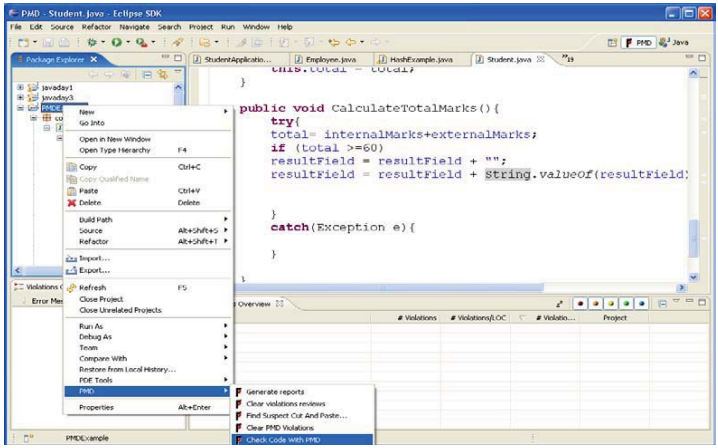


Figure 115: check code with PMD

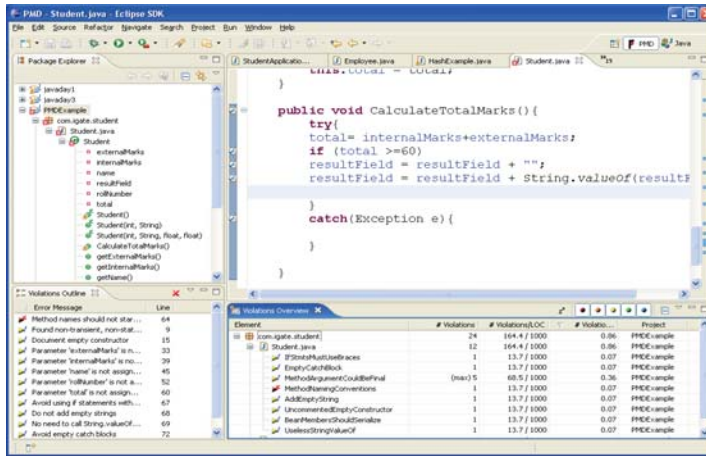


Figure 116: Eclipse showing with Violations

Step 4: Right click project select PMD – Generate Report

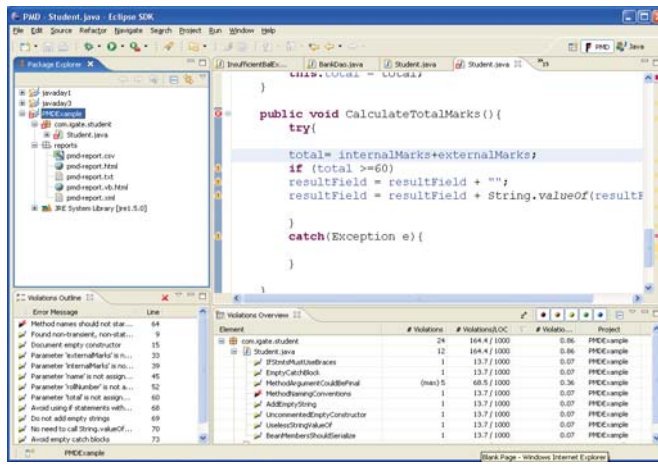


Figure 117: Generate Report

Step 5: Double click pmd-report.html

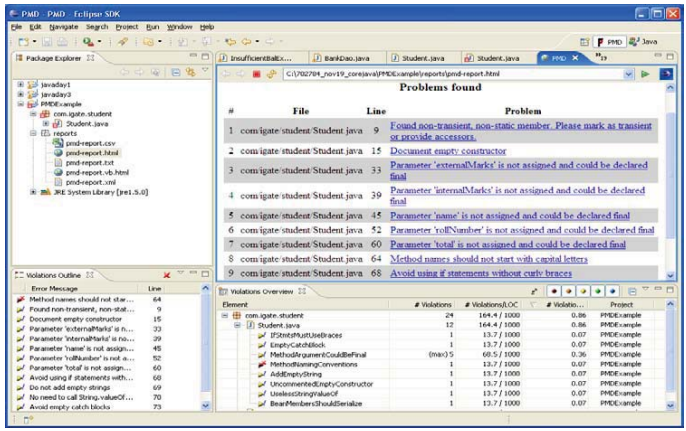


Figure 18: HTML Reporting Format

Step 6: Double click pmd-report.html

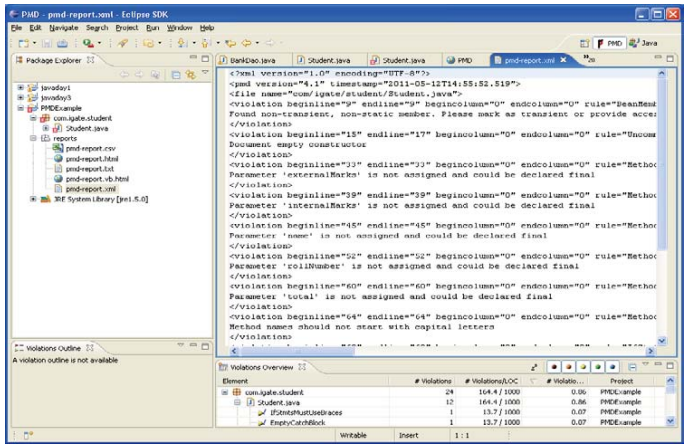


Figure 19: XML Reporting Format

<< TO DO>>

Assignment-1:
Review the java code created in the core java Lab Assignment - 4 using PMD tool. Generate the HTML and XML reports.

Appendices

Appendix A: Table of Figures

Figure 1: Environmental Variable 5
Figure 2: System Properties 6
Figure 3: Environment Variables 7
Figure 4: Edit User Variable 8
Figure 5: Edit User Variable 8
Figure 6: Edit User Variable 8
Figure 7: settings.xml 9
Figure 8: Creating first standalone Maven application using archetype 10
Figure 11: Select Wizard in Eclipse 14
Figure 12: New Maven Project 15
Figure 13: Selecting an archetype 16
Figure 14: Specifying Archetype Parameters 17
Figure 15: Building an Application 18
Figure 16: Enable PMD 21
Figure 17: check code with PMD 21
Figure 18: Eclipse showing with Violations 22
Figure 19: Generate Report 22
Figure 20: HTML Reporting Format 23
Figure 21: XML Reporting Format 23

