Maven Interview Questions and Answers

Is there a way to use the current date in the POM?

Take a look at the buildnumber plugin. It can be used to generate a build date each time I do a build, as follows:

```
<plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>maven-buildnumber-plugin</artifactId>
      <version>0.9.4</version>
      <configuration>
        <format>{0,date,yyyy-MM-dd HH:mm:ss}</format>
        <items>
          <item>timestamp</item>
        </items>
        <doCheck>false</doCheck>
        <doUpdate>false</doUpdate>
      </configuration>
      <executions>
        <execution>
          <phase>validate</phase>
          <goals>
            <goal>create</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
```

pom.xml or settings.xml? What is the best practice configuration usage for these files?

The best practice guideline between settings.xml and pom.xml is that configurations in settings.xml must be specific to the current user and that pom.xml configurations are specific to the project.

For example, <repositories> in pom.xml would tell all users of the project to use the <repositories> specified in the pom.xml. However, some users may prefer to use a mirror instead, so they'll put <mirrors> in their settings.xml so they can choose a faster repository server.

so there you go:

settings.xml -> user scope
pom.xml -> project scope

How do I indicate array types in a MOJO configuration?

```
<tags>
  <tag>value1</tag>
  <tag>value2</tag>
</tags>
```

How should I point a path for maven 2 to use a certain version of JDK when I have different versions of JDK installed on my PC and my JAVA_HOME already set?

If you don't want to change your system JAVA_HOME, set it in maven script instead.

How do I setup the classpath of my antrun plugin to use the classpath from maven?

The maven classpaths are available as ant references when running your ant script. The ant reference names and some examples can be found here: maven-antrun-plugin

Is it possible to use HashMap as configurable parameter in a plugin? How do I configure that in pom.xml?

Yes. Its possible to use a HashMap field as a parameter in your plugin. To use it, your pom configuration should look like this:

```
<myMap>
   <yourkey>yourvalue</yourkey>
   .....
 </myMap>
```

How do I filter which classes should be put inside the packaged jar?

All compiled classes are always put into the packaged jar. However, you can configure the compiler plugin to exclude compiling some of the java sources using the compiler parameter excludes as follows:

```
<project>
 ...
 <build>
   ...
   <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-compiler-plugin</artifactId>
       <configuration>
         <excludes>
           <exclude>**/NotNeeded*.java</exclude>
```

```
        </excludes>
      </configuration>
    </plugin>
  </plugins>
  ...
 </build>
</project>
```

How can I change the default location of the generated jar when I command "mvn package"?

By default, the location of the generated jar is in ${project.build.directory} or in your target directory.
We can change this by configuring the outputDirectory of maven-jar-plugin.

```
<plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-jar-plugin</artifactId>
          <configuration>
             <outputDirectory>${project.build.directory}/<!-- directory --
></outputDirectory>
          </configuration>
       </plugin>
```

How does maven 2 implement reproducibility?

- Add the exact versions of plugins into your pluginDepenencies (make use of the release plugin)
- Make use of ibiblio for your libraries. This should always be the case for jars. (The group is working on stabilising metadata and techniques for locking it down even if it changes. An internal repository mirror that doesn't fetch updates (only new) is recommended for true reproducibility.)

Why there are no dependency properties in Maven 2?

They were removed because they aren't reliable in a transitive environment. It implies that the dependency knows something about the
environment of the dependee, which is back to front. In most cases, granted, the value for war bundle will be the same for a particular
dependency - but that relies on the dependency specifying it.

In the end, we give control to the actual POM doing the building, trying to use sensible defaults that minimise what needs to be
specified, and allowing the use of artifact filters in the configuration of plugins.

What does aggregator mean in mojo?

When a Mojo has a @aggregator expression, it means that It can only build the parent project of your multi-module-project, the one who has the packaging of pom. It can also give you values for the expression ${reactorProjects} where reactorProjects are the MavenProject references to the parent pom modules.

Where is the plugin-registry.xml?

From the settings.xml, you may enable it by setting <usePluginRegistry/> to true and the file will be in ~/.m2/plugin-registry.xml

How do I create a command line parameter (i.e., -Dname=value) in my mojo?

In your mojo, put "expression=${<exp>}" in your parameter field

```
/**
 * @parameter expression="${expression.name}"
 */
private String exp;
```

You may now able to pass parameter values to the command line.
"mvn -Dexpression.name=value install"

How do I convert my <reports> from Maven 1 to Maven 2?

In m1, we declare reports in the pom like this:

```
<project>
  ...
  <reports>
    <report>maven-checkstyle-plugin</report>
    <report>maven-pmd-plugin</report>
  </reports>
</project>
```

In m2, the <reports> tag is replaced with <reporting>

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```
      <artifactId>maven-checkstyle-plugin</artifactId>
      <configuration>
        <!-- put your config here -->
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <configuration>
        <!-- put your config here -->
      </configuration>
    </plugin>
  </plugins>
 <reporting>
</project>
```

What does the "You cannot have two plugin executions with the same (or missing) elements" message mean?

It means that you have executed a plugin multiple times with the same <id>. Provide each <execution> with a unique <id> then it would be ok.

How do I add my generated sources to the compile path of Maven, when using modello?

Modello generate the sources in the generate-sources phase and automatically adds the source directory for compilation in maven. So you don't have to copy the generated sources. You have to declare the modello-plugin in the build of your plugin for source generation (in that way the sources are generated each time).

What is Maven's order of inheritance?

1. parent pom
2. project pom
3. settings
4. CLI parameters

where the last overrides the previous.

How do I execute the assembly plugin with different configurations?

Add this to your pom,

```
<build>
  ...
```

```
<plugins>
 <plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
   <execution>
 <id>1</id>
     <phase>install</phase>
  <goals>
      <goal>assembly</goal>
  </goals>
    <configuration>
        <descriptor>src/main/descriptors/bin.xml</descriptor>
        <finalName>${project.build.finalName}-bin</finalName>
  </configuration>
    </execution>

    <execution>
      <id>2</id>
     <phase>install</phase>
   <goals>
      <goal>assembly</goal>
  </goals>
    <configuration>
      <descriptor>src/main/descriptors/src.xml</descriptor>
    <finalName>${project.build.finalName}-src</finalName>
  </configuration>
  </execution>
  </executions>
</plugin>
</plugins>
...

</build>
```

and run mvn install, this will execute the assembly plugin twice with different config.

How do I configure the equivalent of maven.war.src of war plugin in Maven 2.0?

```
<build>
 ...
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-war-plugin</artifactId>
   <configuration>
```

```
        <warSourceDirectory><!-- put the path of the directory --
></warSourceDirectory>
      </configuration>
    </plugin>
  </plugins>
  ...
</build>
```

How do I add main class in a generated jar's manifest?

Configure the maven-jar-plugin and add your main class.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
   <mainClass>com.mycompany.app.App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

What does the FATAL ERROR with the message "Class org.apache.commons.logging.impl.Jdk14Logger does not implement Log" when using the maven-checkstyle-plugin mean?

Checkstyle uses commons-logging, which has classloader problems when initialized within a Maven plugin's container. This results in the above message - if you run with '-e', you'll see something like the following:

—
Caused by: org.apache.commons.logging.LogConfigurationException: org.apache.commons.logging.LogConfigurationException: Class org.apache.commons.logging.impl.Jdk14Logger does not implement Log
—

buried deep in the stacktrace.

The only workaround we currently have for this problem is to include another commons-logging Log implementation in the plugin itself. So, you can solve the problem by adding the following to your plugin declaration in your POM:

```
<project>
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <dependencies>
          <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.12</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
  ...
  <reporting>
    ...
    <plugins>
      <!-- your checkstyle report is registered here, according to Maven
documentation -->
    </plugins>
  </reporting>
</project>
```

While this may seem a counter-intuitive way of configuring a report, it's important to remember that Maven plugins can have a mix of reports and normal mojos. When a POM has to configure extra dependencies for a plugin, it should do so in the normal plugins section.
We will probably try to fix this problem before the next release of the checkstyle plugin.

UPDATE: This problem has been fixed in the SVN trunk version of the checkstyle plugin, which should be released very soon.
Plugins and Lifecycle, Sites & Reporting, Errors

How do I determine the stale resources in a Mojo to avoid reprocessing them?

This can be done using the following piece of code:

```
// Imports needed
import org.codehaus.plexus.compiler.util.scan.InclusionScanException;
import org.codehaus.plexus.compiler.util.scan.StaleSourceScanner;
import org.codehaus.plexus.compiler.util.scan.mapping.SuffixMapping;
```

```
// At some point of your code
    StaleSourceScanner scanner = new StaleSourceScanner( 0,
Collections.singleton( "**/*.xml" ), Collections.EMPTY_SET );
    scanner.addSourceMapping( new SuffixMapping( ".xml", ".html" ) );
    Set<File> staleFiles = (Set<File>) scanner.getIncludedSources(
this.sourceDirectory, this.targetDirectory );
```

The second parameter to the StaleSourceScanner is the set of includes, while the third parameter is the set of excludes. You must add a source mapping to the scanner (second line). In this case we're telling the scanner what is the extension of the result file (.html) for each source file extension (.xml). Finally we get the stale files as a Set<File> calling the getIncludedSources method, passing as parameters the source and target directories (of type File). The Maven API doesn't support generics, but you may cast it that way if you're using them.
In order to use this API you must include the following dependency in your pom:

```
<dependencies>
  <dependency>
    <groupId>org.codehaus.plexus</groupId>
    <artifactId>plexus-compiler-api</artifactId>
    <version>1.5.1</version>
  </dependency>
</dependencies>
```

Is there a property file for plug-in configuration in Maven 2.0?

No. Maven 2.x no longer supports plug-in configuration via properties files. Instead, in Maven 2.0 you can configure plug-ins directly from command line using the -D arguement, or from the plug-in's POM using the <configuration> element.

How do I determine which POM contains missing transitive dependency?

run "mvn -X"

How do I integrate static (x) html into my Maven site?

You can integrate your static pages in this several steps,

- Put your static pages in the resources directory, ${basedir}/src/site/resources.
- Create your site.xml and put it in ${basedir}/src/site. An example below:
- <project name="Maven War Plugin">
  <bannerLeft>
    <name>Maven War Plugin</name>

```
      <src>http://maven.apache.org/images/apache-maven-project.png</src>
      <href>http://maven.apache.org/</href>
    </bannerLeft>
    <bannerRight>
      <src>http://maven.apache.org/images/maven-small.gif</src>
    </bannerRight>
    <body>
      <links>
        <item name="Maven 2" xhref="http://maven.apache.org/maven2/"/>
      </links>

      <menu name="Overview">
        <item name="Introduction" xhref="introduction.html"/>
        <item name="How to Use" xhref="howto.html"/>
      </menu>
      ${reports}
    </body>
  </project>
```

Link the static pages by modifying the <menu> section, create items and map it with the filename of the static pages.

```
<menu name="Overview">
  <item name="Introduction" xhref="introduction.html"/>
  <item name="How to Use" xhref="howto.html"/>
  <item name="<put-name-here>" xhref="<filename-of-the-static-page>"/>
</menu>
```

How do I run an ant task twice, against two different phases?

You can specify multiple execution elements under the executions tag, giving each a different id and binding them at different phases.

```
<plugin>
      <artifactId>maven-antrun-plugin</artifactId>
      <executions>
        <execution>
          * <id>one</id>*
          <phase>generate-sources</phase>
          <configuration>
            <tasks>
              <echo message="generate-sources!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"/>
            </tasks>
          </configuration>
          <goals>
            <goal>run</goal>
```

```
      </goals>
    </execution>

    <execution>
       *<id>two</id>*
      <phase>package</phase>
      <configuration>
        <tasks>
          * <echo message="package!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"/>*

        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
 </plugin>
```

Can a profile inherit the configuration of a "sibling" profile?

No. Profiles merge when their ID's match - so you can inherit them from a parent POM (but you can't inherit profiles from the same POM).

Inheritence and Interpolation, Plugins and Lifecycle, POM

How do I invoke the "maven dist" function from Maven 1.0, in Maven 2.0?

mvn assembly:assembly

See the Assembly Plugin documentation for more details.

General, Plugins and Lifecycle

How do I specify which output folders the Eclipse plugin puts into the .classpath file?

```
<build>
...
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-eclipse-plugin</artifactId>
        <configuration>
```

```
        <outputDirectory>target-eclipse</outputDirectory>
      </configuration>
    </plugin>
  </plugins>
 </pluginManagement>
...
</build>
```

What is a Mojo?

A mojo is a Maven plain Old Java Object. Each mojo is an executable goal in Maven, and a plugin is a distribution of one or more related mojos.

How to produce execution debug output or error messages?

You could call Maven with *-X* parameter or *-e* parameter. For more information, run:

mvn --help

Maven compiles my test classes but doesn't run them?

Tests are run by the surefire plugin. The surefire plugin can be configured to run certain test classes and you may have unintentionally done so by specifying a value to ${test}. Check your settings.xml and pom.xml for a property named "test" which would like this:

```
 ...
 <properties>
   <property>
     <name>test</name>
     <value>some-value</value>
   </property>
</properties>
 ...
```

Or

```
 ...
 <properties>
   <test>some-value</test>
</properties>
 ...
```

How do I include tools.jar in my dependencies?

The following code includes tools.jar on Sun JDKs (it is already included in the runtime for Mac OS X and some free JDKs).

```
...
  <profiles>
    <profile>
      <id>default-tools.jar</id>
      <activation>
        <property>
          <name>java.vendor</name>
          <value>Sun Microsystems Inc.</value>
        </property>
      </activation>
      <dependencies>
        <dependency>
          <groupId>com.sun</groupId>
          <artifactId>tools</artifactId>
          <version>1.4.2</version>
          <scope>system</scope>
          <systemPath>${java.home}/../lib/tools.jar</systemPath>
        </dependency>
      </dependencies>
    </profile>
  </profiles>
  ...
```

I have a jar that I want to put into my local repository. How can I copy it in?

If you understand the layout of the maven repository, you can copy the jar directly into where it is meant to go. Maven will find this file next time it is run.

If you are not confident about the layout of the maven repository, then you can adapt the following command to load in your jar file, all on one line.

```
mvn install:install-file
  -Dfile=<path-to-file>
  -DgroupId=<group-id>
  -DartifactId=<artifact-id>
  -Dversion=<version>
  -Dpackaging=<packaging>
  -DgeneratePom=true
```

Where: <path-to-file> the path to the file to load
<group-id> the group that the file should be registered under
<artifact-id> the artifact name for the file

<version> the version of the file
<packaging> the packaging of the file e.g. jar

This should load in the file into the maven repository, renaming it as needed.

How do I set up Maven so it will compile with a target and source JVM of my choice?

You must configure the source and target parameters in your pom. For example, to set the source and target JVM to 1.5, you should have in your pom :

```
...
  <build>
  ...
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
       </configuration>
     </plugin>
   </plugins>
  ...
 </build>
  ...
```

How can I use Ant tasks in Maven 2?

There are currently 2 alternatives:

- For use in a plugin written in Java, Beanshell or other Java-like scripting language, you can construct the Ant tasks using the instructions given in the Ant documentation
- If you have very small amounts of Ant script specific to your project, you can use the AntRun plugin.

Maven 2.0 Eclipse Plug-in

Plugins are great in simplifying the life of programmers; it actually reduces the repetitive tasks involved in the programming. In this article our experts will show you the steps required to download and install the Maven Plugin with your eclipse IDE.

Why Maven with Eclipse

Eclipse is an industry leader in IDE market, it is used very extensively in developing projects all around the world. Similarly, Maven is a high-level, intelligent project management, build and deployment tool provided by Apache's software foundation group. Maven deals with application development lifecycle management.

Maven–Eclipse Integration makes the development, testing, packaging and deployment process easy and fast. Maven Integration for Eclipse provides a tight integration for Maven into the IDE and avails the following features:

· It helps to launch Maven builds from within Eclipse

· It avails the dependency management for Eclipse build path based on Maven's pom.xml

· It resolves Maven dependencies from the Eclipse workspace withoutinstalling to local Maven repository

· It avails an automatic downloading of the required dependencies from the remote Maven repositories

· It provides wizards for creating new Maven projects, pom.xml or to enable Maven support on plain Java project

· It helps to search quickly for dependencies in Maven remote repositories

· It quickly fixes in the Java editor for looking up required dependencies/jars by the class or package name.

What do you Need?

1. Get the Eclipse Development Environment :
In this tutorial we are using the eclipse-SDK-3.3-win32, which can be downloaded fromhttp://www.eclipse.org/downloads/
2. Get Maven-eclipse-plugin-plugin :
It is available at http://mevenide.codehaus.org/maven-eclipse-plugin-plugin/

Download and Install Eclipse
First download and install the eclipse plugin on your development machine then proceed with the installation process of the eclipse-maven plugin.

A Maven 2.0 Repository: An Introduction

Maven repository Types:

- **Public remote external repository:** This public external repository exists at ibiblio.org and maven synchronizes with this repository.
- **Private remote internal repository:** We set up this repository and make changes in the maven's pom.xml or settings.xml file to use this repository.
- **Local repository:** This repository is maintained by the developer and stays on the developer's machine. It is synchronous to the maven repository defined in the settings.xml file that exists in the .m2 directory at its standard location i.e. **C:\Documents and Settings\Administrator.** If no private internal repository is setup and not listed in the pom.xml or in the setting.xml then the local repository exists on the developer's machine is synchronized with the public maven repository at ibiblio.org.

Advantages of having an internal private repository :

- Reduces conflicts among likelihood versions.
- To build first time it requires less manual intervention.
- Rather than having several separate independent libraries it provides a single central reference repository for all the dependent software libraries.
- It quickly builds the project while using an internal repository as maven artifacts are retrieved from the intranet server rather than retrieving from the server on internet.

Use cases for maven repository:

- It creates two sub-repository inside the internal repository.
    - Downloads ibiblio-cache from ibiblio for artifacts and make it available publically. This synchronizes with external repository from ibiblio.
    - internal-maven-repository**:** used for internal artifacts of an organization. It contains unique artifacts for the organization and is not synchronized with any repository.
    - Alternatively, another sub-repository that is not at ibiblio can be created for artifacts. This does not synchronize with any external repository.
    - Browse the remote repository by using a web browser.
    - Search the artifacts in the repository.
    - Download code from version control and make changes in settings.xml to point to the internal repository and build without any manual intervention.
    - Install new version of the artifacts.
    - Import artifacts into the repository in bulk.
    - Export artifacts from the repository in bulk.
    - Setup the task to backup the repository automatically.

Criteria for choosing a maven repository implementation: In ideal condition a maven repository implementation should be:

- Free and open source
- Provide admin tools
- Easy to setup and use
- Provide backup facility
- Able to create, edit and delete sub repositories.
- Anonymous read only access and also access control facility.
- Deployable in any standard web server such as Tomcat or Apache.
- Issue tracker, forums and other independent source of information.
- Active community developers make the product enhanced and bugs fixed.
- Bulk import/export facility to move groups of artifacts into the repository and out of the repository.
- Provide a repository browser**:** should be a web browser instead of the desktop application.

Shifting from Apache Ant to Maven

Maven is entirely a different creature from Ant. Ant is simply a toolbox whereas Maven is about the application of patterns in order to achieve an infrastructure which displays the characteristics of visibility, reusability, maintainability, and comprehensibility. It is wrong to consider Maven as a build tool and just a replacement for Ant.

Ant Vs Maven

There is nothing that Maven does that Ant cannot do. Ant gives the ultimate power and flexibility in build and deployment to the developer. But Maven adds a layer of abstraction above Ant (and uses Jelly). Maven can be used to build any Java application. Today JEE build and deployment has become much standardized. Every enterprise has some variations, but in general it is all the same: deploying EARs, WARs, and EJB-JARs. Maven captures this intelligence and lets you achieve the build and deployment in about 5-6 lines of Maven script compared to dozens of lines in an Ant build script.

Ant lets you do any variations you want, but requires a lot of scripting. Maven on the other hand mandates certain directories and file names, but it provides plugins to make life easier. The restriction imposed by Maven is that only one artifact is generated per project (A project in Maven terminology is a folder with a project.xml file in it). A Maven project can have sub projects. Each sub project can build its own artifact. The topmost project can aggregate the artifacts into a larger one. This is synonymous to jars and wars put together to form an EAR. Maven also provides inheritance in projects.

Maven : Stealing the show

Maven simplifies build enormously by imposing certain fixed file names and acceptable restrictions like one artifact per project. Artifacts are treated as files on your computer by the build script. Maven hides the fact that everything is a file and forces you to think and script to create a deployable artifact such as an EAR. Artifact has a dependency on a particular version of a third party library residing in a shared remote (or local) enterprise repository, and then publish your library into the repository as well for others to use. Hence there are no more classpath issues. No more mismatch in libraries. It also gives the power to embed even the Ant scripts within Maven scripts if absolutely essential.

Maven 2.0: Features

Maven is a high-level, intelligent project management, build and deployment tool provided by Apache's software foundation group. Maven deals with application development lifecycle management. Maven was originally developed to manage and to minimize the complexities of building the Jakarta Turbine project. But its powerful capabilities have made it a core entity of the Apache Software Foundation projects. Actually, for a long time there was a need to standardized project development lifecycle management system and Maven has emerged as a perfect option that meets the needs. Maven has become the de- facto build system in many open source initiatives and it is rapidly being adopted by many software development organizations.

Maven was borne of the very practical desire to make several projects at Apache work in a consistence manner. So that developers could freely move between these projects, knowing clearly how they all worked by understanding how one of them worked.

If a developer spent time understanding how one project built it was intended that they would not have to go through this process again when they moved on to the next project. The same idea extends to testing, generating documentation, generating metrics and reports, testing and deploying. All projects share enough of the same characteristics, an understanding of which Maven tries to harness in its general approach to project management.

On a very high level all projects need to be built, tested, packaged, documented and deployed. There occurs infinite variation in each of the above mentioned steps, but these variation still occur within the confines of a well defined path and it is this path that Maven attempts to present to everyone in a clear way. The easiest way to make a path clear is to provide people with a set of patterns that can be shared by anyone involved in a project.

The key benefit of this approach is that developers can follow one consistent build lifecycle management process without having to reinvent such processes again. Ultimately this makes developers more productive, agile, disciplined, and focused

on the work at hand rather than spending time and effort doing grunt work understanding, developing, and configuring yet another non-standard build system.

Maven: Features

1. **Portable:** Maven is portable in nature because it includes:
    - Building configuration using maven are portable to another machine, developer and architecture without any effort
    - **Non trivial:** Maven is non trivial because all file references need to be relative, environment must be completely controlled and independent from any specific file system.
2. **Technology:** Maven is a simple core concept that is activated through IoC container (Plexus). Everything is done in maven through plugins and every plugin works in isolation (ClassLoader). Plugings are downloaded from a plugin-repository on demand.

Maven's Objectives:
The primary goal of maven is to allow the developers to comprehend the complete state of a project in the shortest time by using easy build process, uniform building system, quality project management information (such as change Log, cross-reference, mailing lists, dependencies, unit test reports, test coverage reports and many more), guidelines for best practices and transparent migration to new features. To achieve to this goal Maven attempts to deal with several areas like:

- It makes the build process easy
- Provides a uniform building system
- Provides quality related project information
- Provides guidelines related to development to meet the best goal.
- Allows transparent migration to new features.

Introduction to Maven 2.0

Maven2 is an Open Source build tool that made the revolution in the area of building projects. Like the build systems as "**make"** and "**ant"** it is not a language to combine the build components but it is a build lifecycle framework. A development team does not require much time to automate the project's build infrastructure since maven uses a standard directory layout and a default build lifecycle. Different development teams, under a common roof can set-up the way to work as standards in a very short time. This results in the automated build infrastructure in more stable state. On the other hand, since most of the setups are simple and reusable immediately in all the projects using maven therefore many important reports, checks, build and test animation are added to all the projects. Which was not possible without maven because of the heavy cost of every project setup.

Maven 2.0 was first released on 19 October 2005 and it is not backward compatible

with the plugins and the projects of maven1. In December 2005, a lot of plugins were added to maven but not all plugins that exists for maven1 are ported yet. Maven 2 is expected to stabilize quickly with most of the Open Source technologies. People are introduced to use maven as the core build system for Java development in one project and a multi-project environment. After a little knowledge about the maven, developers are able to setup a new project with maven and also become aware of the default maven project structure. Developers are easily enabled to configure maven and its plugins for a project. Developers enable common settings for maven and its plugins over multiple projects, how to generate, distribute and deploy products and reports with maven so that they can use repositories to set up a company repository. Developers can also know about the most important plugins about how to install, configure and use them, just to look for other plugins to evaluate them so that they can be integrated in their work environment.

Maven is the standard way to build projects and it also provides various other characters like clearing the definition of the project, ways to share jars across projects. It also provides the easy way to publish project information (OOS).

Originally maven was designed to simplify the building processes in the Jakarta Turbine project. Several projects were there containing their own slightly different Ant build files and JARs were checked into CVS. An apache group's tool that can build the projects, publish project information, defines what the project consists of and that can share JARs across several projects. The result of all these requirement was the maven tool that builds and manages the java-based-project.

Why maven is a great build tool? how does it differ from other Build tools?

Tell me more about Profiles and Nodes in Maven?

Tell me more about local repositories?

How did you configured local repositories in different environment (Development, Testing , Production etc)?

What is Transcend Dependencies in maven 2?

Did you write plugins in maven? if so what are they?

Why a matrix report is required during a new release? How does this benefit QA Team?

What are pre-scripts and post-scripts in maven? Illustrate with an example?

What are the checklists for artifacts ? and what are the checklists for source code artifact?

Tell me the experience about Static Analysis Code?

Reference:

http://www.javabeat.net