Maven

Maven is a powerful *project management tool* that is based on POM (project object model). It is used for projects build, dependency and documentation.

Understanding the problem without Maven

There are many problems that we face during the project development. They are discussed below:

1) Adding set of Jars in each project: In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

2) Creating the right project structure: We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

3) Building and Deploying the project: We must have to build and deploy the project so that it may work.

How to install Maven on windows

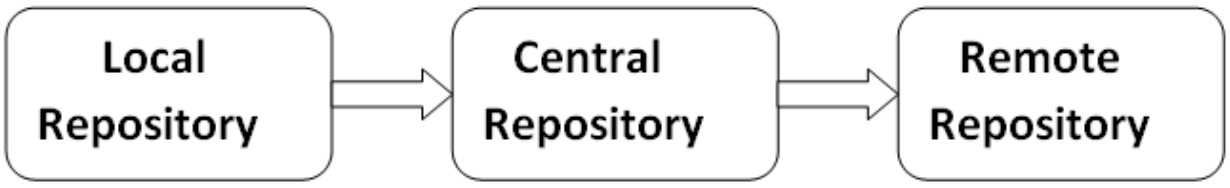
You can download and install maven on windows, linux and MAC OS platforms. Here, we are going to learn how to install maven on windows OS.

To install maven on windows, you need to perform following steps:

1. Download maven and extract it
2. Add JAVA_HOME and MAVEN_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

Maven searches for the dependencies in the following order:

Local repository then **Central repository** then **Remote repository**.



Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

Maven **central repository** is located on the web. It has been created by the apache maven community itself.

The path of central repository is: http://repo1.maven.org/maven2/.

Remote repository

Apart from central repository, you may have needed artifacts deployed on other remote locations. For example, in your corporate office there may be projects or modules specific to organization only. In this cases, organization can create remote repository and deploy these **private artifacts**. This remote repository will be accessible only inside organization.

You can configure a remote repository in the POM file or super POM file in remote repository itself.

```
<repositories>
   <repository>
       <id>org.source.repo</id>
       <url>http://maven.orgName.com/maven2/</url>
   </repository>
</repositories>
```

Maven Settings File

- You can configure location of local repository
- Remote repository username and password

Maven Directory Structure

Maven has a standard directory structure. If you follow that directory structure for your project, you do not need to specify the directories of your source code, test code etc. in your POM file.

Here are the most important directories:

```
- src
  - main
    - java
    - resources
    - webapp
  - test
    - java
```

```
    - resources

- target
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.javatpoint.application1</groupId>
  <artifactId>my-application1</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

How to build the maven project or how to package maven project?

The **mvn package** command completes the build life cycle of the maven project such as:

Phase	Handles	Description

prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

Maven clean goal (clean:clean) is bound to the *clean* phase in the clean lifecycle. Its **clean:cleangoal** deletes the output of a build by deleting the build directory

Spring AOP example

```
public  class Operation{
   public void msg(){System.out.println("msg method invoked");}
   public int m(){System.out.println("m method invoked");return 2;}
   public int k(){System.out.println("k method invoked");return 3;}
}

public class TrackOperation{
   public void myadvice(JoinPoint jp)//it is advice
   {
      System.out.println("additional concern");
      System.out.println("Method Signature: "  + jp.getSignature());
   }
```

```
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

<aop:aspectj-autoproxy />

<bean id="opBean" class="com.Operation">   </bean>
<bean id="trackAspect" class="com.TrackOperation"></bean>
```

//before

```
<aop:config>
  <aop:aspect id="myaspect" ref="trackAspect" >
    <!-- @Before -->
    <aop:pointcut id="pointCutBefore"   expression="execution(* com.Operation.*(..))" />
    <aop:before method="myadvice" pointcut-ref="pointCutBefore" />
  </aop:aspect>
</aop:config>
```

//after

```
<aop:config>
  <aop:aspect id="myaspect" ref="trackAspect" >
    <!-- @After -->
    <aop:pointcut id="pointCutAfter"    expression="execution(* com.Operation.*(..))" />
    <aop:after method="myadvice" pointcut-ref="pointCutAfter" />
  </aop:aspect>
</aop:config>
```

//after returning

```
<aop:config>
  <aop:aspect id="myaspect" ref="trackAspect" >
```

```
    <!-- @AfterReturning -->
    <aop:pointcut id="pointCutAfterReturning"   expression="execution(* com.Operation.*(..))"
 />
    <aop:after-returning method="myadvice" returning="result" pointcut-
ref="pointCutAfterReturning" />
  </aop:aspect>
</aop:config>
```

//around

```
<aop:config>
  <aop:aspect id="myaspect" ref="trackAspect" >
    <!-- @Around -->
    <aop:pointcut id="pointCutAround"   expression="execution(* com.javatpoint.Operation.*(.
.))" />
    <aop:around method="myadvice" pointcut-ref="pointCutAround" />
  </aop:aspect>
</aop:config>



</beans>
```

```
public class Test{
   public static void main(String[] args){
      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.x
ml");
      Operation e = (Operation) context.getBean("opBean");
      System.out.println("calling msg...");
      e.msg();
      System.out.println("calling m...");
      e.m();
      System.out.println("calling k...");
      e.k();
   }
}
```

//after throwing

```
public  class Operation{
   public void validate(int age)throws Exception{
   if(age<18){
      throw new ArithmeticException("Not valid age");
   }
   else{
      System.out.println("Thanks for vote");
   }
   }

}

public class TrackOperation{

   public void myadvice(JoinPoint jp,Throwable error)//it is advice
   {
      System.out.println("additional concern");
      System.out.println("Method Signature: "  + jp.getSignature());
      System.out.println("Exception is: "+error);
      System.out.println("end of after throwing advice...");
   }
}

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/aop
   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">
<aop:aspectj-autoproxy />
<bean id="opBean" class="com.javatpoint.Operation">   </bean>
<bean id="trackAspect" class="com.javatpoint.TrackOperation"></bean>

<aop:config>
  <aop:aspect id="myaspect" ref="trackAspect" >
    <!-- @AfterThrowing -->
```

```
    <aop:pointcut id="pointCutAfterThrowing"    expression="execution(* com.javatpoint.Operation.*(..))" />
    <aop:after-throwing method="myadvice" throwing="error" pointcut-ref="pointCutAfterThrowing" />
  </aop:aspect>
</aop:config>

</beans>

public class Test{
   public static void main(String[] args){
      ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
      Operation op = (Operation) context.getBean("opBean");
      System.out.println("calling validate...");
      try{
         op.validate(19);
      }catch(Exception e){System.out.println(e);}
      System.out.println("calling validate again...");

      try{
         op.validate(11);
      }catch(Exception e){System.out.println(e);}
   }
}
```