Maven

Ingeniería de Software
Laboratorio

L. en C.C. Miguel Angel Piña Avelino

Maven

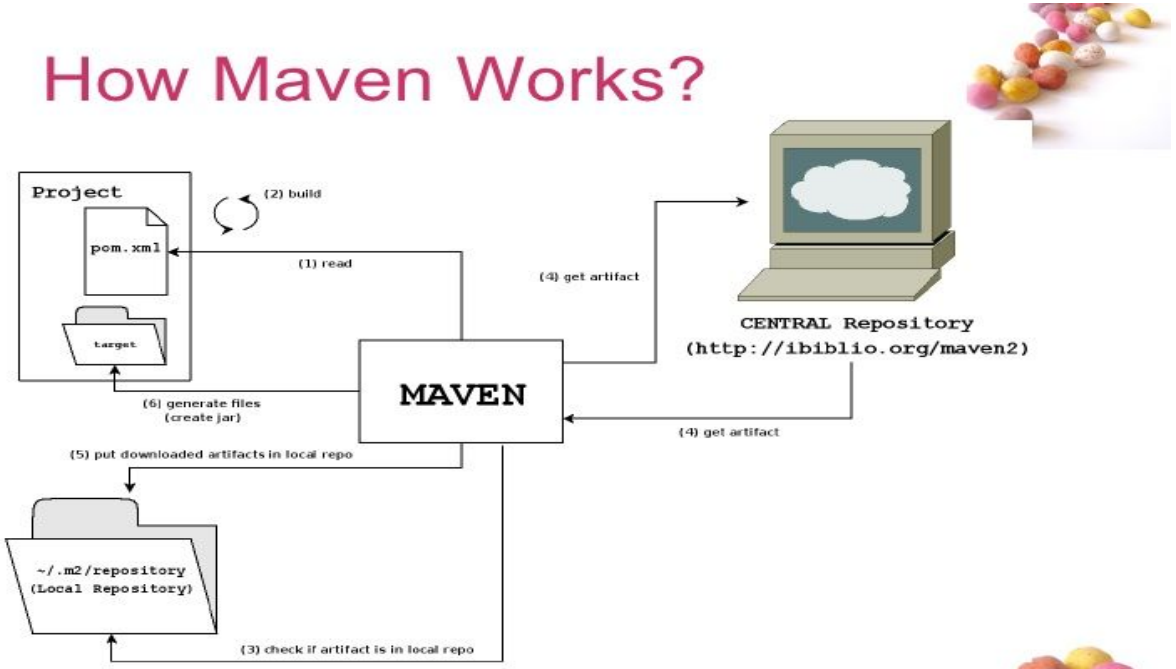
¿Qué es Maven?

- Herramienta para la gestión y construcción de proyectos de Java
- Tiene un funcionamiento similar a Ant
- Trabaja sobre configuraciones en XML
- Proyecto de nivel superior de la Apache Software Foundation
- Utiliza Project Object Models (POM) para describir el proyecto a construir
- Listo para usarse desde la red

Filosofía de Maven

- Estandarización de las construcciones
- Convención sobre configuración
- Restringe ampliamente la variabilidad de construir proyectos de software
- Ideal para proyectos nuevos
- Los proyectos complejos y existentes podrían no ser adaptables a Maven

¿Cómo funciona Maven?



- Image Credit--
http://blogs.exist.com/oching/wp-content/uploads/image/how_maven_works.png

Ciclo de vida

- **Compile**: Genera archivos .class desde los fuente.
- **Test**: Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso de ejecución de Maven si las pruebas fallan.
- **Package**: Genera archivos *.jar* o *.war* con los .class compilados.
- **Install**: Copia el fichero *.jar* (*.war*) a una carpeta donde Maven comparte todos los *.jar* generados. Esto permite que un mismo proyecto esté disponible para otros.
- **Deploy**: Copia el *.jar* a un servidor remoto, poniéndolo disponible para que sea ejecutado.

Ciclo de vida

Cuando se ejecuta alguno de los comandos anteriores, por ejemplo, mvn install, maven irá verificando que todas las fases previas a el se hayan cumplido.

Maven también tiene disponibles algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida estándar.

Metas adicionales

- **Clean**: Elimina todos los .class y .jar generados.
- **Assembly**: Genera un archivo .zip con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero .xml que se debe incluir en ese zip.
- **Site**: Genera un sitio web con la información de nuestro proyecto.
- **Site-deploy**: Sube el sitio web anterior.

Creando un primer proyecto en Maven

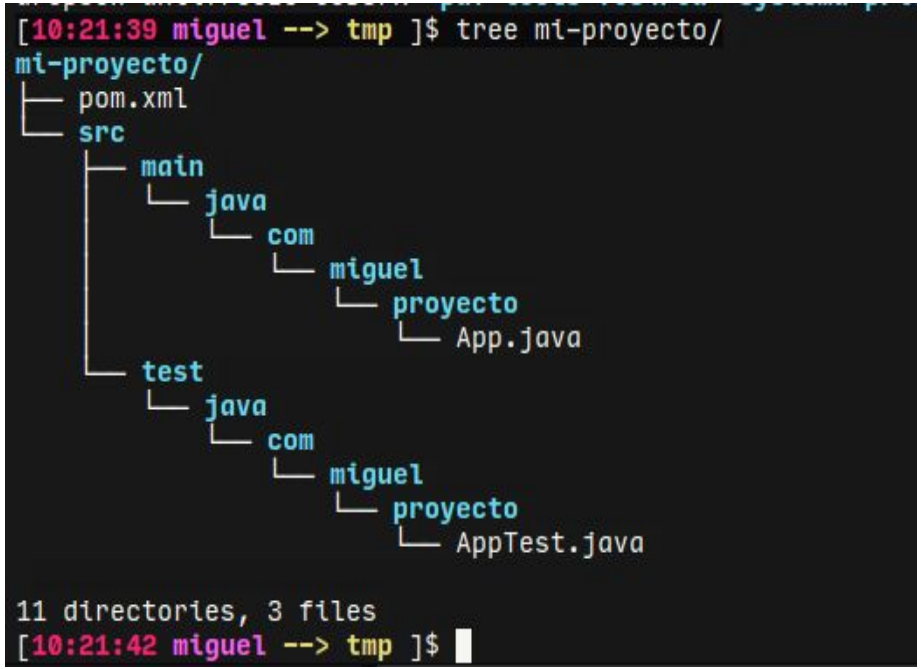
Primer proyecto

Para crear un proyecto Maven simple, basta con ejecutar la siguiente instrucción (previamente con maven instalado).

```
$ mvn archetype:generate -DgroupId="com.miguel.proyecto" \
-DartifactId="mi-proyecto" -Dversion="0.0.1"
```

Primer proyecto

El código anterior va a crear un proyecto de maven con una serie de paquetes por defecto para diferenciarlo de otros proyectos *com.miguel.proyecto*, así como tener por nombre *mi-proyecto* y empezar con la versión *0.0.1*.



Primer proyecto

Podemos editarlo, compilarlo y jugar un poco con el proyecto. Para que pueda agregar la referencia de donde se encuentra el archivo principal (main) dentro del jar, esto una vez que sea empaquetado, hay que agregar la siguiente instrucción en el pom.xml, justo después de las dependencias del proyecto:

Primer proyecto

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

Primer proyecto

Compilamos con:

```
mvn clean install
```

Ejecutamos con:

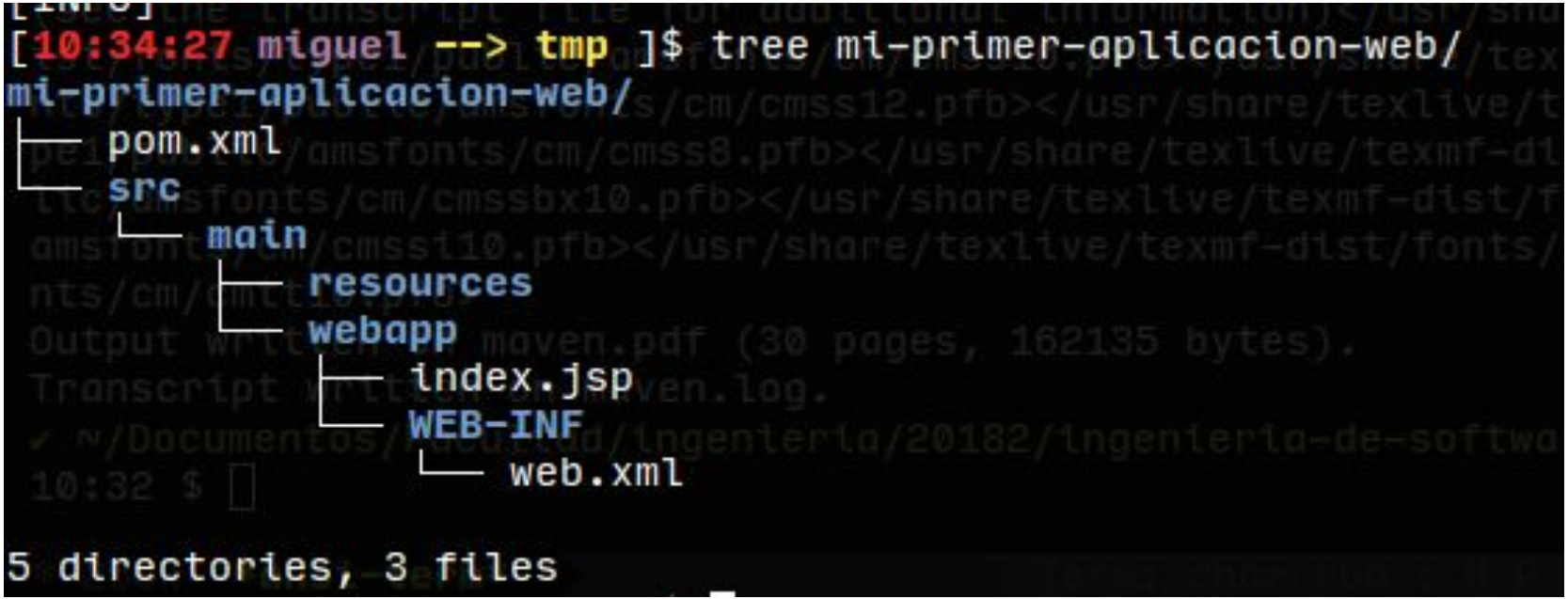
```
java -jar target/mi-proyecto-0.0.1.jar
```

Primer proyecto web

De forma similar al ejemplo anterior, para construir un proyecto, podemos hacer uso de las herramientas que maven nos provee para construir una aplicación web.

```
mvn archetype:generate -DgroupId="com.miguel.proyecto" \
    -DartifactId="mi-primer-aplicacion-web" \
    -DarchetypeArtifactId=maven-archetype-webapp \
    -DinteractiveMode=false
```

Primer proyecto web



Primer proyecto web

Para tener un servidor http integrado con la aplicación, basta con agregar el siguiente snippet al archivo pom.xml

```
<build>
  <finalName>mi-primer-aplicacion-web</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration><source>1.8</source><target>1.8</target></configuration>
    </plugin>
```

Primer proyecto web

```
<!-- For Maven Tomcat Plugin -->
     <plugin>
       <groupId>org.apache.tomcat.maven</groupId>
       <artifactId>tomcat7-maven-plugin</artifactId>
       <version>2.2</version>
       <configuration>
          <path>/mi-primer-aplicacion-web</path>
       </configuration>
     </plugin>
  </plugins>
</build>
```

Primer proyecto web

Compilamos con:

```
mvn clean install
```

Ejecutamos con:

```
mvn tomcat7:run
```

En un navegador, accedemos a:

```
http://localhost:8080/mi-primer-aplicacion-web/
```

Agregando soporte a JSF

Aunque ya tenemos un proyecto de maven pensando para usarse en la web, aún nos falta agregarle ciertas características para crear aplicaciones dinámicas en internet.

Para ello, usaremos JSF. JSF (*Java Server Faces*) es un framework MVC (Modelo-Vista-Controlador) construido sobre la **API** de *Servlets*.

Este framework proporciona componentes a través de etiquetas XHTML a través de Facelets, que son componentes web definidos como plantillas.

Agregando soporte a JSF

Para usar JSF en nuestro proyecto Maven, debemos añadir las siguientes dependencias:

```
   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
```

Agregando soporte a JSF

Para usar JSF en nuestro proyecto Maven, debemos añadir las siguientes dependencias:

```
   <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.2.18</version>
      <type>jar</type>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.2.18</version>
    </dependency>
```

Agregando soporte a JSF

Lo anterior nos permite tener disponible en el PATH de nuestra aplicación las bibliotecas de JSF. Con ello ya podemos comenzar a agregar código en Java y en XHTML.

Consideremos en agregar un Java Bean que va a contener el nombre. Lo vamos a utilizar para comunicar información entre páginas.

También vamos a tener una página llamada hello.xhtml y otra nombrada welcome.xhtml.

main/java/com/miguel/app/HelloBean.java

```
package com.miguel.app;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import java.io.Serializable;
@ManagedBean
@SessionScoped
public class HelloBean implements Serializable {
     private static final long serialVersionUID = 1L;
     private String name;
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
}
```

main/webapp/hello.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Hola Mundo</title>
    </h:head>
    <h:body>
     <h2>Ejemplo Hola mundo - hello.xhtml</h2>
     <h:form>
        <h:inputText value="#{helloBean.name}"></h:inputText>
        <h:commandButton value="Me doy la bienvenida" action="welcome"></h:commandButton>
     </h:form>
    </h:body>
</html>
```

main/webapp/welcome.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">

    <h:head>
     <title>JSF Hello World</title>
    </h:head>
    <h:body bgcolor="white">
     <h2>JSF Ejemplo Hola mundo - welcome.xhtml</h2>
     <h2>Bienvenido #{helloBean.name}</h2>
    </h:body>
</html>
```

Configurando la aplicación

Aunque ya podemos compilar de forma correcta las clases y plantillas anteriores de forma correcta, aún falta configurar la aplicación para que sepa resolver las peticiones que son enviadas a JSF.

Para ello modificaremos el archivo main/webapp/WEB-INF/web.xml. En el indicaremos cuál va a ser el archivo de inicio de la aplicación, así como las extensiones que van a ser resueltas por JSF.

En las siguientes diapositivas está el contenido del archivo web.xml

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

  <display-name>Archetype Created Web Application</display-name>

  <display-name>JavaServerFaces</display-name>
     <context-param>
          <param-name>javax.faces.PROJECT_STAGE</param-name>
          <param-value>Development</param-value>
     </context-param>
     <welcome-file-list>
          <welcome-file>faces/hello.xhtml</welcome-file>
     </welcome-file-list>
```

web.xml

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.faces</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
</web-app>
```

Ejecutando la aplicación web

Compilamos con:

```
mvn clean install
```

Ejecutamos con:

```
mvn tomcat7:run
```

En un navegador, accedemos a:

```
http://localhost:8080/mi-primer-aplicacion-web/
```