TP Spring n°3
Les annotations Spring

Nous allons aborder dans ce TP l'utilisation des annotations avec Spring Framework. Les annotations ont été une avancée majeure de Java 5, cette technique a été intégrée dans Spring Framework à partir de la version 2.5.

Les annotations Java permettent de simplifier grandement les fichiers de configuration de Spring et de séparer plus nettement les beans techniques, qui restent configurés en fichier XML, des beans fonctionnels pour lesquels les annotations apportent des simplifications.

Comme nous avons des beans fonctionnels dans notre projet, on va maintenant les instanciés grâce aux annotations de Spring.

1° - Spring Bean Annotations

Selon l'utilisation de nos bean, on va pouvoir les déclarer avec un annotation d'un certain type. On retrouve les annotations suivantes que l'on peut mettre au dessus du nom de la classe du bean :

- ***@Component*** : l'annotation "mère" aux quatre autres ci-dessous.
- ***@Repository*** : Défini un bean représentant la couche d'accès à la base de données.
- ***@Service*** : Défini un bean représentant la couche métier avec la définition des services (fonctionnalités proposées).
- ***@Controller*** : Défini un bean représentant le couche "controller" dans Spring-MVC
- ***@Configuration*** : Défini un bean représentant la couche de configuration de méthodes qui sont annotées de *@Bean*.

L'annotation **@Service** permet de déclarer un bean de service, c'est-à-dire de la couche métier. Cette annotation peut avoir un paramètre afin de donner un nom à ce bean sinon c'est le nom de la classe par défaut. L'annotation @Scope permet de préciser la portée du bean; les valeurs possibles pour cette annotation sont "singleton" et "prototype".

Ajoutez cette annotation à la classe ProductServiceImpl :

```
@Service
public class ProductServiceImpl implements ProductService {
    ...
}
```

On peut le nommer via un id :

```
@Service("products")
public class ProductServiceImpl implements ProductService {
    ...
}
```

Le bean déclaré ici est exactement le même que celui du fichier XML ci-dessus : son id est products. Attention cela va faire un conflit et c'est la configuration du fichier XML qui est prise en compte, quel que soit l'ordre de déclaration dans le fichier XML.

Dans notre cas on va utiliser les annotations et donc on ne va plus charger le fichier XML.

2° - Création des services

Pour tester cette annotation nous allons lancer le programme non plus en chargeant les configurations depuis le fichier XML, mais grâce aux annotations.

Mettez en commentaire tout ce qu'il y a dans la méthode main (ou renommez ce main en *mainTP2* et mettez une méthode *main* pour ce TP3), ajoutez la ligne suivante seulement pour le moment :

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(ECommerceApplication.class);
```

On va récupérer le context, non plus par l'instanciation de *ClassPathXmlApplicationContext.* Mais par l'instanciation de *AnnotationConfigApplicationContext.*

On avait donné en paramètre le fichier XML de configuration, ici on va donner en paramètre notre Application (ECommerceApplication.class).

Mais pour que ça fonctionne, on va ajouter l'annotation à notre application de scanner tous les composants qu'elle possède. On ajoute au dessus de la déclaration de la classe *ECommerceApplication* l'annotation suivante :

```
@ComponentScan("fr.*")
```

Testez votre programme, vous devez voir apparaître dans la console l'affichage de la création de votre bean "products" :

```
... - Creating shared instance of singleton bean 'products'
```

Vous pouvez maintenant de la même façon, ajouter l'annotation devant les autres services.

3° - Spring Core Annotations

Ajoutez dans votre main, exactement de la même manière que pour le TP2, la création d'une commande ainsi que l'affichage de celle-ci après la mise à jour de son statut à "Payée".

Si vous exécutez ce programme, vous allez très certainement avoir une exception de type *NullPointException* qui sera levée dans la classe *OrderServiceImpl* lors de la suppression des stocks au moment de la mise à jour de la commande. Et oui, car le service commande doit connaître les produits pour fonctionner. On avait fait cette injection de dépendance dans le fichier de configuration mais que l'on utilise plus…

Du coup on va le faire via les annotations. On ajoute l'annotation **@Autowired** que l'on va placer au-dessus de l'attribut *productService* dans la classe *OrderServiceImpl.*

To wire : câbler en français, on va traduire autowired par câbler automatiquement.

Si on exécute à nouveau le programme, cela devrait fonctionner.

C'est un peu magique, ce sont les annotations… Bienvenue dans le framework Spring.