

# TP Spring n°1

## La modélisation

Durant cette semaine nous allons travailler sur un projet simplifié d'e-commerce. Nous allons créer une solution avec un back-end en Spring afin de créer cette application de vente en ligne (simplifiée). Nous allons dans un premier temps modéliser cette solution. Faire le schéma UML de diagramme de classe, créer le projet sous Spring et retranscrire cette modélisation.

### 1° - Diagramme de classe

Nous avons besoin des classes suivantes :

- **Product** : classe qui représente un produit.
- **Order** : classe qui représente une commande.
- **Client** : classe qui représente un client.

On part du principe qu'une commande peut contenir plusieurs produits, et que l'on peut commander plusieurs fois le même produit dans une même commande. Une commande est passée par un client, et un client peut passer plusieurs commandes.

Il y a bien une association à faire entre Product et Order, nous allons modéliser cette association avec une classe associative :

- **OrderProduct** : classe qui représente un produit et sa quantité dans une commande.

1° - Créez le diagramme de classe avec UML avec les informations suivantes, certaines sont "manquantes" à vous de tirer les bonnes associations avec les bonnes cardinalités.

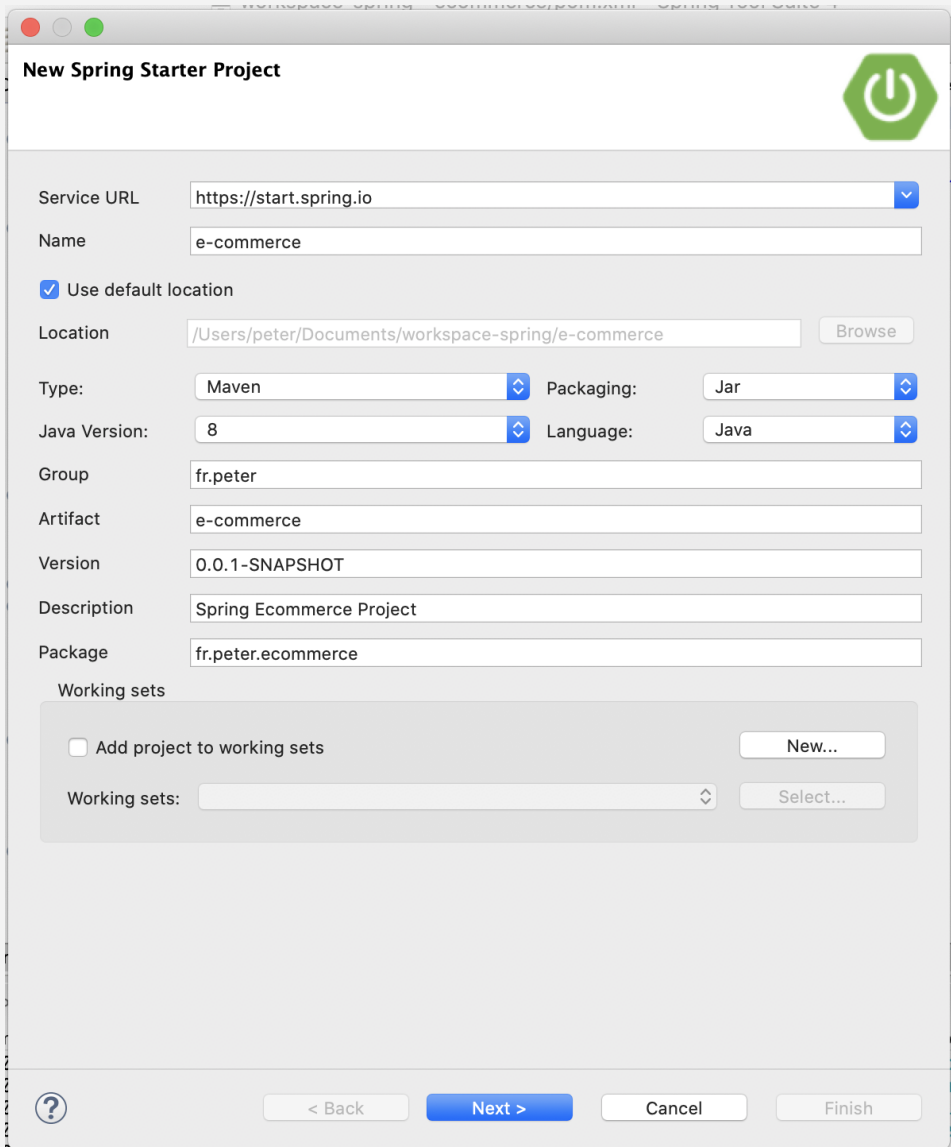
<b>Classe Product :</b> <ul style="list-style-type: none"><li>• <b>id</b> : Long</li><li>• <b>name</b> : String</li><li>• <b>description</b> : String</li><li>• <b>price</b> : Double</li><li>• <b>picture</b> : String</li><li>• <b>quantity</b> : Integer</li></ul>	<b>Classe Order :</b> <ul style="list-style-type: none"><li>• <b>id</b> : Long</li><li>• <b>dateCreated</b> : LocalDate</li><li>• <b>status</b> : String</li></ul>
<b>Classe Client :</b> <ul style="list-style-type: none"><li>• <b>id</b> : Long</li><li>• <b>username</b> : String</li><li>• <b>password</b> : String</li></ul>	<b>Classe OrderProduct :</b> <ul style="list-style-type: none"><li>• <b>quantity</b> : Integer</li></ul>

## 2° - Création d'un projet sous Spring

Utilisation de l'IDE : Spring Tool Suite 4. Téléchargement [ici](#).

File > New > Spring Starter Project :

### Création du projet



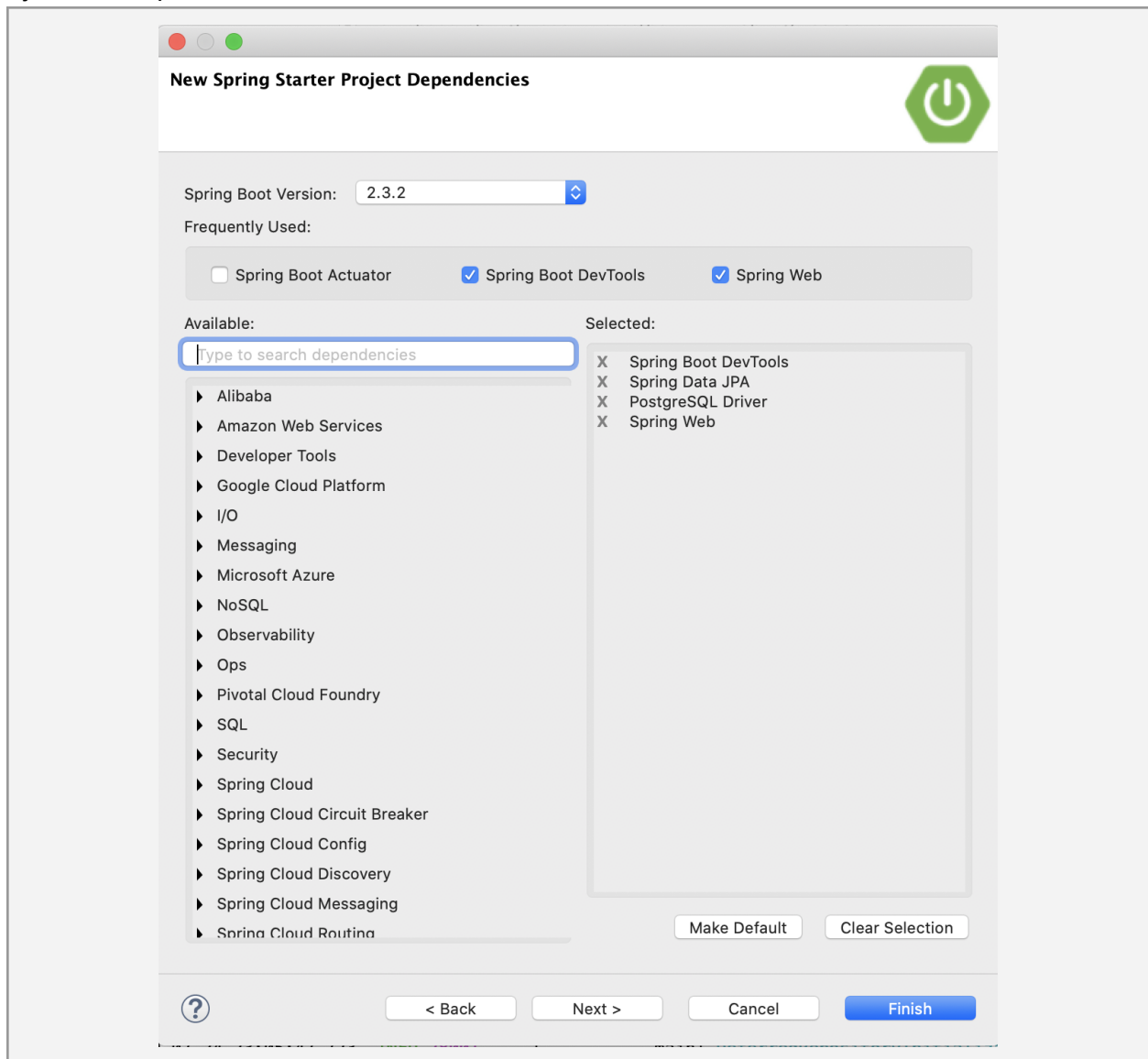
The screenshot shows the 'New Spring Starter Project' dialog box in Spring Tool Suite 4. The dialog has a title bar with standard macOS window controls and a green power button icon in the top right corner. The main content area contains several fields and options for configuring a new project:

- Service URL:** A dropdown menu set to 'https://start.spring.io'.
- Name:** A text field containing 'e-commerce'.
- Use default location:** A checked checkbox.
- Location:** A text field showing the path '/Users/peter/Documents/workspace-spring/e-commerce' and a 'Browse' button.
- Type:** A dropdown menu set to 'Maven'.
- Packaging:** A dropdown menu set to 'Jar'.
- Java Version:** A dropdown menu set to '8'.
- Language:** A dropdown menu set to 'Java'.
- Group:** A text field containing 'fr.peter'.
- Artifact:** A text field containing 'e-commerce'.
- Version:** A text field containing '0.0.1-SNAPSHOT'.
- Description:** A text field containing 'Spring Ecommerce Project'.
- Package:** A text field containing 'fr.peter.ecommerce'.
- Working sets:** A section with an unchecked checkbox 'Add project to working sets', a 'New...' button, a 'Working sets:' dropdown menu, and a 'Select...' button.

At the bottom of the dialog, there is a help icon (question mark) and four buttons: '< Back', 'Next >' (highlighted in blue), 'Cancel', and 'Finish'.

Next >

Ajoute de dépendances :



Selectionnez :

- Spring Boot DevTools
- Spring Data JPA
- PostgreSQL Driver
- Spring Web

Les dépendances sont en prévision de la suite.

Finish.

### 3° - Création du model

1° - Créez un nouveau package “model” et ajoutez les classes de votre modélisation. Implémentez vos classes avec les attributs tels qu’ils sont dans votre diagramme UML.

2° - Chaque classe contient un constructeur par défaut que ne fait rien que l’appel au constructeur parent via `super()`.

3° - Ajoutez des méthodes de type Get pour chaque attribut.

4° - Ajoutez les méthodes spécifiques suivantes :

Dans la classe `OrderProduct` :

- **`getTotalPrice`** : Double - calcule le prix de cette ligne de commande.

Dans la classe `Order` :

- **`getTotalOrderPrice`** : Double - calcule le prix total de cette commande.
- **`getNumberOfProducts`** : int - calcule le nombre de types de produits.
- **`getTotalNumberOfProducts`** : int - calcule le nombre total de produits.
- **`addProduct`** : void - ajoute un produit selon une quantité passée en paramètre (crée une instance de `OrderProduct` et l’ajoute à la liste).
- **`setStatus`** : void - Setter de l’attribut status.
- **`setClient`** : void - Setter de l’attribut client.

Dans la classe `Product` :

- **`setQuantity`** : void - Setter de l’attribut quantity.

5° - Ajouter des méthodes `toString` dans chaque classe.

Source > Generate `toString()`

### 4° - Instantiation

Si on souhaitait tester notre modèle, on peut faire un jeu de test en java “pure” dans le main.

Mettez en commentaire les deux lignes suivantes :

```
//@SpringBootApplication
```

```
//SpringApplication.run(EOcommerceApplication.class, args);
```

Créez 3-4 produits, créez un client. Puis créez une commande passée par ce client, ajoutez lui des produits en quantités. Passez la commande en changeant le statut à “payé”.

Affichez le résultat à l’aide de la méthode `toString` dans la console.

**Attention** : pour la méthode `toString` de `OrderProduct` à ne pas afficher les informations de `Order`... sinon... ça boucle...