

Spring Framework

Βασίλειος Καραβασίλης
Μονάδα Αριστείας ΕΛΛΑΚ | ΕΤΕΠΗ | 3/11/2014

Τι είναι;

- Είναι μια συλλογή από βιβλιοθήκες σε java.
- Απλοποιεί την διαδικασία ανάπτυξης web εφαρμογών σε Java Enterprise Edition.
- Γενικά, είναι περίπλοκο στην χρήση του.
- Άδεια: Apache License 2.0
- Πληροφορίες:
 - <https://spring.io>

Γιατί να το χρησιμοποιήσουμε

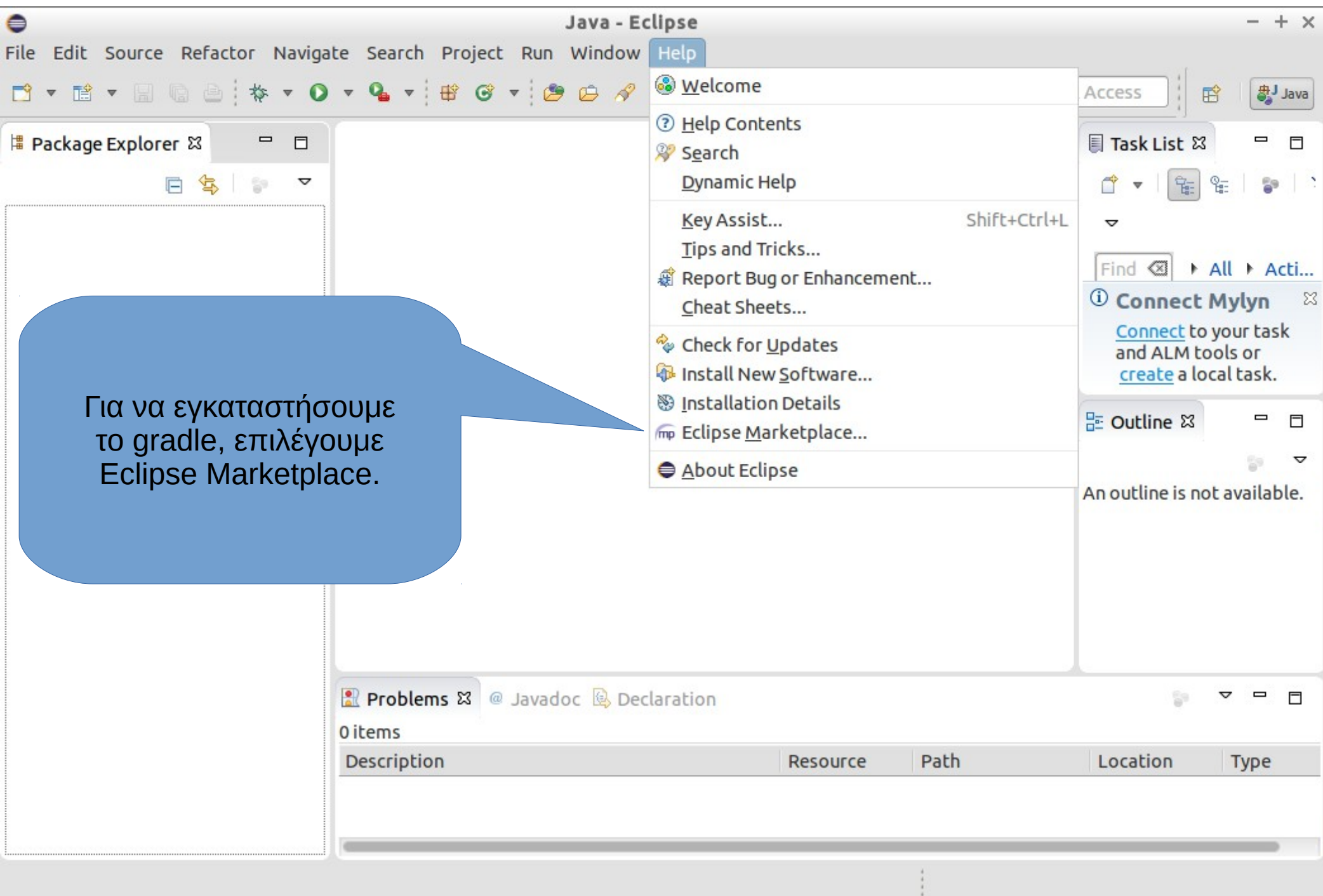
- Απλοποιεί την διαδικασία ανάπτυξης JEE εφαρμογών.
- Μπορεί να συνεργαστεί με άλλα framework (πχ για σύνδεση σε κάποια βάση μέσω hibernate).
- Βασικό πλεονέκτημα: Ευέλικτες εφαρμογές:
 - Μπορούμε να διαχωρίσουμε τον κώδικα από το configuration.
 - Πχ το password της βάσης να μην το βάλουμε μέσα στον κώδικα της εφαρμογής, αλλά σε ένα configuration file.

Δομή

- Περιέχει πολλά projects
 - <https://spring.io/projects>
- Έχει IDE βασισμένο σε eclipse
 - <https://spring.io/tools/sts>
- Χρησιμοποιεί Maven ή Gradle για να κατεβάσει διάφορα jar που χρειάζεται η εφαρμογή, να ελέγξει και να κατασκευάσει την τελική εφαρμογή
 - <http://maven.apache.org>
 - <http://www.gradle.org>

Τι θα παρουσιάσουμε

- Web εφαρμογή που θα χρησιμοποιεί gradle και το project spring boot
- Gradle:
 - Είναι πιο καινούριο από το Maven, οπότε υπάρχουν λιγότερες πληροφορίες στο internet.
 - Γενικά στο μέλλον πιστεύουμε ότι θα χρησιμοποιείται περισσότερο (πχ Android studio).
 - Έχει σύνταξη που μοιάζει με java (αντί για XML).
- Spring boot (<http://projects.spring.io/spring-boot/>)
 - Απλοποιεί **πάρα-πάρα** πολύ την διαδικασία ανάπτυξης μιας spring εφαρμογής.
 - Μπορεί να πράξει εκτελέσιμο jar αντί για war (ουσιαστικά παράγει ένα jar με τον web server και τα αρχεία του site μαζί).



Γράφουμε gradle

Eclipse Marketplace

to install. Press Finish to proceed with installation.
Pre information button to see a detailed overview and a link to more information.

Search Recent Popular Installed September 09/25

Find: All Markets All Categories Go



Gradle IDE Pack 3.6.x, 0.17
Install Pivotal Gradle IDE & Enide Gradle for Eclipse in on
Nodeclipse/Enide Gradle for Eclipse - Gradle Integration
[more info](#)
by Nodeclipse/Enide, EPL
[gradle IDE editor](#) [run build](#)
★ 4 Installs: 23.1K (4,423 last month)



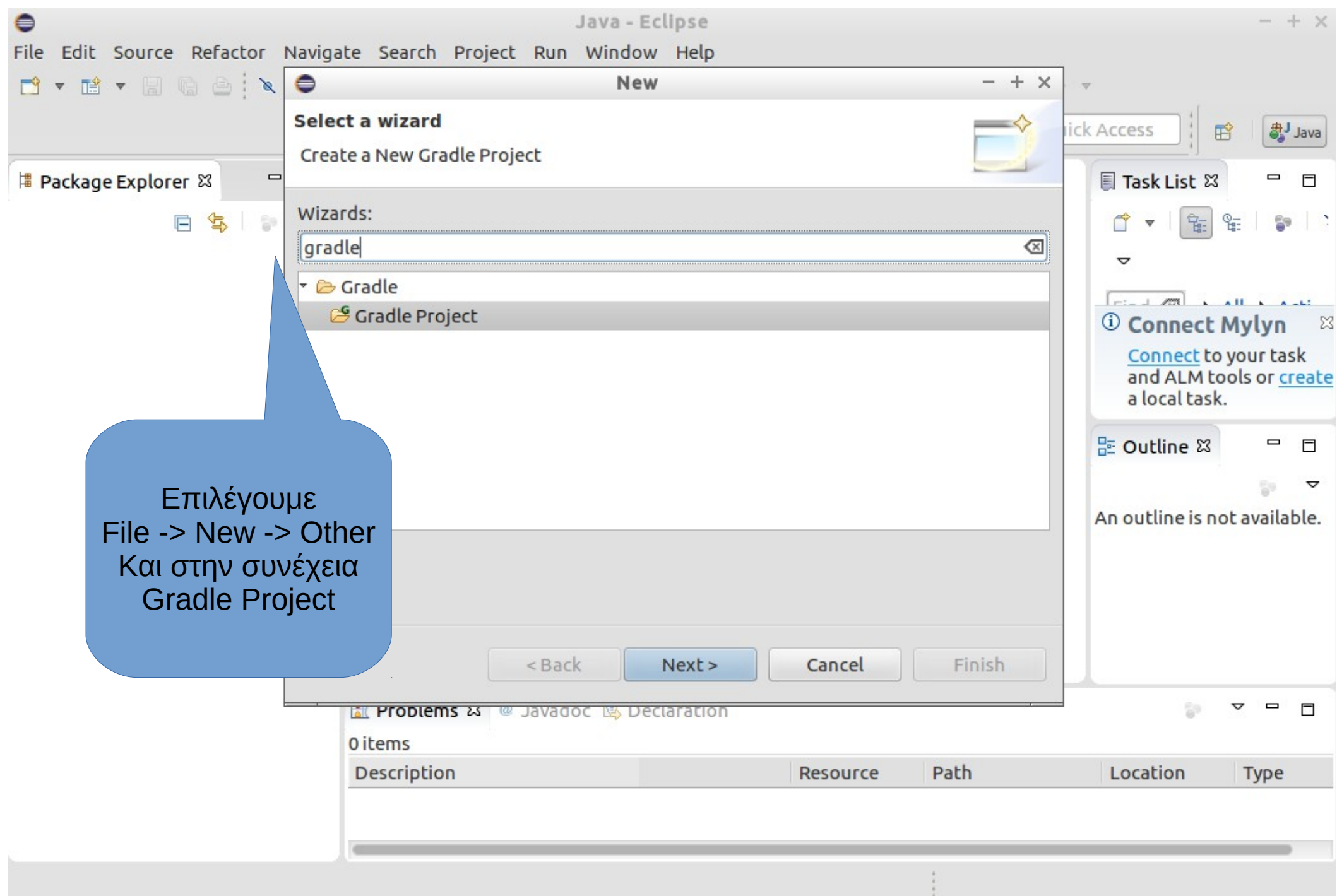
Nodeclipse/Enide Gradle for Eclipse 0.17
For any project that gradle can build. Launch build, gradle GUI, run Jetty or deploy
Android .apk to AVD by right-clicking build.gradle. Project does not need to...
[more info](#)
by Nodeclipse/Enide, Other Open Source

Marketplaces

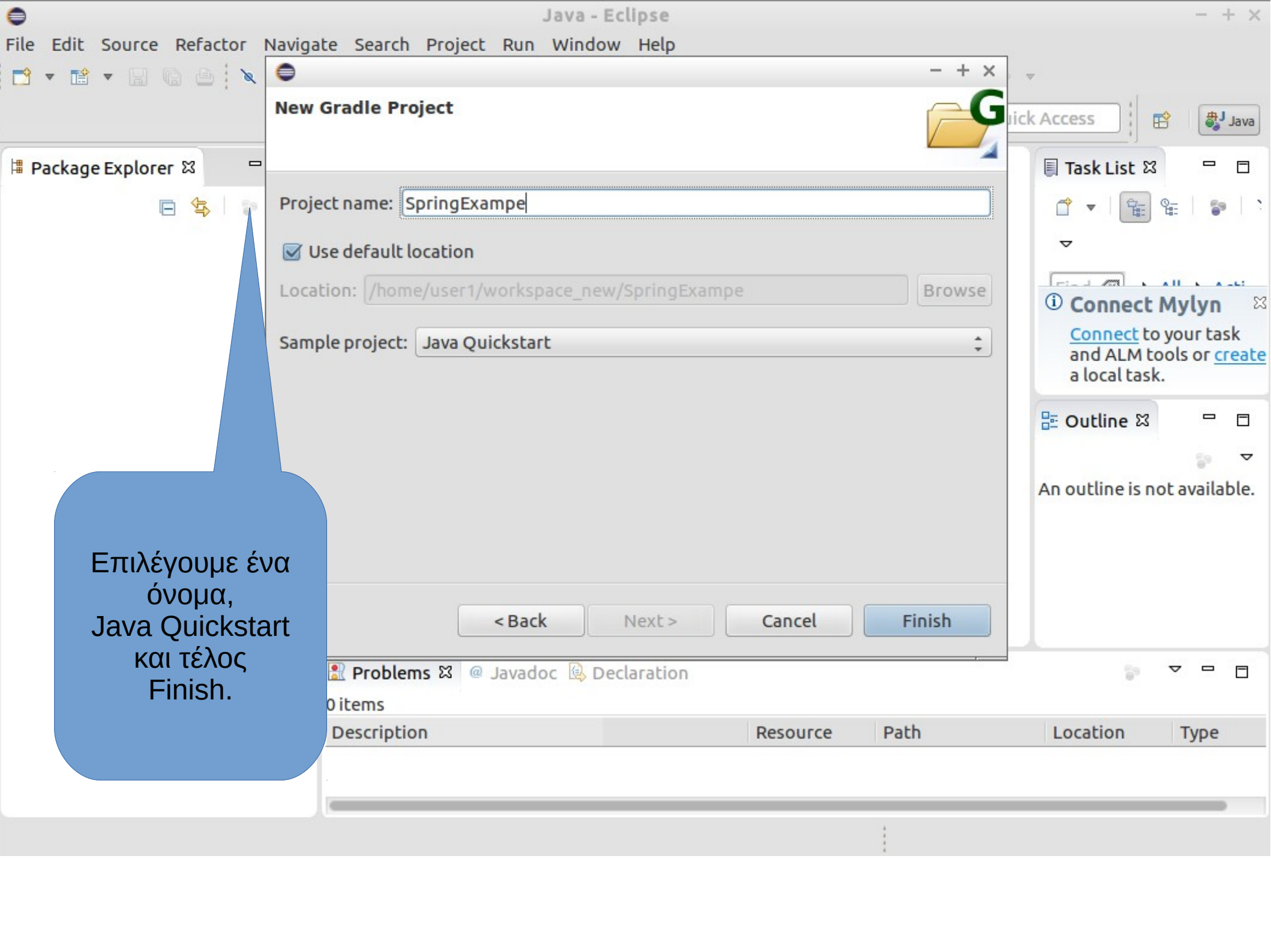


? < Back Install Now > Cancel Finish

Επιλέγουμε Gradle IDE Pack
και στην συνέχεια Install



Επιλέγουμε
File -> New -> Other
Και στην συνέχεια
Gradle Project



New Gradle Project

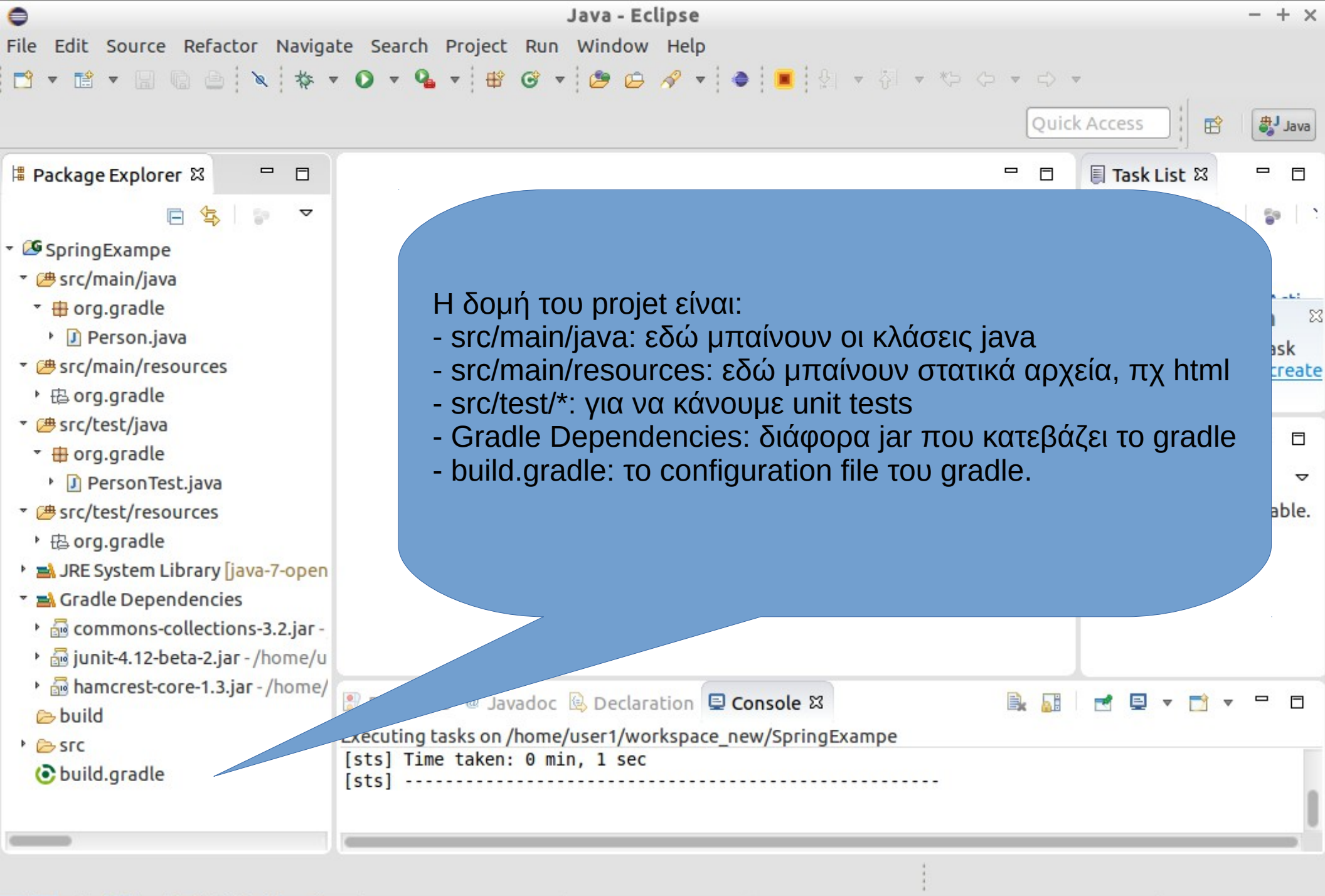
Project name:

☒ Use default location

Location:

Sample project:

Επιλέγουμε ένα
όνομα,
Java Quickstart
και τέλος
Finish.



Αρχείο build.gradle

- Εδώ ορίζουμε διάφορα πράγματα που αφορούν το project.
- Μπορούμε να ορίσουμε:
- repositories: από πού θα κατεβάζει τα jar
- dependencies: ποια jar θα κατεβάσει
- Το όνομα του project
- Τι εκτελέσιμο θα κατεβάσει
- Και άλλα



Quick Access



build.gradle

```
1 apply plugin: 'java'
2 apply plugin: 'eclipse'
3
4 sourceCompatibility = 1.5
5 version = '1.0'
6 jar {
7     manifest {
8         attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': version
9     }
10 }
11
12 repositories {
13     mavenCentral()
14 }
15
16 dependencies {
17     compile group: 'commons-collections', name: 'commons-collections', version: '3.2'
18     testCompile group: 'junit', name: 'junit', version: '4.+'
19 }
20
21 test {
22     systemProperties 'property': 'value'
23 }
24
25 uploadArchives {
26     repositories {
27         flatDir {
28             dirs 'repos'
29         }
30     }
31 }
```

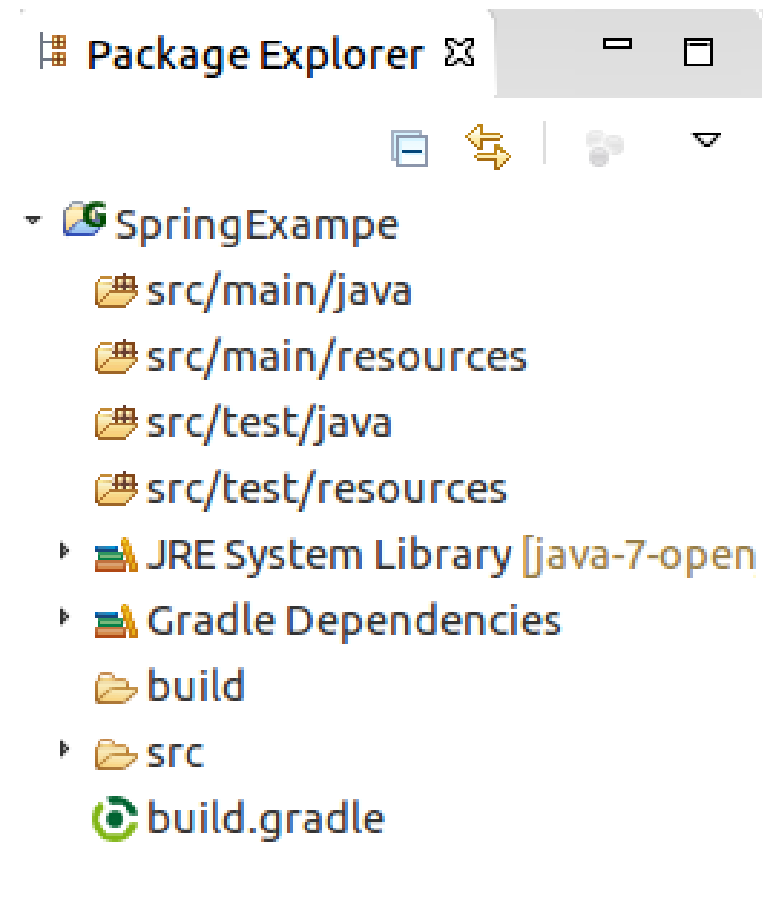
Writable

Insert

1:1

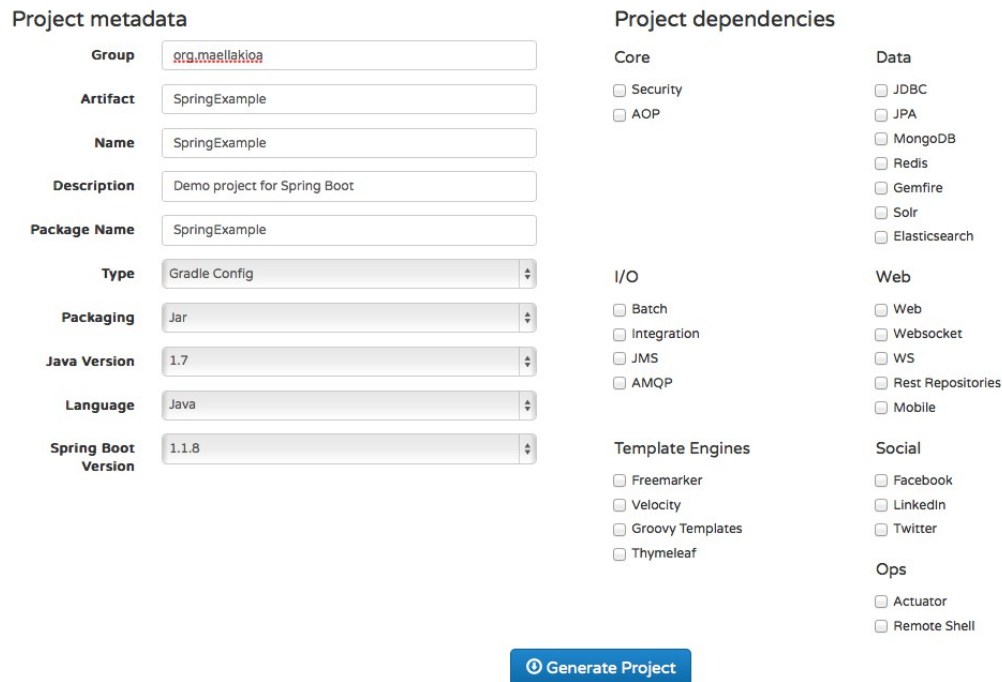
Αλλαγές που κάνουμε για το spring boot

- Σβήνουμε τα αρχεία που δεν χρειαζόμαστε (σχεδόν όλα).



Αλλαγές που κάνουμε για το spring boot

- Στην σελίδα <https://spring.io/guides/gs/spring-boot/> έχει οδηγίες για το πως φτιάχνουμε ένα project για spring boot.
- Στην σελίδα <http://start.spring.io> μπορούμε να φτιάξουμε το βασικό build.gradle αυτόματα και να επιλέξουμε αν θέλουμε επιπλέον πακέτα.



The image shows the 'Spring Boot Start' web form. It is divided into two main sections: 'Project metadata' on the left and 'Project dependencies' on the right. The 'Project metadata' section includes fields for Group (org.maellakioa), Artifact (SpringExample), Name (SpringExample), Description (Demo project for Spring Boot), Package Name (SpringExample), Type (Gradle Config), Packaging (Jar), Java Version (1.7), Language (Java), and Spring Boot Version (1.1.8). The 'Project dependencies' section is organized into several categories with checkboxes: Core (Security, AOP), Data (JDBC, JPA, MongoDB, Redis, Gemfire, Solr, Elasticsearch), I/O (Batch, Integration, JMS, AMQP), Template Engines (Freemarker, Velocity, Groovy Templates, Thymeleaf), Web (Web, Websocket, WS, Rest Repositories, Mobile), Social (Facebook, LinkedIn, Twitter), and Ops (Actuator, Remote Shell). A blue 'Generate Project' button is located at the bottom center.

Project metadata		Project dependencies	
Group	org.maellakioa	Core	
Artifact	SpringExample	☐ Security	
Name	SpringExample	☐ AOP	
Description	Demo project for Spring Boot		Data
Package Name	SpringExample		☐ JDBC
Type	Gradle Config		☐ JPA
Packaging	Jar		☐ MongoDB
Java Version	1.7		☐ Redis
Language	Java		☐ Gemfire
Spring Boot Version	1.1.8		☐ Solr
			☐ Elasticsearch
		I/O	Web
		☐ Batch	☐ Web
		☐ Integration	☐ Websocket
		☐ JMS	☐ WS
		☐ AMQP	☐ Rest Repositories
			☐ Mobile
		Template Engines	Social
		☐ Freemarker	☐ Facebook
		☐ Velocity	☐ LinkedIn
		☐ Groovy Templates	☐ Twitter
		☐ Thymeleaf	
			Ops
			☐ Actuator
			☐ Remote Shell

[Generate Project](#)

Το build.gradle που κατεβάσαμε και επεξεργαστήκαμε

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:1.1.8.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'spring-boot'

jar {
    baseName = 'SpringExample'
    version = '0.0.1-SNAPSHOT'
}
sourceCompatibility = 1.7
targetCompatibility = 1.7

repositories {
    mavenCentral()
}

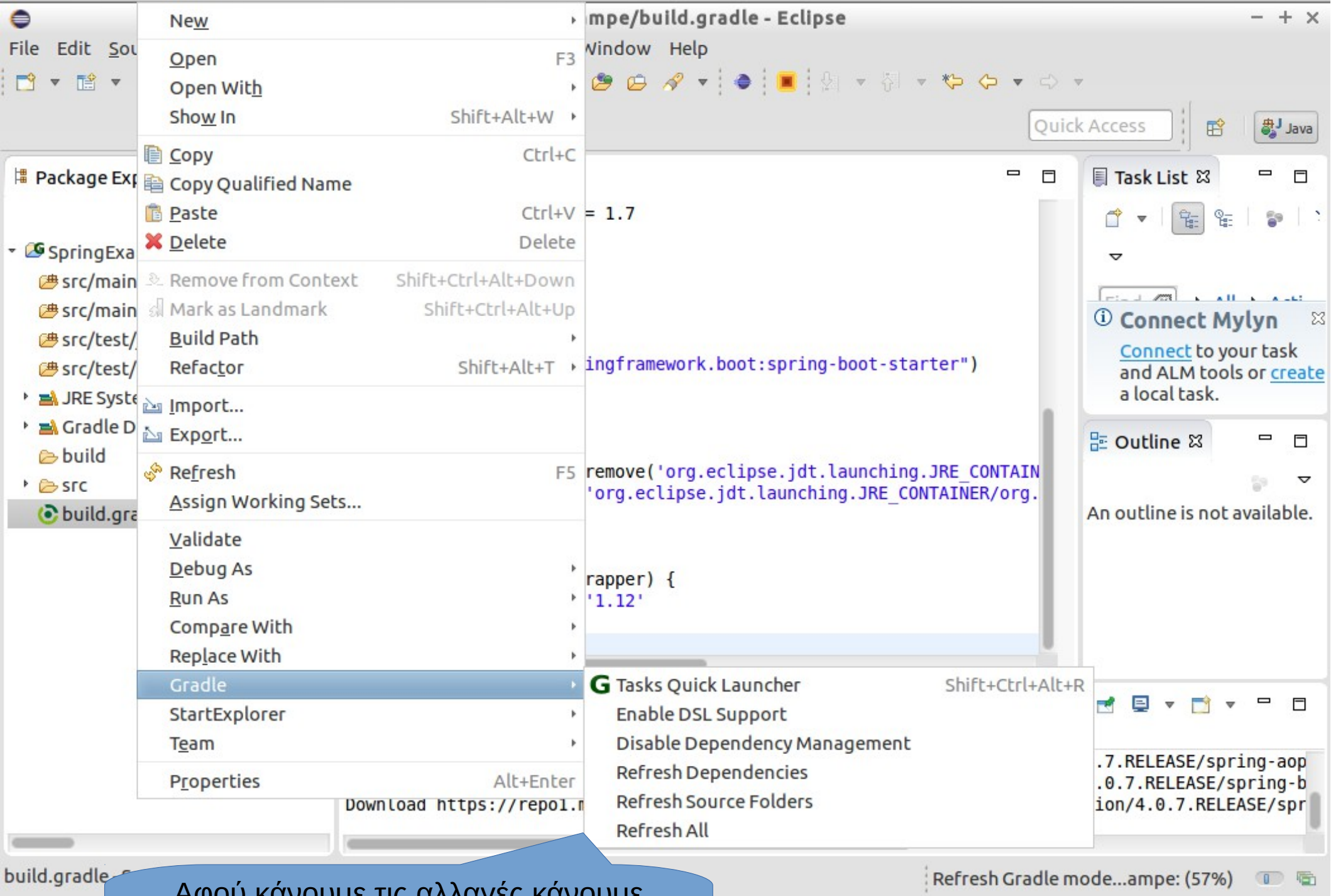
dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
}

eclipse {
    classpath {
        containers.remove('org.eclipse.jdt.launching.JRE_CONTAINER')
        containers 'org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.7'
    }
}

task wrapper(type: Wrapper) {
    gradleVersion = '1.12'
}
```

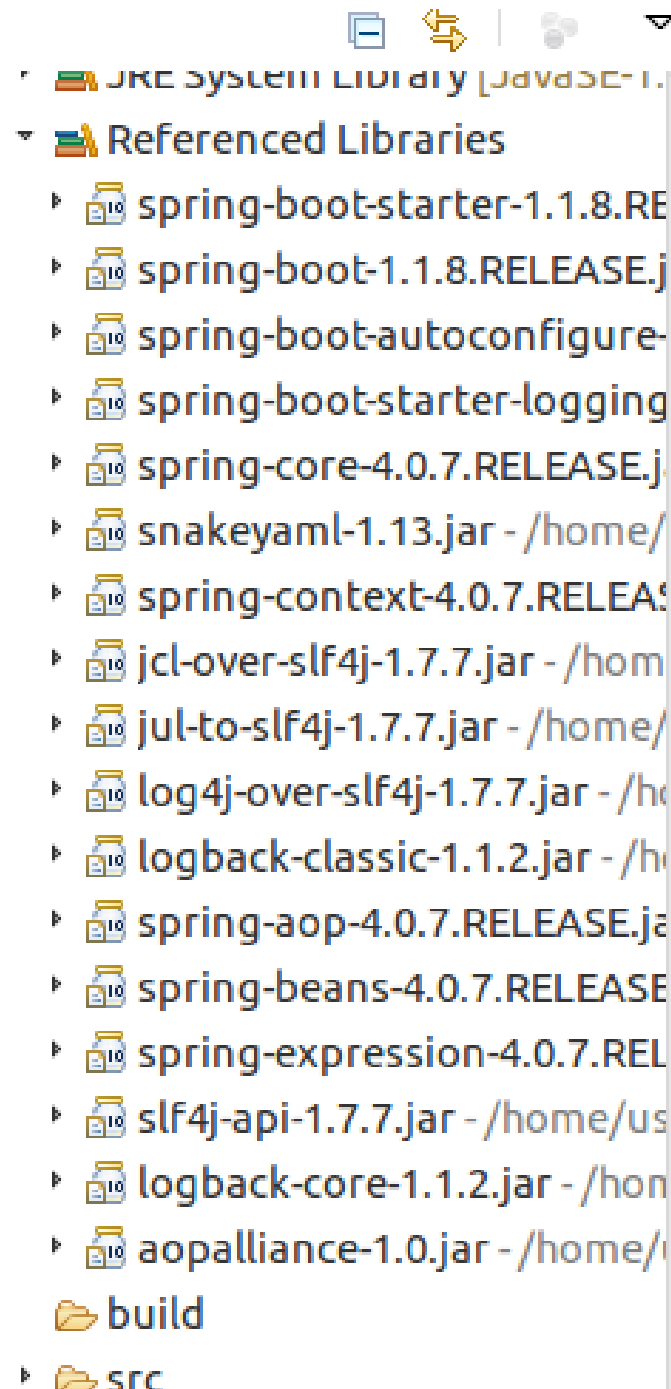
Αντιστοιχεί στο maven dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
```

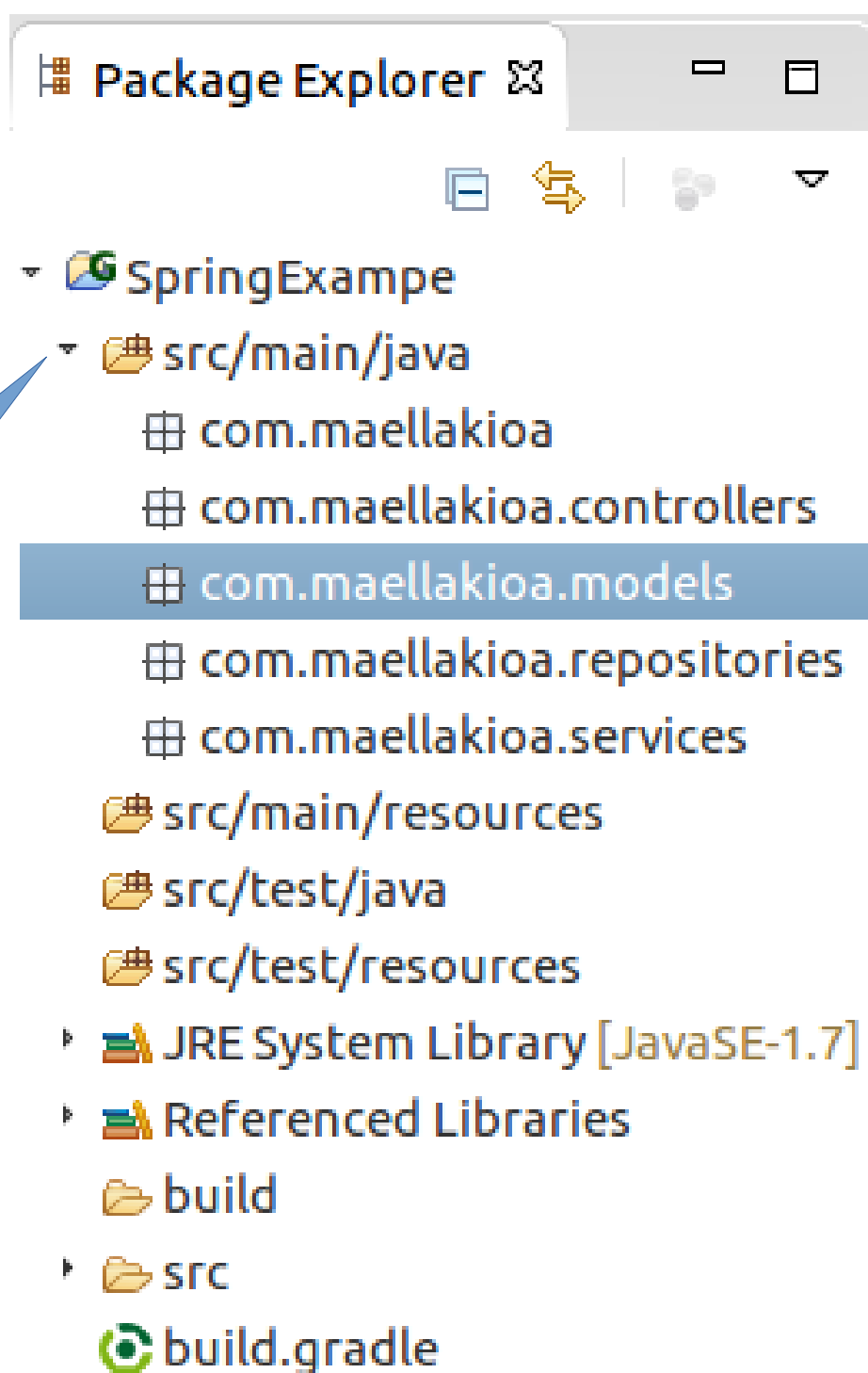



Αφού κάνουμε τις αλλαγές κάνουμε δεξί κλικ στο build.gradle και επιλέγουμε Gradle -> Refresh All

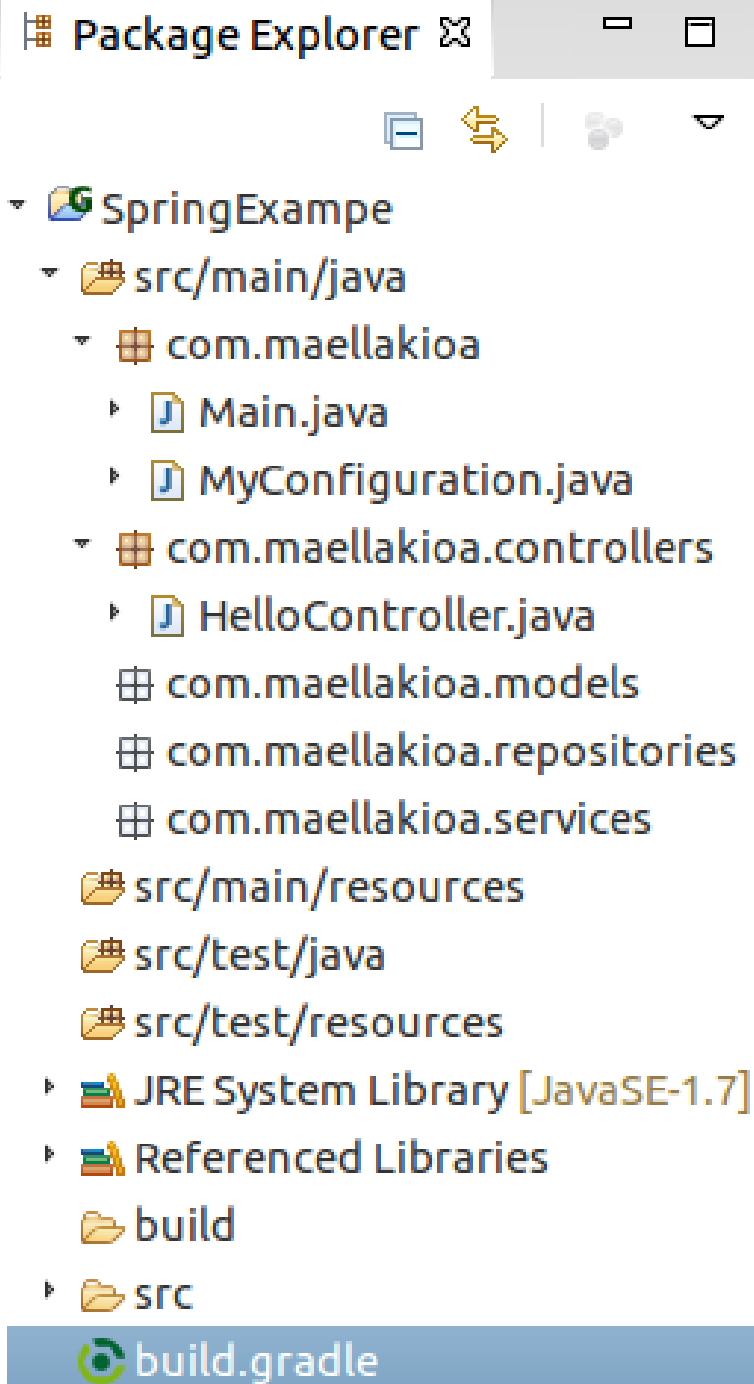
Κατέβηκαν όλα τα
απαραίτητα jar στο
References Libraries



Φτιάχνουμε μερικά πακέτα
για την java.
Το καθένα θα έχει
κλάσεις με συγκεκριμένες
λειτουργίες.



Φτιάχνουμε
και κάποια
αρχεία.



Main.java

```
package com.maellakioa;

import
org.springframework.boot.SpringApplication;

public class Main {
    public static void main(String[] args) throws
Exception {
        SpringApplication app = ne
SpringApplication(MyConfiguration.class);
app.setShowBanner(false);
app.run(args);
    }
}
```

Την κλάση MyConfiguration θα την
δούμε στην συνέχεια.
Ουσιαστικά ξεκινά τον tomcat server
με το configuration που θέλουμε.

MyConfiguration.java

```
package com.maellakioa;

import
org.springframework.boot.autoconfigure.EnableAutoConfigu
ration;
import
org.springframework.context.annotation.ComponentScan;
import
org.springframework.context.annotation.Configuration;

@Configuration
@EnableAutoConfiguration
@ComponentScan
public class MyConfiguration {

}
```

Αυτά τα annotations λένε τι να κάνει.

Αν θέλουμε να δημιουργήσουμε επιπλέον beans ή να αλλάξουμε κάτι άλλο, το κάνουμε μέσα στην κλάση.

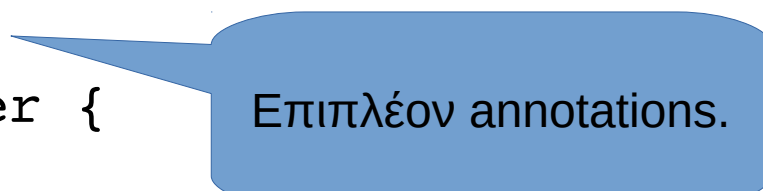
HelloController.java

```
package com.maellakioa.controllers;

import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @RequestMapping("/hello")
    public String index() {
        return "Greetings from Spring Boot!";
    }
}
```

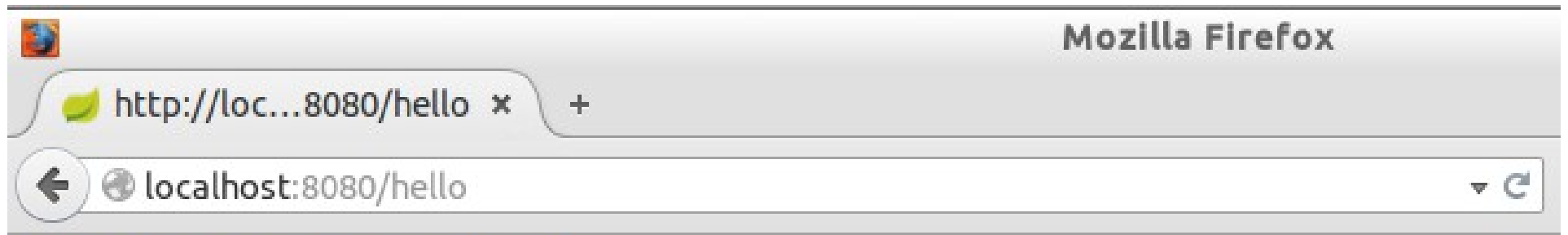


Επιπλέον annotations.

Έλεγχος

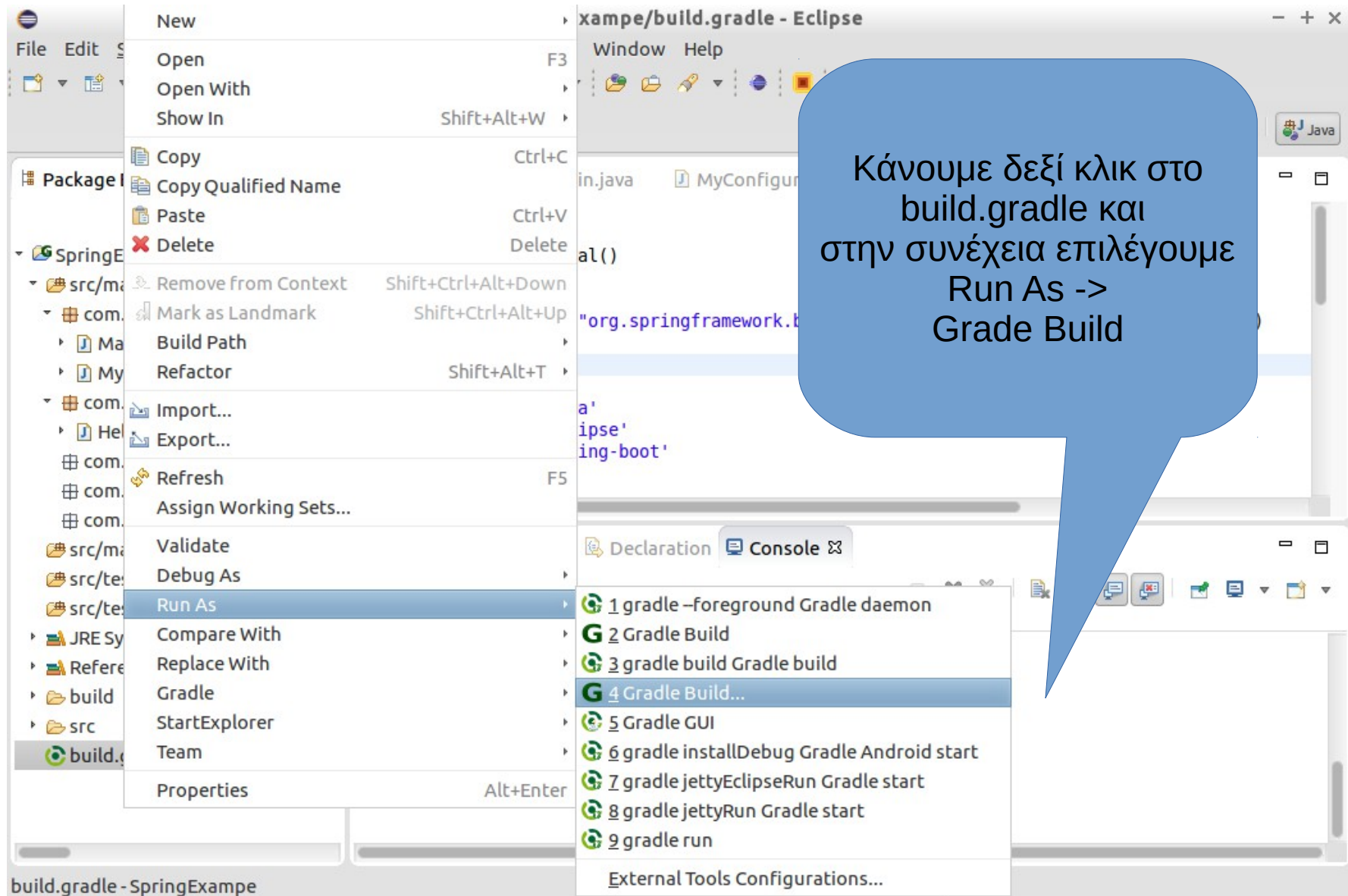
- Κάνουμε δεξί κλικ στο Main.java, Επιλέγουμε Run As -> Java Application.
- Εμφανίζονται κάποια μηνύματα στην κονσόλα.
- Περιμένουμε να εμφανιστεί το μήνυμα:
 - Started Main in 12.345 seconds
- Ουσιαστικά, έχει ξεκινήσει ο web server στην πόρτα 8080.
- Αν πάμε στην διεύθυνση

Τι θα δούμε



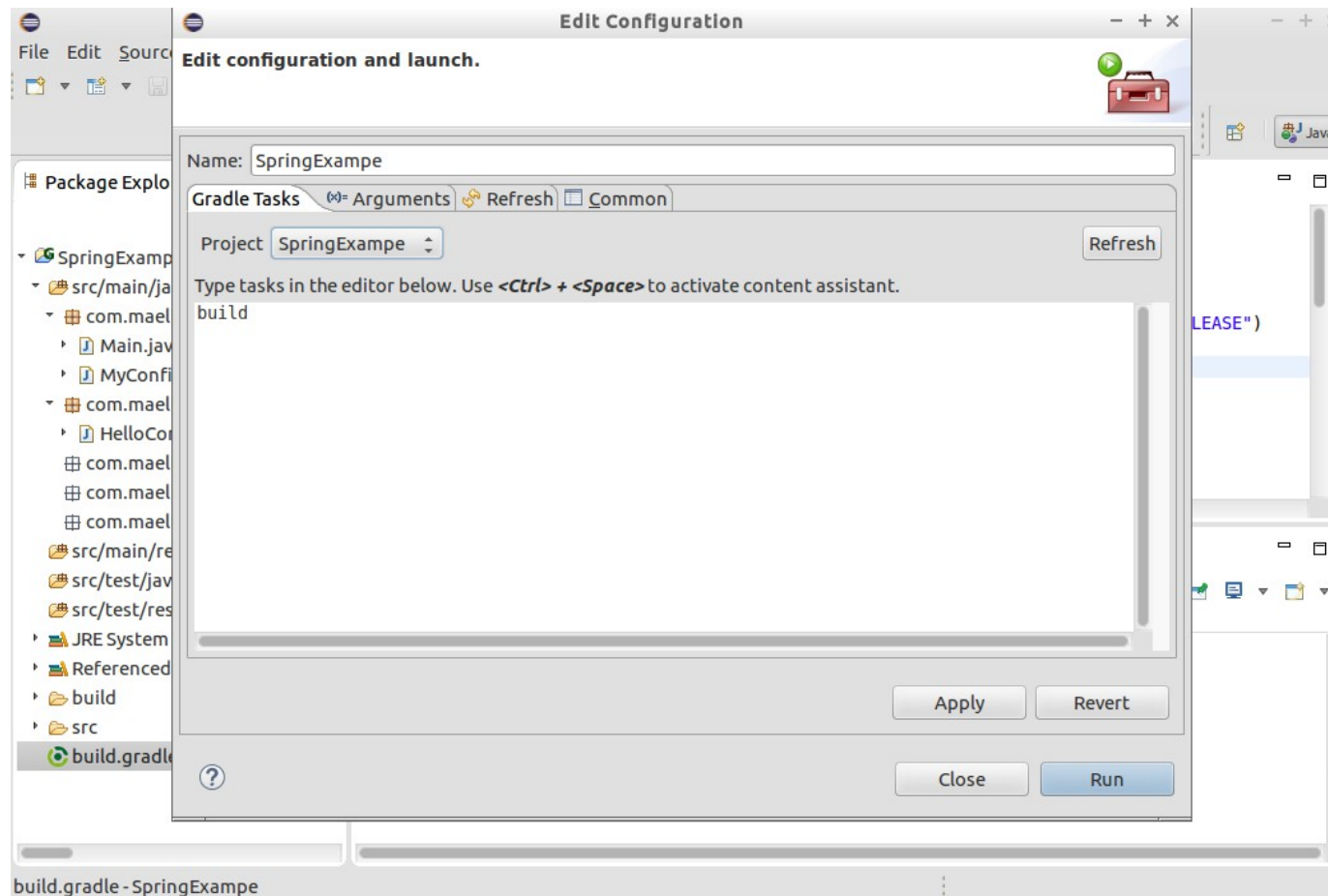
Greetings from Spring Boot!

Δημιουργία εκτελέσιμου jar



Δημιουργία εκτελέσιμου jar

- Στο νέο παράθυρο πληκτρολογούμε build και στην συνέχεια πατάμε Run.



Δημιουργία εκτελέσιμου jar

- Το εκτελέσιμο θα φτιαχτεί μέσα στο project, στην θέση: `build/libs/SpringExample-0.0.1-SNAPSHOT.jar`.
- Μπορούμε να το μεταφέρουμε όπου θέλουμε. Δεν χρειαζόμαστε κάτι άλλο εκτός από αυτό το jar.
- Για να δούμε τα περιεχόμενά του:
 - `java -tzn SpringExample-0.0.1-SNAPSHOT.jar`
- Για να το τρέξουμε:
 - `java -jar SpringExample-0.0.1-SNAPSHOT.jar`

Annotations

- Γενικά είναι οδηγίες για τον compiler ή το runtime processing.
 - <http://docs.oracle.com/javase/tutorial/java/annotations/>
- Ωστόσο, όταν εκτελείται ο κώδικας τον επηρεάζουν
- Τα περισσότερα annotations που θα χρησιμοποιήσουμε είναι ορισμένα από το spring.
 - Κάποια είναι από το hibernate

Επεξήγηση annotations

- **@Configuration**
 - Ορίζει ότι η κλάση που ακολουθεί μπορεί να χρησιμοποιηθεί για τον ορισμό beans. Ουσιαστικά, αυτή η κλάση έχει αντικαταστήσει το configuration XML που χρησιμοποιούνταν παλαιότερα (και τώρα μπορεί να χρησιμοποιηθεί, αλλά καλύτερα όχι).
- **@EnableAutoConfiguration**
 - Πολύ σημαντικό. Λέμε στο spring να προσπαθήσει να καταλάβει τι beans (αντικείμενα) πρέπει να φτιάξει **αυτόματα** από τις κλάσεις και τα annotations που έχει η κάθε κλάση. Αν δεν βρει ορισμένο κάποιο bean για κάποια κλάση, θα προσπαθήσει να φτιάξει ένα. Αν βρει ένα ορισμένο από εμάς, τότε δεν θα κάνει τίποτα.
- **@ComponentScan**
 - Λέμε ποιά πακέτα να εξετάσει για annotations. Αν δεν ορίσουμε πακέτο, τότε ξεκινά από αυτό που είναι η κλάση και συνεχίζει σε όλα τα υποπακέτα του.
- **@RestController**
 - Η κλάση είναι controller που μέσα της ορίζουμε μεθόδους που επιστρέφουν κατευθείαν αποτέλεσμα (δεν πρέπει να καλέσουμε κάποιο jsp ή template generator για να φτιάξει το html).
- **@RequestMapping("/hello")**
 - Ορίζουμε το path που πρέπει να βάλουμε στο URL ώστε να κληθεί η συγκεκριμένη κλάση.

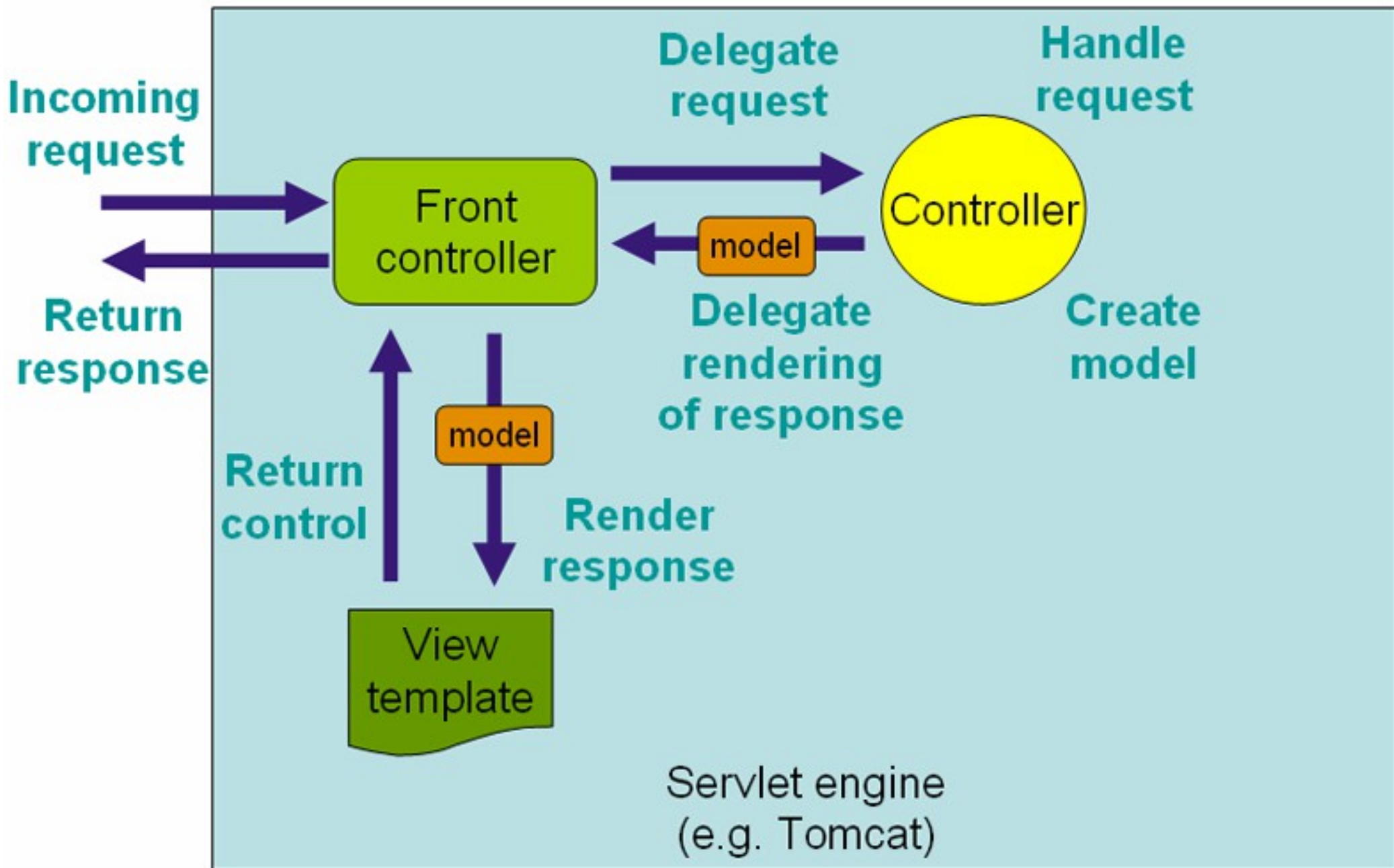
Επεξήγηση HelloController.java

```
@RestController  
public class HelloController {  
  
    @RequestMapping("/hello")  
    public String index() {  
        return "Greetings from Spring Boot!";  
    }  
}
```

- Ορίζουμε ότι αν καλέσουμε την διεύθυνση localhost:8080/hello θα κληθεί η μέθοδος index ενός αντικειμένου (bean) της κλάσης HelloController.
- Το αντικείμενο αυτό θα φτιαχτεί αυτόματα από το spring.
 - Δεν έχουμε γράψει κάπου new HelloController().

Spring MVC

- Ορίζει κάποιους κανόνες για το πως πρέπει να φτιάξουμε μια σελίδα.
- Υπάρχει ένας controller που έχει φτιαχτεί από το spring.
- Όταν ζητάμε μια διεύθυνση, ο βασικός controller επιλέγει ποιόν από τους controller που φτιάξαμε εμείς θα κληθεί.
- Καλεί τον controller, αυτός επιστρέφει ένα μοντέλο.
- Στην συνέχεια ο βασικός controller περνά το μοντέλο σε ένα view template (πχ JSP, Freemake) και παράγεται μια σελίδα.
 - Στο spring boot δεν συστήνεται η χρήση JSP, γιατί δεν μπορεί να δημιουργήσει εκτελέσιμο jar.



Προτεινόμενη αρχιτεκτονική

- Το μοντέλο το παράγει ο controller συνδεδεμένος σε κάποια βάση.
- Ωστόσο, δεν συνδέεται κατευθείαν ο controller στην βάση.
- Η αρχιτεκτονική που προτείνεται είναι:
 - Controller
 - Service
 - Repository
 - Model
- Θα τα εξηγήσουμε στην συνέχεια από κάτω προς τα πάνω.

Model

- Κάνει την συσχέτιση σχεσιακής βάσης – αντικειμένου.
- Ουσιαστικά δημιουργούμε μια κλάση στην οποία μπορούν να μπουν τα περιεχόμενα μιας γραμμής ενός πίνακα της βάσης.
- Έχει επιπλέον annotations για διάφορες συσχετίσεις
 - Σε ποιο πίνακα αντιστοιχεί η βάση
 - Σε ποια στήλη αντιστοιχεί το κάθε γνώρισμα
 - Αν έχει κάποιο περιορισμό κάποιο γνώρισμα (πχ not null).

Repository

- Χρησιμοποιεί το model.
- Εδώ υπάρχει η διεπαφή με την βάση.
- Μπορούμε να ορίσουμε τι μεθόδους μπορούμε να κάνουμε στην βάση (πχ create, read, update, delete).
- Συνήθως παρέχει αυτές τις δυνατότητες σε ένα πίνακα μιας βάσης.

Service

- Χρησιμοποιεί το repository.
- Παρόμοιο με το repository.
- Η διαφορά είναι ότι το service μπορεί να χρησιμοποιεί πολλά repository ώστε να κάνει κάποιες πράξεις σε όλα από αυτά.
- Υποστηρίζει transactions αυτόματα (δεν γράφουμε καθόλου κώδικα).
 - Αν έχουμε μια σειρά από πράξεις σε κάποια repository και αν στην μέση, κάποια πράξη διακοπεί, τότε οι προηγούμενες θα γίνουν rollback.

Controller

- Χρησιμοποιεί το service.
- Σε κάθε path του URL που ορίζει, αντιστοιχεί κάποιες πράξεις του repository.
- Μπορεί να πράξει κατευθείαν το αποτέλεσμα ή να δώσει το μοντέλο που δημιουργήθηκε από το service σε κάποιο template engine.

Παράδειγμα

- Θα δημιουργήσουμε μια σελίδα που θα μπορούμε να εισάγουμε πληροφορίες για ανθρώπους (όνομα, ηλικία) και μια που θα εμφανίζονται οι πληροφορίες που έχουμε εισάγει.
- Θα χρειαστούμε μια βάση.

Βάση δεδομένων

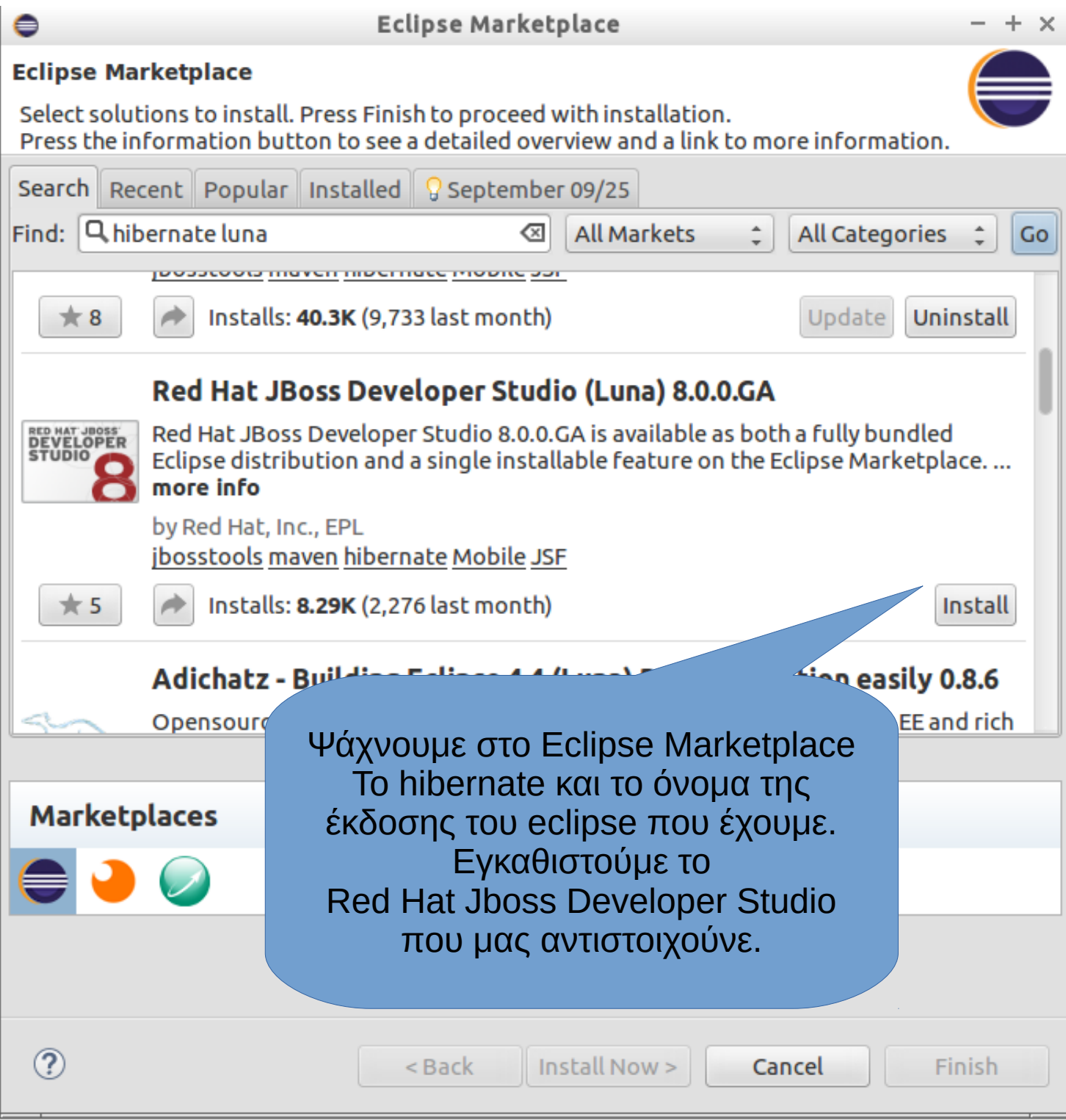
- Χρησιμοποιούμε mysql (5.5).
- Δημιουργούμε την βάση:
 - `CREATE DATABASE IF NOT EXISTS personsdb DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8_general_ci;`
 - `GRANT ALL ON `personsdb`.* TO `user1`@localhost IDENTIFIED BY '1234';`
 - `FLUSH PRIVILEGES;`
 - `USE personsdb;`
 -
 - `CREATE TABLE IF NOT EXISTS persons (`
 - `id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,`
 - `name VARCHAR(255) NOT NULL,`
 - `age TINYINT UNSIGNED NOT NULL,`
 - `PRIMARY KEY ( id )`
 - `) ENGINE = InnoDB;`

Δημιουργία μοντέλου

- Μπορούμε να χρησιμοποιήσουμε το Hibernate και για να πράξουμε το μοντέλο και για να συνδεθούμε στην βάση.
 - Το spring δεν παρέχει τρόπο για σύνδεση στην βάση.

Hibernate

- Είναι ένα ORM framework (Object Relational Mapping).
- Η Java EE ορίζει τα interfaces του JPA (Java Persistence API) που λένε πως θα γίνεται η διασύνδεση με μια βάση δεδομένων.
- Ωστόσο δεν δίνει κλάσεις που τα υλοποιούνε.
- Το hibernate δίνει αυτές τις κλάσεις, και ορίζει και κάποιες επιπλέον δικές του.
- Γενικά, χρησιμοποιούμε τους ορισμούς του JPA όποτε είναι δυνατό, αλλά σαν αντικείμενα (beans) χρησιμοποιούμε το Hibernate.
 - Πχ `SomeInterfaceJPA myObject = new SomeClassHibernate()`.



Ψάχνουμε στο Eclipse Marketplace
Το hibernate και το όνομα της
έκδοσης του eclipse που έχουμε.
Εγκαθιστούμε το
Red Hat Jboss Developer Studio
που μας αντιστοιχούνε.

Download Connector/J

MySQL Connector/J is the official JDBC driver for MySQL.

Online Documentation:

- MySQL Connector/J Installation Instructions, Documentation and Change History

Please report any bugs or inconsistencies you observe to our [Bugs Database](#).

Thank you for your support!

MySQL open source software is provided under the [GPL License](#).

OEMs, ISVs and VARs can purchase commercial licenses.

Generally Available (GA) Releases

Connector/J 5.1.33

Select Platform:

Platform Independent

[Looking for previous GA versions?](#)

Platform Independent (Architecture Independent), Compressed TAR Archive

(mysql-connector-java-5.1.33.tar.gz)

5.1.33

3.6M

[Download](#)

MD5: 5952133669ef4d2c3dfdef8232baf782 | [Signature](#)

Platform Independent (Architecture

5.1.33

3.9M

[Download](#)

Κατεβάζουμε τον driver της mysql.

MySQL on Windows

MySQL Yum Repository

MySQL APT Repository

MySQL Community Server

MySQL Cluster

MySQL Fabric

MySQL Utilities

MySQL Workbench

MySQL Proxy

MySQL Connectors

Connector/ODBC

Connector/Net

Connector/J

Connector/Python

Connector/C++

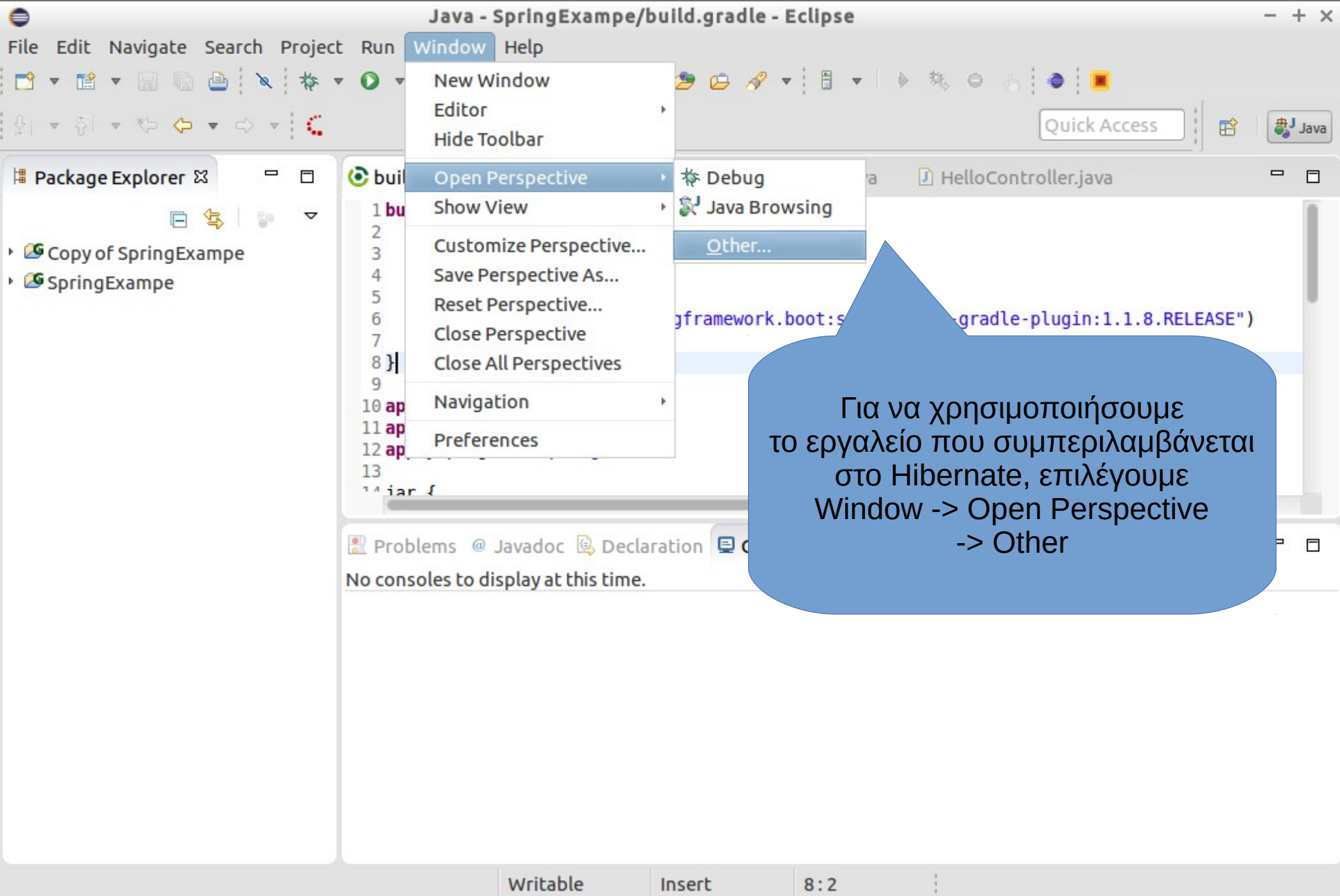
Connector/C

MySQL Native Driver for PHP

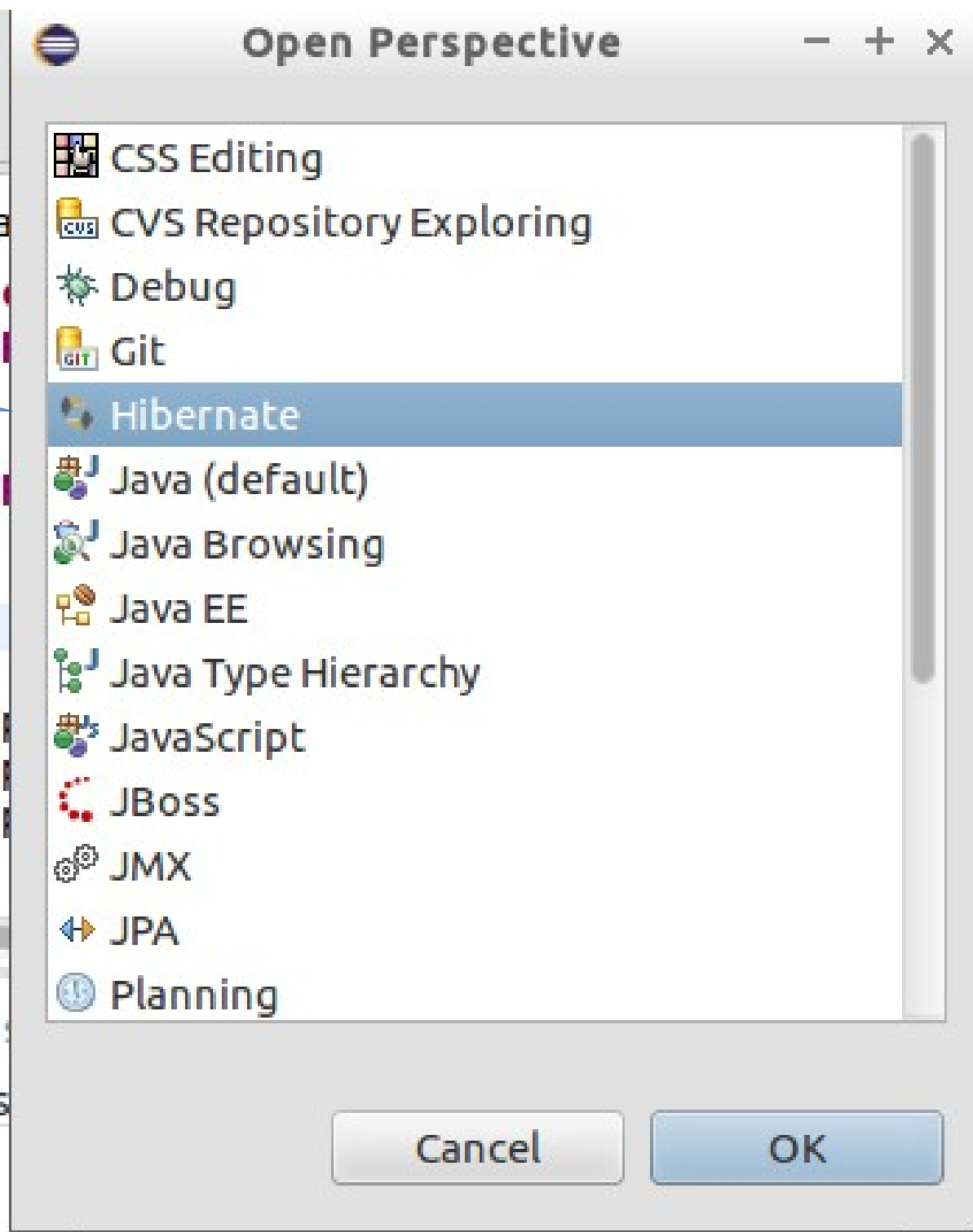
Other Downloads

- SpringExample
 - src/main/java
 - src/main/resources
 - src/test/java
 - src/test/resources
 - JRE System Library [JavaSE-1.7]
 - Referenced Libraries
 - build
 - myLibs
 - mysql-connector-java-5.1.33-bin.jar
 - src
 - build.gradle

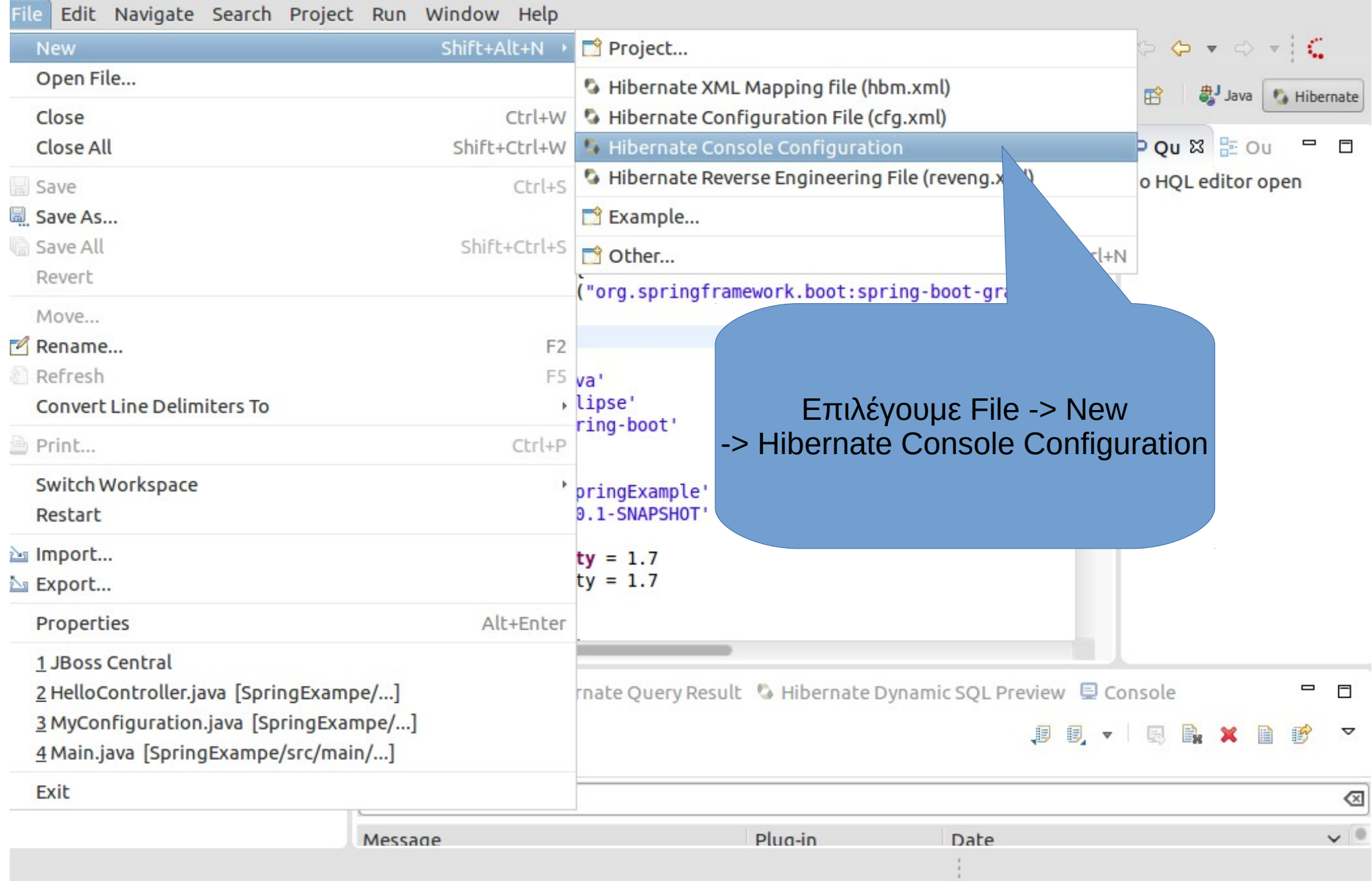
Δημιουργούμε ένα φάκελο myLibs μέσα στο project και κάνουμε αντιγραφή εκεί τον driver.
(απλώς, για να τον έχουμε μέσα στο project).



Για να χρησιμοποιήσουμε
το εργαλείο που συμπεριλαμβάνεται
στο Hibernate, επιλέγουμε
Window -> Open Perspective
-> Other



Επιλέγουμε
Hibernate



Συμπληρώνουμε ένα όνομα.
Στο Main tab
Επιλέγουμε την τελευταία
Hibernate Version,
επιλέγουμε το projet και
στο Database conection
επιλέγουμε Hibernate
configured connection
και πατάμε New.

Create Hibernate Console Configuration

This wizard allows you to create a configuration for Hibernate Console.

Name:

MainOptionsClasspathMappingsCommon

Type:

☒ Core

☐ Annotations (jdk 1.5+)

☐ JPA (jdk 1.5+)

Hibernate Version:

4.3

Project:

Browse...

Database connection:

[Hibernate configured connection]

New...Edit...

Property file:

Setup...

Configuration file:

Setup...

Persistence unit:

Browse...

?

CancelFinish

**Connection Profile**

Create a MySQL connection profile.



Connection Profile Types:

type filter text

- DB2 for Linux, UNIX, and Windows
- DB2 for i5/OS
- DB2 for z/OS
- Derby
- Generic JDBC
- HSQldb
- Informix
- Ingres
- MaxDB
- MySQL**
- Oracle
- PostgreSQL

Name:

SpringExampleMysql

Description (optional):



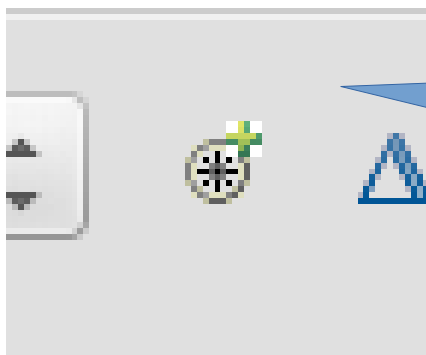
< Back

Next >

Cancel

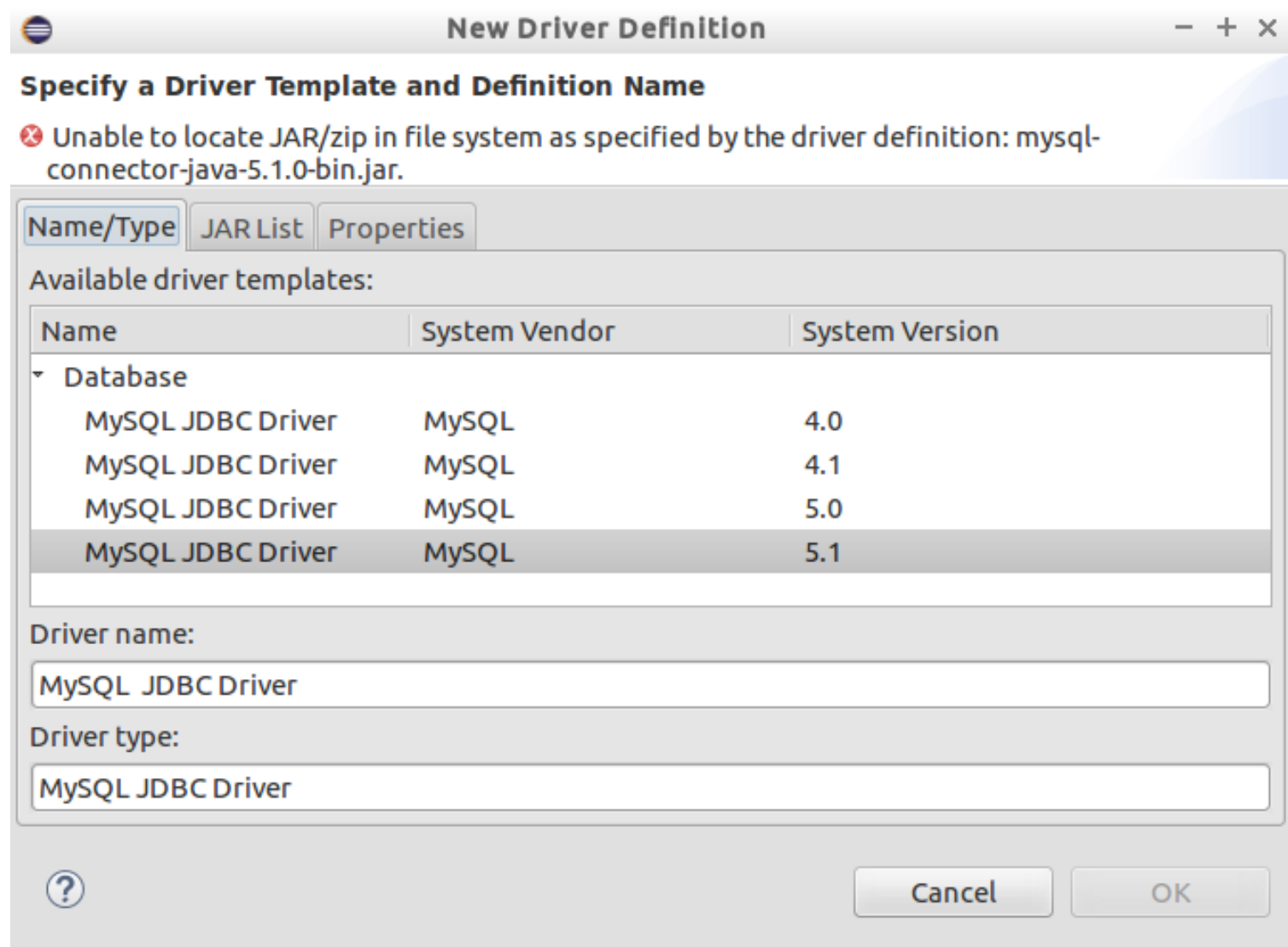
Finish

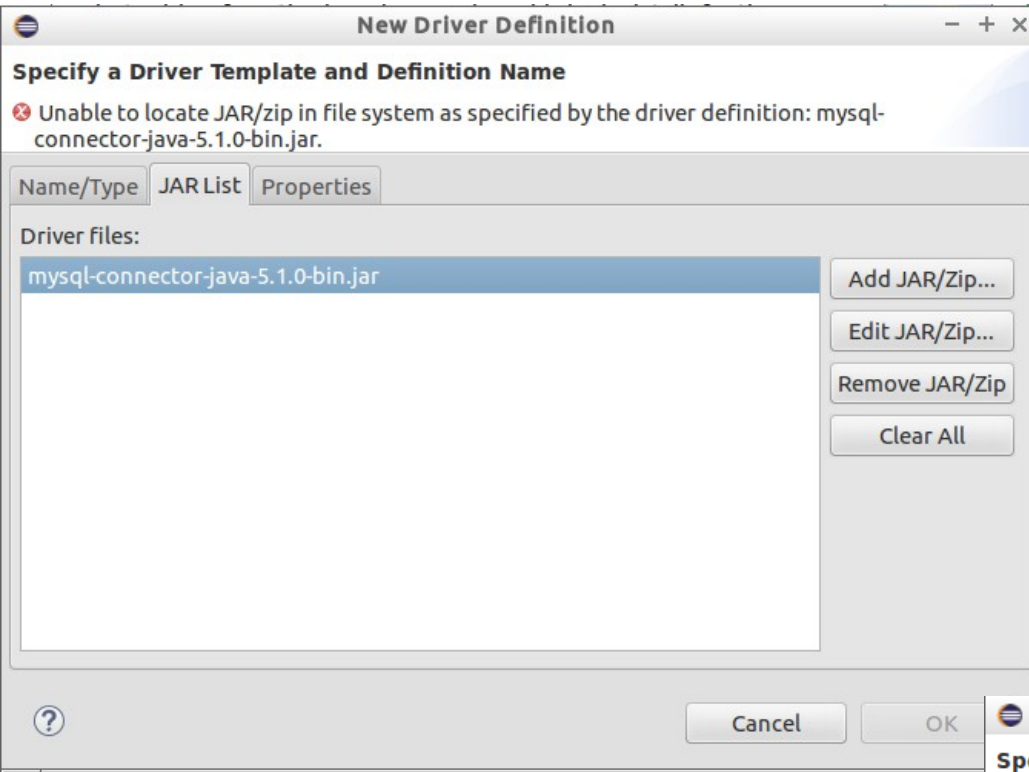
Επιλέγουμε MySQL
και δίνουμε ένα
όνομα.
Πατάμε Next.



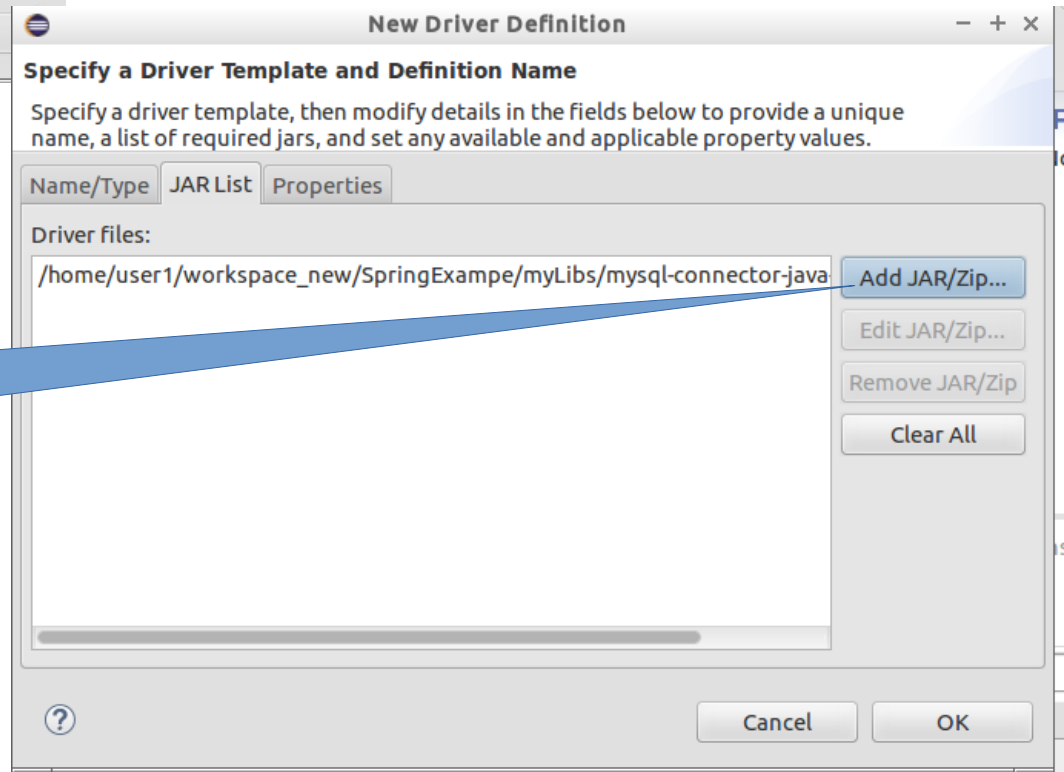
Επιλέγουμε αυτό το εικονίδιο για να φτιάξουμε ένα νέο driver.

Στο Name/Type tab επιλέγουμε την τελευταία έκδοση.





Στο tab JAR List, σβήνουμε
το υπάρχον jar.



Και προσθέτουμε το jar
που κατεβάσαμε.
Πατάμε OK.

New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: MySQL JDBC Driver

Properties

General Optional

Database: personsdb

URL: jdbc:mysql://localhost:3306/personsdb

User name: user1

Password:

☒ Save password

☒ Connect when the wizard completes

☐ Connect every time the workbench is started

Test Connection

Finish

< Back **Next >** **Cancel**

Βάζουμε τις πληροφορίες της βάσης και πατάμε Finish.

Create Hibernate Console Configuration

This wizard allows you to create a configuration for Hibernate Console.

Name:

Main Options Classpath Mappings Common

Type:

☒ Core ☐ Annotations (jdk 1.5+) ☐ JPA (jdk 1.5+)

Hibernate Version:

Project:

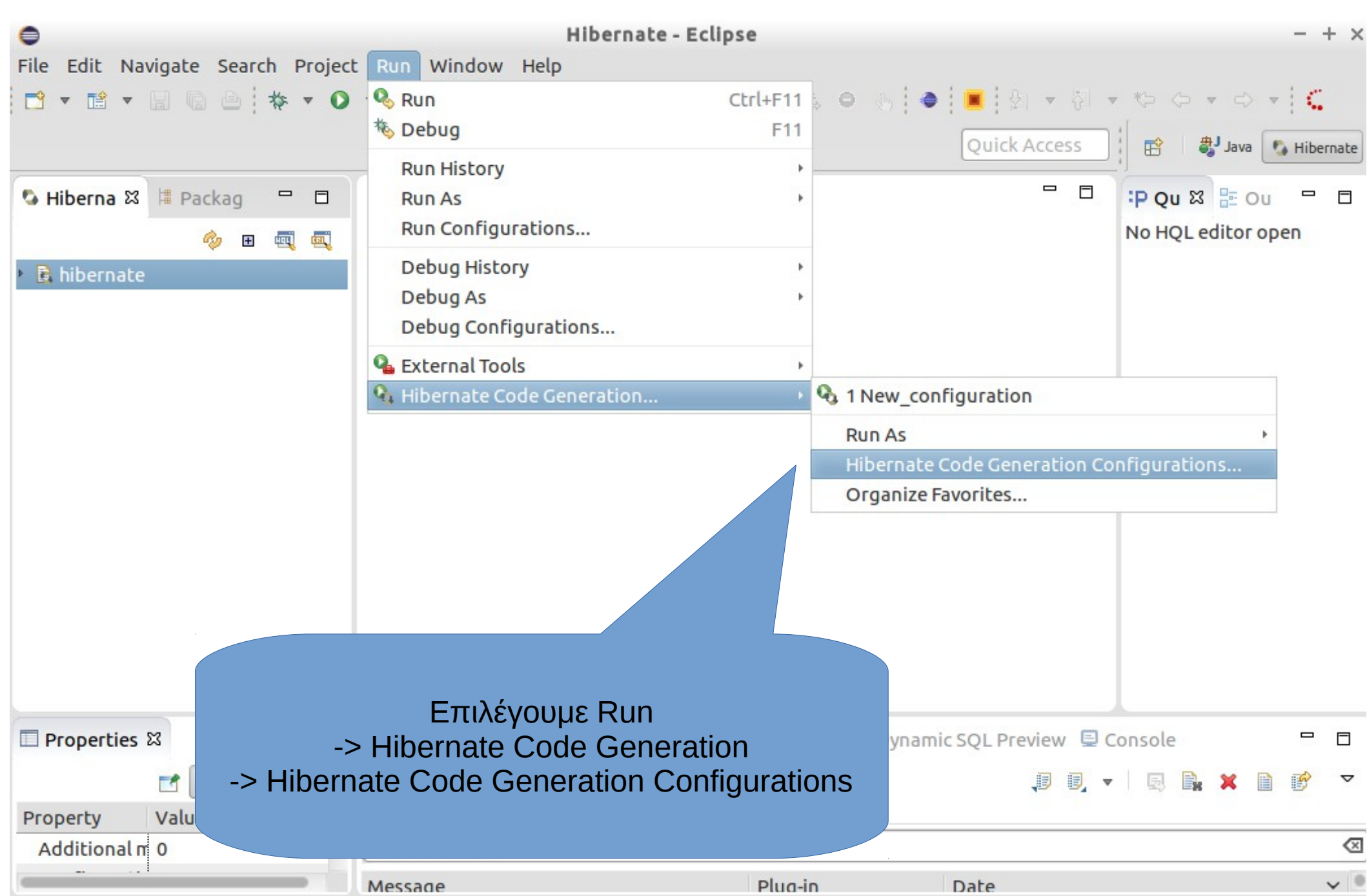
Database connection:

Property file:

Configuration file:

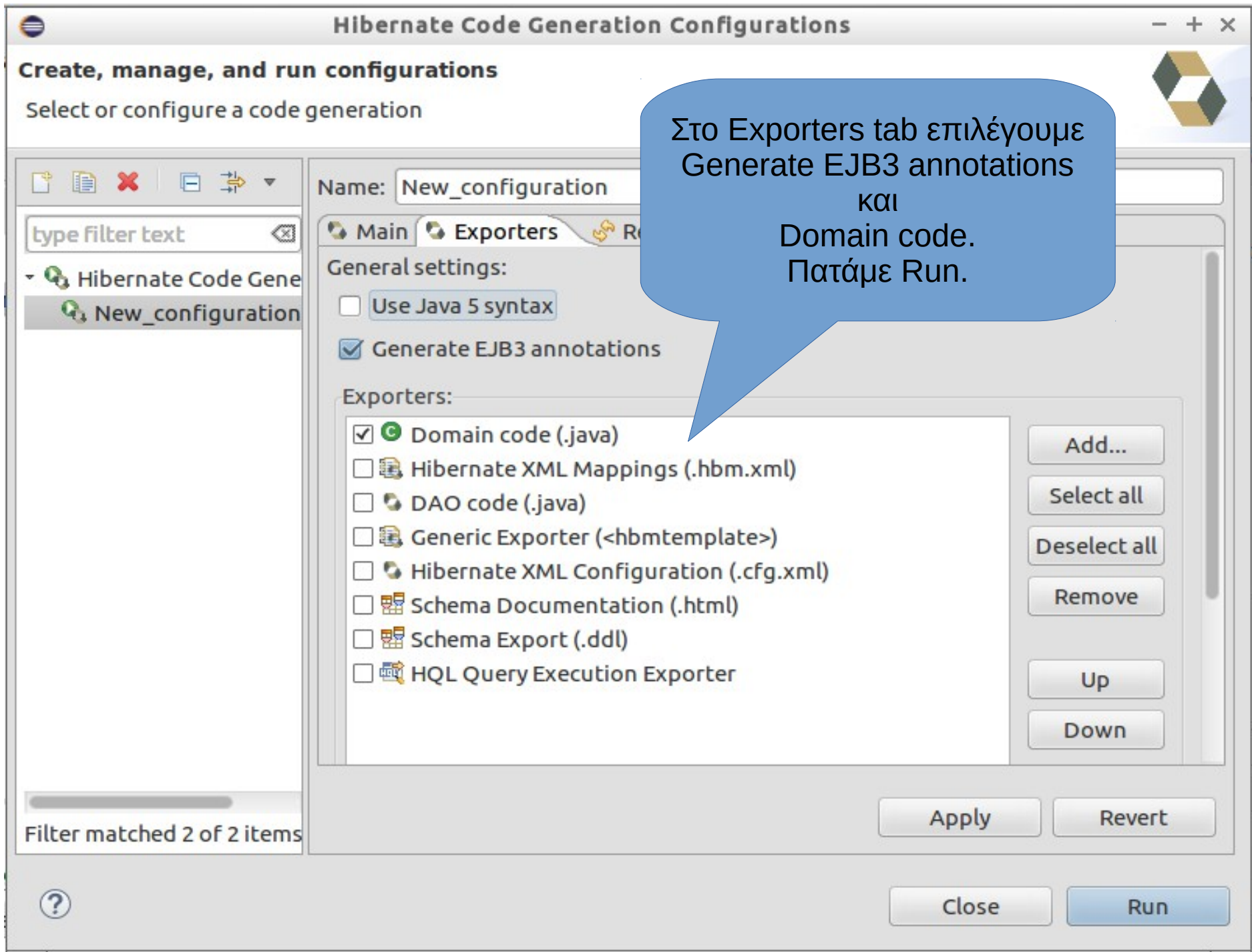
Persistence unit:

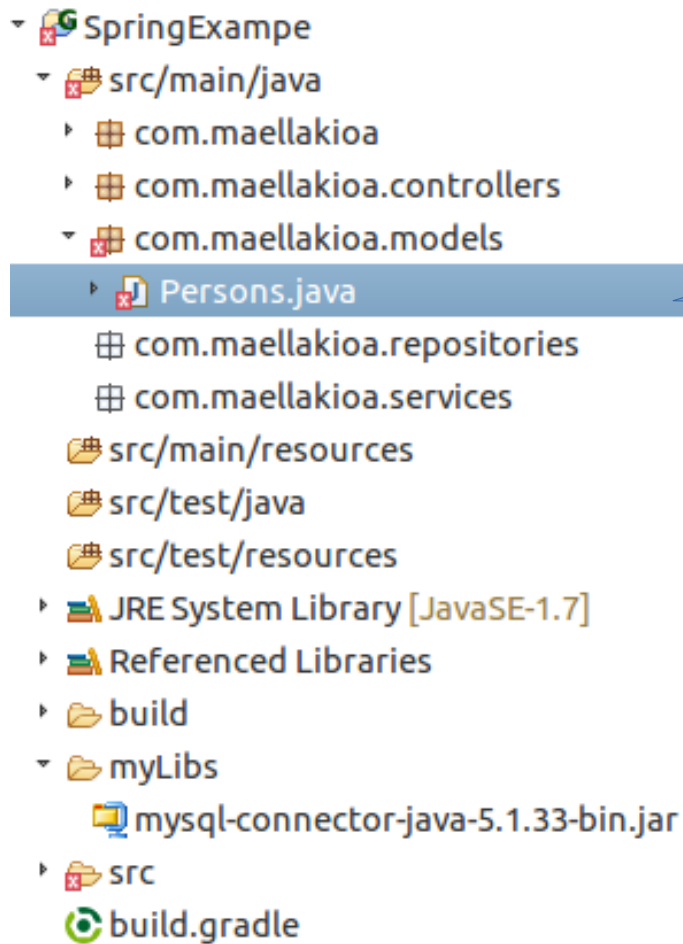
Πατάμε
Finish.





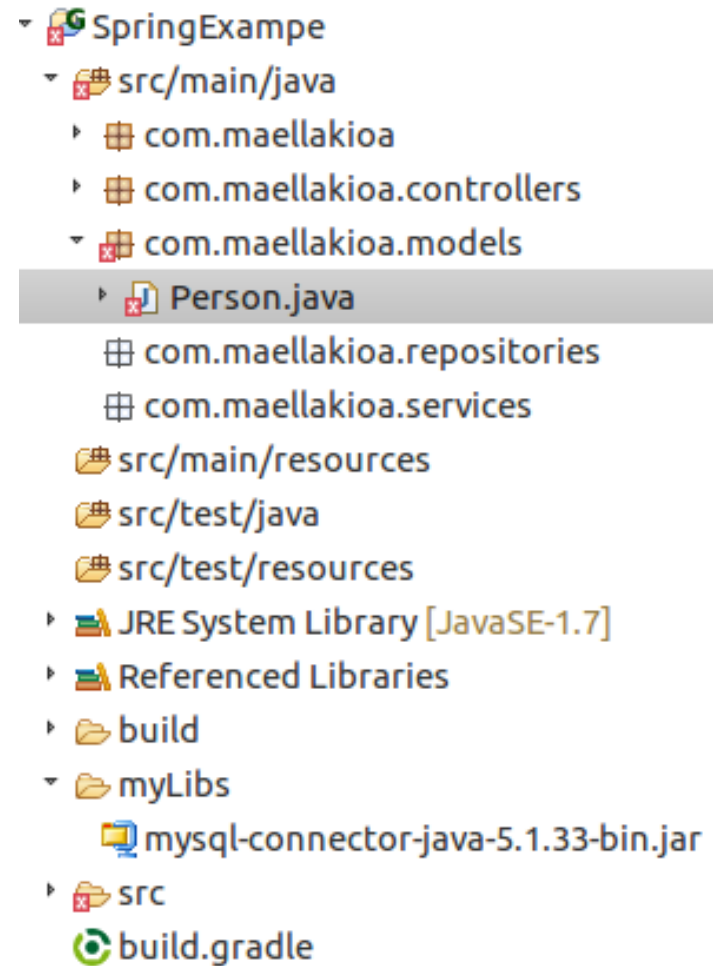
Στο Main tab συμπληρώνουμε τα
Console configuration,
Output directory,
Reverse engineer from
JDBS Connection,
Package.





Δημιουργήθηκε η κλάση Persons. Θα την μετονομάσουμε σε Person.

Έχει κάποια λάθη, που οφείλονται στο ότι δεν έχουμε τα κατάλληλα jar.



```
dependencies {  
    compile("org.springframework.boot:spring-boot-starter-web")  
    compile("org.springframework.boot:spring-boot-starter-data-jpa")  
}  
}
```

Στο build.gradle προσθέτουμε
`compile("org.springframework.boot:spring-boot-starter-data-jpa")`
και κάνουμε Gradle -> Update All.

```

@Entity
@Table(name = "persons", catalog = "personsdatabase")
public class Person implements java.io.Serializable {

    private Long id;

```

Σβήνουμε το
implements java.io.Serializable

```

/*
 * Persons generated by hibernate
 */
@Entity
@Table(name = "persons", catalog = "personsdatabase")
public class Person {

    private Long id;

```

Person.java

- Package com.maellakioa.models;

```
// Generated Nov 1, 2014 10:08:00 PM by Hibernate  
Tools 4.3.1
```

```
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import static  
    javax.persistence.GenerationType.IDENTITY;  
import javax.persistence.Id;  
import javax.persistence.Table;
```

```
/**  
 * Persons generated by hbm2java  
 */
```

```
@Entity
```

```
@Table(name = "persons", catalog = "personsdb")
```

```
public class Person {
```

```
    private Long id;  
    private String name;  
    private byte age;
```

```
    public Person() {  
    }
```

```
    public Person(String name, byte age) {  
        this.name = name;  
        this.age = age;  
    }
```

```
@Id
```

```
@GeneratedValue(strategy = IDENTITY)
```

```
@Column(name = "id", unique = true, nullable =  
false)
```

```
    public Long getId() {  
        return this.id;  
    }
```

```
    public void setId(Long id) {  
        this.id = id;  
    }
```

```
@Column(name = "name", nullable = false)
```

```
    public String getName() {  
        return this.name;  
    }
```

```
    public void setName(String name) {  
        this.name = name;  
    }
```

```
@Column(name = "age", nullable = false)
```

```
    public byte getAge() {  
        return this.age;  
    }
```

```
    public void setAge(byte age) {  
        this.age = age;  
    }
```

```
}
```

Person.java

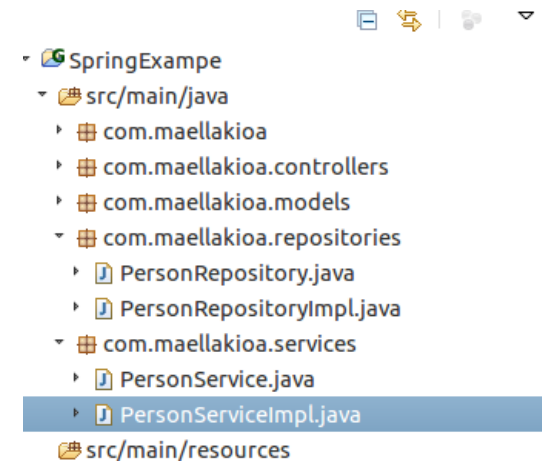
- Το hibernate έφτιαξε μια κλάση για τον πίνακα που έχουμε.
 - Αν είχαμε παραπάνω πίνακες θα έφτιαχνε μια κλάση για κάθε πίνακα.
 - Υποστηρίζει και συσχετίσεις one-to-one, one-to-many και many-to-many
- Κάθε κλάση έχει τα αντίστοιχα πεδία που έχει και ο πίνακας ως στήλες.
 - Φτιάχνει αντίστοιχους τύπους.
- Προσθέτει και κάποια annotations.
 - Τα annotations μπορούν να πάνε είτε πάνω από τα πεδία είτε πάνω από τις αντίστοιχες μεθόδους get. Καλύτερα πάνω από μεθόδους.

Annotations του Hibernate

- `@Entity`
 - Ορίζει ότι η επόμενη κλάση αντιστοιχεί σε κάποιον πίνακα κάποιας βάσης.
- `@Table(name = "persons", catalog = "personsdb")`
 - Ορίζει σε ποιο ακριβώς πίνακα ποιας κλάσης αντιστοιχεί.
- `@Id`
 - Ορίζει ότι το επόμενο πεδίο είναι κλειδί.
- `@GeneratedValue(strategy = IDENTITY)`
 - Ορίζει τον τρόπο που αυξάνεται το κλειδί (auto increment).
- `@Column(name = "id", unique = true, nullable = false)`
 - Ορίζει σε ποια στήλη του πίνακα αντιστοιχεί το επόμενο πεδίο.
- `@Column(name = "name", nullable = false)`
 - Ορίζει σε ποια στήλη του πίνακα αντιστοιχεί το επόμενο πεδίο.

Repository και service

- Για τα repository και τα service προτείνεται να δημιουργούμε ένα interface και όσες κλάσεις χρειάζονται.
 - PersonRepository.java
 - PersonRepositoryImpl.java
 - PersonService.java
 - PersonServiceImpl.java
- Είναι η ίδια οδηγία με το να χρησιμοποιούμε interfaces και όχι κλάσεις όταν δηλώνουμε αντικείμενα.



PersonRepository.java

- ```
package com.maellakioa.repositories;

import java.util.List;

import com.maellakioa.models.Person;

public interface PersonRepository {
 public List<Person> getPersons();
 public void createPerson(Person p);
}
```

# PersonRepositoryImpl.java

```
package com.maellakioa.repositories;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.stereotype.Repository;

import com.maellakioa.models.Person;

@Repository
public class PersonRepositoryImpl
implements PersonRepository {

 @Autowired
 private SessionFactory
 sessionFactory;

 private Session getSession() {
 return
 sessionFactory.getCurrentSession();
 }

 @SuppressWarnings("unchecked")

 @Override
 public List<Person> getPersons() {
 return
 getSession().createQuery("from
 Person").list();
 }

 @Override
 public void createPerson(Person p)
 {
 getSession().save(p);
 }
}
```

# PersonService.java

```
package com.maellakioa.services;

import java.util.List;

import
com.maellakioa.models.Person;

public interface PersonService {
 public List<Person> getPersons();
 public void createPerson(Person
p);
}
```

# PersonServiceImpl.java

```
package com.maellakioa.services;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.maellakioa.models.Person;
import com.maellakioa.repositories.PersonRepository;

@Service
@Transactional
public class PersonServiceImpl implements PersonService {

 @Autowired
 private PersonRepository personRepository;

 @Override
 public List<Person> getPersons() {
 return personRepository.getPersons();
 }

 @Override
 public void createPerson(Person p) {
 personRepository.createPerson(p);
 }
}
```

# PersonController.java

```
package com.maellakioa.controllers;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

import com.maellakioa.models.Person;
import com.maellakioa.services.PersonService;

@Controller
@RequestMapping("mypersons")
public class PersonController {
 @Autowired
 private PersonService personService;
```

```
 // GET http://localhost:8080/mypersons/listApi
 @RequestMapping(value = "/listApi",
 method = RequestMethod.GET, produces = "application/json; charset=utf-8")
 @ResponseBody
 public String listApi(Model model)
 throws JSONException {
 JSONArray menuArray = new
 JSONArray();
 for (Person p :
 personService.getPersons()) {
 JSONObject menuJSON = new
 JSONObject();
 menuJSON.put("name",
 p.getName());
 menuJSON.put("age", p.getAge());
 menuArray.put(menuJSON);
 }
 return menuArray.toString();
 }
}
```

# build.gradle dependencies

```
dependencies {
 compile("org.springframework.boot:spring-boot-starter-web")
 compile("org.springframework.boot:spring-boot-starter-data-jpa")
 compile("org.json:json:20140107")
 compile("mysql:mysql-connector-java:5.1.33")
 compile("mysql:mysql-connector-java:5.1.33")
}
```

# MyConfiguration.java

```
package com.maellakioa;
import java.util.Properties;
import javax.sql.DataSource;
import org.apache.commons.dbcp.BasicDataSource;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.orm.hibernate4.HibernateTransactionManager;
import org.springframework.orm.hibernate4.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableAutoConfiguration
@ComponentScan
@EnableTransactionManagement
public class MyConfiguration {
 @Autowired
 private Environment env;
```

```
 @Bean
 public DataSource dataSource() {
 BasicDataSource dataSource = new BasicDataSource();
 dataSource.setUrl(env.getProperty("datasource.url"));

 dataSource.setDriverClassName(env.getProperty("datasource.driverClassName"));
 dataSource.setUsername(env.getProperty("datasource.username"));
 dataSource.setPassword(env.getProperty("datasource.password"));
 return dataSource;
 }

 @Bean
 @Autowired
 public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {
 Properties properties = new Properties();
 properties.setProperty("hibernate.dialect", env.getProperty("hibernate.dialect"));
 properties.setProperty("hibernate.show_sql", env.getProperty("hibernate.show_sql"));
 properties.setProperty("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl-auto"));
 LocalSessionFactoryBean localSessionFactoryBean = new LocalSessionFactoryBean();
 localSessionFactoryBean.setDataSource(dataSource);
 localSessionFactoryBean.setHibernateProperties(properties);
 localSessionFactoryBean.setPackagesToScan(new String[] { "com.maellakioa.models" });
 return localSessionFactoryBean;
 }

 @Bean
 @Autowired
 public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {
 HibernateTransactionManager txManager = new HibernateTransactionManager();
 txManager.setSessionFactory(sessionFactory);
 return txManager;
 }
}
```



# src/test/resources/application.yml

```
server:
 port: 8080
datasource:
 url: jdbc:mysql://localhost:3306/personsδβ?
 autoReconnect=true&useUnicode=true&characterE
 ncoding=UTF-8
 driverClassName: com.mysql.jdbc.Driver
 username: user1
 password: 1234
hibernate:
 dialect:
 org.hibernate.dialect.MySQLInnoDBDialect
 show_sql: false
```

# Annotations

- **@Repository**
  - Ορίζει ότι η κλάση που ακολουθεί είναι ένα repository (ή DAO = Data Access Object)
- **@Service**
  - Ορίζει ότι η κλάση που ακολουθεί είναι service.
- **@Controller**
  - Ορίζει ότι η κλάση που ακολουθεί είναι controller.
- **@Transactional**
  - Ορίζει ότι η λειτουργίες θα εκτελούνται ως transactions (ή ολοκληρώνονται όλες οι υπό-επενέργειες χωρίς λάθος ή καμία)

# Annotations

- **@Autowired**
  - Το αντικείμενο που ακολουθεί θα προσπαθήσει να συνδέσει από μόνο του το spring με κάποιο ήδη υπάρχον Bean. Για τους κανόνες που ακολουθεί:  
<http://springindepth.com/book/in-depth-ioc-autowiring.html>
- **@RequestMapping**
  - Αντιστοιχεί web resources σε κλάσεις ή μεθόδους.
- **@ResponseBody**
  - Τα δεδομένα που επιστρέφει η μέθοδος επιστρέφονται κατευθείαν. Δεν χρειάζεται να τα δώσουμε σε κάποιο view.
- **@EnableTransactionManagement**
  - Ενεργοποιεί τα annotation που σχετίζονται με transactions.
- **@Bean**
  - Ορίζει ότι η μέθοδος που ακολουθεί παράγει ένα bean (αντικείμενο) που το διαχειρίζεται το spring.

# Freemarker

- Είναι ένα template engine που παράγει html σελίδες με βάση κάποιο template και κάποια δεδομένα.
  - Με το spring boot δεν προτείνεται το JSP γιατί δεν μπορεί να ενσωματωθεί στο εκτελέσιμο jar.
- Πληροφορίες: <http://freemarker.org>
- Βάζουμε τα αρχεία στο φάκελο `src/main/resources/templates`.
- Τα αρχεία έχουν κατάληξη `.ftl`.
- Ο controller πρέπει να επιστρέψει το όνομα του αρχείου.
- Στο `build.gradle` προσθέτουμε μέσα στα `dependencies`:
  - `compile("org.springframework.boot:spring-boot-starter-freemarker")`

# PersonController.java

- Προσθέτουμε:
- `/http://localhost:8080/mypersons/list`  

```
@RequestMapping(value = "/list",
method = RequestMethod.GET)
public String list(Map<String,
Object> model) {
 model.put("persons",
personService.getPersons());
 return "list";
}
```

# src/main/resources/templates/list.ftl

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Persons List</title>
</head>
<body>
<#list persons as p>
 <div> Person: ${p.name} - ${p.age} </div>
</#list>
</body>
</html>
```

# Αυτόματη δημιουργία repository

- Μπορούμε να μην γράψουμε εμείς το repository.
- Χρησιμοποιούμε το interface `crudRepository`.
  - Το spring δημιουργεί αυτόματα ένα implementation του repository που δηλώσαμε με crud δυνατότητες
  - Επίσης, μόνο δηλώνοντας τις συναρτήσεις, μπορούμε να κάνι αναζήτηση.
  - Μπορούμε να το κάνουμε αυτόματα διαθέσιμο σαν HATEOAS resource (json+επιπλέον πληροφορίες).
- Πληροφορίες:  
<http://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>

# Αλλαγές

- Στο build.gradle, μέσα στο dependencies προσθέτουμε:
  - compile("org.springframework.boot:spring-boot-starter-data-rest")
- Στο MyConfiguration.java, πριν την κλάση προσθέτουμε:
  - @EnableJpaRepositories("com.maellakioa.repositories")
  - @Import(RepositoryRestMvcConfiguration.class)



# PersonCrudRepository.java

```
package com.maellakioa.repositories;
import java.util.List;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import com.maellakioa.models.Person;
```

## **@RepositoryRestResource**

```
public interface PersonCrudRepository extends CrudRepository<Person, Long> {
```

```
 public List<Person> findByAge(@Param("age") int age);
```

```
 @Query(value = "SELECT * FROM persons WHERE age=:age1 OR age=:age2",
 nativeQuery = true)
```

```
 public List<Person> findDay(@Param("age1") byte age1, @Param("age2") byte
 age2);
```

```
}
```

# Παραδείγματα

- **`curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/"`**
  - HTTP/1.1 200 OK
  - Server: Apache-Coyote/1.1
  - Content-Type: application/json;charset=UTF-8
  - Transfer-Encoding: chunked
  - Date: Sun, 02 Nov 2014 20:39:30 GMT
  - 
  - {
  - "\_links" : {
  - "persons" : {
  - "href" : "http://localhost:8080/persons"
  - }
  - }
  - }

# Παράδειγμα

- **curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons"**

```
- HTTP/1.1 200 OK
- Server: Apache-Coyote/1.1
- Content-Type: application/json; charset=UTF-8
- Transfer-Encoding: chunked
- Date: Sun, 02 Nov 2014 20:40:17 GMT
-
- {
- "_links" : {
- "search" : {
- "href" : "http://localhost:8080/persons/search"
- }
- },
- "_embedded" : {
- "persons" : [{
- "name" : "Mike",
- "age" : 10,
- "_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/1"
- }
- }
- }
-]
- }
```

```
- }, {
- "name" : "Mike",
- "age" : 20,
- "_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/2"
- }
- }
- }, {
- "name" : "Peter",
- "age" : 20,
- "_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/3"
- }
- }
- }]
- }
```

# Παράδειγμα

- **`curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons/2"`**
  - HTTP/1.1 200 OK
  - Server: Apache-Coyote/1.1
  - Content-Type: application/json;charset=UTF-8
  - Transfer-Encoding: chunked
  - Date: Sun, 02 Nov 2014 20:42:22 GMT
  - 
  - {
  - "name" : "Mike",
  - "age" : 20,
  - "\_links" : {
  - "self" : {
  - "href" : "http://localhost:8080/persons/2"
  - }
  - }
  - }

# Παράδειγμα

- **curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons/search"**
  - HTTP/1.1 200 OK
  - Server: Apache-Coyote/1.1
  - Content-Type: application/json; charset=UTF-8
  - Transfer-Encoding: chunked
  - Date: Sun, 02 Nov 2014 20:42:54 GMT
  - 
  - {
  - "\_links" : {
  - "findByAge" : {
  - "href" : "http://localhost:8080/persons/search/findByAge"
  - },
  - "findDay" : {
  - "href" : "http://localhost:8080/persons/search/find2Age(?age1,age2)",
  - "templated" : true
  - }
  - }
  - }

# Παράδειγμα

- `curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons/search/findByAge?age=10"`
  - HTTP/1.1 200 OK
  - Server: Apache-Coyote/1.1
  - Content-Type: application/json; charset=UTF-8
  - Transfer-Encoding: chunked
  - Date: Sun, 02 Nov 2014 20:45:27 GMT
  - 
  - {
  - "\_embedded" : {
  - "persons" : [ {
  - "name" : "Mike",
  - "age" : 10,
  - "\_links" : {
  - "self" : {
  - "href" : "http://localhost:8080/persons/1"
  - }
  - }
  - } ]
  - }
  - }

# Παράδειγμα

- **`curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons/search/fin d2Age?age1=10&age2=20"`**

- HTTP/1.1 200 OK
- Server: Apache-Coyote/1.1
- Content-Type: application/json; charset=UTF-8
- Transfer-Encoding: chunked
- Date: Sun, 02 Nov 2014 20:54:22 GMT

- {
- "\_embedded" : {
- "persons" : [ {
- 

- "name" : "Mike",
- "age" : 10,
- "\_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/1"
- }
- }
- }, {
- "name" : "Mike",
- "age" : 20,
- "\_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/2"
- }
- }
- }, {
- "name" : "Peter",
- "age" : 20,
- "\_links" : {
- "self" : {
- "href" : "http://localhost:8080/persons/3"
- }
- }
- }]
- }
- }

# Εμφάνιση υποσυνόλου χαρακτηριστικών

- Αν θέλουμε να εμφανίσουμε μόνο ένα υποσύνολο των στιλών της βάσης, τότε μπορούμε να δημιουργήσουμε interfaces.
- Μπορούμε να έχουμε όλες τις μεθόδους που ορίσαμε στο βασικό repository, αλλά θα επιστρέφουν μόνο τις στήλες που θέλουμε.



# PersonAgeProjection.java

- ```
package com.maellakioa.models;

import
org.springframework.data.rest.core.config.
Projection;

@Projection(name = "age_summary", types =
Person.class)

public interface PersonAgeProjection {
    public byte getAge();
}
```

Παράδειγμα

- `curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons"`
 - HTTP/1.1 200 OK
 - Server: Apache-Coyote/1.1
 - Content-Type: application/json; charset=UTF-8
 - Transfer-Encoding: chunked
 - Date: Sun, 02 Nov 2014 21:02:31 GMT
 - ...
 - `"_embedded" : {`
 - `"persons" : [ {`
 - `"name" : "Mike",`
 - `"age" : 10,`
 - `"_links" : {`
 - `"self" : {`
 - `"href" : "http://localhost:8080/persons/1{?projection}",`
 - `"templated" : true`
 - `}`
 - `}`
 - `}, {`
 - `...`
 - `}`

Παράδειγμα

- `curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons?projection=age_summary"`
 - HTTP/1.1 200 OK
 - Server: Apache-Coyote/1.1
 - ...
 - "_embedded" : {
 - "persons" : [{
 - "age" : 10,
 - "_links" : {
 - "self" : {
 - "href" : "http://localhost:8080/persons/1{?projection}",
 - "templated" : true
 - }
 - }
 - }, {
 - "age" : 20,
 - "_links" : {
 - "self" : {
 - "href" : "http://localhost:8080/persons/2{?projection}",
 - "templated" : true
 - }
 - }
 - }, {
 - "..."
 - }
 - }

Παράδειγμα

- `curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET "http://localhost:8080/persons/search/find2Age?age1=10&age2=20&projection=age_summary"`
 - HTTP/1.1 200 OK
 - ...
 - {
 - "_embedded" : {
 - "persons" : [{
 - "age" : 10,
 - "_links" : {
 - "self" : {
 - "href" : "http://localhost:8080/persons/1{?projection}",
 - "templated" : true
 - }
 - }
 - }, {
 - ...
 - }
 - }

Φόρμα εισαγωγής

- Την φόρμα εισαγωγής θα την φτιάξουμε κάνοντας κλήση του `PersonCrudRepository`.
- Θα φτιάξουμε μια στατική σελίδα και ένα αρχείο σε `AngularJS`.

src/main/resources/static/insert.html

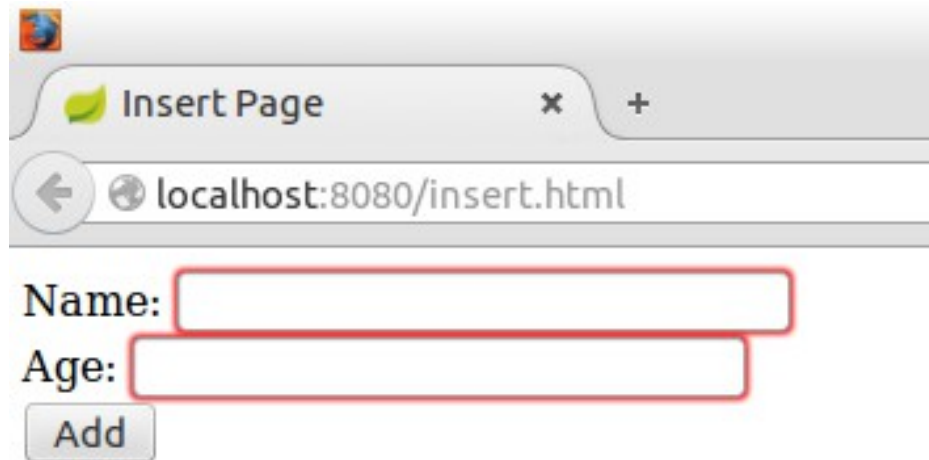
```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8"><title>Insert Page</title>
<script src="
https://ajax.googleapis.com/ajax/libs/angularjs/1.2.16/angular.js
"></script>
<script src="js/InsertModule.js"></script></head>
<body ng-app="InsertModule" ng-controller="InsertController">
<div>
<label>Name:</label>
<input type="text" ng-required="true" ng-model="name" ></input>
</div>
<div>
<label>Age:</label>
<input type="text" ng-required="true" ng-model="age" ></input>
</div>
<button type="button" ng-click="addPerson()"> Add </button>
</body>
</html>
```

src/main/resources/js/InsertModule.js

```
var menuModule =  
angular.module('InsertModule',  
[]);  
  
menuModule.controller('InsertController', function($scope, $http)  
{  
    var urlBase = "  
http://localhost:8080";  
    $scope.name = "";  
    $scope.age = "";  
    $http.defaults.headers.post["Content-Type"] = "application/json;  
charset=UTF-8";  
    $scope.addPerson = function  
addPerson() {  
    if($scope.name==" " ||  
    $scope.age==""){  
    alert("All values must be set.");  
    }  
}
```

```
else{  
    $http.post(urlBase +  
    '/persons', {  
    name: $scope.name,  
    age: $scope.age  
    })  
    .success(function(data,  
status, headers) {  
        $scope.name = "";  
        $scope.age = "";  
    }).error(function(data,  
status, headers) {  
        alert("Person not  
added.");  
    });  
    }  
};});
```

Αποτέλεσμα

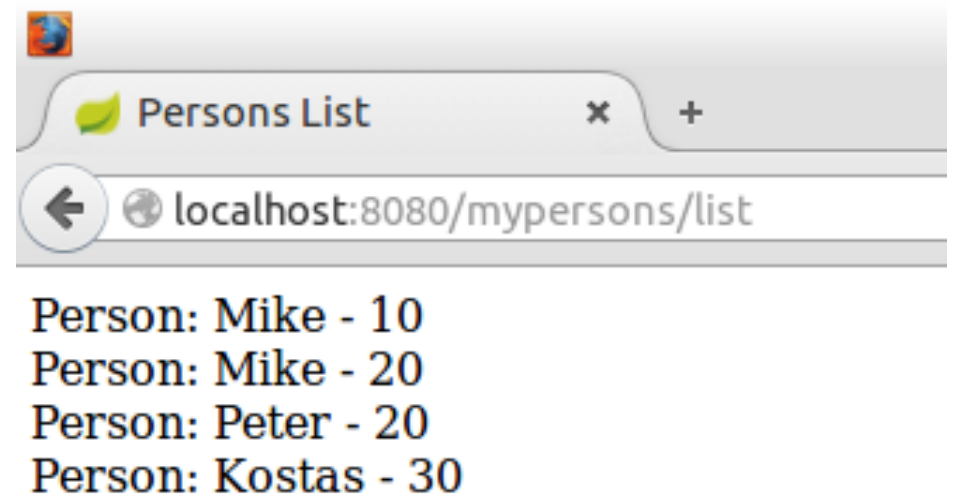


A screenshot of a web browser window with a single tab titled "Insert Page". The address bar shows "localhost:8080/insert.html". The page contains a form with two input fields: "Name:" and "Age:". Both fields are empty and have a red border. Below the fields is a button labeled "Add".

Name:

Age:

Add



A screenshot of a web browser window with a single tab titled "Persons List". The address bar shows "localhost:8080/mypersons/list". The page displays a list of four persons, each with a name and an age, separated by a hyphen.

Person: Mike - 10
Person: Mike - 20
Person: Peter - 20
Person: Kostas - 30

Επιπλέον πληροφορίες

- <http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>
- <https://github.com/spring-projects/spring-boot>
- <http://spring.io/docs>
- <http://www.infoq.com/articles/microframeworks1-spring-boot>

Ερωτήσεις;