FORMATION SPRING

spring by Pivotal

PLAN DES TP

- TP spring IOC / AOP
- TP spring-boot
- TP spring-boot / Hibernate.
- TP Spring-boot REST.
- TP spring-mvc
- TP sping-security

TP IOC/AOP

TP IOC

- Adapter l'exemple 1 comme suit:
  - Supprimer toutes les déclaration de bean en xml
  - Remplacer par le scan des annotations:
    - <context:component-scan base-package="*fr.training.spring*" />
  - Passer par les annotations pour toutes les injections de beans
    - Injecter dans la classe Developer, une instance de la classe ProgrammingTask
    - Tester dans le Main, l'invocation de la méthode doTask
    - Injecter dans la classe Developer, une instance de la classe DocumentingTask
    - Tester dans le Main, l'invocation de la méthode doTask

TP AOP - INTERCEPTOR

- Adapter le projet example2-aop, comme suit:
  - Ecrire un interceptor qui permet de logger :
    - qu'on est au début d'une méthode
    - qui qu'on est à la fin de la méthode
    - Le temps passé dans chaque méthode
- NB: Vous pouvez vous inspirer du projet m-spring-000-hello

TP AOP - ASPECT

- Adapter le projet example3-aop, comme suit:
  - Créer un Aspect *fr.training.spring.LoggingAspect*
  - L'aspect permet de logger :
    - Les paramètres d'entrée, avant l' exécution de la méthode
  - Dans le fichier applicationContext.xml ajouter:
    - <aop:aspectj-autoproxy />
    - <bean id="*loggingAspect*" class="*fr.training.spring.LoggingAspect*" />
  - Evoluer l'aspect pour, logger Le temps passé dans chaque méthode

SPRING-BOOT INTRODUCTION.

TP: SPRING-BOOT HELLO WORLD !

- Importer le projet: sprring-boot-hello
- Quelle version de spring le projet embarque-t-il ?
- Lancer la classe MainHappy, puis analyser les résultats

TP: UTILISATION DES PROPERTIES SPRING-BOOT

- Créer un projet: spring-boot-properties
- Le projet référence la dépendance:

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

- Créer un fichier application.properties, dans le répertoire: src/main/resources.
- Ce fichier contient les propriétés suivantes:

```
logging.level.root = INFO
hello.message = Allo !
```

TP: UTILISATION DES PROPERTIES SPRING-BOOT

- Créer une classe de propriétés: HelloProperties
  - Le préfixe des propriétés est: hello
  - Cette classe contient une propriété message de type String
  - Utiliser l'annotation @ConfigurationProperties pour injecter des propriétés peronnalisées
- Créer une interface HelloService avec la méthode
  - void sayHello(String message);
- Créer une classe HelloServiceImpl (bean spring), qui implémente HelloService
  - Cette classe injecte et affiche : la propriété message de la classe HelloProperties
- Créer une classe main spring-boot, DemoApplication
  - Cette classe initialise le contexte spring-boot
  - Appel la méthode sayHello, du bean helloService
- Lancer la classe DemoApplication, puis analyser les résultats

TP: SPRING-BOOT-STARTER

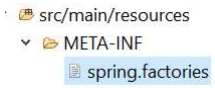
- Un starter permet d'avoir une configuration dynamique
- Un starter est déclaré comme une dépendance Maven
- Le processus de création peut être:
  - Automatique
  - Conditionné
- Le comportement d'un starter peut également être surchargé

TP: SPRING-BOOT-STARTER

- Importer le projet: sprring-boot-starter
- Le projet contient une interface StarterHello, avec son implémentation StarterHelloImpl
- Notre objectif est le suivant:
  - Quand un utilisateur importe notre starter, si un bean StarterHello n'existe pas, le starter va le créer

TP: SPRING-BOOT-STARTER

- Dans le projet sprring-boot-starter:
  - Ajouter le fichier src/main/resources/META-INF/spring.factories



  - Ce fichier contient la ligne suivante:
    - org.springframework.boot.autoconfigure.EnableAutoConfiguration=my.component.service.starter.AppConfig

TP: SPRING-BOOT-STARTER

- Pour créer votre propre starter, ajouter le bean de configuration suivant:

```
package my.component.service.starter;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

    @Bean
    StarterHello myHello() {
        return new StarterHelloImpl();
    }

}
```

TP: SPRING-BOOT-STARTER

- Pour tester le starter, ajouter la dépendance suivante dans le projet sprring-boot-hello:

```
<dependency>
    <groupId>apollotp</groupId>
    <artifactId>sprring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

- Lancer le MainHappy du projet spring-boot-hello
- Aller dans le projet: spring-boot-starter
  - Dans la classe AppConfig, renomer la méthode myHello en myHello1
- Lancer le MainHappy du projet spring-boot-hello
- Qu'est-ce qu'on constate ?

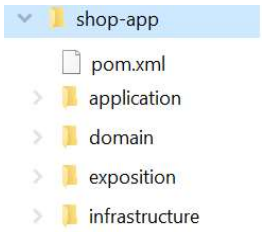
15

TP SPRING / HIBERNATE

16

TP: SHOP-APP

- L ’application suivante permet de gérer les commandes d’articles d’un client sur site web, pour cela:
  - Créer un répertoire shop-app dans workspace
  - Initialiser un projet shop-app avec la structure suivante:



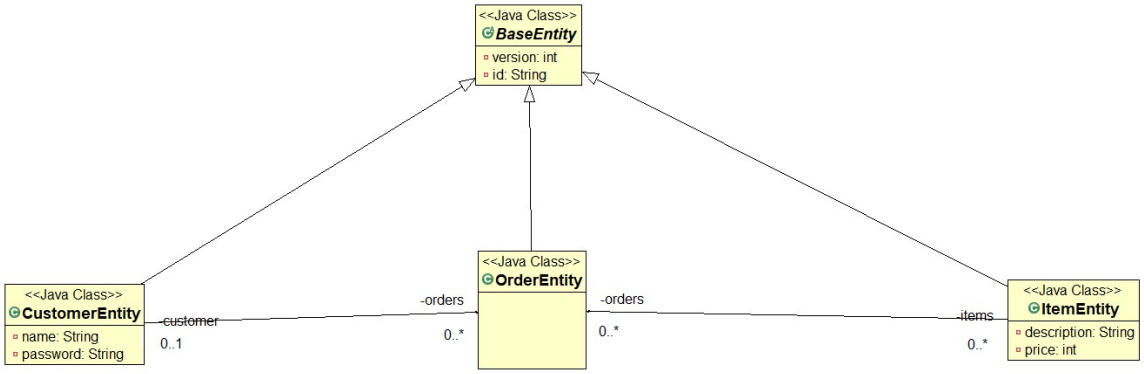
- ✓ Contient l’implémentation des services métiers
- ✓ Contient la couche domaine métier
- ✓ Contient l’exposition des API REST
- ✓ Contient les interface/Implémentation de la couche DAO

TP: SHOP-APP

- Le module domaine ne dépend d’aucun autre module
- Le module infrastructure dépend du module domaine
- Le module application dépend du module domaine
- Le module exposition dépend du module application
- Le module exposition dépend du module infrastructure en runtime

TP: SHOP-APP – LE DOMAINE

- La couche domaine, peut être modélisée comme suit:



TP: SHOP-APP – L'INFRA

- Installation de MySQL:
  - Lancer mysql-installer-community-5.7.11.0.msi
    - Choisir MySQL Server
    - Mot de passe: admin
  - Si besoin, il faudra lancer au préalable: dotNetFx40_Full_x86_x64.exe
  - Lancer vcredist_x86.exe
  - Lancer mysql-workbench-community-6.3.6-win32.msi
  - Créer le schéma en utilisant mysql-workbench: **formationdb**
  - Définir ce schéma comme schéma par defaut

TP: SHOP-APP - INFRA

- Ajouter la dépendance du dirver de la BD au niveau de la couche infrastructure

```
<dependency>
    <groupId>fr.training.samples</groupId>
    <artifactId>shop-app-domain</artifactId>
    <version>0.0.11-SNAPSHOT</version>
</dependency>


<!-- To replace with target DB driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

TP: SHOP-APP - INFRA

- Implémenter les repository dans le projet infrastructure
  - CustomerRepository -> en créant CustomerRepositoryImpl
  - ItemRepository -> en créant ItemRepositoryImpl
  - OrderRepository -> en créant OrderRepositoryImpl
- Vous pouvez utiliser l'interface JpaRepository de spring-data-jpa au niveau de la couche infrastructure
- Créer les classes de tests comme suit:
  - CustomerJpaRepositoryTest -> pour tester CustomerRepository
  - ItemRepositoryImplTest -> pour tester ItemRepository
  - OrderRepositoryImplTest -> pour tester OrderRepository

TP: SHOP-APP - APPLICATION

- Implémenter les interfaces suivantes
  - CustomerManagement -> en créant CustomerManagementImpl
  - ItemManagement -> en créant ItemManagementImpl
  - OrderManagement -> en créant OrderManagementImpl
- Il faudra utiliser spring pour injecter les repository dans les implémentations au niveau de la couche application
- Ecrire les tests unitaires avec Mockito

TP: SHOP-APP – APPLICATION -AOP

- Ajouter un Aspect, qui permet de logger le temps passé dans chaque méthode de la couche application

TP: SHOP-APP – APPLICATION - CACHE

- Ajouter la dépendance sur le projet shop-app-application

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

- Importer le fichier ehcache.xml, dans le répertoire src/main/resources/cache
- Mettre en Cache les résultats de l'appel de service getAllItems
  - Utilsier @Cacheable("itemCache")
- Implémenter la méthode addItem qui vide le cache

SPRING-BOOT REST.

26

TP: INITIATION À SPRING-BOOT-REST

- Créer le projet: spring-boot-hello-rest; avec le pom.xml suivant:

```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>${spring-boot.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <!-- DevTools -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

27

TP: INITIATION À SPRING-BOOT-REST

- On va créer une API REST, pour une entité sans association

```java
@Entity
public class Info {

    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Id
    private Long id;
    @Version
    private int version;
    private String message;
```

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface InfoRepository extends JpaRepository<business.drh.model.Info, Long> {

}
```

TP: INITIATION À SPRING-BOOT-REST

```java
@RestController
@RequestMapping("/infos")
public class InfoController {

    private @Autowired InfoRepository infoRepository;

    /**
     * findAll
     * GET
     * http://localhost:8080/infos
     */
    @RequestMapping(method = GET)
    public List<Info> findAll() {
        List<Info> infos = infoRepository.findAll();
        return infos;
    }
```

TP: INITIATION À SPRING-BOOT-REST

- Créer un fichier data.sql dans le répértoire: src/main/resources. Le contenu est comme suit:
  - **insert into info(id,version,message) values (1, 1, 'Alex');**
- Il faudra créer un classe Main de type @SpringBootApplication
- Tester l'accès:
  - http://localhost:8080/infos

TP: INITIATION À SPRING-BOOT-REST

- Implémenter l'API REST suivante, avec la méthode GET:
  - /infos/{id} => /infos/12

```
/**
 * findOne by repo (by default : no exception if not found)
 * GET
 * http://localhost:8080/repo/infos/25
 */
@RequestMapping(path = "/{idInfo}", method = GET)
public Info findOneRepo(@PathVariable Long idInfo) {
    Optional<Info> info = infoRepository.findById(idInfo);
    return info.get();
}
```

TP: INITIATION À SPRING-BOOT-REST

- Tester InfoController avec un id info existant
  - GET /infos/1
- Tester des cas limites avec un id info inexistant
  - GET /infos/55
  - Si un id info est non existant, il faudra retourner un code HTTP 404

TP: INITIATION À SPRING-BOOT-REST

- Evoluer l'exemple, pour avoir une gestion transverse des exceptions
- Vous pouvez passer @ControllerAdvice et @ExceptionHandler de spring

```java
@ControllerAdvice
public class RestExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<?> handleResourceNotFoundException(ResourceNotFoundException rnfe,
            HttpServletRequest request) {
        ErrorDetail errorDetail = new ErrorDetail();
        errorDetail.setTimeStampLong(new Date().getTime());
        errorDetail.setTimeStamp(new Date());
        errorDetail.setStatus(HttpStatus.NOT_FOUND.value());
        errorDetail.setTitle("Resource Not Found");
        errorDetail.setDetail(rnfe.getMessage());
        errorDetail.setDeveloperMessage(rnfe.getClass().getName());
        return new ResponseEntity<>(errorDetail, null, HttpStatus.NOT_FOUND);
    }
```

```json
{
  "title": "Resource Not Found",
  "status": 404,
  "detail": "info.not.found",
  "timeStamp": 1458729430742,
  "timeStampLong": 1458729430742,
  "developerMessage": "business.drh.controller.exception.ResourceNotFoundException"
}
```

33

TP: SHOP-APP – EXPOSITION

- Pour exposer les API REST sur l'application shop-app, ajouter les dépendances suivantes:

```xml
<!-- spring-boot dependencies -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

TP: SHOP-APP – EXPOSITION DES APIS REST

- En utilisant spring-boot-starter-web, exposer les ressources REST suivantes:

GET	/api/customers/{customerID}
PUT	/api/customers
POST	/api/customers
GET	/api/items
POST	/api/items
GET	/api/orders
POST	/api/orders

TP: SHOP-APP – DOCUMENTATION DES API

- Ajouter la documentation avec la librairie swagger

```
<dependency>
   <groupId>org.springdoc</groupId>
   <artifactId>springdoc-openapi-ui</artifactId>
   <version>1.4.4</version>
</dependency>
```

- Accéder à la liste des ressources:
  - http://localhost:8080/swagger-ui.html

TP: SHOP-APP – INTÉGRATION D'ACTUATOR

- Ajouter spring-boot-actuator au projet
- Ajouter la propriété:

```
#Monitoring endpoints
management:
   endpoints:
      web:
         exposure:
            include: info, health, configprops, logfile, metrics, env, loggers
```

- Exposer les métriques de chaque controlleur REST à l'aide de l'annotation @Timed
- Accéder au différentes urls d'actuator

SPRING-MVC

38

TP SPRING MVC – HELLO WORLD

- Créer un nouveau module Web: shop-app-webmvc
- Utiliser les dépendances fournies dans la fichier pom.xml
- Ajouter le dossier layouts dans le répertoire:
  - src/main/resources/templates/
- Ajouter un contrôleur HelloWorld, qui retourne un message « Hello World » à une vue
- Vous pouvez vous inspirer du lien suivants:
  - https://spring.io/guides/gs/serving-web-content/
- Tester l'accès à partir de l'url:
  - http://localhost:8081/welcome

TP SPRING MVC – SPRING SHOP - ITEM CONTROLLER

- Le but de cette étape de est créer une page Web qui affiche une liste de tous les items de la base de données.
- Pour cela, on va suivre les étapes suivantes:
  - Ajouter dans les dépendances des projets:
    - shop-app-application
    - shop-app-infrastructure
  - Créer un ItemController annoté avec une méthode qui appelle le ItemManagement et retourne un ModelAndView avec la liste des éléments en tant que modèle
- Ajouter le vue items.html dans le répertoire:
  - src/main/resources/templates/items.html
- Tester l'accès:
  - http://localhost:8081/items

TP SPRING MVC – SPRING SHOP – ORDER CONTROLLER - VALIDATIONS

- Le but de cette étape de est créer une page Web qui qui permet d'ajouter une commande
  - Ajouter le vue addOrder.html dans le répertoire: src/main/resources/templates/
    - Cette vue contient:
      - N ° de client
      - Numéro de commande
      - Pour chaque items de la base de données, une case à cocher pour indiquer si elle doit être incluse dans la commande
  - Valider que tous les éléments sont renseignées
  - Tester l'accès:
    - http://localhost:8081/addOrder

TP SPRING MVC – SPRING SHOP

- L'application shop-app-webmvc, ne devrait gérer que la couche présentation
  - Supprimer les dépendances des projets:
    - shop-app-application
    - shop-app-infrastructure
- Faire les appels des APIs REST du projet shop-app-exposition, depuis le projet: shop-app-webmvc. (Vous pouvez passer RestTemplate ou WebClient)