

Materiale fornito come supporto per lo studio del corso di SWAM tenuto presso UNIFI. Alcune delle figure o dei loghi presenti possono essere tratte da pubblicazioni o siti web e quindi soggette a copyright. Pertanto l'adistribuzione di questo materiale è lecita se effettuata a fini di insegnamento o di ricerca scientifica o di utilizzo sia per finalità illustrative o di studio e per fini non commerciali.

Java EE in action: How To

Materiale di supporto al corso di
Software Architectures and Methodologies
Laboratory

Ing. Jacopo Parri, Ing. Samuele Sampietro
Software Technologies Lab (STLab)
Dipartimento di Ingegneria dell'Informazione (UNIFI)



Argomenti trattati

- Predisporre il proprio SO per il Java
 - JDK e JRE
- Alcuni IDE
 - Eclipse
- Alcuni Application Server
 - WildFly
- Configurare un progetto Maven
 - le Specifiche JEE nel progetto
 - esempio

SWAM

2

Premessa

Nonostante il Java sia un **linguaggio multiplatforma**, lavorare con gli strumenti che lo riguardano può avere differenze a seconda del Sistema Operativo (SO) adottato dallo sviluppatore.

In questo modulo del corso, quanto detto vale per ogni SO:

- ma si farà maggiore attenzione per il caso di **Windows**
 - perché solitamente gli utenti Linux sono più "abili"
 - perché come si dice "Mac OS X è il miglior SO al mondo"
 - perché "in realtà" il laboratorio utilizza *in primis* Windows

SWAM

3

Introduzione

Per poter iniziare a sviluppare in **Java** è obbligatorio configurare il proprio ambiente:

- predisponendo il proprio Sistema Operativo per la **Java Platform**
 - installando una JDK e una JRE
- installando e configurando un **IDE adeguato**
 - ne esistono vari (anche open-source)
- installando e configurando un **Application Server**
 - se si è interessati a sviluppare applicazioni Java EE
 - ne esistono vari (anche open-source)

SWAM

4

Predisposizione del Sistema Operativo - JDK

Per **sviluppare** applicazioni Java si deve installare un **Java Development Kit (JDK)**.
La scelta possibile è tra:

- **Java Standard Edition (SE) Development Kit** di *Oracle Corporation*
 - <https://www.oracle.com/java/technologies/javase-downloads.html#javasejdk>
 - licenza d'uso per uso personale non commerciale
- **Java Enterprise Edition (EE) Development Kit** di *Oracle Corporation*
 - <https://www.oracle.com/java/technologies/javaee-8-sdk-downloads.html>
 - licenza d'uso per uso personale non commerciale (ultima versione JEE8)
- **Open Java SE Development Kit** rilasciato dalla *OpenJDK Community*
 - <https://jdk.java.net/>
 - licenza open-source di tipo GNU GPLv2

SWAM

5

Predisposizione del Sistema Operativo - JRE

Per **eseguire** programmi Java, non necessariamente da noi sviluppati, si deve installare un **Java Runtime Environment (JRE)**.

In questo caso conviene installare quella ufficiale:

- **JRE** di *Oracle Corporation*
 - <https://www.java.com/it/download/>
 - licenza d'uso per uso personale non commerciale

Potrebbe servire per lanciare alcuni strumenti Java-based (e.g. IDE)

SWAM

6

Predisposizione del Sistema Operativo - Variabili di ambiente



In Windows si deve anche configurare le **variabili di ambiente** affinché il Java sia disponibile (sia da "shell" sia per i programmi che lo richiedono).

In questo caso, su Windows 10, cercare:

- "Impostazioni" → "Modifica le variabili di ambiente relative al sistema"
 - Scheda "Avanzate" → "Variabili d'ambiente..."
 - creare una variabile di sistema chiamata **JAVA_HOME**
 - assegnargli come valore il path alla vostra JDK
 - aggiungere alla variabile Path il valore **%JAVA_HOME%\bin**

SWAM

7

Prova da "shell" del Java

Per testare se nel Sistema Operativo il Java **funziona correttamente** si può aprire il proprio terminale e digitare il comando:

```
java -version
```

Se tutto è OK dovrebbe comparire una indicazione simile a questa:

```
openjdk version "13.0.1" 2019-10-15
OpenJDK Runtime Environment (build 13.0.1+9)
OpenJDK 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing)
```

SWAM

8

Integrated Development Environment (IDE)

Lo sviluppatore deve prendere in considerazione anche l'utilizzo di un IDE standalone.
La scelta possibile è tra:

- **Eclipse by Eclipse Foundation**
 - <https://www.eclipse.org/>
 - licenza open-source
- **IntelliJ IDEA by JetBrains**
 - <https://www.jetbrains.com/idea/>
 - licenza d'uso open-source (ma senza JEE) oppure commerciale (con JEE)
- **Visual Studio Code by Microsoft**
 - <https://code.visualstudio.com/>
 - licenza open-source

SWAM

9

IDE - varianti ulteriori

Ovviamente di IDE ne esistono anche di più, ma abbiamo voluto elencare i principali.

Ogni sviluppatore deve scegliere quello che preferisce.

Ulteriori IDE standalone:

- **Android Studio by Google**
- **JDeveloper by Oracle Corporation**
- **NetBeans by The Apache Software Foundation**

Ulteriori IDE cloud:

- **Codenvy by Red Hat**
- **Eclipse Che by Eclipse Foundation**

SWAM

10

IDE - Eclipse

La soluzione consigliata per molte esigenze è offerta da **Eclipse**.

Dopo aver scaricato l'ultima versione dal sito ufficiale:

<https://www.eclipse.org/downloads/packages/installer>

Sarà sufficiente seguire le istruzioni del **wizard**.

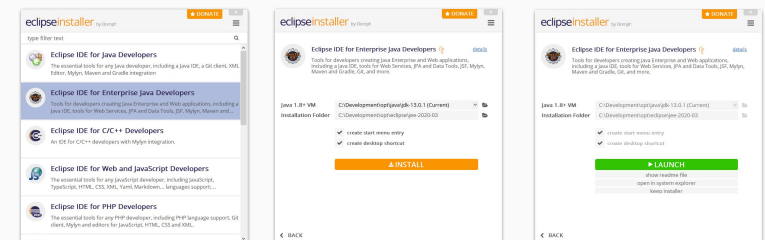
Vediamo i passi principali per Windows 10.



SWAM

11

IDE - Eclipse "installer wizard"



STEP 1

STEP 2

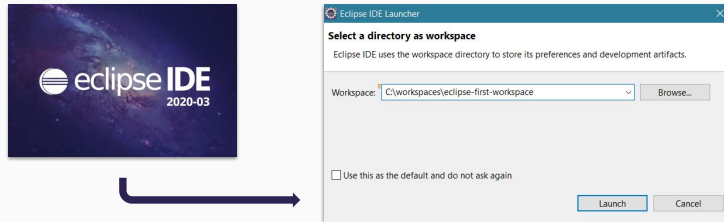
STEP 3

SWAM

12

IDE - Eclipse "primo avvio" [1/3]

Al lancio, dopo lo *splash screen*, viene subito chiesto di indicare il **Workspace**.

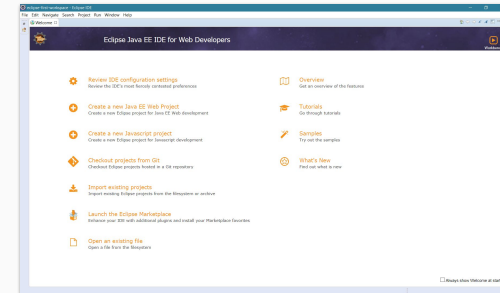


SWAM

13

IDE - Eclipse "primo avvio" [2/3]

All'apertura si visualizza una specie di **"Welcome page"**.

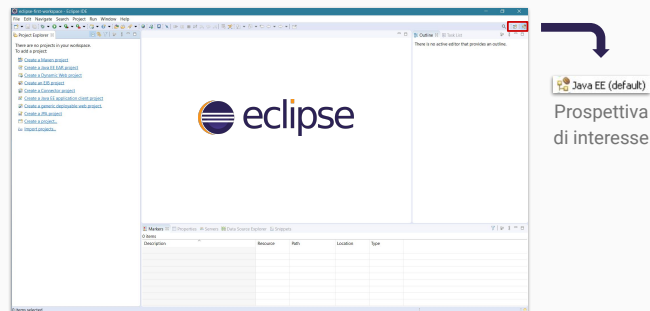


SWAM

14

IDE - Eclipse "primo avvio" [3/3]

Chiudendo la pagina di benvenuto si accede al proprio **"Workspace"**.



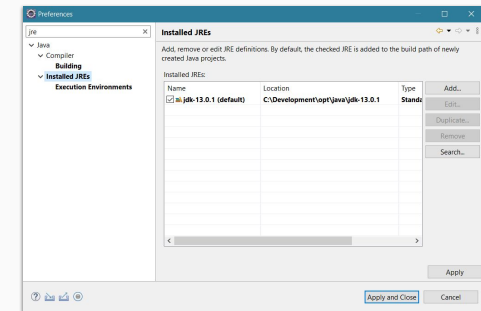
SWAM

15

IDE - Eclipse "Is Java Installed ?"

Se tutto sta procedendo bene, aprendo **"Window" → "Preferences"**

- digitando **"jre"** nel campo di ricerca
- si ritrova la JDK
 - puntata anche nel ruolo di JRE



SWAM

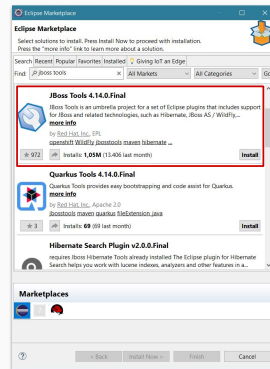
16

IDE - Eclipse "configurazioni ulteriori"

"Help" → "Eclipse Marketplace..."

Conviene **installare** anche un **pacchetto aggiuntivo** dal Marketplace dell'IDE (**JBoss Tools**) per avere un supporto ad alcune Specifiche Java EE e ad alcuni prodotti sviluppati da **JBoss Red Hat**, come:

- JPA + Hibernate
- JBoss AS / WildFly
- CDI + Weld
- JSF
- etc.



17

Application Server (AS)

Si deve anche prevedere l'utilizzo di un **server per lo sviluppo di applicazioni web**. La scelta possibile è tra:

- **GlassFish** by *Eclipse Foundation* (in origine proprietà di *Oracle Corporation*)
 - <https://github.com/eclipse-ee4j/glassfish>
 - licenza open-source
- **Tomcat** by *The Apache Software Foundation*
 - <https://tomcat.apache.org/>
 - licenza open-source
- **WildFly** by *JBoss Red Hat*
 - <https://wildfly.org/>
 - licenza open-source

SWAM

18

AS - utili precisazioni

Tra i server indicati si deve fare una distinzione:

- **GlassFish e WildFly**
 - sono certificati JEE e servono per lo sviluppo di applicazioni Enterprise di varia natura
 - JavaServer Faces (JSF)
 - Web Services
 - RESTful Services
- **Tomcat**
 - è una implementazione open-source delle seguenti tecnologie
 - Java Servlet
 - JavaServer Pages (JSP)
 - Java Expression Language
 - Java WebSocket

19

AS - WildFly

La scelta consigliata è di adottare **WildFly** come server per lo sviluppo.

Dopo aver scaricato l'ultima versione dal sito ufficiale:

<https://wildfly.org/downloads/>



Si deve collocare in una directory il contenuto dell'archivio compresso scaricato:

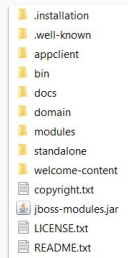
C:\Development\opt\application-server\wildfly-19.0.0.Final

SWAM

20

AS - WildFly "struttura cartelle" [1/2]

La cartella principale **wildfly-19.0.0.Final** presenta questa struttura:



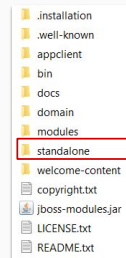
- **modules** contiene
 - i moduli applicativi
 - le librerie di runtime
- **standalone** contiene
 - le configurazioni dell'istanza del server
 - le applicazioni da eseguire sull'istanza
 - ossia i cosiddetti *deployment*

SWAM

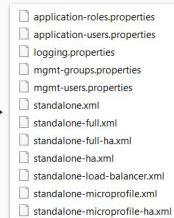
21

AS - WildFly "struttura cartelle" [2/2]

In particolare, la cartella **standalone** presenta questa struttura:



- **configuration** contiene
 - i file di *config* dell'istanza
- **deployments** contiene
 - i cosiddetti WAR files
 - Web application ARchive



SWAM

22

AS - WildFly "aggiunta di moduli" [1/4]

Spesso torna utile **aggiungere nuovi moduli** all'istanza del server.

Ad esempio, in molti progetti Java EE si necessita di un DBMS per salvare su di una base di dati lo stato delle entità in vita nel modello di dominio.

Tra i moduli del server è quindi utile predisporre almeno i **connettori**:

- connettori per **MySQL**
 - <https://www.mysql.com/it/products/connector/>
- connettori per **MariaDB**

SWAM

23

AS - WildFly "aggiunta di moduli" [2/4]

L'aggiunta di un modulo è documentata dai fornitori interessati ad integrarsi con WildFly.

Nel caso del **connettore per MySQL** si deve:

1. scaricare il connettore dal sito ufficiale (e.g. `mysql-connector-java-8.0.19`)
2. creare dentro alla cartella dei moduli di WildFly un percorso per il connettore
 - `/wildfly-19.0.0.Final/modules/system/layers/base/com/mysql/main`
3. copiare dentro il JAR del connettore
 - `mysql-connector-java-8.0.19.jar`
4. creare un file `module.xml`
 - come segue

Analogamente, con gli opportuni accorgimenti, si può configurare per il connettore **MariaDB**.

SWAM

24

AS - WildFly "aggiunta di moduli" [3/4]

Il contenuto del file `module.xml` è una configurazione per il modulo stesso.

Il formato è XML:

```
<module xmlns="urn:jboss:module:1.5" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-8.0.19.jar" />
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

SWAM

25

AS - WildFly "aggiunta di moduli" [4/4]

L'ultimo passo prevede di registrare il modulo dentro al file principale di WildFly:

- `wildfly-19.0.0.Final/standalone/configuration/standalone.xml`

Basta cercare il TAG `<drivers>` e aggiungervi il seguente blocco:

```
<drivers>
...
<driver name="mysql" module="com.mysql">
  <driver-class>com.mysql.cj.jdbc.Driver</driver-class>
  <xa-datasource-class>com.mysql.cj.jdbc.MysqlXADataSource</xa-datasource-class>
</driver>
</drivers>
```

SWAM

26

AS - WildFly "predisposizione dentro a Eclipse" [1/7]

L'Application Server può essere messo in esecuzione (i.e. *running*) in vari modi:

- tramite **comandi da shell**
 - non lo vedremo
- tramite l'**IDE di Eclipse**
 - più comodo mentre si sviluppa

Per questo motivo, dovremo configurare l'IDE affinché questo metta in esecuzione gli "artefatti" (rappresentati dai WAR file delle applicazioni sviluppate) all'interno di una istanza di WildFly.

SWAM

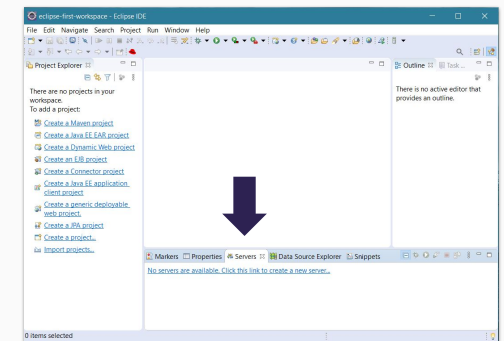
27

AS - WildFly "predisposizione dentro a Eclipse" [2/7]

Tramite l'IDE Eclipse si può **aggiungere l'istanza del server**.

Dalla scheda "Servers" cliccare il link che dice:

*"No servers are available.
Click this link to create a new server..."*



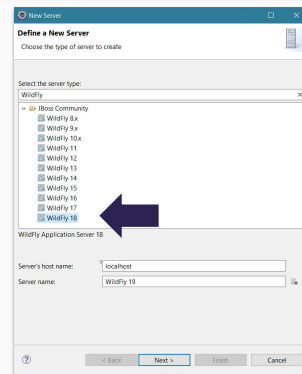
SWAM

28

AS - WildFly "predisposizione dentro a Eclipse" [3/7]

Si apre una **finestra di dialogo** che ci guiderà attraverso i passi.

- si deve selezionare il tipo di AS
 - può accadere (come nello screenshot) che non vi sia nell'elenco la vostra versione
 - WildFly 19
 - **NON** importa !
 - si può procedere

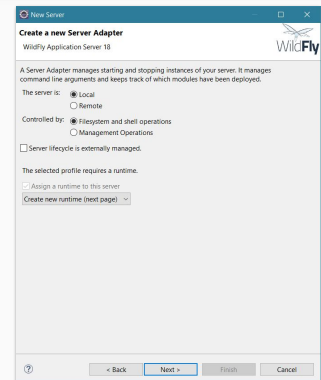


29

AS - WildFly "predisposizione dentro a Eclipse" [4/7]

Si prosegue con le impostazioni di default.

- indicano che l'istanza sarà di tipo **locale**
- e che sarà gestita tramite **filesystem**



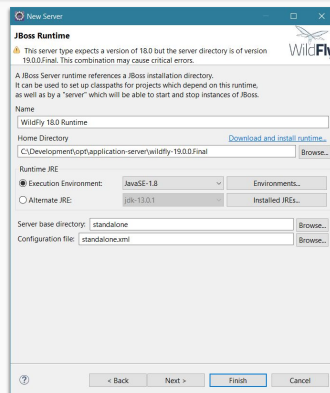
SWAM

30

AS - WildFly "predisposizione dentro a Eclipse" [5/7]

Si prosegue con le impostazioni.

- è importante selezionare la **cartella di installazione di WildFly**
 - non preoccupiamoci della *warning* in alto
- si noti che le configurazioni saranno lette dal solito **standalone.xml**

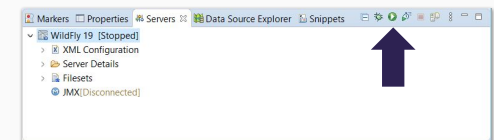


31

AS - WildFly "predisposizione dentro a Eclipse" [6/7]

Al termine della procedura, nella scheda "Servers" compare la **nuova istanza**.

- al momento è **ferma**
- quando ci servirà di farla partire, basterà cliccare il pulsante indicato.



Per ora clicchiamo 2 volte sulla voce **WildFly 19** e si aprirà una scheda di informazioni utili... (segue)

SWAM

32

AS - WildFly "predisposizione dentro a Eclipse" [7/7]

The screenshot shows the WildFly 19 Overview page in Eclipse IDE. The page is divided into several sections:

- Overview**
- General Information**
 - Specify the host name and other common settings.
 - Server name: Wildfly 19
 - Host name: localhost
 - Runtime Environment: Wildfly 18.0 Runtime
 - Open launch configuration
- Management Login Credentials**
 - Set the management login and password for your server.
 - This is used by all management commands, and during server shutdown.
 - User name: admin
 - Password: *****
- Server Behavior**
 - Behavior Profile: Local
 - ☐ Server lifecycle is externally managed.
 - ☐ Listen on all interfaces to allow remote web connections
 - ☒ Expose your management port as the server's host name
- Publishing**
 - Specify the time limit to complete server operations.
 - Timeouts
 - Start (in seconds): 450
 - Stop (in seconds): 450
- Deployment Scanners**
- Application Reload Behavior**
- Server State Detectors**
 - Startup Poller: Web Port
 - Shutdown Poller: Web Port
- Server Ports**
 - The ports entered here are which ports the tools will poll the server on. Changing these fields will not change the ports the server itself listens on.
 - Port Offset: 0
 - ☒ Detect from Local Runtime [Configure...](#)
 - Web: 8080
 - ☒ Detect from Local Runtime [Configure...](#)
 - Management: 9990
 - ☒ Detect from Local Runtime [Configure...](#)
 - ☐ Attach remote debugger
 - Debug Port: 8787

e.g. le porte del server

SWAM

33

E adesso...?

A questo punto si è **quasi predisposto** il proprio ambiente di sviluppo.

Conviene quindi procedere con la creazione di un progetto Java EE che ci serva da "palestra" di test per le configurazioni.

Il progetto che va creato deve essere:


- un **Maven project**
 - ossia, un progetto Java le cui dipendenze e il cui processo di "build" possono essere gestite tramite configurazioni e metadati, servendosi di uno strumento con ruolo di "dependency manager & build automation"

Maven™

SWAM

34

Maven [1/5]



Tool, offerto da *The Apache Software Foundation*, che permette di:

- definire la **struttura** di un progetto Java
- semplificare il processo di costruzione degli **artefatti** Java
 - JAR, EAR o WAR
- favorire la gestione delle **dipendenze** di un'applicazione Java
 - artefatti
 - codici sorgente (i.e. *sources*)
 - documentazione (i.e. *javadocs*)

SWAM

35



- definire la **struttura** di un progetto Java
- semplificare il processo di costruzione degli **artefatti** Java
 - JAR, EAR o WAR
- favorire la gestione delle **dipendenze** di un'applicazione Java
 - artefatti
 - codici sorgente (i.e. *sources*)
 - documentazione (i.e. *javadocs*)

Maven [2/5]

- **Maven Central Repository**

<https://search.maven.org/classic/>

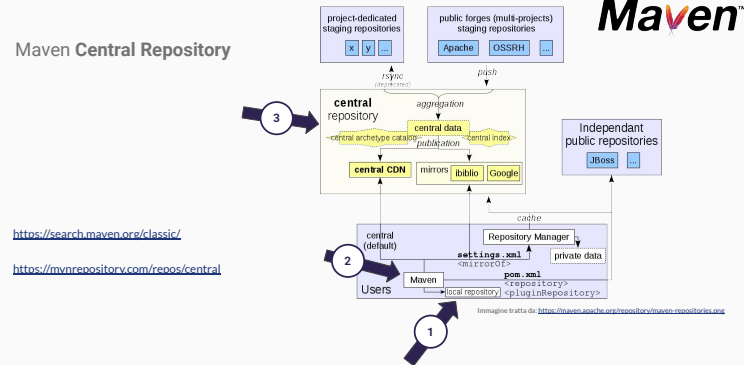
<https://mvnrepository.com/repos/central>

The diagram illustrates the Maven Central Repository architecture. At the bottom, a box labeled 'Users' (1) interacts with 'Maven'. 'Maven' contains 'local repository' and 'Repository Manager'. The 'Repository Manager' handles 'settings.xml', 'pom.xml', and 'pluginRepository'. It also manages 'private data' and 'cache'. The 'Repository Manager' connects to the 'central (default)' repository (2). The 'central (default)' repository contains 'central archetype catalog', 'central data', 'central index', 'central CDN', and 'mirrors'. The 'central (default)' repository connects to the 'central repository' (3). The 'central repository' contains 'central archetype catalog', 'central data', 'central index', 'central CDN', and 'mirrors'. The 'central repository' connects to 'project-dedicated staging repositories' (X, Y, Z) and 'public forges (multi-projects) staging repositories' (Apache, OSSRH, ...). The 'central repository' also connects to 'Independent public repositories' (JBoss, ...). The 'central repository' is connected to 'Mirrors' (liblibto, Google). The 'central repository' is connected to 'Maven' via 'push' and 'sync (optional)'.

Immagine tratta da <https://maven.apache.org/repositories/maven-repository.html>

SWAM

36



<https://myrepository.com/repos/central>

Immagine tratta da: <https://maven.apache.org/repository/maven-repositories.png>

Maven [3/5]

L'unità fondamentale di lavoro per Maven è il cosiddetto **Project Object Model**.

- un file in formato XML (i.e. `pom.xml`)
- conforme all'XML schema <http://maven.apache.org/POM/4.0.0>
- contiene indicazioni su:
 - informazioni del progetto (e.g., version, description, developers e mailing lists)
 - source directory (`src/main/java`) e test source directory (`src/test/java`)
 - resource directory (`src/main/resources`)
 - plugin da attivare
 - build directory

Maven [4/5]

Si notino i TAG:

- `groupId + artifactId + version`
 - identificano il progetto
- `artifactId + packaging`
 - denominano e tipizzano l'artefatto
- `dependencies + dependency`
 - sono autoesplicativi
- `build`
 - struttura il progetto

```
<?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>
  <groupId>it.unife.inf.ats</groupId>
  <artifactId>sample.artifact</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>sample.project.name</name>
  <description>sample.project.description</description>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javax-api</artifactId>
      <version>0.0.1</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <testSourceDirectory>src/test/java</testSourceDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
      </resource>
    </resources>
  </build>
</project>
```

pom.xml

Maven [5/5]

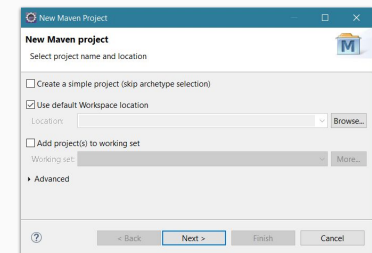
Per ogni `<dependency>` si può indicare uno `<scope>` tra 6 disponibili:

- **compile**
 - *default scope*, che indica che la dipendenza è disponibile in tutti i *classpath* del progetto (i.e. compile, runtime e test)
- **provided**
 - indica che la dipendenza dovrà essere fornita dalla JDK o dall'Application Server durante l'esecuzione ed è disponibile a *compile-time*
- **runtime**
 - indica che la dipendenza non è richiesta per la compilazione ma solo per l'esecuzione
- **test**
 - indica che la dipendenza è richiesta solamente per il testing
- **system**
 - simile a *provided* ma con il vincolo che si punti uno specifico JAR dentro al sistema
- **import**
 - applicabile solamente con il tipo *pom* (ossia per importare altre dipendenze di un altro Maven project)

Creazione di un progetto Maven in Eclipse [1/3]

Dal Workspace di Eclipse si deve:

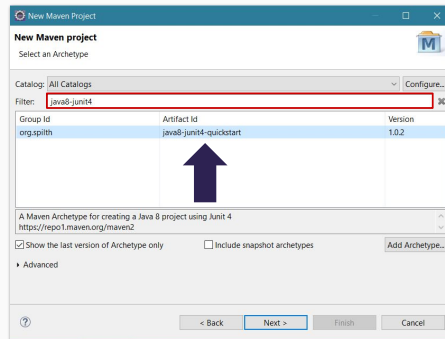
- selezionare **"File" → "New" → "Maven Project"**
- si aprirà una finestra wizard
 - al primo passo lasciare spuntata **"Use default Workspace location"**



Creazione di un progetto Maven in Eclipse [2/3]

Al secondo passo viene chiesto di selezionare un **Archetipo Maven**:

- e.g. java8-junit4
- si riferisce ad un progetto di base predisposto per Java 8 e JUnit 4

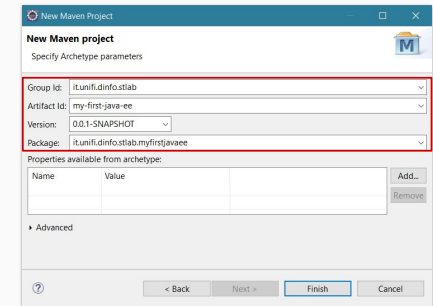


41

Creazione di un progetto Maven in Eclipse [3/3]

Al terzo (e ultimo) passo viene chiesto di indicare per il nuovo progetto:

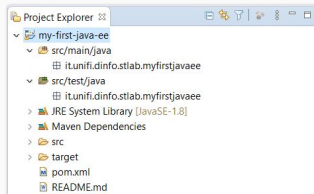
- **Group Id**
- **Artifact Id**
- **Versione**
- **Package di base**



42

Struttura del progetto Maven in Eclipse [1/5]

Il nuovo progetto Maven presenta di default una **struttura minimale**:



Se vogliamo renderlo Java EE (con supporto a JPA) si devono fare alcune configurazioni.

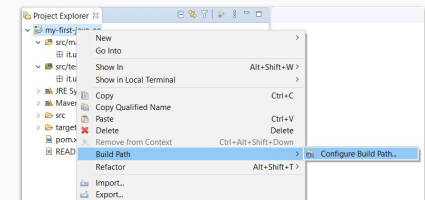
43

Struttura del progetto Maven in Eclipse [2/5]

Innanzitutto, conviene creare una nuova cartella sorgente, che servirà per contenere i file di configurazione XML richiesti da alcune specifiche Java.

La cartella da creare è **src/main/resources**.

- Aprire il menu contestuale del progetto
 - tasto destro del mouse in Win10
- selezionare **"Build Path"**
 - **"Configure Build Path..."**



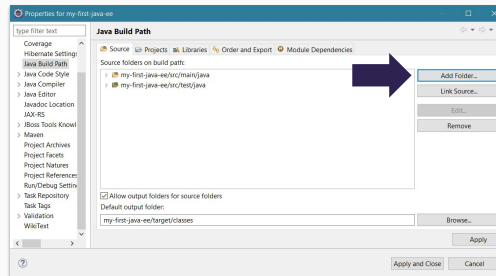
SWAM

44

Struttura del progetto Maven in Eclipse [3/5]

Si apre una finestra di dialogo; spostarsi sulla Scheda "Source".

- cliccare il pulsante "Add Folder..."

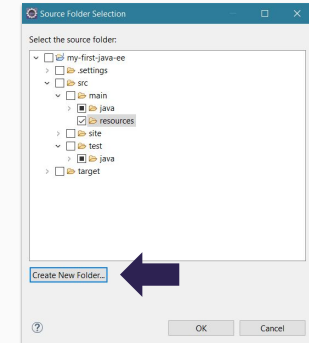


45

Struttura del progetto Maven in Eclipse [4/5]

Dalla ulteriore finestra di dialogo che si apre:

- cliccare "Create New Folder..."
- generare la nuova cartella `src/main/resources`
 - nell'anteprima laterale era già stata creata :)



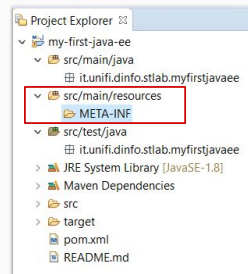
SWAM

46

Struttura del progetto Maven in Eclipse [5/5]

Al termine del wizard la **struttura del progetto** è cambiata:

- conviene anche creare la sotto-cartella `/META-INF`
- servirà a contenere 2 importanti file configurazione
 - `beans.xml`
 - `persistence.xml`



47

Le "Facet base" del progetto [1/4]

Per poter fruire a pieno delle funzionalità offerte dall'IDE, è necessario configurare i **facet**.

- letteralmente "facet" significa sfaccettatura o aspetto
 - la loro configurazione permette all'IDE di capire la "natura" del progetto
- in questo modo, l'IDE potrà **assisterci nello sviluppo** dell'applicazione
 - fornendo una struttura idonea al progetto
 - mettendo a disposizione autocompletamento del codice
 - automatizzando gli import
 - mostrando gli errori nell'utilizzo di sintassi non supportate

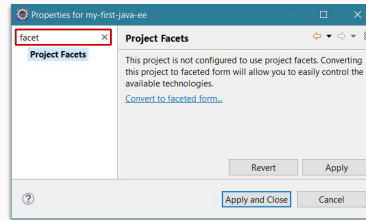
SWAM

48

Le "Facet base" del progetto [2/4]

A questo punto si può attivare la modalità "Facet" del progetto all'interno dell'IDE.

- basta selezionare il progetto
- quindi selezionare "File" → "Properties"
- e nella finestra di dialogo digitare **facet** nel campo di ricerca



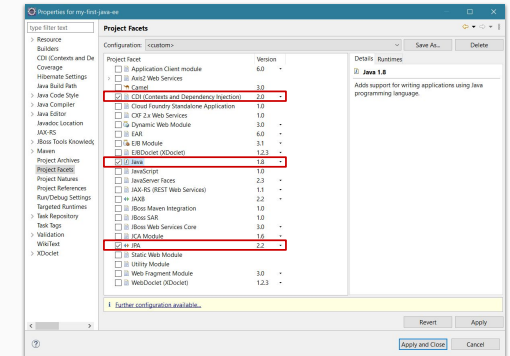
SWAM

49

Le "Facet base" del progetto [3/4]

Dopo aver cliccato "Convert to faceted form..." si apre una schermata dalla quale impostare:

- CDI 2.0
- Java 1.8
- JPA 2.2



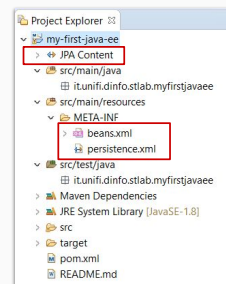
SWAM

50

Le "Facet base" del progetto [4/4]

Dopo aver cliccato "Apply and Close" e concesso il permesso di creare i file di configurazione per CDI e JPA il progetto risulterà cambiato.

- compaiono anche i preannunciati file di configurazione dentro a **src/main/resources/META-INF**



SWAM

51

Aggiungere dipendenze al progetto Maven [1/3]

Volendo provare a sviluppare un'applicazione **JEE** che adotti **JPA** per compiere il processo di *Object-Relational Mapping*, interfacciandosi ad un DBMS relazionale; si devono aggiungere alcune dipendenze al progetto:

- le dipendenze vanno scritte dentro al file **pom.xml**
- in particolare dentro al TAG **<dependencies>**

SWAM

52

Aggiungere dipendenze al progetto Maven [2/3]

Per i nostri scopi bastano le seguenti dipendenze:

<dependency>

<groupId>javax</groupId>

<artifactId>javaxee-api</artifactId>

<version>8.0.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

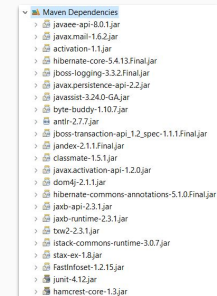
<version>5.4.13.Final</version>

<scope>provided</scope>

</dependency>

Aggiungere dipendenze al progetto Maven [3/3]

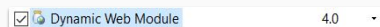
La lista di dipendenze del progetto si è aggiornata:



La "Facet WEB" per il progetto [1/4]

Qualcuno, prima, avrà notato che tra le facet disponibili non abbiamo selezionato quella chiamata **"Dynamic Web Module"**.

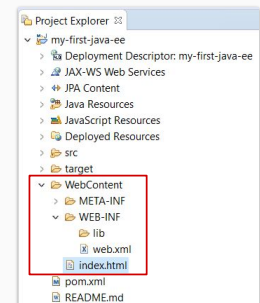
- si deve, adesso, selezionare
- impostando la versione più recente e compatibile con Java EE 8
 - 4.0
- quindi cliccare *"Apply and Close"*



La "Facet WEB" per il progetto [2/4]

In questo modo il progetto diventa **deployable** nell'istanza di WildFly che avevamo predisposto in precedenza.

- la struttura è cambiata nuovamente
- compare la cartella **WebContent**
 - all'interno della quale c'è il file `/WEB-INF/web.xml`
 - dove possiamo includere una "pagina di benvenuto" `index.html`



La "Facet WEB" per il progetto [3/4]

Il file `web.xml` si può lasciare così come è.

La pagina `index.html` potrebbe invece avere questo semplice contenuto:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>my-first-java-ee</title>
  </head>
  <body><h1>Welcome to my-first-java-ee project</h1></body>
</html>
```

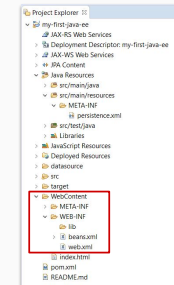
SWAM

57

La "Facet WEB" per il progetto [4/4]

Per come è scritto nella documentazione ufficiale, quando si sviluppa un JEE WAR (https://docs.jboss.org/cdi/spec/2.0/cdi-spec.html#packaging_deployment_ee)

- si dovrebbe porre all'interno della cartella `/WEB-INF` il file `beans.xml`
 - spostandolo dalla cartella `src/main/resources/META-INF`
 - nell'altro caso, valeva per JAR e EAR



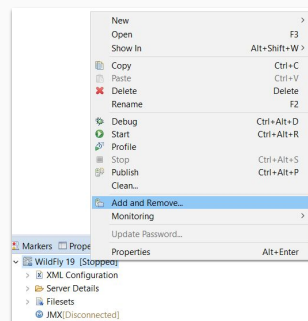
SWAM

58

Proviamo il "deployment" del WAR [1/4]

A questo punto è possibile provare a compiere la **distribuzione sull'istanza di WildFly**.

- aprire il menu contestuale dell'istanza dell'Application Server
- selezionare *"Add and Remove..."*



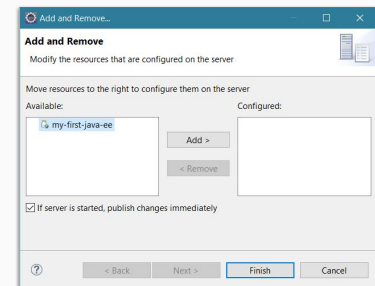
SWAM

59

Proviamo il "deployment" del WAR [2/4]

Si apre una finestra di dialogo per il deployment:

- selezionare `my-first-java-ee`
- cliccare *"Add"*
- quindi *"Finish"*

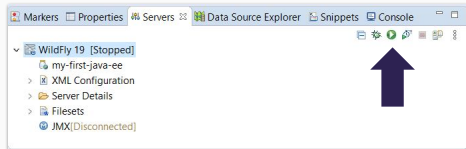


SWAM

60

Proviamo il "deployment" del WAR [3/4]

Si deve adesso **avviare WildFly**; ossia cliccare sul pulsante visto prima:



All'avvio la **console** di Eclipse viene popolata di avvisi utili; tra le righe troviamo:

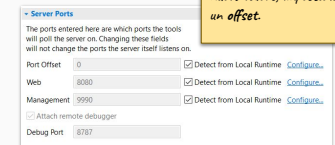
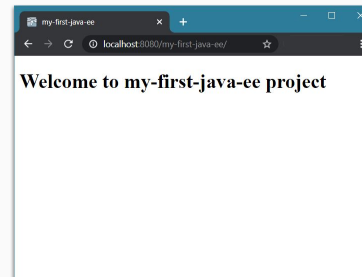
```
WildFly Full 19.0.0.Final (WildFly Core 11.0.0.Final) starting
...
Registered web context: '/my-first-java-ee' for server 'default-server'
Deployed "my-first-java-ee.war" (runtime-name : "my-first-java-ee.war")
```

SWAM

61

Proviamo il "deployment" del WAR [4/4]

Apriamo il browser web, all'indirizzo: `http://localhost:8080/my-first-java-ee`



Se la porta 8080 del SO risultasse già occupata; è possibile configurare un nuovo valore, impostando un offset.

SWAM

62

Per i più curiosi !

Chi navigherà tramite il proprio filesystem verso la cartella dedicata ai deployment di WildFly si accorgerà che è comparsa una **cartella .WAR** del progetto.

`wildfly-19.0.0.Final/standalone/deployments`



- la cartella .WAR è navigabile ulteriormente; poiché siamo in "modalità sviluppo"
- in un rilascio di produzione, il file .WAR risulterà invece un archivio compresso

SWAM

63

ORM in action [1/12]

Ora che abbiamo predisposto il progetto Maven per essere anche eseguibile sull'istanza di WildFly, non ci resta che provare ad attuare un processo di **Object-Relational Mapping**.

Procederemo così:

- il DBMS di riferimento è **MySQL**
 - creeremo tramite il *MySQL Workbench* uno schema chiamato "java-ee-schema"
 - autorizzeremo su tale schema un utente "java-client" con password "password"
- configureremo il file **persistence.xml**
 - metteremo in collegamento con un file datasource "my-first-java-ee-ds.xml"
- annoteremo una classe POJO del modello di dominio con JPA

SWAM

64

ORM in action [2/12]

I primi passi relativi a MySQL (e il suo Workbench) qui **NON** sono documentati poiché la documentazione ufficiale li descrive in modo semplice e chiaro.

Partiamo a configurare il file `persistence.xml`:

- dovrà contenere il binding con un file `datasource`
- dovrà elencare le classi che dovranno essere messe in ORM
- dovrà contenere alcune proprietà di *Hibernate* per istruire l'implementazione di JPA a compiere il suo lavoro in background

SWAM

65

ORM in action [3/12]

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"                                persistence.xml
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="my-first-java-ee" transaction-type="JTA">
    <jta-data-source>myFirstJavaEeDS</jta-data-source>

    <class>it.unifi.dinfo.stlab.myfirstjavaee.model.User</class>

    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect" />
      <property name="hibernate.hbm2ddl.auto" value="create"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.use_sql_comments" value="true" />
      <property name="hibernate.default_batch_fetch_size" value="16" />
      <property name="hibernate.jdbc.batch_size" value="100" />
      <property name="hibernate.order_inserts" value="true" />
      <property name="hibernate.order_updates" value="true" />
      <property name="hibernate.id.new_generator_mappings" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

SWAM

66

ORM in action [4/12]

Avrete notato che un attributo nel file `persistence.xml` indicava:

```
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect" />
```

Il valore dipende dal DBMS specifico:

- indica il dialetto (ossia la sintassi) con la quale saranno definite per l'esecuzione le query SQL
- esistono più dialetti (anche differenziati per versione)
 - `H2Dialect`
 - `MariaDBDialect`
 - `MySQL5Dialect` / `MySQL8Dialect`
 - `PostgreSQL9Dialect`

SWAM

67

ORM in action [5/12]

Avrete notato che un attributo nel file `persistence.xml` indicava:

```
<property name="hibernate.hbm2ddl.auto" value="create" />
```

I valori possibili sono:

- **create-drop e create:** ricreano lo schema allo startup dell'applicazione, riazzinando i dati
 - utili per release di sviluppo, se accompagnati da una inizializzazione del DB in fase di startup dell'applicazione
- **update:** aggiorna lo schema
 - utile per release di produzione
- **validate:** non crea né aggiorna lo schema, ma ne verifica la conformità
- **none:** non effettua alcun cambiamento né validazione sullo schema

SWAM

68

ORM in action [6/12]

Adesso vediamo di comprendere il contenuto del file datasource che dovremo chiamare "my-first-java-ee-ds.xml" e collocare in my-first-java-ee/datasource/:

- dovrà contenere la URL dello schema MySQL precedentemente creato
 - jdbc:mysql://localhost:3306/java-ee-schema
- dovrà selezionare un driver, indicando il nome di quello registrato in precedenza dentro alla configurazione standalone.xml di WildFly
 - mysql
- dovrà specificare le **credenziali** dell'utente autorizzato ad accedere allo schema MySQL

ORM in action [7/12]

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources xmlns="http://www.jboss.org/ironjacamar/schema"                  my-first-java-ee-ds.xml
             xsi:schemaLocation="
               http://www.jboss.org/ironjacamar/schema
               http://docs.jboss.org/ironjacamar/schema/datasources_1_0.xsd">
  <datasource jndi-name="myFirstJavaEeDS" enabled="true" use-java-context="true" pool-name="myFirstJavaEeDS">
    <connection-url>jdbc:mysql://localhost:3306/java-ee-schema?serverTimezone=UTC</connection-url>
    <driver>mysql</driver>
    <security>
      <user-name>java-client</user-name>
      <password>password</password>
    </security>
  </datasource>
</datasources>
```

ORM in action [8/12]

Adesso si deve annotare la classe POJO di prova User.java:

- era infatti l'unica classe elencata dentro a persistence.xml
- per la prova ci accontenteremo di modellare un Utente semplice
 - che possiede solo questi attributi
 - id
 - username
 - email
 - password

ORM in action [9/12]

```
@Entity
@Table(name = "users")
public class User {
    private long id;
    private String username;
    private String email;
    private String password;

    @Id
    @Column(name = "id", nullable = false)
    @NotNull
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

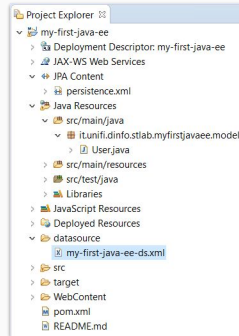
Le annotazioni JPA usate sono prese da:

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
```

ORM in action [10/12]

Prima di far partire WildFly e vedere se il processo di ORM avviene all'avvio:

- si deve fare il **deploy** anche del file **datasource**
- basta aprire il menu contestuale (in Win10 cliccando sopra col tasto destro)
 - selezionare "Mark as Deployable"
 - verificando che venga aggiunto all'istanza del server



SWAM

73

ORM in action [11/12]

Una volta avviato WildFly, la **console** di Eclipse riporta anche **informazioni di mapping**:

```
[org.jboss.as.clustering.infinispan (ServerService Thread Pool -- 79) WFLYCLINF0002: Started client-mappings cache from ejb container
[org.hibernate.annotations.common.Version (ServerService Thread Pool -- 78) HCANN000001: Hibernate Commons Annotations (5.0.5.Final)
[org.jboss.as.jpa (ServerService Thread Pool -- 78) WFLYJPA00002: Starting Persistence Unit (phase 2 of 2) Service 'my-first-java-ee-war/my-first-java-ee'
[org.hibernate.dialect.Dialect (ServerService Thread Pool -- 78) HH00000408: Using dialect: org.hibernate.dialect.MySQLInnoDBDialect
[org.hibernate.envers.boot.internal.EnversServiceImpl (ServerService Thread Pool -- 78) Envers integration enabled! : true
[stdout] (ServerService Thread Pool -- 78) Hibernate:

[stdout] (ServerService Thread Pool -- 78)

[stdout] (ServerService Thread Pool -- 78) drop table if exists users

[stdout] (ServerService Thread Pool -- 78) Hibernate:

[stdout] (ServerService Thread Pool -- 78)

[stdout] (ServerService Thread Pool -- 78) create table users (

[stdout] (ServerService Thread Pool -- 78) id bigint not null,

[stdout] (ServerService Thread Pool -- 78) email varchar(255),

[stdout] (ServerService Thread Pool -- 78) password varchar(255),

[stdout] (ServerService Thread Pool -- 78) username varchar(255),

[stdout] (ServerService Thread Pool -- 78) primary key (id)

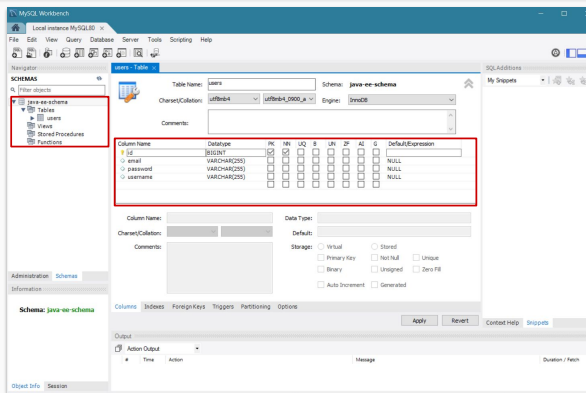
[stdout] (ServerService Thread Pool -- 78) ) engine=InnoDB
```

SWAM

74

ORM in action [12/12]

E se controlliamo
il MySQL
Workbench:



SWAM

75

Miscela di Java EE

Provatela direttamente



SWAM

76