

I - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

https://spring.io/projects/spring-framework	Site officiel de spring

2. Tp spring-framework sans SpringBoot

L'objectif de cette série de Tps est d'appréhender les fonctionnalités essentielles de Spring via une approche très progressive .

2.1. Mise en place d'un projet et configurations de base

- Créer (via eclipse ou autre) une nouvelle application maven appelée "**appliSpringSansSpringBoot**" ou autrement . (skip archetype , packaging="war")
- Ajuster le fichier **pom.xml** en s'inspirant de l'exemple suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>tp</groupId>
  <artifactId>appSpringSansSpringBoot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>appSpringSansSpringBoot</name>
  <packaging>war</packaging>
  <description>appSpringSansSpringBoot</description>
  <properties>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <java.version>11</java.version>
    <spring.version>5.3.21</spring.version>
    <junit.jupiter.version>5.8.1</junit.jupiter.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!-- windows/preferences/general/workspace / UTF8 avec eclipse coherent -->
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax.inject</groupId>
      <artifactId>javax.inject</artifactId>
      <version>1</version>
    </dependency> <!-- pour que Spring puisse interpreter @Inject comme @Autowired -->

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency> <!-- et indirectement spring-bean, spring-core , spring-aop -->
  </dependencies>
</project>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>${spring.version}</version>
</dependency> <!-- et indirectement aspectj-weaver -->

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency> <!-- et indirectement spring-web -->

<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.12.7</version>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope> <!-- provided by tomcat after deploying .war -->
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>${spring.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.17.2</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>${junit.jupiter.version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>2.1.214</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency> <!-- et indirectement spring-jdbc, spring-tx -->

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.6.6.Final</version>
</dependency> <!-- et indirectement jpa -->
```

```

    <dependency>
      <groupId>javax.annotation</groupId>
      <artifactId>javax.annotation-api</artifactId> <!-- @PostConstruct -->
      <version>1.3.2</version>
    </dependency>

    <!--
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
    </dependency>
    -->
  </dependencies>

  <build>
    <finalName>appSpringSansSpringBoot</finalName> <!-- to build appSpringSansSpringBoot.war -->
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.2</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.2</version>
      </plugin>
    </plugins>
  </build>
</project>

```

- effectuer d'éventuels ajustements (java 8 ou 17 , ...), update project, ...
- créer un nouveau package principal "***tp.appliSpring.core***" et un package secondaire "***tp.appliSpring.exemple***" .
- dans package ***tp.appliSpring.exemple*** , Créer une interface élémentaire ***MonCalculateur***

```

package tp.appliSpring.exemple;
public interface MonCalculateur {
    double calculer(double x);
    //...
}

```

- dans package ***tp.appliSpring.exemple*** , Créer la classe ***MonCalculateurCarre***

```

package tp.appliSpring.exemple;
import org.springframework.stereotype.Component;

@Component
public class MonCalculateurCarre implements MonCalculateur {
    public double calculer(double x) {
        return x*x;
    }
}

```

```
}
}
```

- dans package ***tp.appliSpring.exemple*** , Créer la classe ***ExempleConfig***

```
package tp.appliSpring.exemple;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = { "tp.appliSpring.exemple" })
public class ExempleConfig {
    /* @ComponentScan() pour demander à spring de parcourir les classes de certains
    packages pour y trouver des annotations @Component , @Service , @Autowired à
    analyser et interpréter */
}
```

- dans package ***tp.appliSpring.exemple*** , Créer la classe ***ExempleApp***

```
package tp.appliSpring.exemple;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class ExempleApp {
    public static void main(String[] args) {
        ApplicationContext contextSpring = new
            AnnotationConfigApplicationContext(ExempleConfig.class);
        //contextSpring représente un ensemble de composants pris en charge par spring
        //et qui est initialisé selon une ou plusieurs classes de configuration.

        MonCalculateur monCalculateur = contextSpring.getBean(MonCalculateur.class);
        System.out.println("4*4="+monCalculateur.calculer(4)); //4*4=16.0 ou autre
        ((AnnotationConfigApplicationContext) contextSpring).close();
    }
}
```

- Lancer cet l'exécution de cet exemple
- Beaucoup de choses seront approfondies ultérieurement

2.2. Très rapide aperçu sur ancienne config XML

L'application exemple "***oldXmlSpringApp***" (au format "maven") du référentiel git <https://github.com/didier-mycontrib/jee-spring-app-demo> est un **exemple simple** de configuration Spring XML .

Il est possible de charger ce projet dans un IDE tel que eclipse pour ensuite lancer l'application ou bien les tests unitaires .

La partie configuration XML se situe dans le sous répertoire **src/main/resources** .

2.3. Bases de l'injection de dépendance (avec @Autowired)

- dans package *tp.appliSpring.exemple* , Créer une interface élémentaire *MonAfficheur*

```
package tp.appliSpring.exemple;
public interface MonAfficheur {
    void afficher(String message);
    void afficherMaj(String message); //affichage en majuscule via .toUpperCase()
}
```

- dans package *tp.appliSpring.exemple* , coder la classe *MonAfficheurV1* implémentant l'interface *MonAfficheur* via un code de ce genre : *System.out.println(">> "+message);*
- dans package *tp.appliSpring.exemple* , Créer la classe *Coordinateur* avec le début de code suivant (à compléter)

```
package tp.appliSpring.exemple;
//...

@Component
public class Coordinateur {

    //...
    private MonAfficheur monAfficheur=null; //référence vers afficheur à injecter

    //...
    private MonCalculateur monCalculateur=null; //référence vers calculateur à injecter

    public void calculerEtAfficher() {
        double x=4;
        double res=monCalculateur.calculer(x); //x*x ou bien 2*x ou bien ...
        monAfficheur.afficher("res="+res); // >> res=16 en v1 ou bien ** res=16
    }
}
```

- Au sein de la méthode *main()* de la classe *ExempleApp* , ajouter un bloc de code de ce genre :

```
Coordinateur coordonateurPrisEnChargeParSpring =
    contextSpring.getBean(Coordinateur.class);
coordonateurPrisEnChargeParSpring.calculerEtAfficher();
```

- **Compléter le code de la classe *Coordinateur* (et ajuster si besoin d'autres classes) de manière à ce que cet exemple fonctionne bien.**
- On pourra coder et tester successivement plein de variantes d'injection de dépendances :
 - via **@Autowired** (ou bien **@Resource** ou bien **@Inject**) sans ou avec affichage des éléments injectés au sein du constructeur par défaut de la classe *Coordinateur* et d'une méthode *initialiser()* préfixée par **@PostConstruct**
 - via des ajouts de *MonAfficheurV2* (avec préfixe ******* plutôt que **>>>**) et *MonCalculateurDouble* (**2*x** plutôt que **x*x**) de manière à engendrer une ambiguïté.
 - Via des ajouts de **@Qualifier** pour lever les ambiguïtés
 - Via une expérimentation de **l'injection par constructeur** (par exemple dans une classe *"CoordinateurAvecInjectionParConstructeur"*)

2.4. Configurations via classes java (@Configuration, @Bean)

- Dupliquer tout le package *tp.appliSpring.exemple* (et son contenu) dans un nouveau package *tp.appliSpring.exemplev2*
- Au sein de *tp.appliSpring.exemplev2*, remanier tout le code existant en :
 - enlevant toutes les annotations existantes de type `@Autowired`, `@Component`, `@Qualifier` (seules resteront `@Configuration` et `@Bean`)
 - supprimant `@ComponentScan(basePackages = { "tp.appliSpring.exemple" })` au dessus de la classe *exemplev2.ExempleConfig* à renommer ***ExempleConfigExplicite***
 - paramétrant les composants calculateur, afficheur, coordinateur via des méthodes préfixées par ***@Bean*** au sein de la classe *exemplev2.ExempleConfigExplicite*.
 - Ajuster *exemplev2.ExempleApp* utilisant *ExempleConfigExplicite*.
 - Mettre au point une cohérence entre les parties de *exemplev2*.
 - Tester le tout (avec d'éventuelles variantes).
 - On pourra éventuellement analyser un fichier de config de ce type
src/main/resources/exemples.properties
exemple.calculateur=tp.appliSpring.exemplev2.MonCalculateurCarre
#exemple.calculateur=tp.appliSpring.exemplev2.MonCalculateurDouble
 et en tenir compte dans ***ExempleConfigExplicite***
 - On pourra expérimenter différents profils : par exemple `@Profile("V1")` ou `@Profile("V2")` près de `@Bean` sur afficheurs ...V1 et ...V2 au sein d'une variante ***ExempleConfigExpliciteAvecProfils***.
 Tests/Appels via variante ***ExempleAppAvecProfils.main()*** comportant
System.setProperty("spring.profiles.active", "V1");
ApplicationContext contextSpring = new
AnnotationConfigApplicationContext(ExempleConfigExpliciteAvecProfils.class);

NB : cette variante "*exemplev2/explicite*" est plus complexe que l'ancienne variante "*exemple*" et n'a pas beaucoup d'intérêt tel quel.

Par contre, au sein d'un projet plus complexe, la configuration explicite basée sur `@Bean` peut s'avérer très utile pour paramétrer des composants "spring" basés sur des classes (récupérées via maven depuis une librairie externe) dont on n'a pas le droit de changer le code source.

2.5. Mise en place d'un aspect de type "log automatique"

Mettre en place un **aspet/aspect** (via Spring AOP, paramétré via annotations de AspectJ) qui affichera des lignes de logs pour chaque appel d'une méthode d'une classe du package *tp.appliSpring.exemple*.

On pourra par exemple préciser le temps d'exécution et les noms des méthodes invoquées.

NB : ce TP (pas fondamental) n'est pas prioritaire : à faire ou pas selon le temps disponible.

2.6. Accès aux données (DataSource JDBC) , DAO

- Créer package *tp.appliSpring.core.entity*
- Créer classe *Compte.java*

```
package tp.appliSpring.core.entity;

public class Compte {

    private Long numero;
    private String label;
    private Double solde;

    //+get/set , constructeurs , toString()
}
```

- Créer package *tp.appliSpring.core.dao*
- Créer interface *DaoCompte.java*

```
package tp.appliSpring.core.dao;

import java.util.List;
import tp.appliSpring.core.entity.Compte;

public interface DaoCompte{
    Compte findById(Long numCpt);
    Compte save(Compte compte); //sauvegarde au sens saveOrUpdate
    List<Compte> findAll();
    void deleteById(Long numCpt);
    //...
}
```

- dans *src/main/resources* ajouter **application.properties** avec ce contenu :

```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:~/mydbbank
spring.datasource.username=sa
spring.datasource.password=
```

- dans *tp.appliSpring.core* ajouter **MySpringApplication** avec ce contenu :

```
package tp.appliSpring.core;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

//version sans springBoot
@Configuration
@ComponentScan(basePackages = { "tp.appliSpring.core"})

//NB : Tous les sous packages de tp.appliSpring.core seront scrutés pour y découvrir
//@Component... et aussi pour y découvrir d'autres classes avec @Configuration
public class MySpringApplication {
```

```

public static void main(String[] args) {
    //System.setProperty("spring.profiles.active", "p1");

    AnnotationConfigApplicationContext springContext = new
        AnnotationConfigApplicationContext(MySpringApplication.class);
    //...
    springContext.close();
}
}

```

- Créer package **tp.appliSpring.core.config**
- Créer la classe **CommonConfig.java**

```

package tp.appliSpring.core.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.support.PropertySourcesPlaceholderConfigurer;

@Configuration
@PropertySource("classpath:/application.properties")
public class CommonConfig {

    @Bean
    public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
        //pour pouvoir interpréter ${} in @Value()
    }
}

```

- Créer la classe **DataSourceConfig.java**

```

package tp.appliSpring.core.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
public class DataSourceConfig {

    @Value("${spring.datasource.driverClassName}")
    private String jdbcDriver;

    @Value("${spring.datasource.url}")
    private String dbUrl;

    @Value("${spring.datasource.username}")

```



```

private String dbUsername;

@Value("${spring.datasource.password}")
private String dbPassword;

@Bean(name="dataSource")
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(jdbcDriver);
    dataSource.setUrl(dbUrl);
    dataSource.setUsername(dbUsername);
    dataSource.setPassword(dbPassword);
    return dataSource;
}

//seulement utile pour le dao en version Jdbc (avec NamedParameterJdbcTemplate):
@Bean()
public NamedParameterJdbcTemplate namedParameterJdbcTemplate( DataSource dataSource) {
    return new NamedParameterJdbcTemplate(dataSource);
}
}

```

- dans *src/test/java* et dans un package *tp.appliSpring.dao* à créer, ajouter cette classe de test :

```

package tp.appliSpring.dao;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import tp.appliSpring.core.MySpringApplication;
import tp.appliSpring.core.dao.DaoCompte;
import tp.appliSpring.core.entity.Compte;

@ExtendWith(SpringExtension.class) //si junit5/jupiter
@ContextConfiguration(classes= {MySpringApplication.class})
public class TestCompteDao {

    private static Logger logger = LoggerFactory.getLogger(TestCompteDao.class);

    @Autowired
    @Qualifier("jdbc")
    //@Qualifier("jpa")
    private DaoCompte daoCompte; //à tester

    @Test
    public void testAjoutEtRelectureEtSuppression() {
        //hypothese : base avec tables vides et existantes au lancement du test
    }
}

```

```

Compte compte = new Compte(null,"compteA",100.0);
Compte compteSauvegarde = this.daoCompte.save(compte); //INSERT INTO
logger.debug("compteSauvegarde=" + compteSauvegarde);

Compte compteRelu = this.daoCompte.findById(compteSauvegarde.getNumero());
Assertions.assertEquals("compteA",compteRelu.getLabel());
Assertions.assertEquals(100.0,compteRelu.getSolde());
logger.debug("compteRelu apres insertion=" + compteRelu);

compte.setSolde(150.0); compte.setLabel("compte_a");
Compte compteMisAJour = this.daoCompte.save(compte); //UPDATE
logger.debug("compteMisAJour=" + compteMisAJour);

compteRelu = this.daoCompte.findById(compteSauvegarde.getNumero()); //SELECT
Assertions.assertEquals("compte_a",compteRelu.getLabel());
Assertions.assertEquals(150.0,compteRelu.getSolde());
logger.debug("compteRelu apres miseAJour=" + compteRelu);

//+supprimer :
this.daoCompte.deleteById(compteSauvegarde.getNumero());

//verifier bien supprimé (en tentant une relecture qui renvoi null)
Compte compteReluApresSuppression =
    this.daoCompte.findById(compteSauvegarde.getNumero());
Assertions.assertTrue(compteReluApresSuppression == null);
}
}

```

Script de préparation de la base de données (ici en version H2) :

init_db.sql

```

DROP TABLE IF EXISTS Compte;

CREATE TABLE Compte(
    numero integer auto_increment NOT NULL,
    label VARCHAR(64),
    solde double,
    PRIMARY KEY(numero));

INSERT INTO Compte (label,solde) VALUES ('compte courant',100);
INSERT INTO Compte (label,solde) VALUES ('compte codevi',50);
INSERT INTO Compte (label,solde) VALUES ('compte 3',150);

SELECT * FROM Compte;

```

set_env.bat

```

set MVN_REPOSITORY=C:\Users\d2fde\.m2\repository
set MY_H2_DB_URL=jdbc:h2:~/mydbbank
set H2_VERSION=2.1.214

```

```
set H2_CLASSPATH=%MVN_REPOSITORY%\com\h2database\h2\%H2_VERSION%\h2-%H2_VERSION%.jar
```

create_h2_database.bat

```
cd /d %~dp0
call set_env.bat
java -classpath %H2_CLASSPATH% org.h2.tools.RunScript -url %MY_H2_DB_URL% -user sa -script init_db.sql -showResults
pause
```

lancer_console_h2.bat

```
cd /d %~dp0
call set_env.bat
java -jar %H2_CLASSPATH% -user "sa" -url %MY_H2_DB_URL%
```

REM NB: penser à se déconnecter pour éviter des futurs verrous/blocages
pause

NB: Toute cette structure de code et configuration sera utilisée dès le(s) TP(s) suivant(s)

2.7. Petit exemple de DAO via JDBCTemplate

NB: Ce TP pas fondamental est facultatif : à faire ou pas selon le temps disponible

coder le début de ***DaoCompteJdbc.java*** avec le code suivant (*à compléter*)

```
package tp.appliSpring.core.dao;

import java.sql.ResultSet;    import java.sql.SQLException;
import java.util.HashMap;    import java.util.List;    import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Repository;
import tp.appliSpring.core.entity.Compte;

@Repository // @Component de type DAO/Repository
@Qualifier("jdbc")
public class DaoCompteJdbc /*extends JdbcDaoSupport*/ implements DaoCompte {

    private final String INSERT_SQL = "INSERT INTO compte(label, solde) values(:label,:solde)";
    private final String UPDATE_SQL = "UPDATE compte set label=:label , solde=:solde where numero=:numero";
    private final String FETCH_ALL_SQL = "select * from compte";
    private final String FETCH_BY_NUM_SQL = "select * from compte where numero=:numero";
    private final String DELETE_BY_NUM_SQL = "delete from compte where numero=:numero";

    @Autowired
    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    @Override
```

```

public Compte findById(Long numCpt) {
    Compte compte = null;
    Map<String, Long> parameters = new HashMap<String, Long>();
    parameters.put("numero", numCpt);
    List<Compte> comptes = namedParameterJdbcTemplate.query(FETCH_BY_NUM_SQL,
                                                            parameters, new CompteMapper());

    compte = comptes.isEmpty()?null:comptes.get(0);
    return compte;
}

@Override
public Compte save(Compte compte) {
    if(compte==null)
        throw new IllegalArgumentException("compte must be not null");
    return (compte.getNumero()==null)?insert(compte):update(compte);
}

public Compte insert(Compte compte) {
    KeyHolder holder = new GeneratedKeyHolder(); //to retrieve auto_increment value of pk
    SqlParameterSource parameters = new MapSqlParameterSource()
                                    .addValue("label", compte.getLabel())
                                    .addValue("solde", compte.getSolde());
    namedParameterJdbcTemplate.update(INSERT_SQL, parameters, holder);
    compte.setNumero(holder.getKey().longValue()); //store auto_increment pk in instance to return
    return compte;
}

public Compte update(Compte compte) {
    //A CODER/COMPLETER EN TP
}

@Override
public List<Compte> findAll() {
    //A CODER/COMPLETER EN TP
}

@Override
public void deleteById(Long numCpt) {
    //A CODER/COMPLETER EN TP
}

}

//classe auxiliaire "CompteMapper" pour convertir ResultSet jdbc en instance de la classe Compte :
class CompteMapper implements RowMapper<Compte> {
    @Override
    public Compte mapRow(ResultSet rs, int rowNum) throws SQLException {
        Compte compte = new Compte();
        compte.setNumero(rs.getLong("numero"));
        compte.setLabel(rs.getString("label"));
        compte.setSolde(rs.getDouble("solde"));
        return compte;
    }
}

```

- Compléter le code manquant de cette classe
- tester via le lancement de **TestCompteDao** (dans src/test/java)

2.8. Accès aux données via JPA/Hibernate

Ajouter dans le package *tp.appliSpring.core.config* la classe de configuration *DomainAndPersistenceConfig.java* suivante :

```
package tp.appliSpring.core.config;

import java.util.Properties;    import javax.sql.DataSource;
import javax.persistence.EntityManagerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.Database;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement() // "transactionManager" (not "txManager") is expected !!!
@ComponentScan(basePackages = { "tp.appliSpring.core.dao" ,
                                "tp.appliSpring.core.service" , "tp.appliSpring.core.init"})
public class DomainAndPersistenceConfig {

    // JpaVendorAdapter (Hibernate ou OpenJPA ou ...)
    @Bean
    public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter hibernateJpaVendorAdapter =
            new HibernateJpaVendorAdapter();
        hibernateJpaVendorAdapter.setShowSql(false);
        hibernateJpaVendorAdapter.setGenerateDdl(false);
        //hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
        hibernateJpaVendorAdapter.setDatabase(Database.H2);
        return hibernateJpaVendorAdapter;
    }

    // EntityManagerFactory
    @Bean(name = { "entityManagerFactory" })
    public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter,
                                                    DataSource dataSource) {

        LocalContainerEntityManagerFactoryBean factory =
            new LocalContainerEntityManagerFactoryBean();
        factory.setJpaVendorAdapter(jpaVendorAdapter);
        factory.setPackagesToScan("tp.appliSpring.core.entity");
        factory.setDataSource(dataSource);

        Properties jpaProperties = new Properties(); // java.util
        jpaProperties.setProperty("javax.persistence.schema-generation.database.action",
                               "drop-and-create"); //JPA>=2.1
        factory.setJpaProperties(jpaProperties);
        factory.afterPropertiesSet();
        return factory.getObject();
    }
}
```

```

}

// Transaction Manager for JPA or ...
@Bean(name = "transactionManager")
public PlatformTransactionManager transactionManager(
    EntityManagerFactory entityManagerFactory) {
    JpaTransactionManager txManager = new JpaTransactionManager();
    txManager.setEntityManagerFactory(entityManagerFactory);
    return txManager;
}
}

```

Coder au sein du package *tp.appliSpring.core.dao* la classe *DaoCompteJpa* en partant du code suivant (à compléter) :

```

package tp.appliSpring.core.dao;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import tp.appliSpring.core.entity.Compte;

@Repository // @Component de type DAO/Repository
@Qualifier("jpa")
public class DaoCompteJpa implements DaoCompte {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public Compte findById(Long numCpt) {
        // A CODER/COMPLETER EN TP
    }

    /*
    public Compte save(Compte compte) {
        try {
            entityManager.getTransaction().begin();
            if(compte.getNumero()==null)
                entityManager.persist(compte); // INSERT INTO
            else
                entityManager.merge(compte); // UPDATE
            entityManager.getTransaction().commit();
        } catch (Exception e) {
            entityManager.getTransaction().rollback();
            e.printStackTrace();
        }
        return compte; // avec numero plus null (auto_incrémenté)
    }
    */
}

```

```

@Override
@Transactional
public Compte save(Compte compte) {
    if(compte.getNumero()==null)
        entityManager.persist(compte); //INSERT INTO
    else
        entityManager.merge(compte); //UPDATE
    return compte; //avec numero plus null (auto_incrémenté)
}

@Override
public List<Compte> findAll() {
    return entityManager.createQuery("SELECT c FROM Compte c",
                                    Compte.class)
        .getResultList();
}

@Override
@Transactional
public void deleteById(Long numCpt) {
    // A CODER/COMPLETER EN TP
    Compte compte = .....
    entityManager.....(compte);
}
}

```

- Compléter le code de la classe ci-dessus
- Ajouter toutes les **annotations** manquantes et nécessaires dans la classe **tp.appliSpring.core.entity.Compte** (@Entity, @Id ,, @GeneratedValue(strategy = GenerationType.IDENTITY))
- switcher de qualificatif **@Qualifier("jdb")** **Qualifier("jpa")** au sein de la classe **TestCompteDao**
- Lancer le test et corriger les éventuels problèmes/erreurs .

2.9. Service Spring et gestion des transactions

Créer le nouveau package **tp.appliSpring.core.service**

Ajouter y l'interface **ServiceCompte** suivante :

```

package tp.appliSpring.core.service;

import java.util.List;
import tp.appliSpring.core.entity.Compte;

public interface ServiceCompte {
    Compte rechercherCompteParNumero(long numero);
    List<Compte> rechercherTousComptes();
    List<Compte> rechercherComptesDuClient(long numClient);
    Compte sauvegarderCompte(Compte compte);
    void supprimerCompte(long numCpt);
    void transferer(double montant, long numCptDeb, long numCptCred);
}

```


Ajouter la classe d'implémentation **ServiceCompteImpl** suivante (à compléter) :

```
package tp.appliSpring.core.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import tp.appliSpring.core.dao.DaoCompte;
import tp.appliSpring.core.entity.Compte;

@Service //classe de Service prise en charge par spring
public class ServiceCompteImpl implements ServiceCompte{

    @Qualifier("jpa")
    @Autowired
    private DaoCompte daoCompte=null;

    public Compte rechercherCompteParNumero(long numero) {
        return daoCompte.findById(numero);
    }

    public Compte sauvegarderCompte(Compte compte) {
        return daoCompte.save(compte);
    }

    public List<Compte> rechercherTousComptes() {
        // A CODER/COMPLETER EN TP
    }

    public List<Compte> rechercherComptesDuClient(long numClient) {
        // à coder ultérieurement (future version)
        return null;
    }

    public void supprimerCompte(long numCpt) {
        // A CODER/COMPLETER EN TP
    }

    @Transactional(*propagation = Propagation.REQUIRED*) //REQUIRED par défaut
    public void transferer(double montant, long numCptDeb, long numCptCred) {
        try {
            // transaction globale initialisée dès le début de l'exécution de transferer
            Compte cptDeb = this.daoCompte.findById(numCptDeb);
            //le dao exécute son code dans la grande transaction
            //commencée par le service sans la fermer et l'objet cptDeb remonte à l'état persistant
            cptDeb.setSolde(cptDeb.getSolde() - montant);
            //this.daoCompte.save(cptDeb); //facultatif si @Transactional

            //idem pour compte à créditer
            Compte cptCred= this.daoCompte.findById(numCptCred);
```



```

    cptCred.setSolde(cptCred.getSolde() + montant);
    //this.daoCompte.save(cptCred) //facultatif si @Transactional

    //en fin de transaction réussie (sans exception) , toutes les modification effectuées
    //sur les objets à l'état persistant seront répercutées en base (.save() automatiques)
    } catch (Exception e) {
        throw new RuntimeException("echec virement " + e.getMessage() , e);
        //rollback se fait de façon fiable
        //ou bien throw new
        //ClasseExceptionPersonnaliseeHeritantDeRuntimeException("echec virement" , e);
    }
}
}

```

Au sein de `src/test/java` et du package `tp.appliSpring.core.service` (à créer) , ajouter la classe de test ***TestServiceCompte*** suivante :

```

package tp.appliSpring.service;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import tp.appliSpring.core.MySpringApplication;
import tp.appliSpring.core.entity.Compte;
import tp.appliSpring.core.service.ServiceCompte;

@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes= {MySpringApplication.class})
//@ActiveProfiles({ "embeddedDB" , "dev" , "perf" })
public class TestServiceCompte {

    private static Logger logger = LoggerFactory.getLogger(TestServiceCompte.class);

    @Autowired
    private ServiceCompte serviceCompte; //à tester

    @Test
    public void testVirement() {
        Compte compteASauvegarde = this.serviceCompte.sauvegarderCompte(
            new Compte(null,"compteA",300.0));
        Compte compteBSauvegarde = this.serviceCompte.sauvegarderCompte(
            new Compte(null,"compteB",100.0));
        long numCptA = compteASauvegarde.getNumero();
        long numCptB = compteBSauvegarde.getNumero();
        //remonter en memoire les anciens soldes des compte A et B avant virement
        //( +affichage console ou logger) :
        double soldeA_ avant= compteASauvegarde.getSolde();
    }
}

```

```

double soldeB_avant = compteBSauvegarde.getSolde();
logger.debug("avant bon virement, soldeA_avant="+soldeA_avant +
            " et soldeB_avant=" + soldeB_avant);

//effectuer un virement de 50 euros d'un compte A vers vers compte B
this.serviceCompte.transférer(50.0, numCptA, numCptB);

//remonter en memoire les nouveaux soldes des compte A et B apres virement
// (+affichage console ou logger)
Compte compteAReluApresVirement =
    this.serviceCompte.rechercherCompteParNumero(numCptA);
Compte compteBReluApresVirement =
    this.serviceCompte.rechercherCompteParNumero(numCptB);
double soldeA_apres = compteAReluApresVirement.getSolde();
double soldeB_apres = compteBReluApresVirement.getSolde();
logger.debug("apres bon virement, soldeA_apres="+soldeA_apres
            + " et soldeB_apres=" + soldeB_apres);

//verifier -50 et +50 sur les différences de soldes sur A et B :
Assertions.assertEquals(soldeA_avant - 50, soldeA_apres,0.000001);
Assertions.assertEquals(soldeB_avant + 50, soldeB_apres,0.000001);
}

//@Test
public void testMauvaisVirement() {
    /* VARIANTE A CODER/COLPLETER EN TP
    COPIER/COLLER à ADPATER de testVirement()
    AVEC
    try {
        this.serviceCompte.transférer(50.0, numCptA, -numCptB); //erreur volontaire
    } catch (Exception e) {
        logger.error("echec normal du virement " + e.getMessage());
    }
    et
    //verifier -0 et +0 sur les différences de soldes sur A et B
    Assertions.assertEquals(soldeA_avant , soldeA_apres,0.000001);
    Assertions.assertEquals(soldeB_avant , soldeB_apres,0.000001);
    */
}
}

```

Série de tests à effectuer :

1. enlever ~~@Transactional~~ au dessus de la méthode **transférer** et enlever les commentaires sur les lignes *this.daoCompte.save(cptDeb);* et *this.daoCompte.save(cptCred);*
2. lancer le test **testVirement()** et corriger les bugs si nécessaire
3. coder et lancer **testMauvaisVirement()** . c'est normal si ça ne fonctionne pas bien sans l'ajout de @Transactional
4. remplacer @Transactional au dessus de la méthode **transférer** et relancer le test **testMauvaisVirement()** qui devrait normalement fonctionner .
5. remplacer des commentaires sur les lignes *this.daoCompte.save(cptDeb);* et *this.daoCompte.save(cptCred);*
Tous les tests devraient encore bien fonctionner .

2.10. IHM Web basée sur Spring-MVC (jsp ou Thymeleaf)

Pour qu'il y ait un peu de données à afficher, on pourra éventuellement coder une classe servant à initialiser un jeu de données en phase de développement (lorsque le profile "initDataSet" sera activé) :

- créer le package ***tp.appliSpring.core.init***
- ajouter la classe ***InitDataSet*** avec le contenu suivant :

```
package tp.appliSpring.core.init;

import javax.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;
import tp.appliSpring.core.entity.Compte;
import tp.appliSpring.core.service.ServiceCompte;

@Profile("initDataSet")
@Component
public class InitDataSet {

    @Autowired
    private ServiceCompte serviceCompte;

    @PostConstruct
    public void initDefaultDataSet() {
        serviceCompte.sauvegarderCompte(new Compte(null, "compteA", 100.0));
        serviceCompte.sauvegarderCompte(new Compte(null, "compteB", 150.0));
    }
}
```

De manière à configurer le démarrage de l'application spring au sein d'un conteneur web (tel que tomcat), on crée 2 classes de configuration complémentaires :

- créer le package ***tp.appliSpring.web***
- ajouter la classe ***MyWebAppConfig*** avec le contenu suivant :

```
package tp.appliSpring.web;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import tp.appliSpring.core.config.CommonConfig;
import tp.appliSpring.core.config.DataSourceConfig;
import tp.appliSpring.core.config.DomainAndPersistenceConfig;

/*
Cette classe MyWebAppConfig
est utilisée par tp.appliSpring.web.MyWebApplicationInitializer
```

et sert à configurer (sans spring boot) le coeur de Spring-web / Spring-webmvc lorsque l'application .war sera déployée dans tomcat ou un équivalent
*/

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = { "tp.appliSpring.web" }) //to find @Controller , ... @RestController

@Import({CommonConfig.class, DataSourceConfig.class , DomainAndPersistenceConfig.class})

public class **MyWebAppConfig** {

// define a bean for ViewResolver

@Bean

public **ViewResolver** viewResolver() {

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/views/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

public MyWebAppConfig() {

System.out.println("MyWebAppConfig load ...");

}

}

- ajouter également la classe **MyWebApplicationInitializer** avec le contenu suivant :

package tp.appliSpring.web;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class **MyWebApplicationInitializer**

extends **AbstractAnnotationConfigDispatcherServletInitializer** {

MyWebApplicationInitializer(){

System.out.println("MyWebApplicationInitializer ...");

}

protected String[] **getServletMappings**() {

return new String[]{"**/mvc/**"}; //URL en :8080/.../**/mvc/**...

}

@Override

protected Class<?>[] **getRootConfigClasses**() {

return new Class<?>[] {**MyWebAppConfig.class**};

}

@Override

protected Class<?>[] **getServletConfigClasses**() {

return new Class[0];

}

```

@Override
public void onStartup(ServletContext context) throws ServletException {
    super.onStartup(context);

    //String activeProfile = "";
    String activeProfile = "initDataSet";

    context.setInitParameter("spring.profiles.active", activeProfile);
}
}

```

- créer le package **tp.appliSpring.web.ctrl**
- ajouter la classe **BasicController** avec le contenu suivant :

```

package tp.appliSpring.web.ctrl;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/basic")
public class BasicController {

    public BasicController() {
        System.out.println("BasicController load ...");
    }

    //http://localhost:8080/.../mvc/basic/helloworld
    @RequestMapping("/helloWorld")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World!");
        System.out.println("helloWorld returning showMessage ...");
        return "showMessage"; //jsp in /WEB-INF/views/
    }
}

```

- créer les nouveaux sous répertoires (dossiers) **WEB-INF/views** dans **src/main/webapp**
- ajouter dans **src/main/webapp/WEB-INF/views** la page JSP **showMessage.jsp** suivante :

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head><title>showMessage</title></head>
<body>
<p>message=<b>${message}</b></p>
</body>
</html>

```

- ajouter dans **src/main/webapp** la page **index.html** suivante :

```

<html>
<head><title>index</title></head>
<body>
<h1>appSpring</h1>

```

```
<a href="/mvc/basic/helloWorld">helloWorld - SpringMVC avec page JSP</a> <br/>
</body>
</html>
```

- Construire l'application web (appSpring.war) avec **maven** (goal = **package** ou **clean package** ou **install** , avec ou sans skipTests) via par exemple le menu "Run as / maven build ..." de eclipse .
- Installer si besoin **tomcat9** sur le poste (en téléchargeant et extrayant le contenu d'un .zip) → [c:/serveurs/apache-tomcat-9.0.36](#) ou [c:/prog/apache-tomcat-9.0.54](#) ou autres
- **Recopier appSpring.war** vers **apache-tomcat-9.../webapps**
- **Démarrer tomcat** via **/bin/startup** et tester l'application via ces urls : <http://localhost:8080/appSpring> et <http://localhost:8080/appSpring/mvc/basic/helloWorld>
- **NB:** on pourra préférer démarrer tomcat au sein de l'IDE eclipse (ou autre) via par exemple les menus **Window/preferences ... / Server/ Runtime environment /...** et **run as ... / run on server**

NB: Tout ceci n'était qu'un début de TP (configurations nécessaires).

....

3. Tp avec SpringBoot

3.1. Création d'un nouveau projet (Spring initializr)

3.2. Transposition "SpringBoot" de la plupart des Tps précédents

On transposera presque tout sauf la partie IHM/web basée sur Spring-mvc/JSP_ou_thymeleaf

3.3. Simplification des DAO via spring-data-jpa

3.4. API REST via @RestController et Spring-MVC

3.5. Sécurisation via Spring-Security