



Internet y Sistemas Distribuidos

Laboratorio de Prácticas



Índice

- Objetivos
- Entorno de Desarrollo
- Instalación y Configuración
- Apache Maven 3
 - Gestión de Proyectos
 - POM
 - Dependencias
 - Ciclo de Vida
- Ejemplos del Laboratorio
 - Estructura Maven 3
 - Configuración Maven 3
 - Compilación
- IDE (Entorno de Desarrollo Integrado) – IntelliJ IDEA
 - Configuración
 - Ejecución/Depuración
 - Plugin Maven 3
 - Ejecución Ejemplos
- Servidores Web
 - Jetty
 - Tomcat
- Recomendaciones Finales



Objetivos

- Familiarizarse con el entorno y herramientas de desarrollo de aplicaciones empresariales
- Ejecutar los ejemplos según se vaya avanzando en clase de teoría
- Realización de la **Práctica** de la asignatura
 - Enfoque basado en iteraciones, de manera que cada iteración incorpora más funcionalidad sobre la anterior, hasta que en la última iteración se termina con un software que implementa toda la funcionalidad
 - En cada iteración se hace análisis, diseño, implementación y pruebas
 - Grupos: **3** personas



Entorno de Desarrollo

- Sistema Operativo: Windows / Linux / Mac OS
- Software utilizado
 - IntelliJ IDEA
 - Eclipse Temurin (JDK 21)
 - Maven 3.9.x
 - MySQL 8.4.x
 - Tomcat 11.x.y
 - Compilador de Apache Thrift
 - Git (Windows y macOS)
 - Git – clientes gráficos (opcional). Por ejemplo:
 - Cliente de IntelliJ IDEA
 - GitKraken (Windows, macOS y Linux)
 - Gratuito para estudiantes: <https://www.gitkraken.com/github-student-developer-pack-bundle>
 - SourceTree (Windows y macOS)



Instalación y Configuración

- Seguir los pasos de los ficheros que están accesibles desde el campus virtual
 - LEEME_WINDOWS.md: Windows
 - LEEME_UNIX.md: Linux y Mac
- **IMPORTANTE:** Descargar las versiones adecuadas del software para el sistema operativo



Apache Maven 3 - Gestión de Proyectos (1)

- Herramienta de Gestión de proyectos Software
 - Estas herramientas sirven para automatizar las tareas necesarias para construir e instalar un proyecto a partir de sus fuentes, o en general, para automatizar cualquier tarea de desarrollo
 - Compilar el código fuente
 - Empaquetar el código binario y otros recursos (e.g. en ficheros JAR o WAR)
 - Ejecutar las pruebas
 - Crear documentación
 - Generar “distribuciones fuente”
 - Instalar las aplicaciones
 - Etc.



Apache Maven 3 - Gestión de Proyectos (2)

- Open Source
 - Proyecto Apache (<https://maven.apache.org>)
- Desarrollada en Java
- Características
 - Convenciones de nombrado
 - Sistema de plugins
 - Sistema de dependencias similar al existente en el 'mundo linux' con las dependencias entre paquetes
- Documentación
 - Maven en 5 minutos
 - <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
 - Guía básica de Maven
 - <https://maven.apache.org/guides/getting-started/index.html>

- Estandarización de prácticas de gestión de proyectos
 - No se pierde tiempo reinventando estructuras de directorios, convenciones, ni personalizando scripts para cada proyecto
 - Maven permite redefinir la estructura de directorios estándar pero es conveniente respetarla por las siguientes razones
 - Los ficheros de configuración serán más pequeños y sencillos
 - Hace que el proyecto sea más fácil de entender y facilita el mantenimiento futuro por otros
 - Facilita la integración de plugins (asumen también la estructura por defecto)



Apache Maven 3 - POM

- Núcleo: POM (Project Object Model)
 - Contiene una descripción detallada del proyecto, incluyendo información de versiones, gestión de configuración, dependencias, recursos de la aplicación y de pruebas, miembros del equipo, ...
 - Fichero XML situado en el directorio raíz del proyecto -> `pom.xml`
 - Si el proyecto contiene módulos, estos se declaran dentro del `pom.xml` del directorio raíz y cada módulo contiene su propio `pom.xml` (en su subdirectorio) que hereda del `pom.xml` general
 - Esta es la situación de los ejemplos de la asignatura



Apache Maven 3 – Ejemplo pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>es.udc.ws</groupId>
    <artifactId>ws-javaexamples</artifactId>
    <version>3.9.0</version>
  </parent>
  <artifactId>ws-util</artifactId>
  <packaging>jar</packaging>
  <name>WS-JavaExamples Util Subsystem</name>

  <dependencies>
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
    </dependency>
    ...
  </dependencies>
</project>
```



Apache Maven 3 – Dependencias (1)

- En el fichero `pom.xml` se definen las librerías de las que depende un proyecto y los plugins que utiliza
 - Maven automáticamente descarga las librerías y plugins de repositorios remotos en repositorios locales
 - Mantiene una estructura de directorios en base al `groupId` y `artifactId` de los elementos
 - Localización de repositorios (defecto)
 - Remoto -> <https://repo1.maven.org/maven2>
 - Local -> `$HOME/.m2`
 - Para realizar búsquedas sobre el repositorio remoto puede utilizarse el siguiente buscador
 - <https://search.maven.org>



Apache Maven 3 – Dependencias (y 2)

- Gestión de Dependencias Transitivas
 - Sólo es necesario especificar los jars que la aplicación necesita de forma directa
 - Gestiona las dependencias de las librerías utilizadas, para incluirlas de forma automática
- Ámbitos de Dependencias – en función de las fases
 - **compile** -> necesaria en todas las fases (valor por defecto)
 - **provided** -> necesaria para compilar pero no para instalar (e.g. API de servlets)
 - **runtime** -> necesaria sólo para ejecutar (e.g. drivers JDBC)
 - **test** -> necesaria sólo para pruebas (e.g. API JUnit)



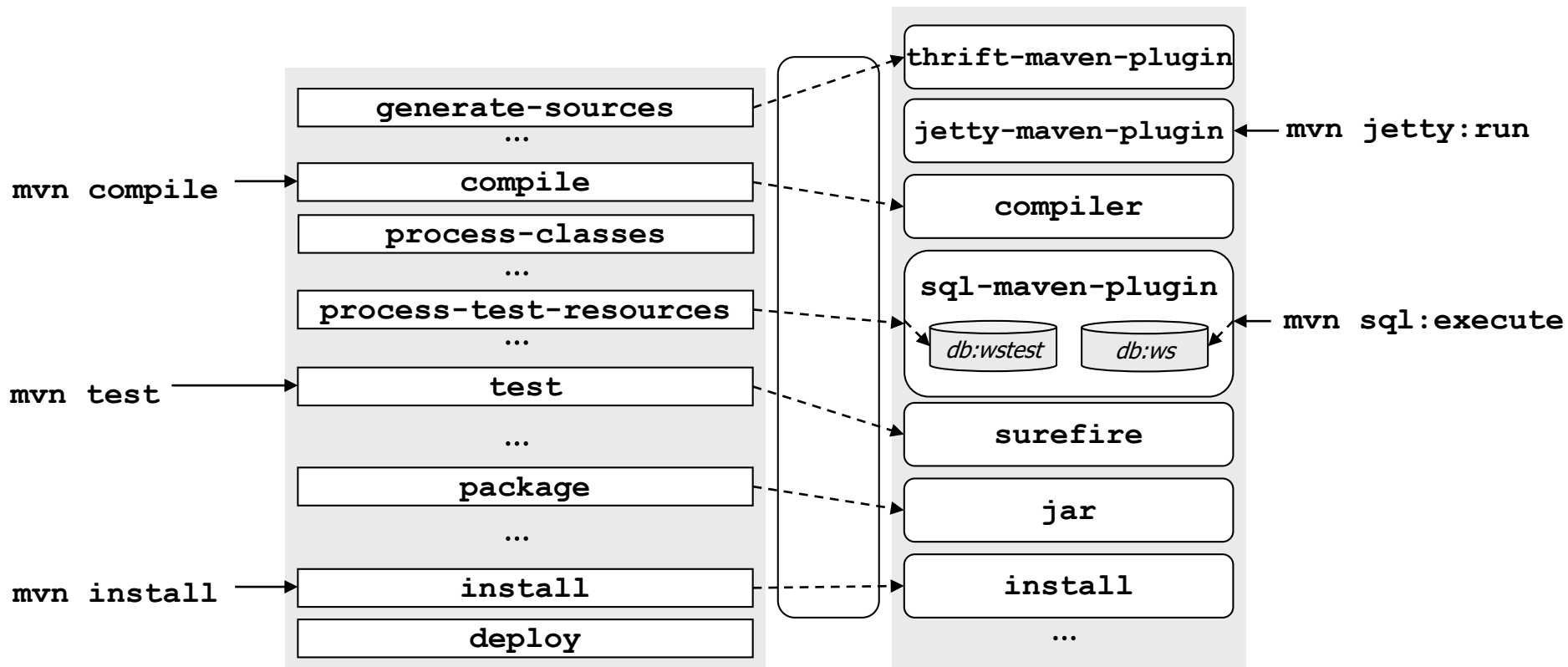
Apache Maven 3 – Ciclo de Vida (1)

- En Maven un ciclo de vida de construcción de un proyecto se compone de *fases*
 - Una *fase* es un *paso* o *etapa* a través de la cual Maven pasa durante la construcción de un proyecto y tiene un objetivo concreto para la construcción del proyecto
 - Existen tres ciclos de vida básicos asociados a la construcción de un proyecto dependiendo de la finalidad u objetivo
 - default: comprende todas las fases relacionadas con la generación e instalación de los artefactos o binarios de nuestro proyecto (e.g. un fichero JAR o WAR) a partir de los fuentes
 - clean: cubre las fases de limpieza del proyecto, es decir, la eliminación de todas las clases y artefactos generados durante el proceso de construcción
 - site: cubre las fases para la generación y distribución de un sitio web con la documentación que describe nuestro proyecto

Apache Maven 3 – Ciclo de Vida (2)

- Fases de un ciclo de vida

Ciclo de vida "default": Fases

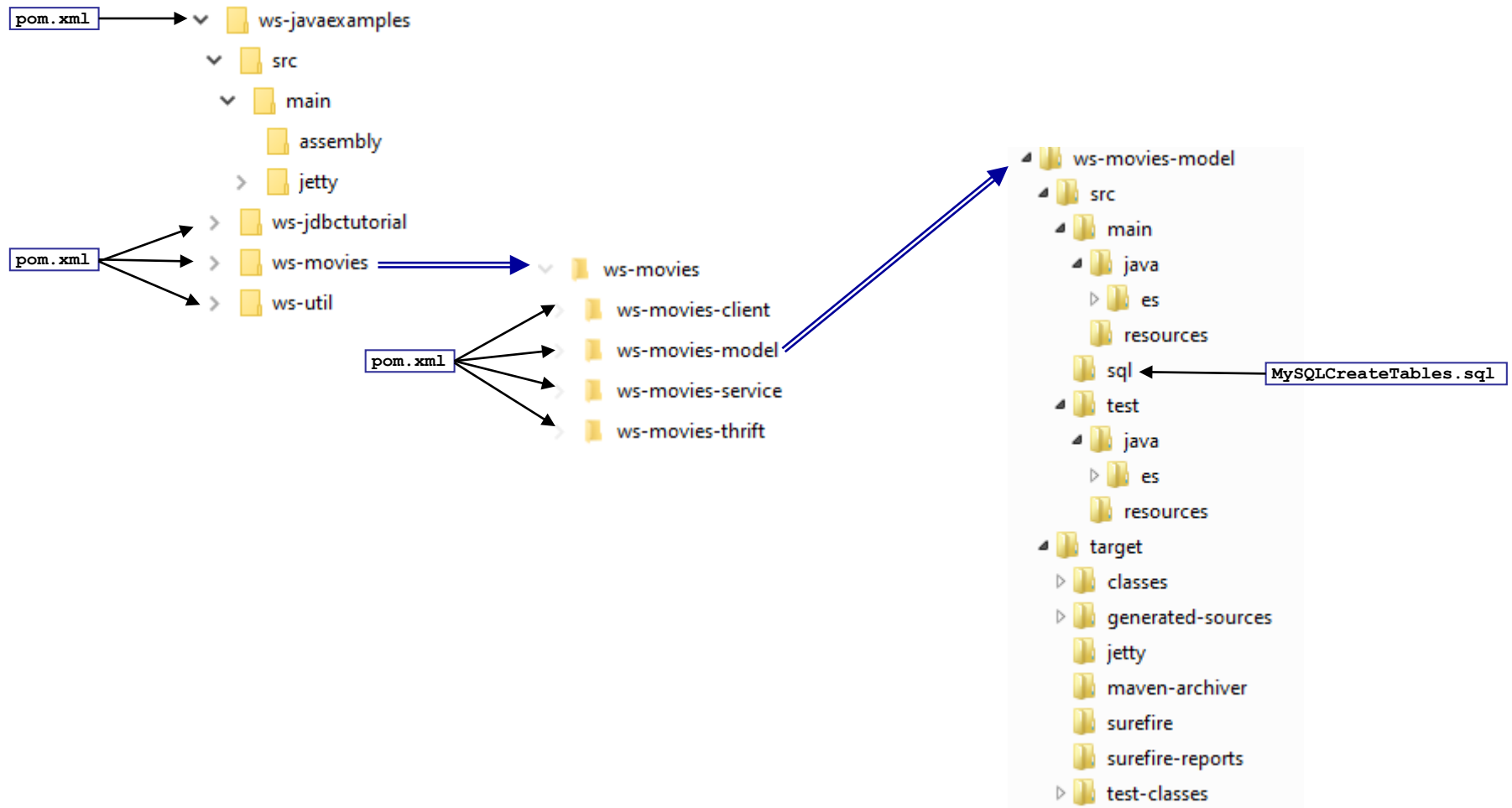




Apache Maven 3 – Ciclo de Vida (y 3)

- Fases de un ciclo de vida (cont)
 - Cada vez que se ejecuta una fase del ciclo de vida, se ejecutan los plugins vinculados a ella
 - Cada plugin tiene un conjunto de *goals* (acciones)
 - Un *goal* se puede vincular a una o varias fases de un ciclo de vida (o a ninguna) con distintas configuraciones
 - Algunos plugins ya están vinculados por defecto a algunas fases (e.g. el *goal* `"compile"` del plugin `"compiler"` está vinculado a la fase `"compile"`)
 - Mediante configuración es posible vincular otros plugins a las fases del ciclo de vida deseadas
 - A Maven se le puede pedir la ejecución de
 - Una fase, y en ese caso, ejecuta también las fases anteriores (e.g. `mvn compile`, `mvn test`)
 - Un goal de un plugin (e.g. `mvn compiler:compile`, `mvn sql:execute`, etc.)

Ejemplos - Estructura Maven 3





Ejemplos – Configuración Maven 3 (1)

■ Filtrados

- Permiten especificar ciertos aspectos de configuración de los recursos como variables, que son reemplazadas por un valor concreto en tiempo de construcción del proyecto (cuando se ejecuta Maven)
 - E.g. las propiedades de configuración del DataSource a utilizar por Jetty están expresadas como variables en el fichero de configuración `src/main/jetty/jetty-env.xml` y en tiempo de construcción se sustituyen por los valores que se hayan especificado para esas variables (más concretamente, cuando se copia el fichero `jetty-env.xml` al directorio `target`)



Ejemplos – Configuración Maven 3 (y 2)

- Plugins

- Compiler
 - Configuración del compilador
- SQL (**`mvn sql:execute`**)
 - Ejecución de los scripts SQL de creación de tablas
- Jetty (**`mvn jetty:run`**)
 - Permite ejecutar una aplicación en el servidor Web Jetty sin necesidad de empaquetarla en un fichero WAR
- Thrift
 - Goal **`compile`**: Genera el código fuente correspondiente al stub y el skeleton a partir del fichero de definición (fichero `.thrift`)
 - Configurado para ejecutarse en la fase **`generate-sources`**

Ejemplos – Compilación

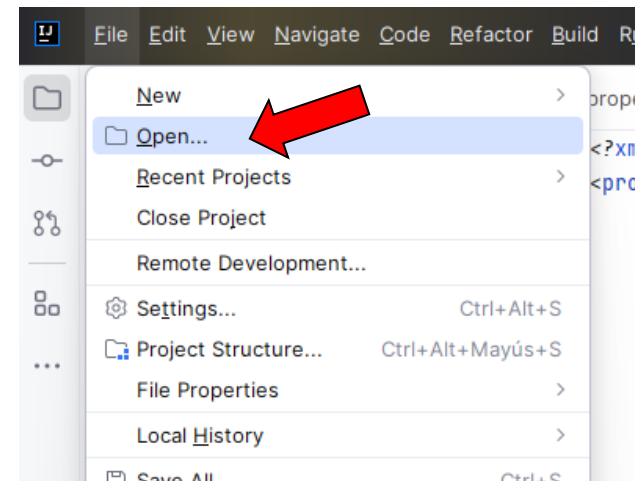
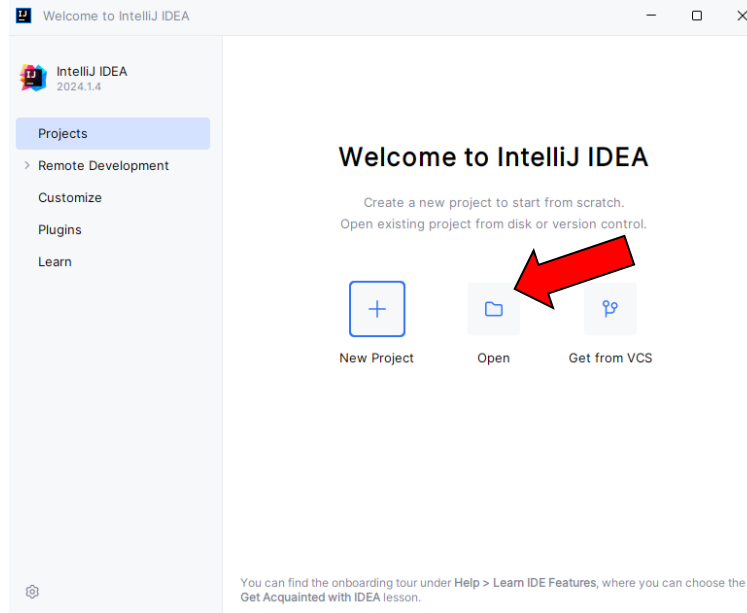
- Configuración de MySQL (ya realizada al seguir los pasos de los ficheros LEEME)
 - BD para la ejecución de la aplicación: **ws**
 - BD para la ejecución de las pruebas: **wstest**
 - En ambos casos => usuario **ws**, password **ws**
 - Disponer de ambas bases de datos, permite que la ejecución de la aplicación no afecte a la ejecución de las pruebas, y viceversa
- Arrancar la BD (si no se instaló como servicio)
- Utilizar Maven para crear las tablas en la base de datos y compilar los ejemplos

```
cd ws-javaexamples-<version>  
mvn sql:execute install
```

IntelliJ IDEA – Abrir / Importar

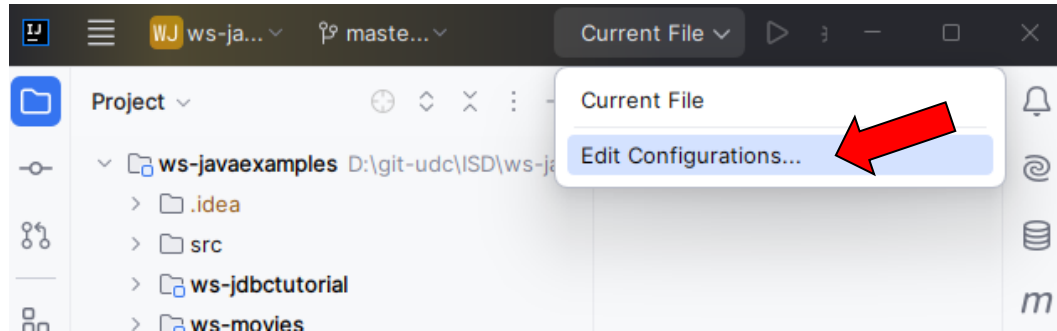
■ Abrir / Importar un proyecto Maven

- Si no hay abierto ningún proyecto previamente, elegir la opción "Open", seleccionar el directorio raíz del proyecto y clic en "Ok"
- Si ya hay abierto un proyecto, elegir la opción del Menú "File > Open", seleccionar el directorio raíz del proyecto y clic en "Ok" (permitirá indicar si abrirlo en una ventana nueva)



IntelliJ IDEA – Configuraciones (1)

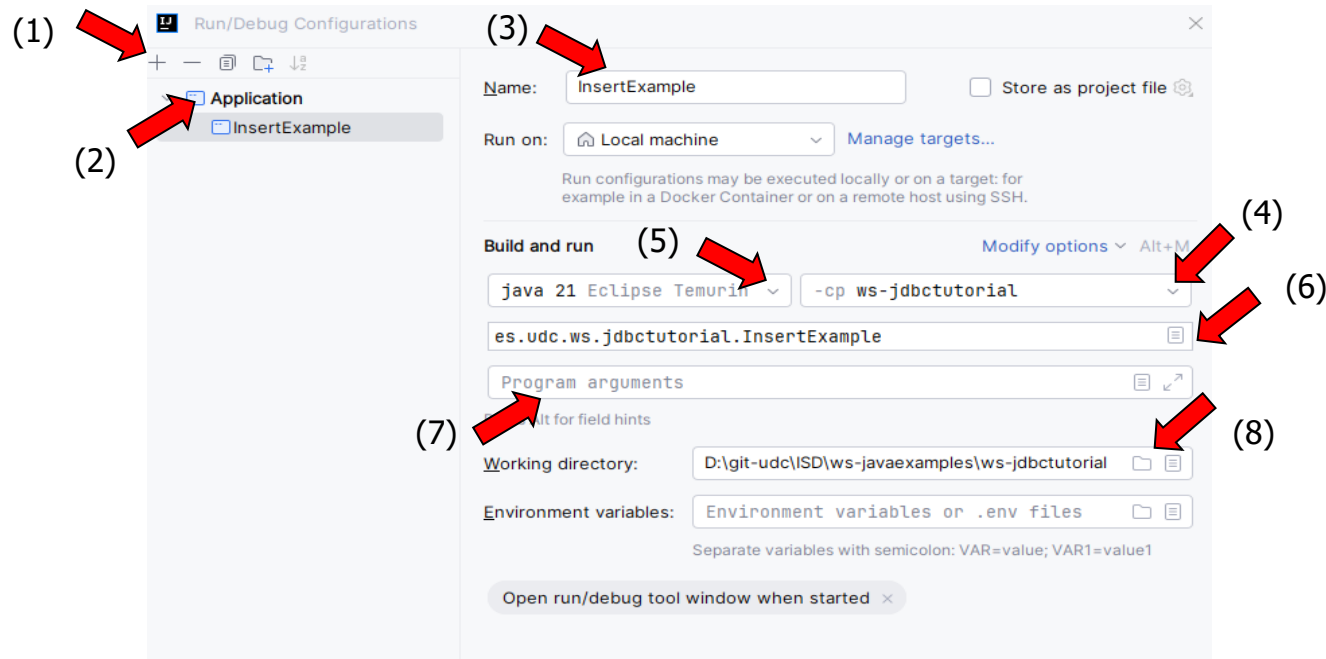
- Crear una configuración
 - En la toolbar hacer clic en “**Edit Configurations...**”



- Equivalente al menú “**Run > Edit Configurations...**”

IntelliJ IDEA – Configuraciones (2)

- Configuración para ejecutar una clase
 - (1) Hacer clic en "+" y (2) elegir "Application"

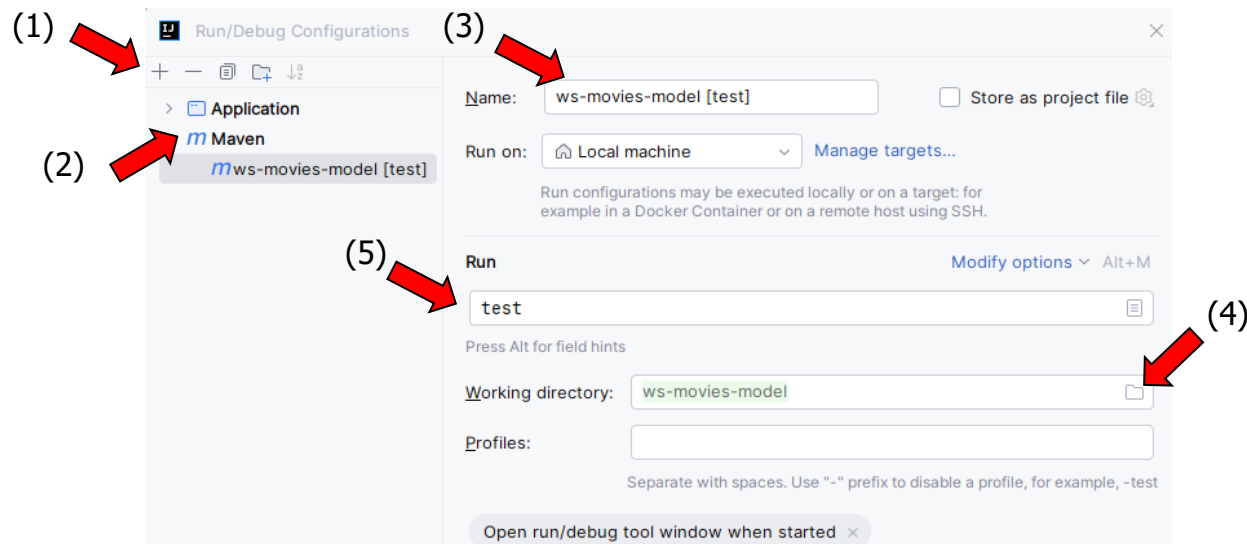


- (3) Rellenar el nombre de la configuración, (4) elegir el módulo maven que tiene la clase, (5) elegir el JRE con el que se desea ejecutar, (6) elegir la clase a ejecutar, (7) [opcional] especificar los argumentos si requiere alguno, y (8) [opcional] cambiar el directorio de trabajo si se desea especificar uno diferente

IntelliJ IDEA – Configuraciones (y 3)

Configuración para ejecutar un *goal* Maven

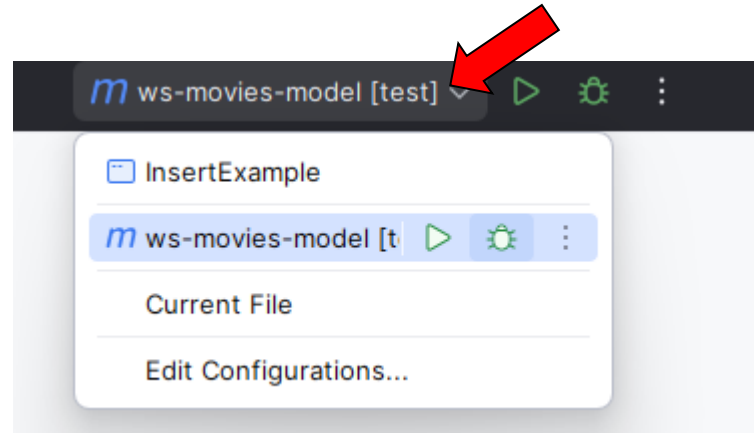
- Equivalente a ejecutar Maven desde línea de comandos, pero nos permite ejecutarlo en modo debug
- (1) Hacer clic en "+" y (2) elegir "Maven"



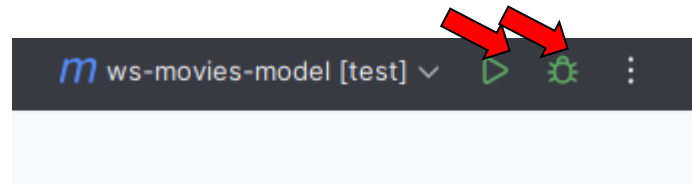
- (3) Rellenar el nombre de la configuración (si no se rellena, se genera uno automáticamente), (4) en "Working directory" especificar el directorio raíz del proyecto Maven (donde está el pom.xml), y (5) en "Command line" el *goal* que se desea ejecutar
 - Nota: La imagen contiene los valores para crear una configuración Maven para ejecutar los tests de la capa modelo de ws-movies (directorio `ws-movies-model` y goal `test`)

IntelliJ IDEA – Ejecutar Configuración

- Seleccionar la configuración

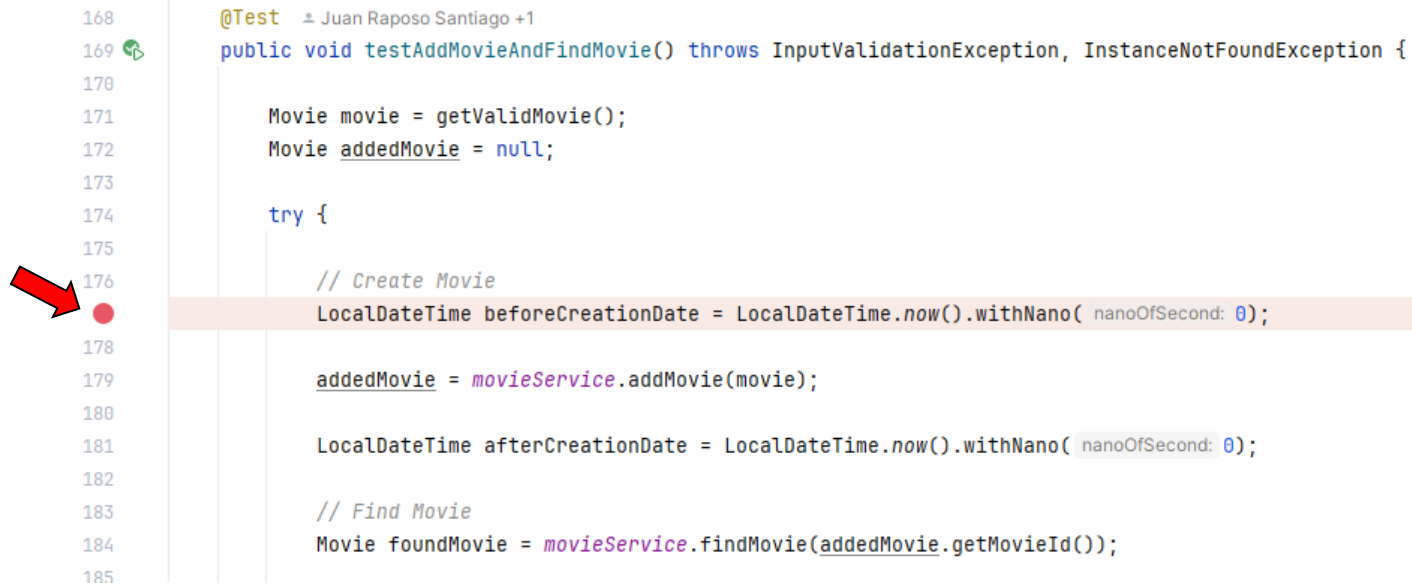


- Ejecutar en modo "normal" o en modo "debug"



IntelliJ IDEA – Breakpoints

- Para añadir un *breakpoint*, doble clic junto al número de línea en la que se quiere establecer



The screenshot shows the IntelliJ IDEA code editor with a Java file. On the left, a vertical line of numbers represents line numbers from 168 to 185. A red circle breakpoint is placed next to line 176. A red arrow points to this breakpoint. The code on line 176 is `LocalDateTime beforeCreationDate = LocalDateTime.now().withNano(nanoOfSecond: 0);`. The code block is as follows:

```
168  @Test  ± Juan Raposo Santiago +1
169  public void testAddMovieAndFindMovie() throws InputValidationException, InstanceNotFoundException {
170
171      Movie movie = getValidMovie();
172      Movie addedMovie = null;
173
174      try {
175
176          // Create Movie
177          LocalDateTime beforeCreationDate = LocalDateTime.now().withNano( nanoOfSecond: 0);
178
179          addedMovie = movieService.addMovie(movie);
180
181          LocalDateTime afterCreationDate = LocalDateTime.now().withNano( nanoOfSecond: 0);
182
183          // Find Movie
184          Movie foundMovie = movieService.findMovie(addedMovie.getMovieId());
185      }
```



Ejecución de los Ejemplos

- Arrancar la base de datos (si no está arrancada como servicio)
- La base de datos debe estar inicializada y los ejemplos compilados (ver diapositiva 19)
- Ejecución desde IntelliJ IDEA o desde línea de comandos

- ws-jdbctutorial: Contiene 3 clases con métodos “main”

```
es.udc.ws.jdbctutorial.InsertExample  
es.udc.ws.jdbctutorial.SelectExample  
es.udc.ws.jdbctutorial.TransactionExample
```

- ws-movies

- Seguir pasos del fichero **README.md** que se encuentra en el directorio raíz de los ejemplos

Jetty & Maven 3 (1)



- Jetty: Servidor HTTP y contenedor de servlets *open source* escrito 100% en Java
- Maven 3 + plugin Jetty
 - Desarrollo ágil: Permite ejecutar una aplicación Web sin empaquetarla en un fichero WAR (entiende los directorios "**src**" y "**target**" de la distribución fuente)
 - El plugin puede configurarse para que detecte automáticamente cambios en la aplicación web y la recargue, o para permitir una recarga manual
 - Ejecución:
 - `cd <module>`
 - `mvn jetty:run`
 - Cargar en el navegador `http://localhost:9090/<module>`
 - Ejemplo:
 - `cd ws-movies/ws-movies-service`
 - `mvn jetty:run`
 - `http://localhost:9090/ws-movies-service/`
 - Para parar Jetty: **CTRL-C**
 - Para recargar la aplicación web: **ENTER**



Jetty - Modificación del Puerto por Defecto (2)

- Por defecto se arranca en el puerto 8080
- En los ejemplos de la asignatura se ha configurado para que arranque en el puerto 9090
- Se ha incluido el siguiente fragmento en la configuración del plugin de Jetty (dentro del tag `configuration`)

```
<configuration>
  <httpConnector>
    <port>9090</port>
  </httpConnector>
  ...
</configuration>
```

- También es posible especificar un puerto distinto en el momento de arrancar el servidor Jetty desde Maven, usando la propiedad de entorno `jetty.port` (es necesario eliminar el puerto de la configuración del plugin en el `pom.xml`)
 - E.g. Arrancar jetty en el puerto 9090

```
mvn -Djetty.port=9090 jetty:run
```



Jetty - Detección de Cambios (3)

- Cambios a ficheros fuente Java y recursos
 - Es posible configurar Jetty para que recargue la aplicación Web
 - Para aplicar cambios en ficheros java o recursos de la aplicación (ficheros de configuración)
 - En la asignatura hemos optado por configurar Jetty para que recargue la aplicación Web pulsando **ENTER** en la consola de Jetty
 - Es necesario haber compilado previamente, por tanto, antes de recargar se recomienda ejecutar `mvn install` desde el directorio raíz de `ws-app`



Jetty - Depuración (y 4)

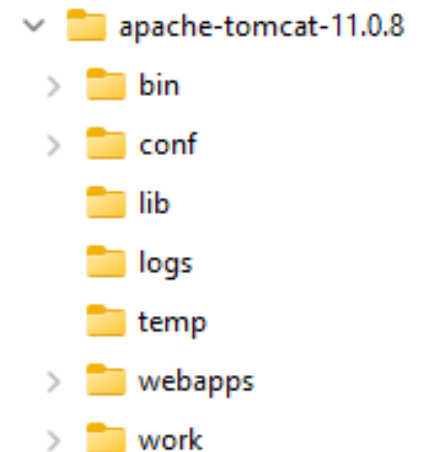
■ Depuración Web

- Crear en IntelliJ IDEA una configuración para ejecutar el *goal* `jetty:run` en el directorio `ws-app-service`
- Desde la barra de herramientas elegir la configuración y arrancarla en modo debug (icono **Debug**)
- Para finalizar la ejecución de Jetty, hay que pulsar sobre el botón rojo de la consola de Jetty en IntelliJ IDEA

Apache Tomcat



- No lo utilizaremos hasta la segunda iteración de la práctica
- Configuración
 - Es necesario haber seguido los pasos del apartado “Configuración de Tomcat” del fichero LEEME_WINDOWS.md o LEEME_UNIX.md
- Arranque -> <http://localhost:8080>
 - `bin/startup.bat` (Windows)
 - `bin/startup.sh` (Linux / macOS)
- Finalización:
 - `bin/shutdown.bat` (Windows)
 - `bin/shutdown.sh` (Linux / macOS)
- Instalación de Aplicaciones
 - Copiar el fichero WAR al directorio **webapps**
- Logs
 - `logs/localhost.<FECHA>.log`
 - `logs/catalina.out`
- Ejecución del ejemplo ws-movies
 - Seguir pasos del fichero **README.md** que se encuentra en el directorio raíz de los ejemplos



Herramientas HTTP

- Existen herramientas que permiten probar la parte *servidora* de un servicio web (e.g. el de la práctica) mientras no se implemente la parte *cliente*
- Postman
 - Herramienta con versión gratuita y open source
 - Permite invocar servicios web REST



POSTMAN



Comentarios

- La práctica debe poder ser compilada y empaquetada con Maven 3 desde línea de comandos, y ejecutada desde Tomcat, aunque se utilicen IntelliJ IDEA y Jetty como entorno de desarrollo