Séance 3

Spring Core & Tests unitaires

Application de gestion de location de véhicules

Exercice 1 : Injection de dépendances

Exercice 1.1 : Injection dans l'interface en ligne de commande

1. Dans le fichier pom.xml de votre application, ajoutez la dépendance à **spring-context**.

```
<dependencies>
  <!-- other dependencies -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.6.RELEASE</version>
  </dependency>
</dependencies>
```

2. Mettez en place une classe **AppConfiguration** permettant de scanner les beans que vous allez définir :

```
@Configuration
@ComponentScan({ "com.epf.rentmanager.service", "com.epf.rentmanager.dao",
"com.epf.rentmanager.persistence" }) // packages dans lesquels chercher les beans
public class AppConfiguration {
}
```

3.
4. Remplacez vos **DAOs** par des beans Spring (ces **DAOs** ont le scope **singleton,** en utilisant l'annotation : `@Repository` , et en supprimant la fonction getInstance) avec un contructeur sans argument.
5. Remplacez vos **services** par des beans Spring injectant les **DAOs** (ces services ont le scope singleton, `@Service` et avec pour constructeur :

```
        private ClientService(ClientDao clientDao){
                this.clientDao = clientDao;
        }
```

6. Testez le fonctionnement de l'interface en ligne de commande en utilisant :

```
ApplicationContext context = new AnnotationConfigApplicationContext(AppConfiguration.class);
ClientService clientService = context.getBean(ClientService.class);
VehicleService vehicleService = context.getBean(VehicleService.class);
```

Exercice 1.2 : Injection dans l'application web

1. Dans le fichier pom.xml de votre application, ajoutez la dépendance à la version **5.1.6-RELEASE** de l'artifact **spring-webmvc**, proposé par **org.springframework**.
2. Créez un classe **WebConfiguration**, implémentant **WebApplicationInitializer**, permettant de lier le contexte de Spring au contexte des Servlets :

```
@Configuration
```

```
@EnableWebMvc
public class WebConfiguration implements WebApplicationInitializer {

@Override
public void onStartup(ServletContext servletContext) throws ServletException {
AnnotationConfigWebApplicationContext rootContext = new
AnnotationConfigWebApplicationContext();
rootContext.register(AppConfiguration.class);
servletContext.addListener(new ContextLoaderListener(rootContext));
}
}
```

3. Utilisez de l'injection par attribut afin de fournir les services aux Servlets :

```
@Autowired
VehicleService vehicleService;

@Override
public void init() throws ServletException {
super.init();
SpringBeanAutowiringSupport.processInjectionBasedOnCurrentContext(this);
}
```

4. Testez le fonctionnement de l'application web.

Exercice 2 : Mise en place de tests

Exercice 2.1 : Création du dossier de tests

A l'aide de votre explorateur de fichiers, créez l'arborescence de dossiers suivante : **src/test/java/com/ensta/rentmanager**.

Exercice 2.2 : Mise en place de tests sur la couche DAO et Service

Faites quatre ou cinq cas de test, plutôt au niveau des Services, pour prendre en main le fonctionnement des tests et des mocks.

Exercice 3 : Ajouter des contraintes

- un client n'ayant pas 18ans ne peut pas être créé
- un client ayant une adresse mail déjà prise ne peut pas être créé
- le nom et le prénom d'un client doivent faire au moins 3 caractères
- une voiture ne peux pas être réservé 2 fois le même jour
- une voiture ne peux pas être réservé plus de 7 jours de suite par le même utilisateur
- une voiture ne peux pas être réservé 30 jours de suite sans pause

- si un client ou un véhicule est supprimé, alors il faut supprimer les réservations associées
- une voiture doit avoir un modèle et un constructeur, son nombre de place doit être compris entre 2 et 9