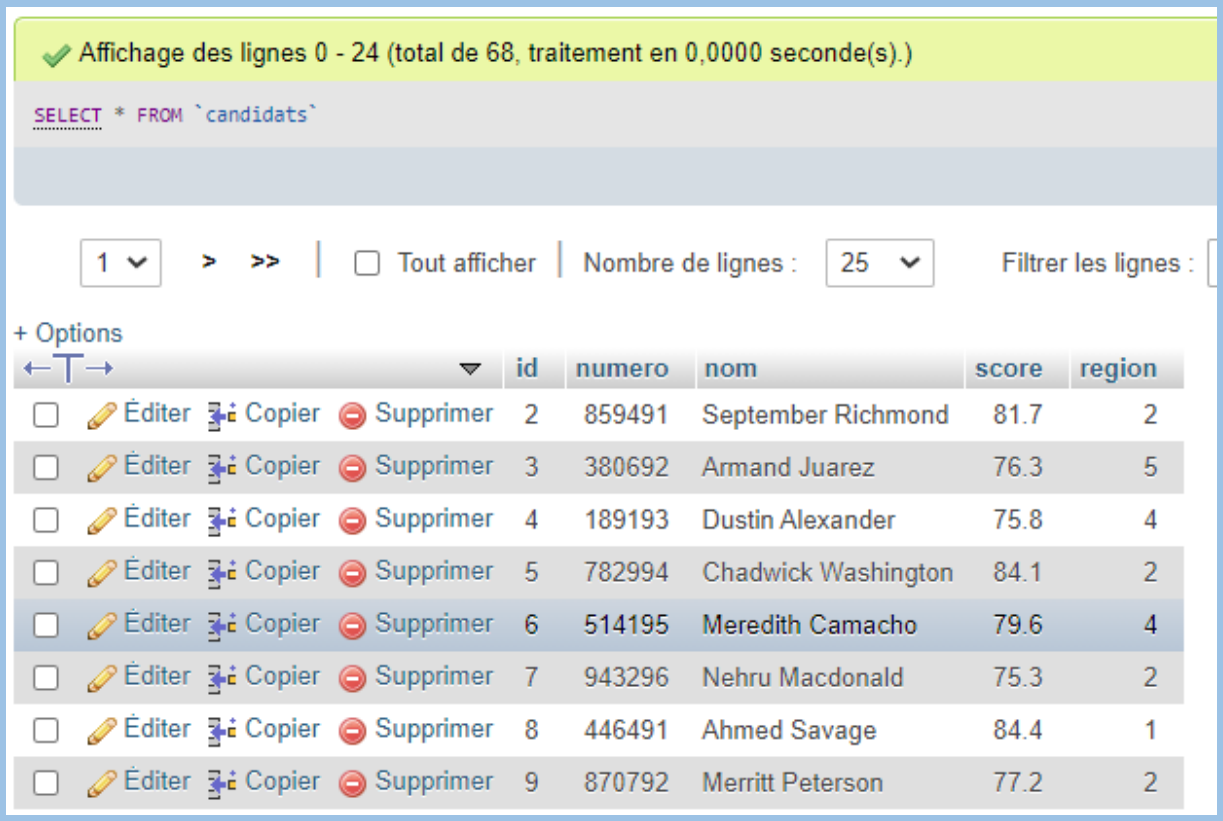
Etapes du TD5

1 Créer Alimenter la Base de Données (Wamp Server)

Exécuter le script db_concours.sql pour créer/alimenter la Base

Vérifier les enregistrements de la table (80 enregistrements)



Affichage des lignes 0 - 24 (total de 68, traitement en 0,0000 seconde(s).)

```
SELECT * FROM `candidats`
```

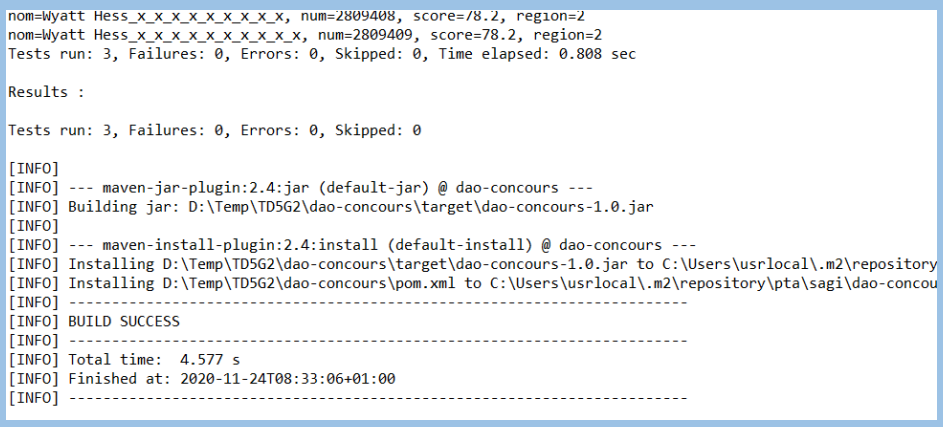
1 > >> | Tout afficher | Nombre de lignes : 25 | Filtrer les lignes :

+ Options

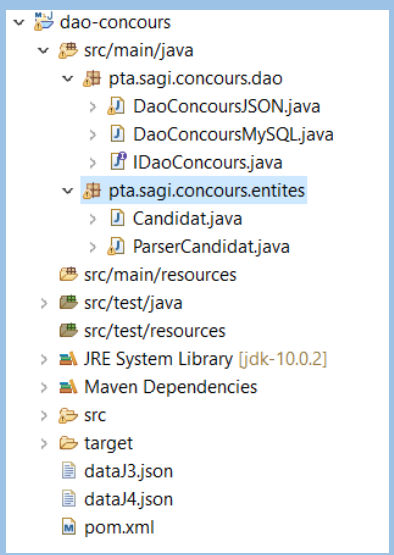
	id	numero	nom	score	region
Éditer Copier Supprimer	2	859491	September Richmond	81.7	2
Éditer Copier Supprimer	3	380692	Armand Juarez	76.3	5
Éditer Copier Supprimer	4	189193	Dustin Alexander	75.8	4
Éditer Copier Supprimer	5	782994	Chadwick Washington	84.1	2
Éditer Copier Supprimer	6	514195	Meredith Camacho	79.6	4
Éditer Copier Supprimer	7	943296	Nehru Macdonald	75.3	2
Éditer Copier Supprimer	8	446491	Ahmed Savage	84.4	1
Éditer Copier Supprimer	9	870792	Merritt Peterson	77.2	2

2 Ouvrir le projet Maven de la couche DAO de l'application Concours (projet fourni)

Installer le projet pta.sagi/dao-concours dans le dépôt Maven (Run As/Maven Install)



```
nom=Wyatt Hess_x_x_x_x_x_x_x_x_x, num=2809408, score=78.2, region=2
nom=Wyatt Hess_x_x_x_x_x_x_x_x_x_x, num=2809409, score=78.2, region=2
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.808 sec

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ dao-concours ---
[INFO] Building jar: D:\Temp\TD5G2\dao-concours\target\dao-concours-1.0.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ dao-concours ---
[INFO] Installing D:\Temp\TD5G2\dao-concours\target\dao-concours-1.0.jar to C:\Users\usrlocal\.m2\repository
[INFO] Installing D:\Temp\TD5G2\dao-concours\pom.xml to C:\Users\usrlocal\.m2\repository\pta\sagi\dao-concou
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.577 s
[INFO] Finished at: 2020-11-24T08:33:06+01:00
[INFO] ------------------------------------------------------------------------
```



3 Créer le projet Maven pour le reste de l'application (couche Metier + UI)

Projet Maven : pta.sagi / spring-concours / version 0.1

POM.xml (à la création)

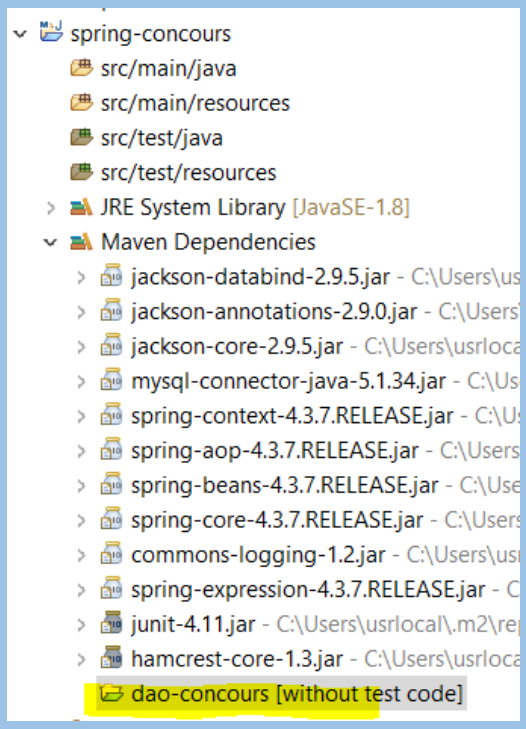
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <modelVersion>4.0.0</modelVersion>
  <groupId>pta.sagi</groupId>
  <artifactId>spring-concours</artifactId>
  <version>0.1</version>
  <name>spring-concours</name>
  <description>Projet des couches Metier et UI de l'application Concours</description>
</project>
```

4 Ajouter les dépendances : couche DAO + JUnit + Spring Framework

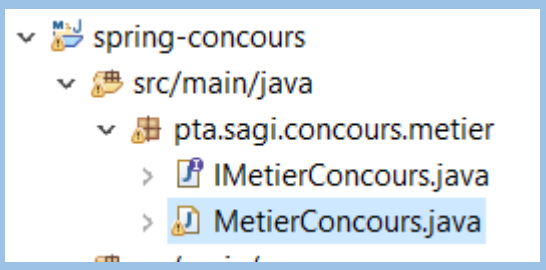
POM.xml (modifié)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSch
  <modelVersion>4.0.0</modelVersion>
  <groupId>pta.sagi</groupId>
  <artifactId>spring-concours</artifactId>
  <version>0.1</version>
  <name>spring-concours</name>
  <description>Projet des couches Metier et UI de l'application Concours</description>

  <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

  <dependencies>
    <dependency>
        <groupId>pta.sagi</groupId>
        <artifactId>dao-concours</artifactId>
        <version>1.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

5 Vérifier que les dépendances sont importées par Maven



6 Ajouter l'interface de la couche Metier + implémentation de la couche Metier




```java
package pta.sagi.concours.metier;
import pta.sagi.concours.entites.Candidat;

public interface IMetierConcours
{
    Candidat[] getCandidats(int ... regions);
    Candidat[] getTop(int region);
    void add(Candidat c);
    void setTopSize(int ts);
    int getTopSize();
}
```

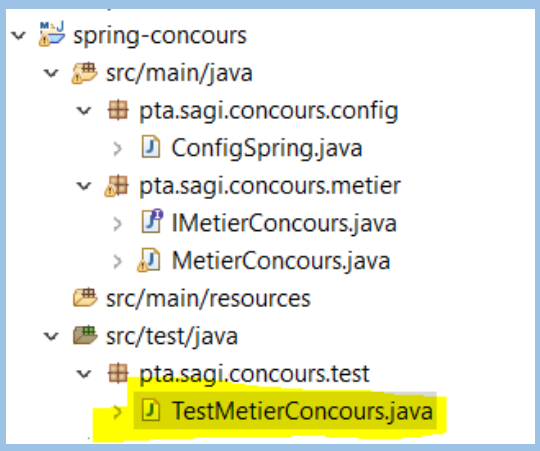
```java
package pta.sagi.concours.metier;
import org.springframework.beans.factory.annotation.Autowired;
import pta.sagi.concours.dao.IDaoConcours;
import pta.sagi.concours.entites.Candidat;

public class MetierConcours implements IMetierConcours
{
    @Autowired
    private IDaoConcours dao;

    private int topsize;

    public MetierConcours(){this.setTopSize(5);}

    @Override
    public Candidat[] getCandidats(int... regions) {
        // TODO Auto-generated method stub
        return null;
    }
```

7 Prévoir les tests unitaires (voir feuille TD)

Comme l'application est en couches, il faut reconstituer les couches Metier->DAO pour les tests. On crée donc un contexte Spring pour les TESTS également

Classe de Configuration Spring

```java
package pta.sagi.concours.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import pta.sagi.concours.dao.DaoConcoursMySQL;
import pta.sagi.concours.dao.IDaoConcours;
import pta.sagi.concours.metier.IMetierConcours;
import pta.sagi.concours.metier.MetierConcours;

@Configuration
public class ConfigSpring
{
    @Bean IDaoConcours dao() { return new DaoConcoursMySQL();}

    @Bean IMetierConcours metier() { return new MetierConcours();}
}
```

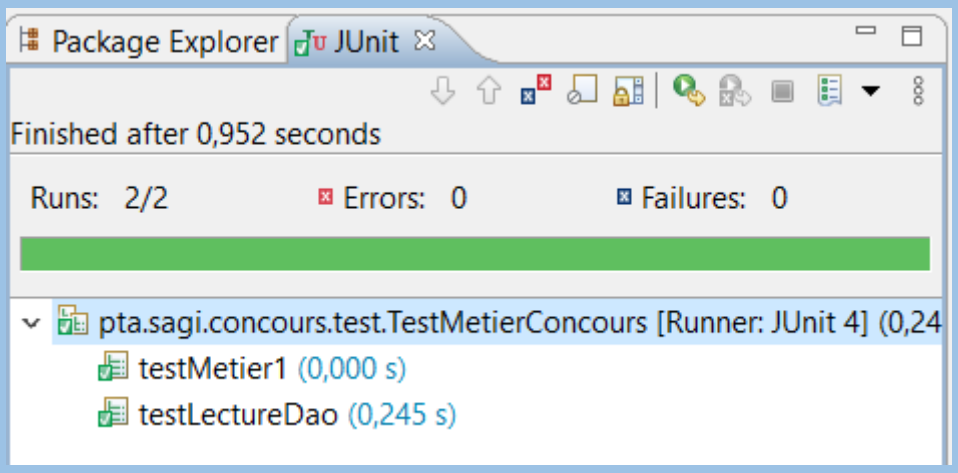



```java
import pta.sagi.concours.metier.IMetierConcours;

public class TestMetierConcours {

    private static AnnotationConfigApplicationContext ctx;

    @BeforeClass
    public static void setUp(){  // crée le contexte Spring
        ctx=null;
        try{ ctx=new AnnotationConfigApplicationContext(ConfigSpring.class);}
        catch(Exception ex){ fail("Problème Spring");}
    }

    @AfterClass
    public static void tearDown(){ if(ctx!=null) ctx.close(); }

    @Test
    public void testLectureDao()  // exemple d'utilisation du contexte Spring
    {
        IDaoConcours dao;
        dao=ctx.getBean(IDaoConcours.class); // obtient le bean Dao
        Candidat[] candidats=dao.getCandidatsIn(75,78);
        for(Candidat c:candidats)
        {
            System.out.println(c);
        }
    }
```

```java
@Test
public void testMetier1()
{
    IMetierConcours metier=ctx.getBean(IMetierConcours.class);
    System.out.println(metier);
    // TODO : écrire la suite du test Metier
}
```

8 Vérification de la bonne exécution des tests

Click droit / Run As / Junit Test



Sortie Console

```
nov. 24, 2020 9:26:02 AM org.springframework.context.annotation.AnnotationCo
INFOS: Refreshing org.springframework.context.annotation.AnnotationConfigApp
pta.sagi.concours.metier.MetierConcours@429bd883
nom=Armand Juarez, num=380692, score=76.3, region=5
nom=Dustin Alexander, num=189193, score=75.8, region=4
nom=Nehru Macdonald, num=943296, score=75.3, region=2
nom=Merritt Peterson, num=870792, score=77.2, region=2
nom=Ursula Bolton, num=367899, score=76.0, region=4
nom=Buffy Gilliam, num=296099, score=75.8, region=3
nom=Gloria Suarez, num=287399, score=76.4, region=4
nom=Rowan Hancock, num=292899, score=78.0, region=5
```

9 Développer les méthodes métier et tester

```
@Test
public void testMetier1()
{
    IMetierConcours metier=ctx.getBean(IMetierConcours.class);
    Candidat[] candidats = metier.getCandidats(1,3);
    // affiche le nombre de candidats région 1 + région 3
    System.out.println(String.format("testMetier1:%d candidats régions 1+3",c
    // Vérifie la région des candidats
    for(Candidat c:candidats){
        assertTrue(c.getRegion()==1 || c.getRegion()==3);
    }

    // Vérifie que les candidats sont triés par score décroissant
    for(int i=0;i<candidats.length-1;i++){
        if(candidats[i].getScore()<candidats[i+1].getScore()){
            fail("Candidats non triés");
        }
    }

    int t1=metier.getCandidats(1,2,3,4).length;
    int t2=metier.getCandidats(1,3).length;
    int t3=metier.getCandidats(2,4).length;
    assertEquals(t1,t2+t3);
}
```

```java
@Test
public void testMetier2()
{
    IMetierConcours metier=ctx.getBean(IMetierConcours.class);
    metier.setTopSize(3);
    Candidat[] topR3 = metier.getTop(3);
    Candidat[] regionR3 = metier.getCandidats(3);

    for(int i=0;i<topR3.length;i++)
    {
        assertEquals(topR3[i].getNumero(),regionR3[i].getNumero());
    }
}
```

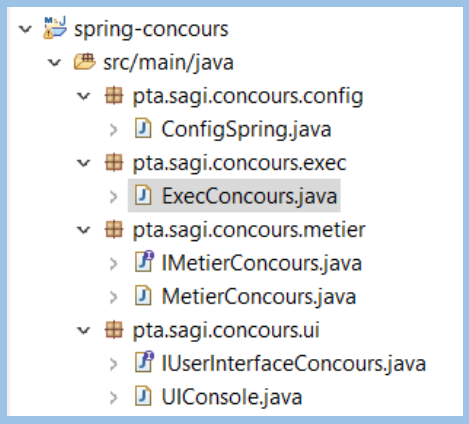
```java
@Test
public void testMetier3()
{
    IMetierConcours metier=ctx.getBean(IMetierConcours.class);
    metier.setTopSize(7);
    Candidat[] topR3 = metier.getTop(3);
    if(topR3.length==7) // il pourrait ne pas y avoir 7 candidats
    {
        System.out.println("Il y a 7 candidats au moins dans le top");
        metier.setTopSize(3);
        Candidat[] topR3b = metier.getTop(3);
        assertEquals(3,topR3b.length);
        assertEquals(topR3b[0].getNumero(),topR3[0].getNumero());
        assertEquals(topR3b[1].getNumero(),topR3[1].getNumero());
        assertEquals(topR3b[2].getNumero(),topR3[2].getNumero());
    }
}
```

```java
@Test
public void testMetier4()
{   // contrainte sur la taille du Top  [3,10]
    IMetierConcours metier=ctx.getBean(IMetierConcours.class);
    metier.setTopSize(7);
    assertEquals(7,metier.getTopSize());
    metier.setTopSize(1);
    assertEquals(3,metier.getTopSize());
    metier.setTopSize(12);
    assertEquals(10,metier.getTopSize());
}
```

10 Couche UI (console)

Lorsque la couche Metier est développée/testée, on peut passer à la couche interface utilisateur.

Pour la version console, on peut voir la couche UI comme une couche qui n'a qu'une seule méthode (run()). Cette méthode contient tous les échanges « console » avec l'utilisateur, depuis le démarrage jusqu'à la fin de l'application.




```java
package pta.sagi.concours.ui;

public interface IUserInterfaceConcours {
    void run();
}
```

```java
public class UIConsole implements IUserInterfaceConcours
{
    @Autowired
    private IMetierConcours metier;

    @Override
    public void run()
    {   //cf Exemple Texte TD4
        System.out.println("Application Concours : UI Console");
        System.out.print("Objet Metier:");
        System.out.println(metier);
        int choix;
        String prompt="0:Q 1:ByRegion 2:TopByRegion 3:Add";
        String line;
        BufferedReader clavier =new BufferedReader(new InputStreamReader(System.in));
        do{
            System.out.println(prompt);
            try{ line=clavier.readLine().trim();
                 choix= Integer.parseInt(line);}
            catch(Exception e){ choix=0; }
            switch(choix){
                case 1:
                    //TODO
                    break;
                case 2:
                    //TODO
                    break;
                case 3:
                    //TODO
                    break;
                default:
```

Compléter la classe de configuration Spring (nouveau Bean UI)

```java
@Configuration
public class ConfigSpring
{
    @Bean IDaoConcours dao() { return new DaoConcoursMySQL();}

    @Bean IMetierConcours metier() { return new MetierConcours();}

    @Bean IUserInterfaceConcours ui() { return new UIConsole();}
}
```

Classe exécutable : crée le contexte Spring (et donc les beans des couches) + lance la méthode run de la couche UI.

```java
package pta.sagi.concours.exec;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import pta.sagi.concours.config.ConfigSpring;
import pta.sagi.concours.ui.IUserInterfaceConcours;

public class ExecConcours {

    public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx;
        ctx=new AnnotationConfigApplicationContext(ConfigSpring.class);
        ctx.getBean(IUserInterfaceConcours.class).run();
        ctx.close();
    }
}
```

Lancer l'exécution de ExecConcours (Run As/Java Application)

Sortie console

```
nov. 24, 2020 10:38:44 AM org.springframework.context.annotation.AnnotationConfig
INFOS: Refreshing org.springframework.context.annotation.AnnotationConfigApplicat
Application Concours : UI Console
Objet Metier:pta.sagi.concours.metier.MetierConcours@aecb35a
0:Q 1:ByRegion 2:TopByRegion 3:Add
```

A ce stade : l'application est structurée en 3 couches UI / Metier / DAO. On peut exécuter l'application mais il reste à développer/compléter les interactions utilisateur (les différents cas du switch) dans la classe UIConsole.

Extrait de la classe UIConsole : IO console pour le menu 2 (obtenir le Top d'une région)

```
case 2:      // Menu 2: le Top par Région
    System.out.println("Région=?");
    try{
        line=clavier.readLine().trim();
        region= Integer.parseInt(line);  // parser la chaîne en int
        Candidat[] candidats = metier.getTop(region);
        for(Candidat c:candidats)   System.out.println(c);
    }
    catch(Exception e){ System.out.println(e.getMessage()); }
    break;
case 3:
    //TODO
    break;
```

Exécution : si l'on ne fournit pas un format compatible avec un entier pour la région, on a une exception

```
nov. 24, 2020 11:03:43 AM org.springframework.context.annotation.AnnotationCo
INFOS: Refreshing org.springframework.context.annotation.AnnotationConfigAppl
Application Concours : UI Console
Objet Metier:pta.sagi.concours.metier.MetierConcours@aecb35a
0:Q 1:ByRegion 2:TopByRegion 3:Add
2
Région=?
2
nom=Beatrice Ewing, num=756699, score=90.9, region=2
nom=Eaton Nieves, num=776099, score=87.0, region=2
nom=Dakota Rhodes, num=505199, score=84.6, region=2
0:Q 1:ByRegion 2:TopByRegion 3:Add
2
Région=?
x
For input string: "x"
0:Q 1:ByRegion 2:TopByRegion 3:Add
```

Pour le Menu 3 (Ajouter un candidat), il y a un parser de candidat fourni dans le package pta.sagi.concours.entites.

Il dispose d'une méthode de classe (statique) pour exploiter une chaîne décrivant un candidat : syntaxe [Nom Prenom ;numéro(6 chiffres) ;score ;region(1chiffre)]

```
Candidat c=ParserCandidat.parse("[John Doe;987654;76.5;2]");
if(c!=null)
{
    System.out.println(c);
}
```

Extrait de la classe UIConsole : IO console pour le menu 3 (ajouter un candidat)

```
case 3:
    System.out.println("Candidat=?");
    try{
        line=clavier.readLine().trim();
        Candidat candidat = ParserCandidat.parse(line);
        if(candidat!=null)
        {
            metier.add(candidat);
        }
    }
    catch(Exception e){ System.out.println(e.getMessage()); }
    break;
```

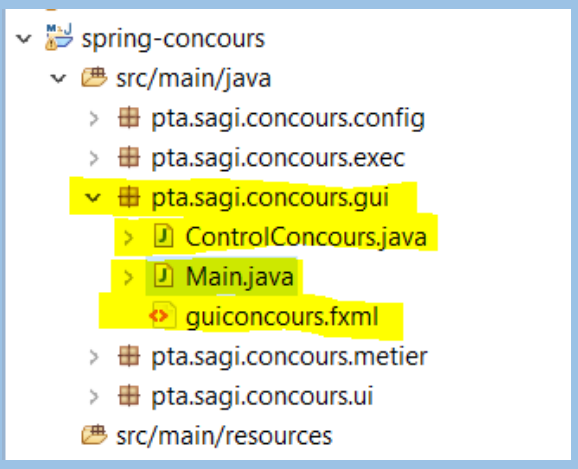
Exemple d'exécution : on demande le top de la région 4, on ajoute un candidat meilleur dans la région 4, et on redemande le top de la région 4

```
Région=?
4
nom=Joy Cole, num=607392, score=88.2, region=4
nom=Joseph Roth, num=923899, score=87.9, region=4
nom=Drew Weaver, num=514199, score=86.4, region=4
0:Q 1:ByRegion 2:TopByRegion 3:Add
3
Candidat=?
[John Moodle;973164;88.5;4]
0:Q 1:ByRegion 2:TopByRegion 3:Add
2
Région=?
4
nom=John Moodle, num=973164, score=88.5, region=4
nom=Joy Cole, num=607392, score=88.2, region=4
nom=Joseph Roth, num=923899, score=87.9, region=4
0:Q 1:ByRegion 2:TopByRegion 3:Add
```

11 GUI JavaFX exploitant les couches Metier / DAO

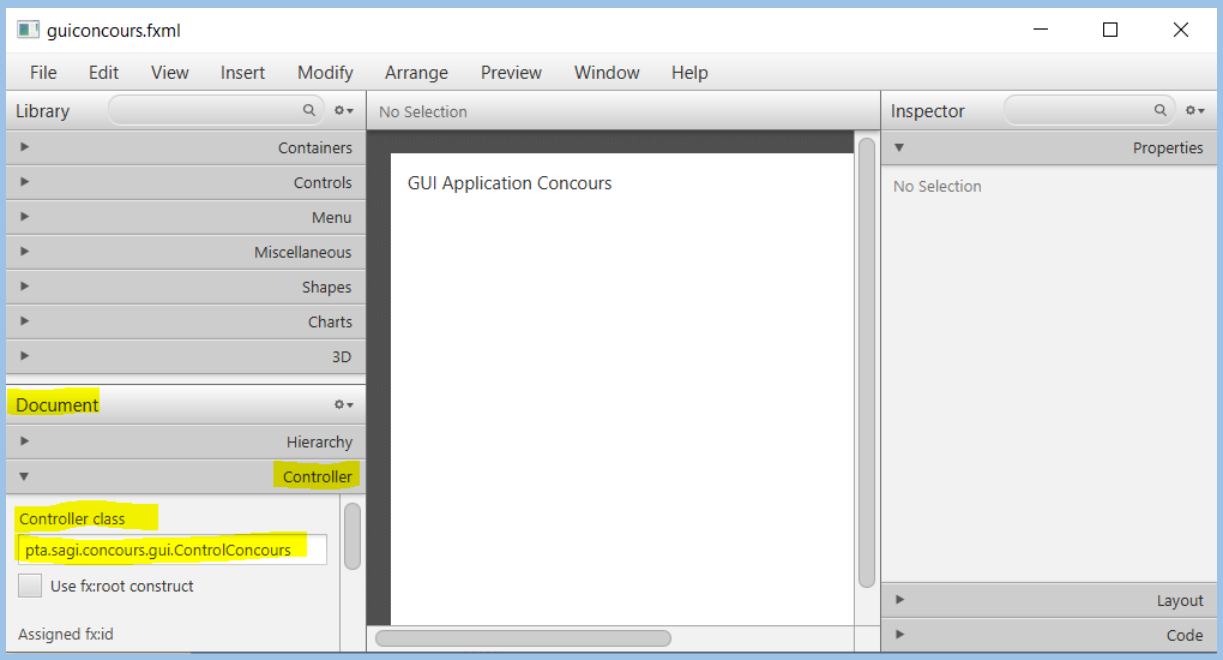
En s'appuyant sur la dernière partie du TD4 (Application Spring avec GUI JavaFX)

- Créer la classe exécutable Main (dérivée de application.Application)
- Créer la classe du contrôleur
- Editer un fichier fxml avec JavaFX SceneBuilder



Editer un fichier guiconcours.fxml minimal

- Un container Pane
- Un contrôle Label
- Définir la classe du contrôleur (cf ci-dessous)



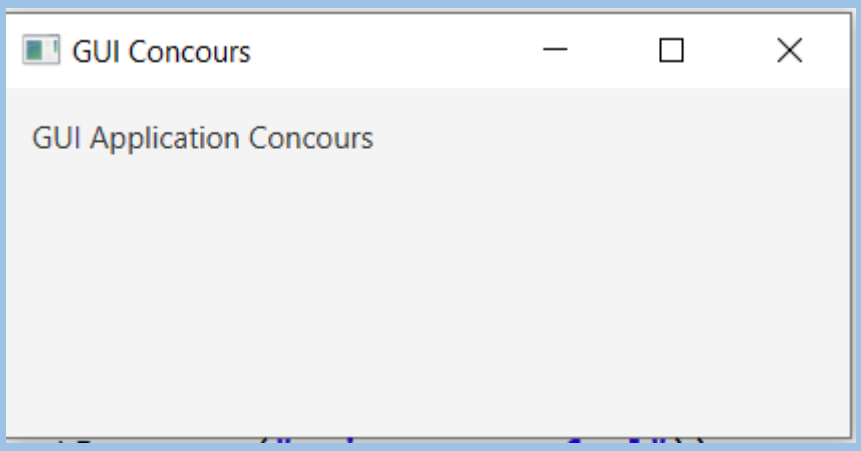
Editer le fichier de la classe Main (classe exécutable pour GUI JavaFX)

On définit un contexte Spring à l'aide de la classe de configuration.

Note : le Bean IUserInterfaceConcours ne sera pas utilisé, peu importe

```
//Adapté de TD4 : ExempleJavaFXSpring.pdf
public class Main extends Application
{
    private AnnotationConfigApplicationContext context;
    @Override
    public void stop(){ context.close();}
    @Override
    public void start(Stage primaryStage) {
    try {
    context = new AnnotationConfigApplicationContext(ConfigSpring.class);
    FXMLLoader loader = new FXMLLoader(getClass().getResource("guiconcours.fxml"));
    Pane root = (Pane)loader.load();
    Scene scene = new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("GUI Concours");
    ControlConcours ctrl = loader.getController();
    context.getAutowireCapableBeanFactory().autowireBean(ctrl);
    primaryStage.show();
    } catch(Exception e) { e.printStackTrace(); }
    }
    public static void main(String[] args){ Launch(args); }
}
```

Vérifier la bonne exécution de cette classe exécutable (doit créer une fenêtre)



En cas de souci :

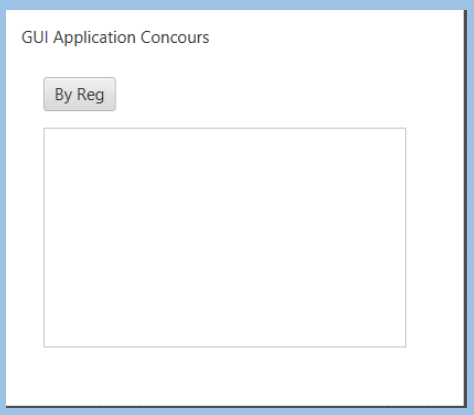
- Vérifier que la classe contrôleur renseignée dans le fichier fxml
- Sauvegarder le fichier FXML
- Dans IDE Eclipse, faire F5 (Refresh)
- Puis essayer l'exécution à nouveau

12 Compléter le fichier FXML et la classe du contrôleur (progressivement)

Penser à définir les Code/fx :id = ? (on utilise ces noms dans le contrôleur)

Pour le ListView : Code/fx :id=listviewcandidats

Pour le Button : On Action = OnByRegion




```
public class ControlConcours
{
    @Autowired
    IMetierConcours metier;

    ObservableList<Candidat> items;
    @FXML
    ListView<Candidat> listviewcandidats;    // fx:id=listviewcandidats

    @FXML
    void initialize(){
        items=FXCollections.observableArrayList();
        this.listviewcandidats.setItems(items);
        // pour modifier le contenu du ListView, lire/écrire dans "items"
    }

    @FXML
    void OnByRegion(){
        System.out.println("click ByRegion");
        System.out.println(metier);
    }

}
```

En cliquant sur le button, vérifier que vous avez un message sur la console

```
nov. 24, 2020 11:58:50 AM org.springframework.context.annot
INFOS: Refreshing org.springframework.context.annotation.Ar
click ByRegion
pta.sagi.concours.metier.MetierConcours@3a11deea
nov. 24, 2020 11:58:56 AM org.springframework.context.annot
INFOS: Closing org.springframework.context.annotation.Annot
```

Un extrait pour lire le top de la région 2 et mettre les objets dans le listview

```java
public class ControlConcours
{
    @Autowired
    IMetierConcours metier;

    ObservableList<Candidat> items;
    @FXML
    ListView<Candidat> listviewcandidats;   // fx:id=listviewcandidats

    @FXML
    void initialize(){
        items=FXCollections.observableArrayList();
        this.listviewcandidats.setItems(items);
        // pour modifier le contenu du ListView, lire/écrire dans "items"
    }

    @FXML
    void OnByRegion(){
        Candidat[] candidats=metier.getTop(2);
        items.clear();
        for(Candidat c:candidats)
        {
            items.add(c);
        }
    }
```