



VIEWNEXT
AN IBM SUBSIDIARY

FORMACIÓN EN NUEVAS TECNOLOGÍAS

Gestión Técnica de Proyectos con Maven

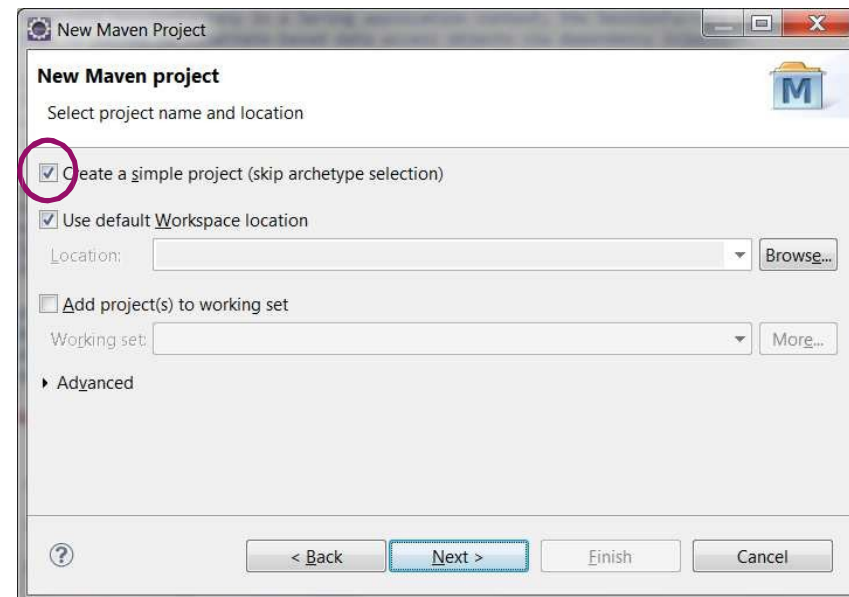
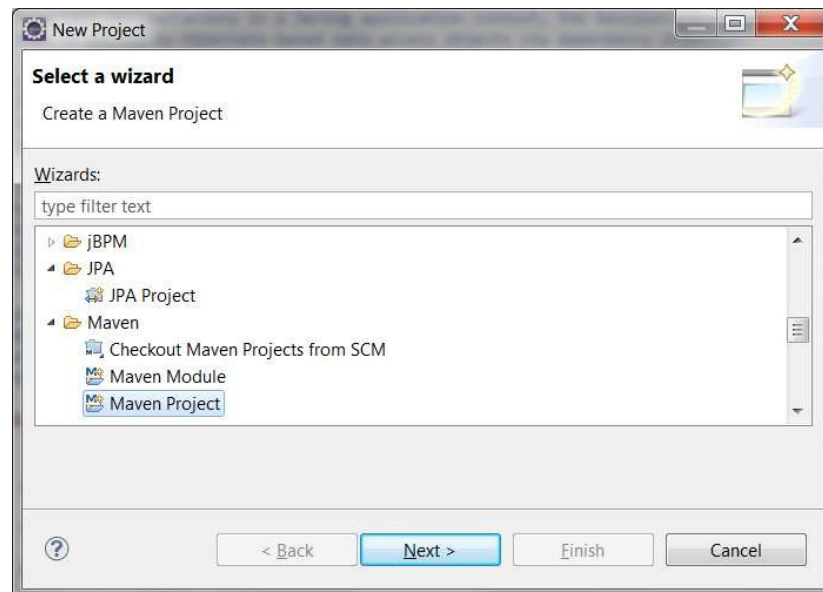
Laboratorio de Maven

- 1.- Creación de proyectos
- 2.- Dependencias y propiedades
- 3.- Comandos
- 4.- Transitividad de dependencias
- 5.- Implementación de los proyectos
- 6.- Despliegue de los proyectos
- 7.- Prueba de los proyectos
- 8.- Módulo parent

1. Creación de proyectos

descripción

El primer paso de este tutorial es crear nuevos proyectos Maven tipo simple. Primero se crea el proyecto base:



1. Creación de proyectos

información

Utilizar los siguientes valores para las "coordenadas":

- Group Id: com.demo.maven **viene a ser el paquete del proyecto**
- Artifact Id: arq-core **nombre del proyecto en eclipse**
- Version: 1.0.0 la **versión del proyecto**
- Packaging: jar

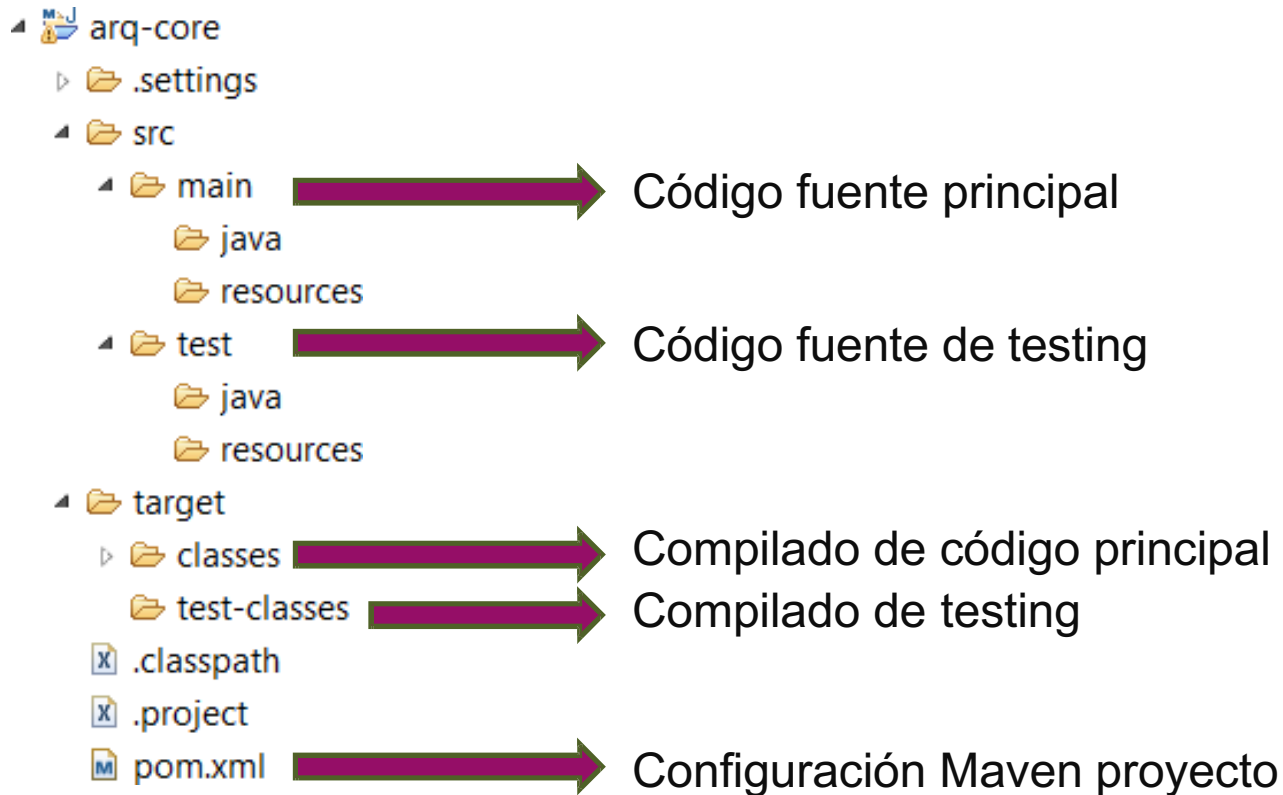
El packaging **jar** indica que es un proyecto Java.

Maven promueve la modularización de los proyectos, estableciendo relaciones entre ellos (dependencias)

1. Creación de proyectos

resultado

En la vista navigator (física), se observan algunas características del proyecto:



1. Creación de proyectos

otros proyectos

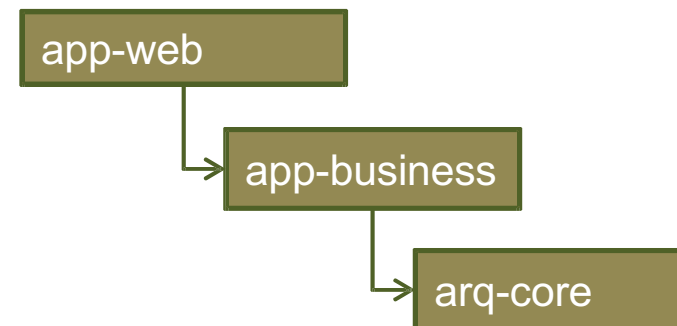
Crear otros dos proyectos:

- Group Id: com.demo.maven
- Artifact Id: app-business
- Version: 1.0.0
- Packaging: jar

- Group Id: com.demo.maven
- Artifact Id: app-web
- Version: 1.0.0
- Packaging: war

El objetivo de cada proyecto es:

- arq-core: Proyecto de componentes comunes.
- app-business: Proyecto de lógica de negocio. Utiliza componentes comunes.
- app-web: Proyecto web. Utiliza lógica de negocio y componentes comunes.



2. dependencias y propiedades

dependencias entre proyectos

Las dependencias entre los proyectos se manejan a través de los pom.xml de Maven, y no directamente con Eclipse. Para ello, se pide lo siguiente:

- Agregar la dependencia de arq-core en el **pom.xml** de app-business. Esto es utilizando la etiqueta <dependencies>, y colocando las "coordenadas" de arq-core:

```
<dependencies>

  <dependency>
    <groupId>com.demo.maven</groupId>
    <artifactId>arq-core</artifactId>
    <version>1.0.0</version>
  </dependency>

</dependencies>
```

2. dependencias y propiedades

dependencias entre proyectos

- Para que esto sea reconocido por Eclipse, debe estar marcada la opción en Project Properties → Maven → Resolve dependencies from Workspace projects. Esto indica que Eclipse intenta buscar el proyecto en el workspace primero, y si no lo encuentra, entonces busca en el repositorio local o remoto.
- De manera similar, agregar la dependencia de app-business en app-web.

2. dependencias y propiedades

configuración de dependencias externas

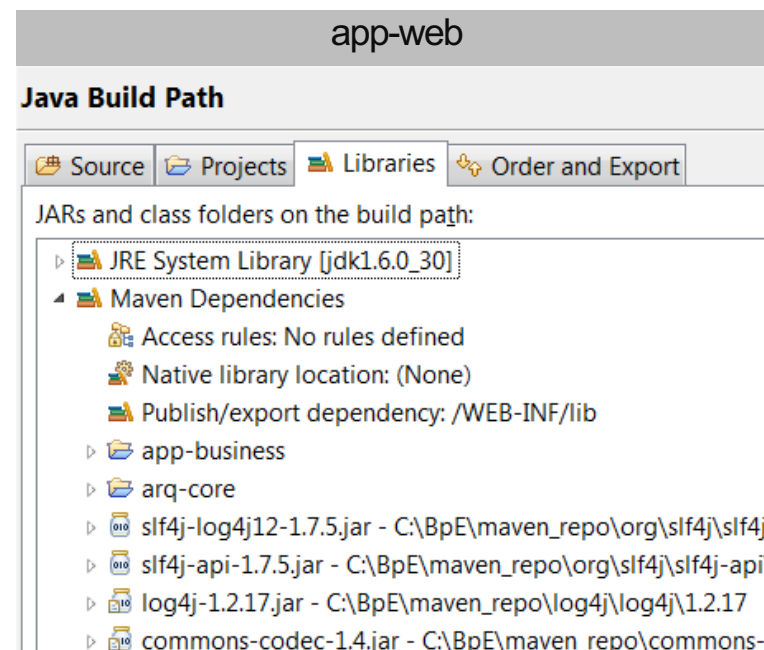
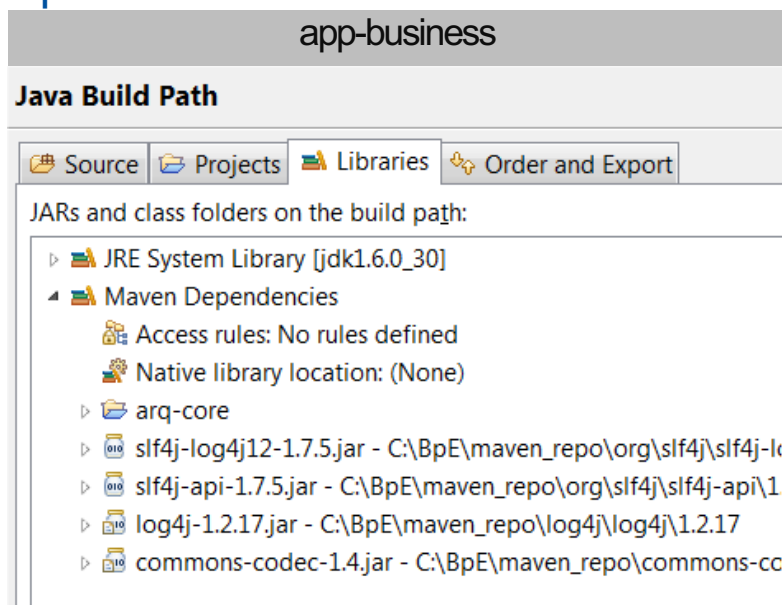
Agregar al proyecto arqu-core las dependencias de las librerías:

- slf4j-log4j12, versión 1.6.6
- commons-codec, versión 1.4

2. dependencias y propiedades

configuración de dependencias externas

En las propiedades de los proyectos app-business y app-web, en Java Build Path → Libraries, se observa la relación directa con los proyectos en Maven Dependencies, a diferencia de las librerías externas, que se referencian desde el repositorio:



2. dependencias y propiedades

conectividad a repositorios externos

En el ejemplo, se utilizan sólo librerías open source, por lo que Maven las puede obtener directamente desde los repositorios internet. Para ello, una forma es configurar dichos repositorios en el **settings.xml**:

```
<profiles>
  <profile>
    <id>development</id>
    <repositories>
      <repository>
        <id>ibiblio</id>
        <url>http://mirrors.ibiblio.org/pub/mirrors/maven2</url>
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>false</enabled></snapshots>
      </repository>
      <repository>
        <id>repo2</id>
        <url>http://repo2.maven.org/maven2</url>
        <releases><enabled>true</enabled></releases>
        <snapshots><enabled>false</enabled></snapshots>
      </repository>
    </repositories>
  </profile>
</profiles>
```

Nota: settings.xml utiliza la misma nomenclatura del pom. Los repositorios se podrían también definir en el pom, aunque no quedarían centralizados.

2. dependencias y propiedades

configuración de propiedades

Para evitar colocar valores "en duro" en la configuración, como por ejemplo el número de versión de una dependencia, se recomienda utilizar **propiedades**. La nomenclatura es la siguiente:

```
<properties>
  <slf4j.version>1.6.6</slf4j.version>
  ...
</properties>

...
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>${slf4j.version}</version>
</dependency>
```

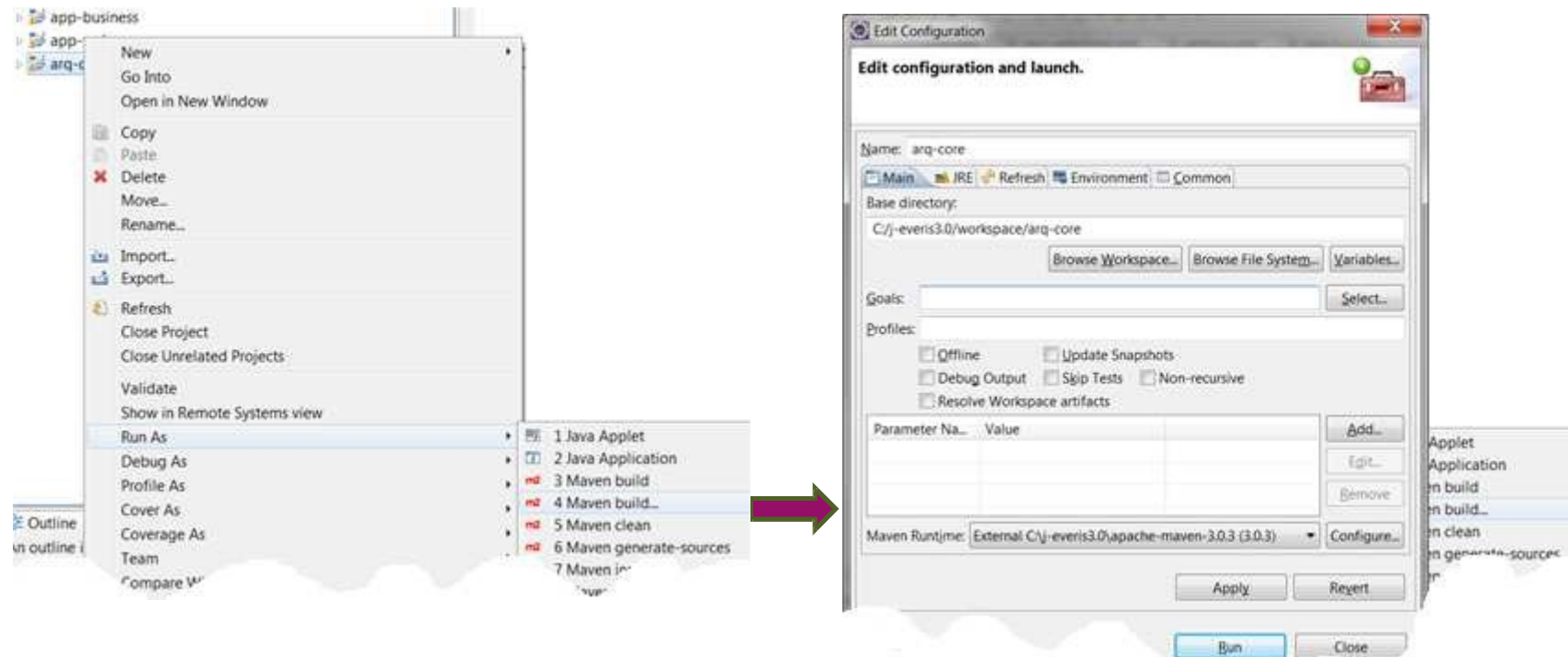
Etiqueta con nombre de la propiedad

Utilización de la propiedad

3. Comandos

ejecución de comandos

Una vez configurado un proyecto con Maven, se pueden ejecutar **comandos** maven para las tareas de desarrollo. El plugin de Eclipse permite ejecutarlos a través del menú de contexto Run As → Maven build...



3. Comandos

sintaxis

- El comando se escribe en "Goals", y permite varios valores separados por espacio. Internamente, equivale a ejecutar la línea de comandos:
`mvn comando`
- El resultado de todas las ejecuciones de comandos de Maven queda en la carpeta "target" del proyecto. Si se quiere limpiar dicha carpeta, el comando es *clean*.

3. Comandos

comando compile

El comando más básico es ***compile***, aunque con el Plugin de Eclipse no es necesario en desarrollo, ya que el propio Eclipse compila el proyecto. Nótese que el código compilado de proyecto queda separado del código de test:



3. Comandos

plugins

Maven funciona internamente con **plugins**, que determinan su comportamiento.

Por ejemplo, "compile" tiene asociado implícitamente el plugin "maven-compiler-plugin", con una configuración default. Si se quiere cambiar dicha configuración, por ejemplo para especificar una versión de java, se debe incluir en el POM.

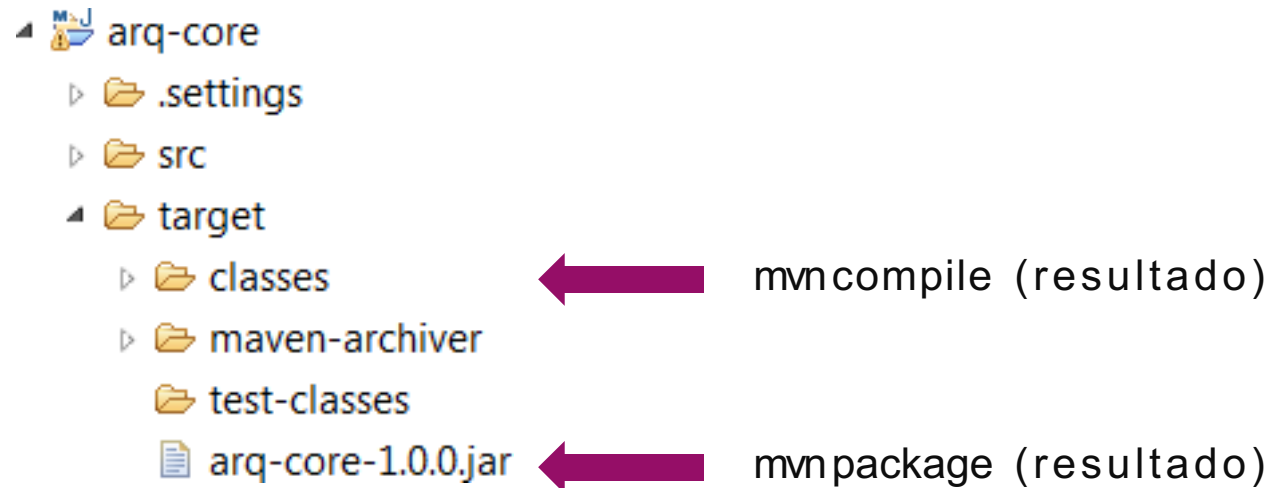
```
<properties>
  <java.version>1.8</java.version>
</properties>
...
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
      </configuration>
    </plugin>
    ...
  </plugins>
  ...
</build>
```

Nota: La lista completa de plugins que utiliza internamente Maven y su configuración se pueden observar en el POM del proyecto, en la pestaña "Effective POM".

3. Comandos

comando package

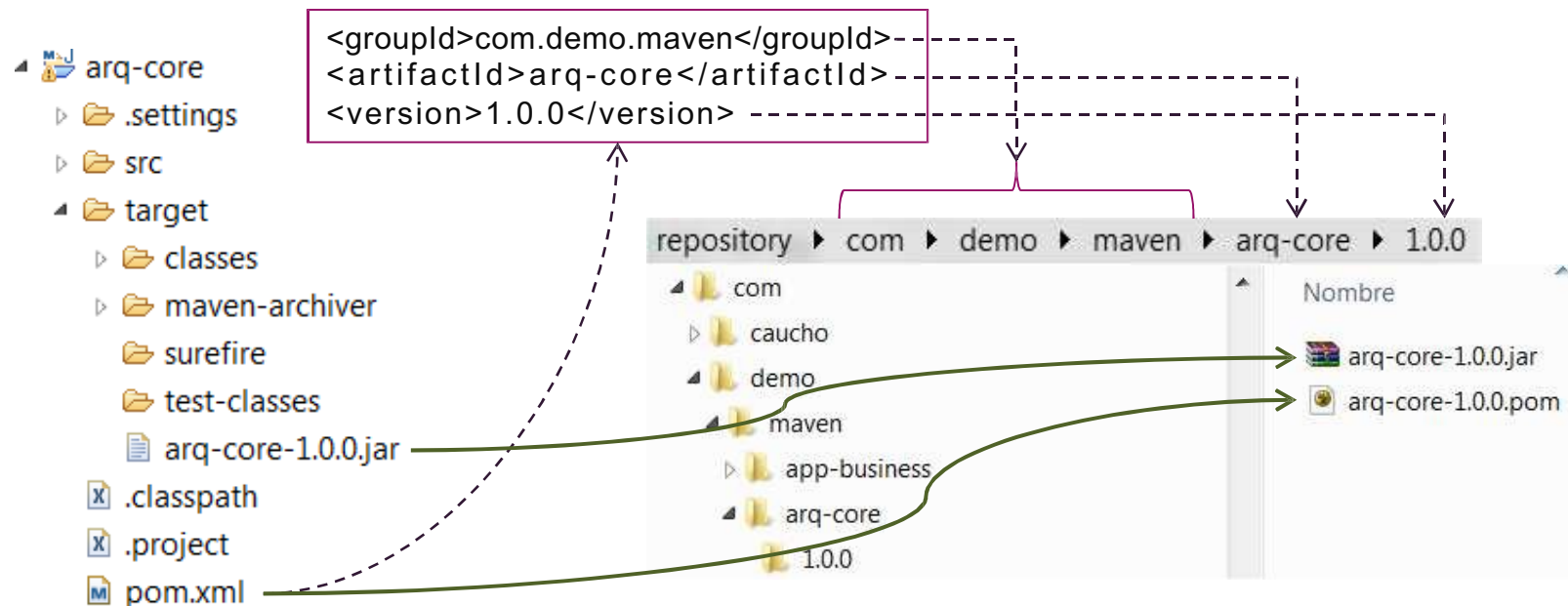
Cuando se ejecuta un comando, Maven internamente ejecuta los anteriores, de acuerdo a un orden predefinido. Por ejemplo, el comando **package**, que paquetiza el proyecto (en un "jar", "war", ...), ejecuta previamente *compile*. Por ejemplo, al ejecutar **package** sobre *arqu-core*, el resultado es:



3. Comandos

Ejecución de comandos maven

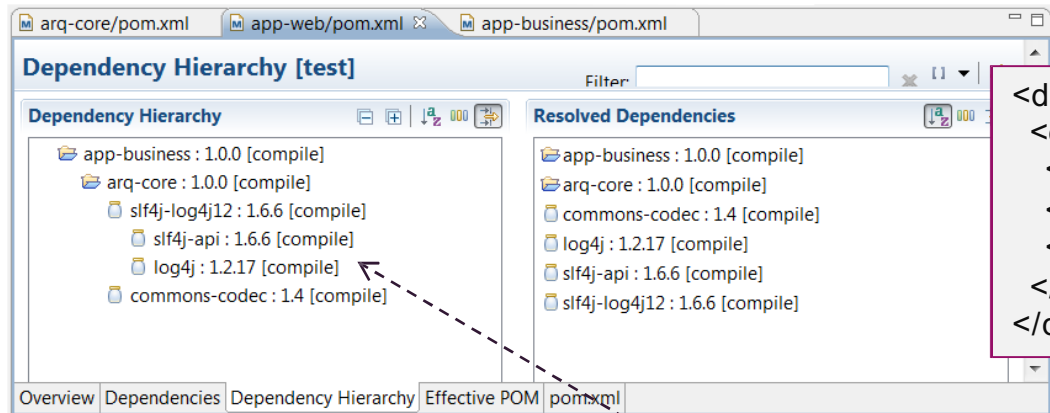
Para instalar la aplicación en el repositorio local, se utiliza el comando **install**, accediendo desde Run As → Maven Install. Ejecuta previamente **compile** y **package**, entre otros, y coloca en el repositorio local definido en el settings.xml la paquetización del proyecto y el pom.xml, con el nombre cambiado a "artifact-version.pom". Si se aplica al proyecto "arq-core":



4. Transitividad de dependencias

descripción

Maven permite también ver las dependencias en forma de árbol, en el editor del POM. Eso sí, para hacerlo no reconoce los proyectos del workspace, por lo que es necesario instalarlos previamente en el repositorio local. Al instalar arq-core y app-business, desde app-web se observa lo siguiente en el POM, pestaña Dependency Hierarchy:



```
<dependencies>  
<dependency>  
  <groupId>com.demo.maven</groupId>  
  <artifactId>app-business</artifactId>  
  <version>1.0.0</version>  
</dependency>  
</dependencies>
```

Esta visualización ilustra cómo maven maneja las dependencias en forma transitiva. Se puede utilizar log4j aunque no esté declarado en su pom, ya que lo "hereda" desde slf4j-log4j12 → arq-core → app-business

4. Transitividad de dependencias

descripción

El hecho de que las dependencias se traten de una forma transitiva puede generar problemas en los scope, en el caso de que una dependencia esté presente en dos proyectos de los que depende el nuestro.

Por defecto prevalece la de versión más actual.

| Direct Scope | vs. Transitive Scope | | | |
|--------------|----------------------|----------|----------|------|
| | compile | provided | runtime | test |
| compile | compile | - | runtime | - |
| provided | provided | - | provided | - |
| runtime | runtime | - | runtime | - |
| test | test | - | test | - |

5. Implementación de los proyectos

arq-core

Una vez configurados los proyectos, se implementa la lógica. En el proyecto arq-core, crear un utilitario HashUtil, con un método estático hash, que dado un número variable de Strings, calcule en formato hexadecimal la codificación SHA-1 de su concatenación. Utilizar DigestUtils.shaHex(...).

```
package com.demo.maven;

import org.apache.commons.codec.digest.DigestUtils;

public class HashUtil {

    public static String hash(String... str) {

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < str.length; i++) {
            sb.append(str[i]);
        }
        return DigestUtils.shaHex(sb.toString());
    }
}
```

Nota: Esto es sólo un ejemplo ilustrativo de lógica en un utilitario, y su implementación no es el foco.

5. Implementación de los proyectos

app-business

En el proyecto app-business, se implementa un servicio PersonaBS, que en el método "hashNombre" recibe el nombre y apellido de una persona, valida que son no null, y utiliza HashUtil para calcular su codificación SHA-1.

```
package com.demo.maven;

public class PersonaBS {

    public String hashNombre(String nombre, String apellido) {

        if (nombre == null) {
            throw new RuntimeException("nombre no debe ser null");
        }

        if (apellido == null) {
            throw new RuntimeException("apellido no debe ser null");
        }


        return HashUtil.hash(nombre, apellido); ←----- Clase de arqu-core
    }
}
```

5. Implementación de los proyectos

app-business

En el proyecto app-business, se implementa un test para el servicio construido. Para ello, debe primero agregarse la dependencia Maven para JUnit, indicando que tiene el scope "test". Con esto, cuando se utiliza desde otros proyectos, esta dependencia no se incluye. Agregar la siguiente dependencia en el POM del proyecto app-business:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.9</version>
  <scope>test</scope>
</dependency>
```



Se puede comprobar en las dependencias de app-web que no está la de junit, dado que utiliza el scope test.

Si para el testing se requieren otras librerías, se pueden agregar también con scope test.

5. Implementación de los proyectos

app-business

El test para el servicio PersonaBS compara el resultado del servicio con el cálculo directo del hash de la concatenación. Implementarlo en el mismo paquete que el servicio. Ejecutarlo y ver su correcto funcionamiento.

```
public class PersonaBSTest {  
  
    private PersonaBS personaBS;  
  
    @Before  
    public void before() {  
        personaBS = new PersonaBS();  
    }  
  
    @Test  
    public void testHashNombre() {  
        String nombre = "John";  
        String apellido = "Doe";  
        String expected = DigestUtils.shaHex(nombre + apellido);  
        String result = personaBS.hashNombre(nombre, apellido);  
        Assert.assertEquals(expected, result);  
    }  
}
```

Clase en carpeta fuente
de testing, src/test/java

Una vez implementado el
test, al ejecutar mvn install
sobre app-business, maven
ejecuta el test utilizando el
plugin **surefire**.

5. Implementación de los proyectos

app-web

A continuación, se implementa una pantalla muy simple que dado el nombre y apellido, obtiene el SHA-1 de su concatenación, utilizando el servicio. Se implementa en forma básica, a través de un servlet que recibe el nombre y apellido por parámetros, y una página JSP que pinta el resultado.

5. Implementación de los proyectos

app-web

Para trabajar con elementos de presentación, que en el servlet y la página incluyen objetos tipo `HttpServletRequest`, `HttpServletResponse`, se debe agregar al POM de app-web la dependencia de la librería API de servlets, teniendo en cuenta que es sólo para compilación, pues el servidor ya la incluye. Por lo tanto se utiliza el scope provided. Agregar la siguiente dependencia al proyecto app-web:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

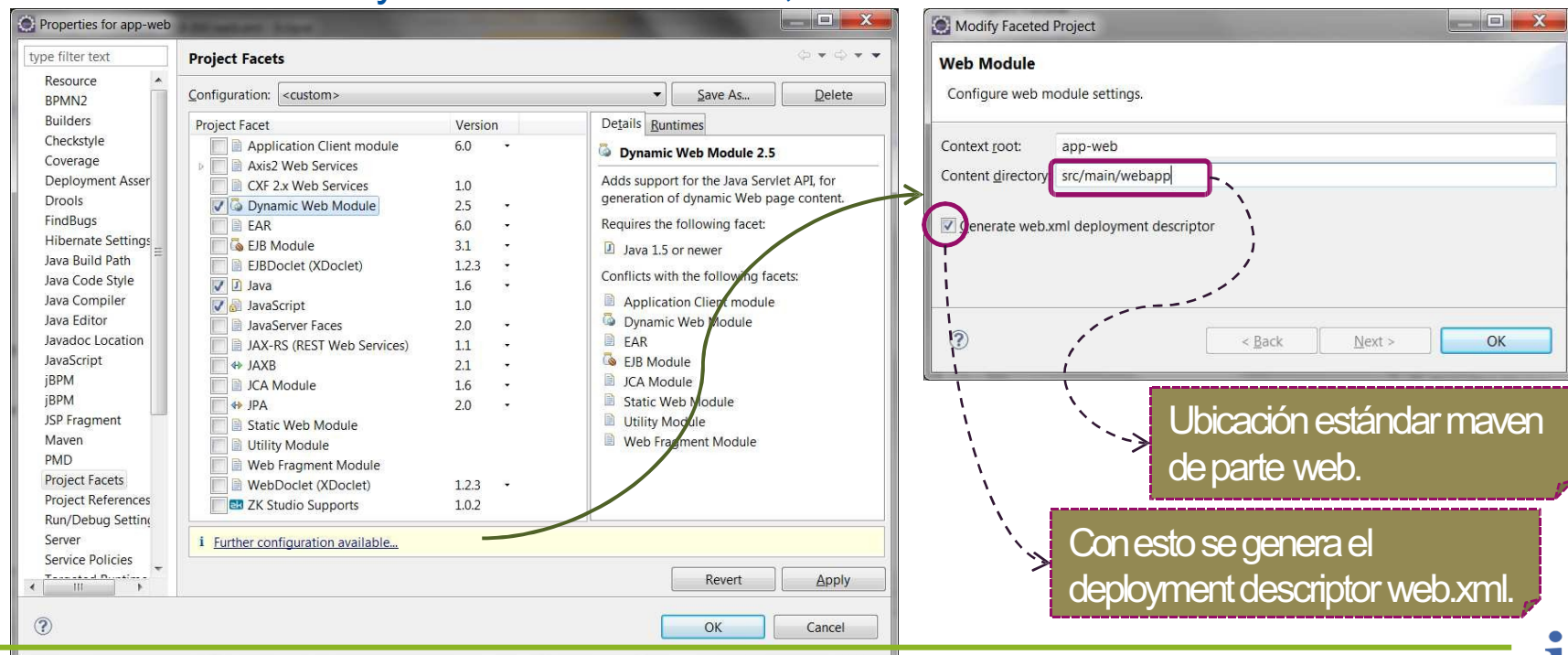


Al paquetizar app-web como un war, esta librería no es incluida.

5. Implementación de los proyectos

app-web

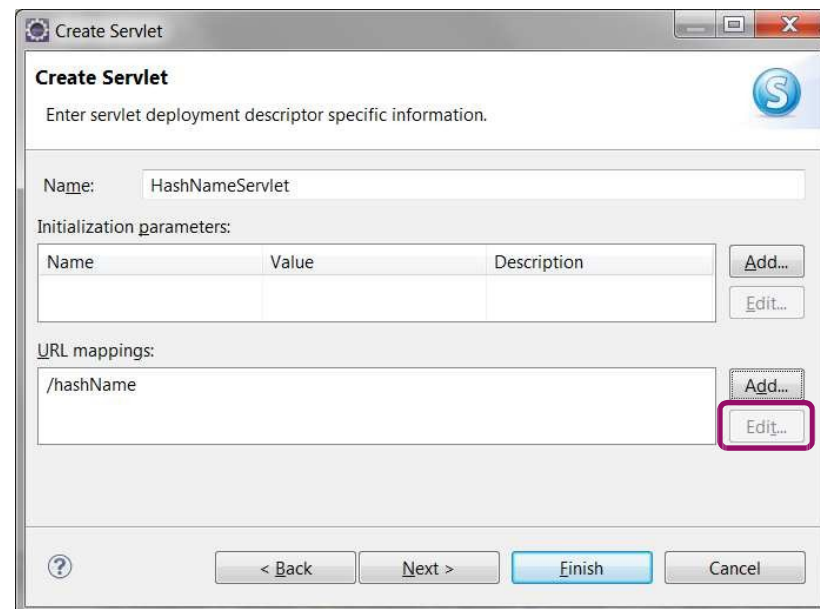
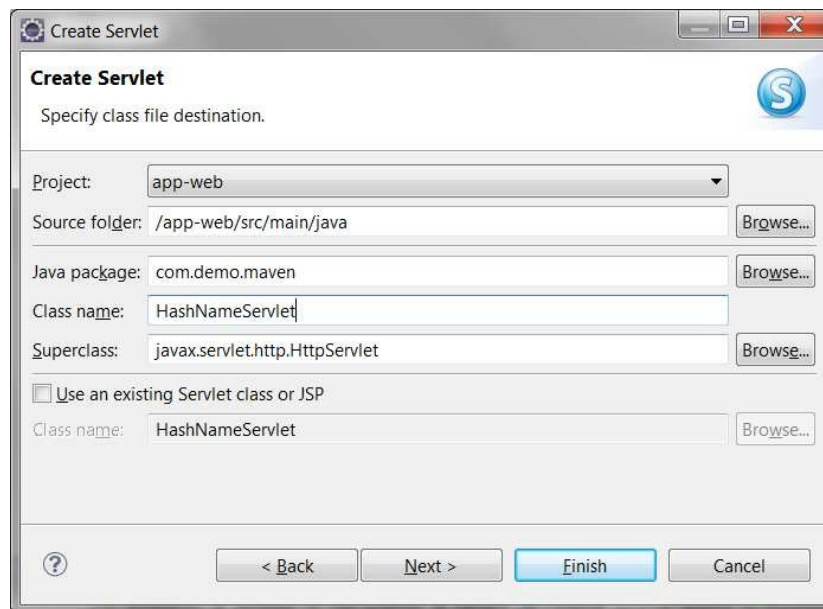
Antes de crear el servlet, es necesario que Eclipse reconozca el proyecto app-web como un proyecto del tipo "Dynamic Web Project", en los "facets". Para ello, en las propiedades del proyecto, opción Project Facets, desmarcar y volver a marcar "Dynamic Web Module", versión 2.5:



5. Implementación de los proyectos

app-web

En app-web, crear un nuevo servlet, invocable con la URL `"/hashName"`:



Nota: un servlet es una clase que se puede invocar a través de una URL. Escribe su resultado en la respuesta, y puede redirigir a una página.

5. Implementación de los proyectos

app-web

En el servlet HashName, implementar el método doGet(...) para que reciba los parámetros y ejecute la lógica. Por simplicidad, no se incluye control de errores:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String nombre = request.getParameter("nombre");
    String apellido = request.getParameter("apellido");

    PersonaBS persona = new PersonaBS();
    String hash = persona.hashNombre(nombre, apellido);

    request.setAttribute("hash", hash);
    request.setAttribute("nombre", nombre);
    request.setAttribute("apellido", apellido);

    request.getRequestDispatcher("/persona.jsp").forward(request, response);
}
```

Recupera los campos
enviados desde el formulario.

Escribe atributos, que pueden ser
recuperados en la página de destino.

Hace un forward a la página de destino.

5. Implementación de los proyectos

app-web

Implementar una página llamada persona.jsp en webapp, que recibe la información del servlet y la muestra en el navegador:

```
<html>
<head>
  <title>Persona</title>
</head>
<body>
<h3>Codificación hash: ${hash}</h3>

  <form action="hashName">
    <input name="nombre" value="${nombre}" />
    <br/>
    <input name="apellido" value="${apellido}" />
    <br/>
    <input type="submit" value="Calcular">
  </form>
</body>
</html>
```

URL de destino del formulario. Es el servlet.

Atributos escritos en el objeto response.

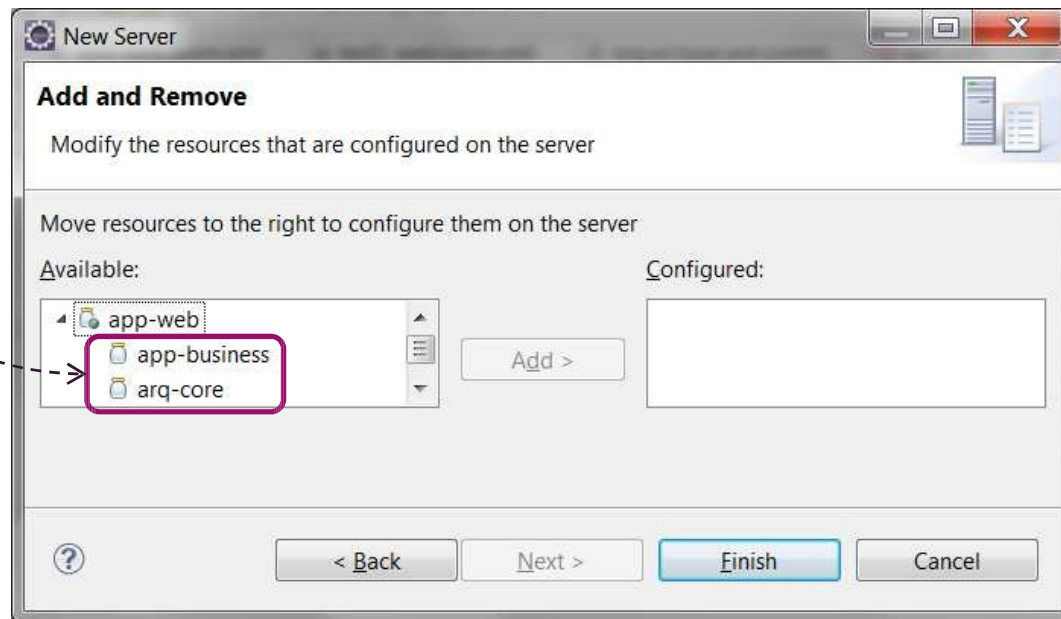
Esta página no incluye tags de control, por simplicidad

6. Despliegue de los proyectos

agregando el proyecto al servidor

Finalmente, para desplegar los proyectos, se crea o utiliza un Tomcat embebido en Eclipse, al cual se le agrega el proyecto app-web. Se observa que la dependencia definida en Maven es también reconocida por Tomcat:

En caso que no estén,
ver siguiente lámina.

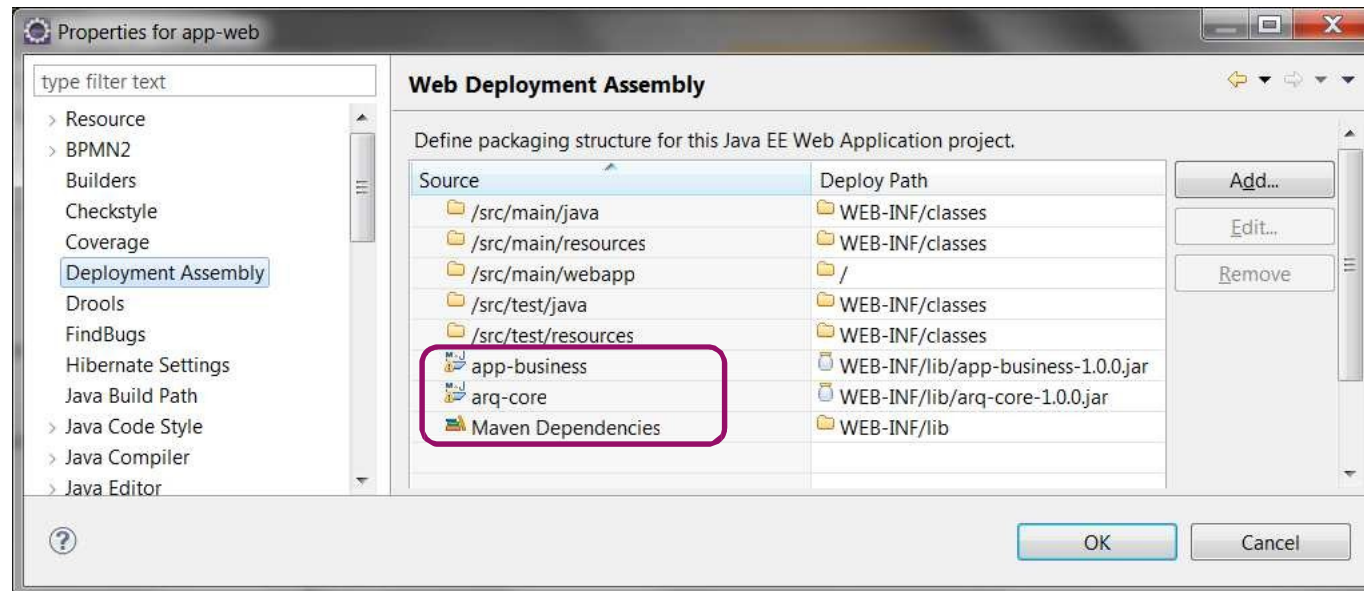


6. Despliegue de los proyectos

deployment assembly

Si se desplegara el war de la paquetización en un Tomcat externo, con lo realizado es suficiente. Bastaría con copiar dicho war en la carpeta webapps.

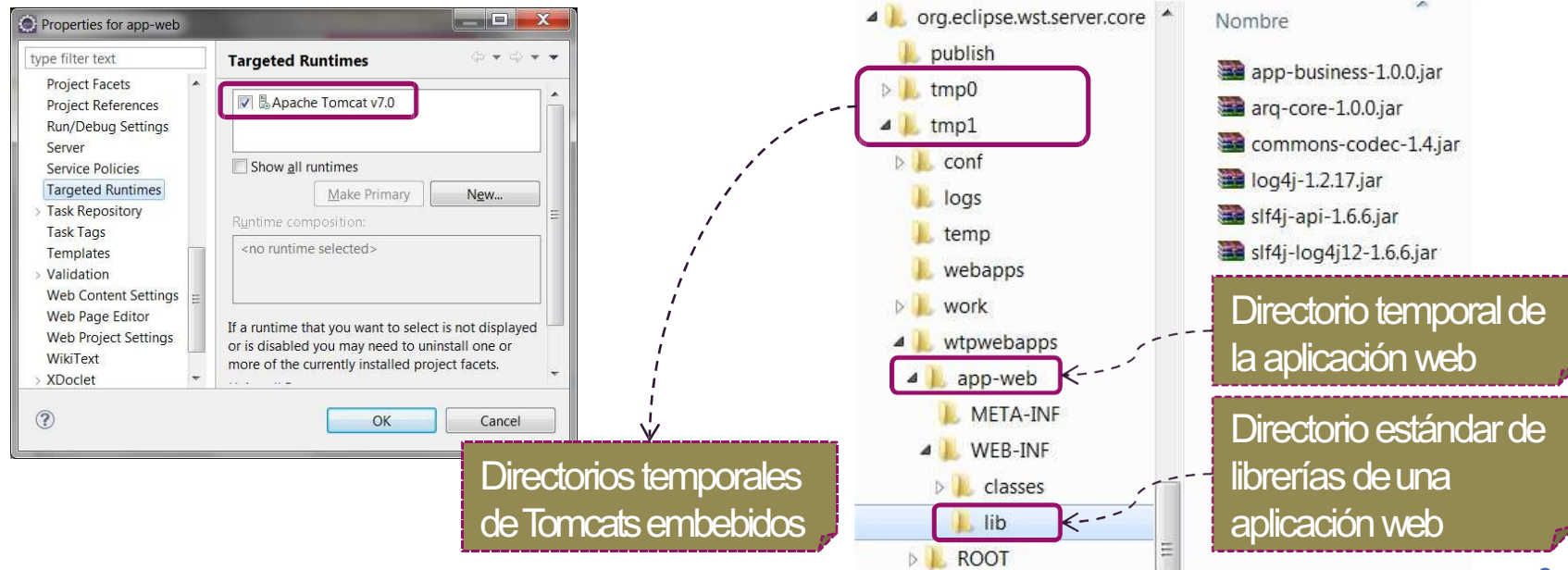
Sin embargo, para poder desplegar en un Tomcat embebido en Eclipse, se debe asegurar que los proyectos dependientes y las dependencias de Maven estén en el "Deployment Assembly". En las propiedades del proyecto app-web:



6. Despliegue de los proyectos

targeted runtime

Otra configuración que se debe tener en cuenta para el funcionamiento con un Tomcat embebido, es que el proyecto debe tener configurado Tomcat como Targeted Runtime. Esto permite que se copien todas las dependencias, tanto de otros proyectos como externas, al directorio temporal que utiliza internamente Eclipse para desplegar.

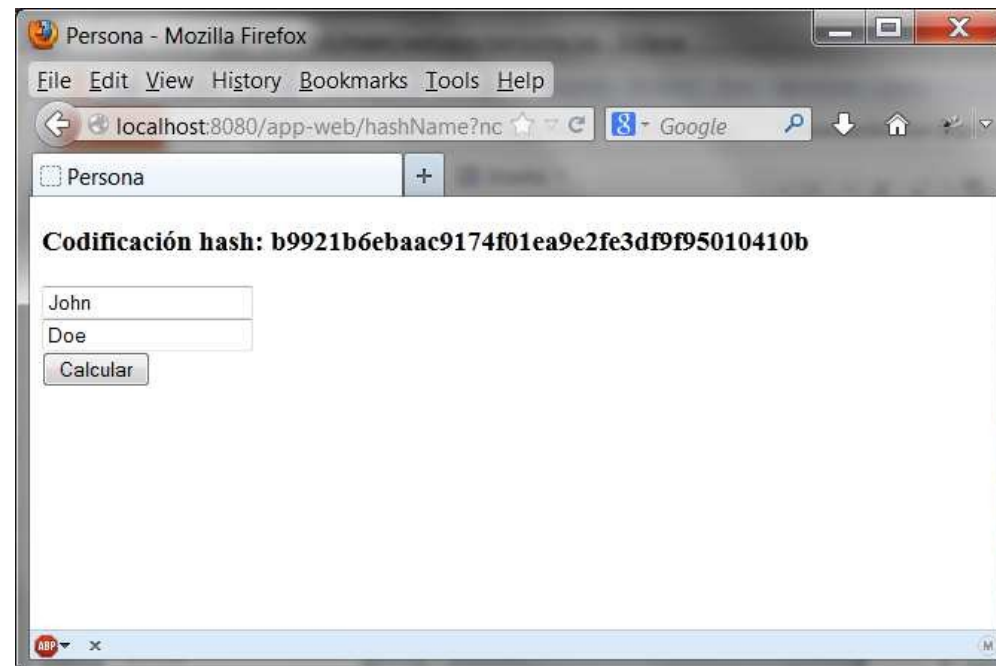


7. Prueba de los proyectos

prueba

Al iniciar Tomcat con la aplicación, al abrir la URL de la página, se observa el resultado:

<http://localhost:8080/app-web/persona.jsp>



8. Modulo parent

características

- Maven permite utilizar un POM "padre", del cual heredan los otros POM. En este POM padre se pueden definir elementos comunes, lo que evita tener que repetir configuraciones entre proyectos.
- En este caso, se adaptan los dos proyectos de la aplicación para que utilicen un proyecto parent, dejando independiente arqu-core.

8. Modulo parent

características

Como primer paso, se crea un nuevo proyecto Maven de tipo simple, con paquetización de tipo "**pom**":

- Group Id: com.demo.maven
- Artifact Id: app-parent
- Version: 1.0.0
- Packaging: pom

Para que los proyectos "hijos" puedan referenciarlo por sus coordenadas Maven, es necesario primero hacer un ***install*** de este proyecto.

8. Modulo parent

Utilización de módulo parent

Luego, se edita el POM del proyecto app-business, para referenciar al proyecto parent, de la manera siguiente:

```
<groupId>com.demo.maven</groupId>  
<artifactId>app-business</artifactId>  
<version>1.0.0</version>
```



```
<parent>  
  <groupId>com.demo.maven</groupId>  
  <artifactId>app-parent</artifactId>  
  <version>1.0.0</version>  
</parent>  
<artifactId>app-business</artifactId>
```

Se observa que:

- El groupId lo define el parent, no el proyecto. Todos los hijos comparten el groupId, y no debe volver a especificarse fuera del parent.
- El proyecto hijo localiza el POM padre a través de sus coordenadas directamente en el repositorio. Por eso es necesario hacer un install del padre.
- La versión también se hereda del padre.
- El hijo solamente especifica el artifactId.

8. Modulo parent

Utilización de módulo parent

Editar de la misma manera el POM del proyecto app-web, para referenciar al proyecto parent.

Configurar en el proyecto parent los siguientes elementos, para que los hereden los proyectos:

- Configuración de los plugin de Maven utilizados.
- Propiedades definidas que son de uso común, como las versiones de los frameworks.

8. Modulo parent

Utilización de módulo parent

Un módulo parent también sirve para agrupar la ejecución de comandos sobre sus hijos. Para ello, en el módulo parent se deben definir los módulos que lo comprenden. Sólo se puede utilizar ruta física, preferentemente relativa. En este caso, se agrega al POM padre lo siguiente:

```
<modules>
  <module>../app-business</module>
  <module>../app-web</module>
</modules>
```

Si se ejecuta el comando maven "install" sobre el parent, entonces realiza el install del proyecto app-business y app-web, en ese orden. Esto es útil para proyectos con varios subproyectos, pues agiliza su despliegue y gestión de configuración.