TP4 : MongoDB en Java graphique

Objectifs
Accès à une base de données mongodb en java
Utilisation de pilotes JDBC
Utilisation de WindowBuilder
Section
R5_10: Département Informatique
IUT GON Campus 3
Auteur
E.Porcq
Date
20/10/2025 durée 8h00

1 Préparation

1.1 Préparation

- Lancer 2 consoles
- Sur la première lancer le démon mongod
- Sur la seconde, lancer mongosh
- Créer une base maBDDTP4
- Lancer Eclipse et pas un autre IDE !

2 MongoDB " à la SQL " en Java

2.1 Présentation

https://unityjdbc.com/mongojdbc/mongo_jdbc.php
https://unityjdbc.com/mongojdbc/mongo_jdbc_free.php
https://unityjdbc.com/mongojdbc/mongosqltranslate.php
https://stackoverflow.com/questions/25815632/how-to-join-two-mongodb-collections-with-the-unity-jdbc-driver

JDBC (Java Database Connectivity) est une interface de programmation créée par Sun Microsystems — depuis racheté par Oracle Corporation — pour les programmes utilisant la plateforme Java. Elle permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC. Normalement, il s'agit d'une base de données *relationnelle*, et des pilotes JDBC sont disponibles pour tous les systèmes connus de bases de données relationnelles.

- Type 1 : Pilotes agissant comme passerelle en permettant l'accès à une base de données grâce à une autre technologie (JDBC-ODBC via ODBC) ;
- Type 2 : Pilotes d'API natifs. C'est un mélange de pilotes natifs et de pilotes Java. Les appels JDBC sont convertis en appels natifs pour le serveur de bases de données (Oracle, Sybase, ou autres) généralement en C ou en C++ ;
- Type 3 : Pilotes convertissant les appels JDBC en un protocole indépendant de la base de données. Un serveur convertit ensuite ceux-ci dans le protocole requis (modèle à 3 couches) ;
- Type 4 : Pilotes convertissant les appels JDBC directement en un protocole réseau exploité par la base de données. Ces pilotes encapsulent directement l'interface cliente de la base de données et sont fournis par les éditeurs de base de données.

Le pilote JDBC pour MongoDB permet **les requêtes SQL sur MongoDB** pour tout logiciel prenant en charge JDBC. La prise en charge de SQL inclut des fonctions, des expressions, des agrégations et des jointures, notamment pour les collections contenant des objets et des tableaux imbriqués. Le pilote JDBC de type 4/5 offre les performances les plus élevées lors des interrogations de MongoDB.

NOUVEAU : Une version gratuite du pilote MongoDB JDBC a été publiée pour les applications clientes où une seule connexion est requise. La version gratuite possède toutes les fonctionnalités et peut être utilisée et distribuée pour toutes les applications non serveur, y compris les projets open source.

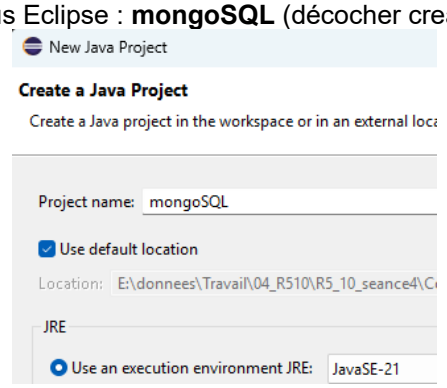
mongodb-jdbc-2.0.3-all.jar

2.2 Réalisation d'une application Java SQL pour MongoDB

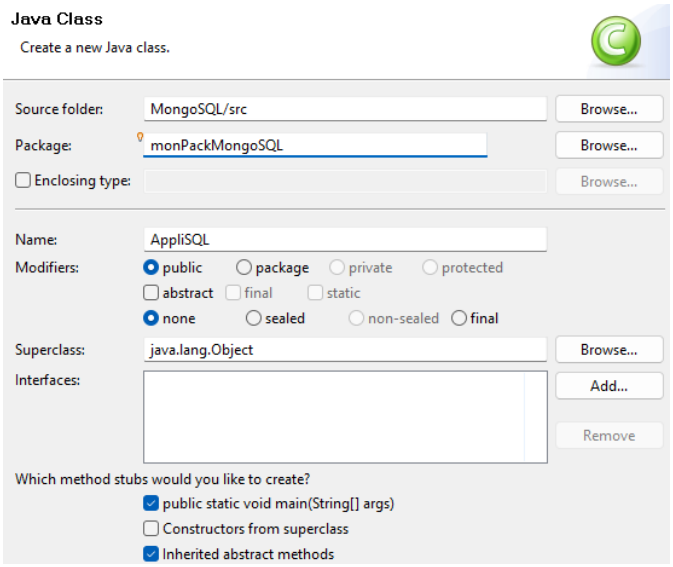
- Créer une collection : mesprofs
- Insérer les 6 documents ci-dessous

```
{'_id' : 1 ,'nom' : "Defromage"    , 'prenom' : "Stéfan"   , ecole : "8" },
{'_id' : 2 ,'nom' : "Scolaire"     , 'prenom' : "Emmanuel" , ecole : "7" },
{'_id' : 3 ,'nom' : "Jort"         , 'prenom' : "Dany"     , ecole : "8" },
{'_id' : 4 ,'nom' : "Tamboule"     , 'prenom' : "Athénaïs" , ecole : "8" },
{'_id' : 5 ,'nom' : "Tergeist"     , 'prenom' : "Paul"     , ecole : "8" },
{'_id' : 6 ,'nom' : "AngeEnPierre" , 'prenom' : "Laure"    , ecole : "9" }
```

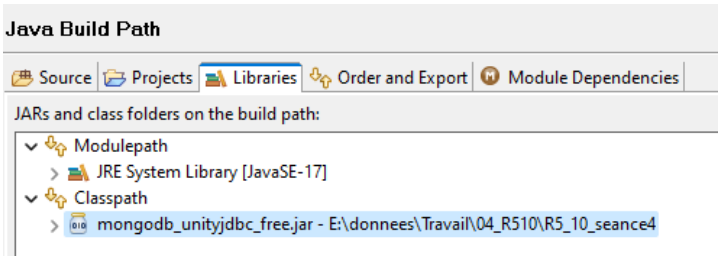
- Créer une application Java sous Eclipse : **mongoSQL** (décocher create module-info.java file)



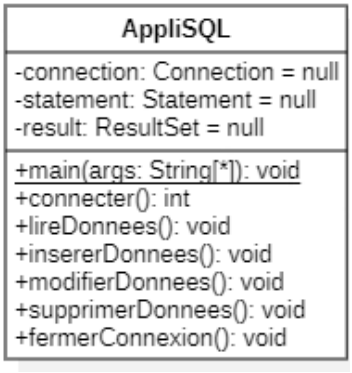
- Ajouter une classe avec un " main() "



- Ajouter le pilote mongodb_unityjdbc_free.jar



- En vous inspirant des exemples du site cité précédemment , réaliser une application permettant de réaliser les méthodes de la classe ci-dessous :



- Pour l'utilisation de **connecter**, se référer à l'excellent code réalisé en R2.06 (TP JDBC MySQL + Oracle)
  - Importer
    - `import java.sql.Connection;`
    - `import java.sql.DatabaseMetaData;`
    - `import java.sql.DriverManager;`
    - `import java.sql.ResultSet;`
    - `import java.sql.ResultSetMetaData;`
    - `import java.sql.SQLException;`
    - `import java.sql.Statement;`
  - Quelques méthodes à utiliser
    - `getMetaData`
    - `getDriverName`
    - `getDriverVersion`
    - `getDatabaseProductName`
    - `getDatabaseProductVersion`
  - Affichage si réussi :

```
Connection to the database has been established...
Nom Driver JDBC : Mongo JDBC
Version Driver JDBC : 4.2
Nom BDD : MongoDB
Version BDD : 6.0.8
```

- Pour l'utilisation de **lireDonnees**, se référer à l'excellent code réalisé en R2.06 (TP JDBC MySQL + Oracle)
  - Affichage si réussi :

```
_id, nom, prenom, ecole,
1 Defromage Stéfan 8
2 Scolaire Emmanuel 7
3 Jort Dany 8
4 Tamboule Athénaïs 8
6 AngeEnPierre Laure 9
```

- Pour l'utilisation de **insererDonnees**, se référer à l'excellent code réalisé en R2.06 (TP JDBC MySQL + Oracle). Appeler **insererDonnees** avant **lireDonnees**
  - Affichage si réussi :

```
_id, nom, prenom, ecole,
1 Defromage Stéfan 8
2 Scolaire Emmanuel 7
3 Jort Dany 8
4 Tamboule Athénaïs 8
6 AngeEnPierre Laure 9
7 Bombadil Tom null
```

- Pour l'utilisation de **modifierDonnees**, cela serait pas mal d'être un peu autonomé, non ? Appeler **modifierDonnees** juste avant **lireDonnees**
  - Affichage si réussi :

```
_id, nom, prenom, ecole,
1 Defromage Stéfan 8
2 Scolaire Emmanuel 7
3 Jort Etama 8
4 Tamboule Athénaïs 8
6 AngeEnPierre Laure 9
7 Bombadil Tom null
```

- Pour l'utilisation de **supprimerDonnees**, débrouillez-vous !
  - Affichage si réussi :

```
_id, nom, prenom, ecole,
1 Defromage Stéfan 8
2 Scolaire Emmanuel 7
3 Jort Etama 8
4 Tamboule Athénaïs 8
```

- Pour l'utilisation de **fermerConnexion**, débrouillez-vous !
- Créer une collection : mesecoles
- Insérer les 3 documents ci-dessous
  - `{'_id' : 1 , 'nom' : 'ENSI' , 'ecole' : '7' },`
  - `{'_id' : 2 , 'nom' : 'IUT' , 'ecole' : '8' },`
  - `{'_id' : 3 , 'nom' : 'ESIX' , 'ecole' : '6' }`
- Ajouter une fonction **lireDonneesEcoles.** Pourquoi cela ne fonctionne pas ?
- Corriger le problème
  - Affichage si réussi :

```
_id, nom, ecole,
1 ENSI 7
2 IUT 8
3 ESIX 6
```

- Ajouter une fonction **lireDonneesJointes.** Pourquoi cela fonctionne ?
  - Affichage si réussi :

```
_id, nom, prenom, ecole, _id, nom, ecole,
5 Tergeist Paul 8 2 IUT 8
4 Tamboule Athénaïs 8 2 IUT 8
3 Jort Etama 8 2 IUT 8
1 Defromage Stéfan 8 2 IUT 8
```

3 MongoDB NoSQL pour Java

3.1 Présentation

https://www.mongodb.com/docs/drivers/java/sync/v5.2/connection/connect/#std-label-connect-to-mongodb
https://www.baeldung.com/java-mongodb
https://www.mongodb.com/docs/atlas/data-federation/query/sql/drivers/jdbc/connect/
https://learn.mongodb.com/learning-paths/using-mongodb-with-java
https://www.mongodb.com/docs/drivers/java/sync/v5.2/fundamentals/builders/filters/
https://www.mongodb.com/docs/drivers/java/sync/v5.2/fundamentals/crud/read-operations/cursor/

3.2 Réalisation d'une application Java pour MongoDB

- Créer une collection : mes_jdbc_profs
- Insérer ces documents :
  - `{'_id' : 1 ,'nom' : "Deraclette" , 'prenom' : "Stéfan" , ecole : "8" },`
  - `{'_id' : 2 ,'nom' : "Scolaire" , 'prenom' : "Emmanuel" , ecole : "7" },`
  - `{'_id' : 3 ,'nom' : "Jort" , 'prenom' : "Dany" , ecole : "8" },`
  - `{'_id' : 4 ,'nom' : "Téria" , 'prenom' : "Alice" , ecole : "8" },`
  - `{'_id' : 5 ,'nom' : "Royce" , 'prenom' : "Carole" , ecole : "8" },`
  - `{'_id' : 6 ,'nom' : "Dinateur" , 'prenom' : "Laure" , ecole : "9" }`
- Créer une application Java sous Eclipse : mongoNoSQL
- Ajouter une classe avec un " main() " : AppliNoSQL
- Ajouter le pilote mongodb-jdbc-2.2.0-all.jar
- En vous inspirant des exemples des sites cités précédemment, réaliser une application permettant de réaliser les mêmes méthodes de la classe du 2-2)
- Faire en sorte que connecter afficher la base sur laquelle l'application est connectée.

```
maBDDTP4
```

- Faire en sorte que connecter afficher la liste des collections

```
mesecoles
mes_jdbc_profs
mesprofs
maBDDTP4.mes_jdbc_profs
```

- Faire en sorte que lireDonnees affiche ceci !

```
Document{{nom=Deraclette, prenom=Stéfan, ecole=8}}
Document{{nom=Scolaire, prenom=Emmanuel, ecole=7}}
Document{{nom=Jort, prenom=Dany, ecole=8}}
Document{{nom=Téria, prenom=Alice, ecole=8}}
Document{{nom=Dinateur, prenom=Laure, ecole=9}}
```

  - Aide : La fonction de recherche peut utiliser 3 documents. Exemple :

```
Document criteres = new Document();
criteres1.put("nom", "JALABERT");
ou
Bson criteres = Filters.ne("nom", "JALABERT" );
Document colonnes = new Document("_id", 0);
Document tri  = new Document("n_coureur",1);

FindIterable<Document> documents1 = collection.find(criteres).projection(colonnes).sort(tri);
```

- Faire en sorte que InsererDonnees permette à lireDonnees d'afficher ceci !

```
Document{{nom=Deraclette, prenom=Stéfan, ecole=8}}
Document{{nom=Scolaire, prenom=Emmanuel, ecole=7}}
Document{{nom=Jort, prenom=Dany, ecole=8}}
Document{{nom=Téria, prenom=Alice, ecole=8}}
Document{{nom=Dinateur, prenom=Laure, ecole=9}}
Document{{nom=Bombadil, prenom=Tom, ecole=7}}
```

- Faire en sorte que modifierDonnees permette à lireDonnees d'afficher ceci !

```
données modifiéesAcknowledgedUpdateResult{matchedCount=1, modifiedCount=1, upsertedId=null}
Document{{nom=Deraclette, prenom=Stéfan, ecole=8}}
Document{{nom=Scolaire, prenom=Emmanuel, ecole=7}}
Document{{nom=Jort, prenom=Etama, ecole=8}}
Document{{nom=Téria, prenom=Alice, ecole=8}}
Document{{nom=Dinateur, prenom=Laure, ecole=9}}
Document{{nom=Bombadil, prenom=Tom, ecole=7}}
```

- Faire en sorte que supprimerDonnees permette à lireDonnees d'afficher ceci !

```
Deleted document count: 2
Document{{nom=Deraclette, prenom=Stéfan, ecole=8}}
Document{{nom=Scolaire, prenom=Emmanuel, ecole=7}}
Document{{nom=Jort, prenom=Etama, ecole=8}}
Document{{nom=Téria, prenom=Alice, ecole=8}}
```

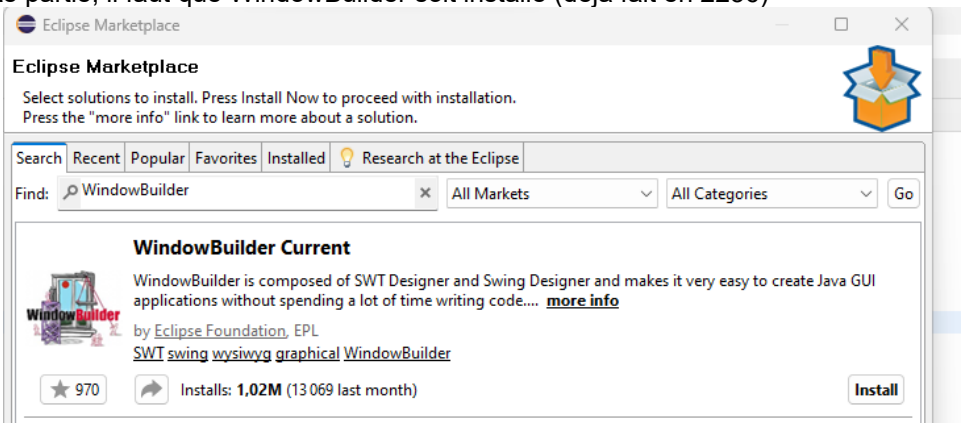
4 Réalisation d'une interface graphique avec WindowBuilder

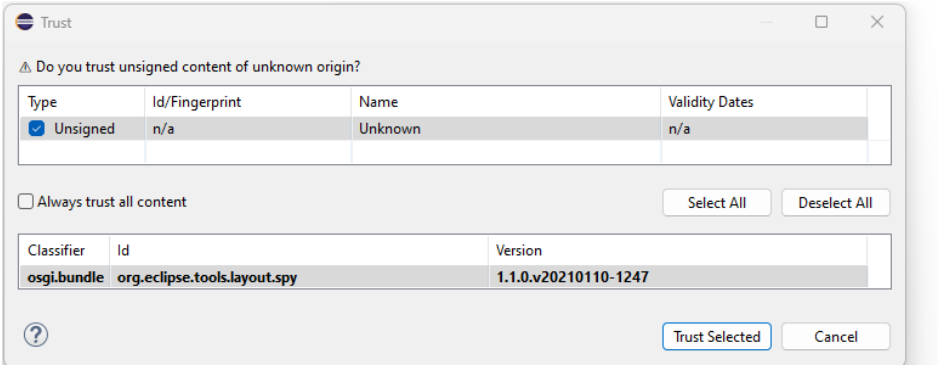
4.1 Installation

https://kb.objectrocket.com/mongo-db/how-to-connect-java-to-mongodb-using-eclipse-ide-284

Voir doc " Guide d'utilisation d'Eclipse pour créer des applications en Java" de M. Dalmau

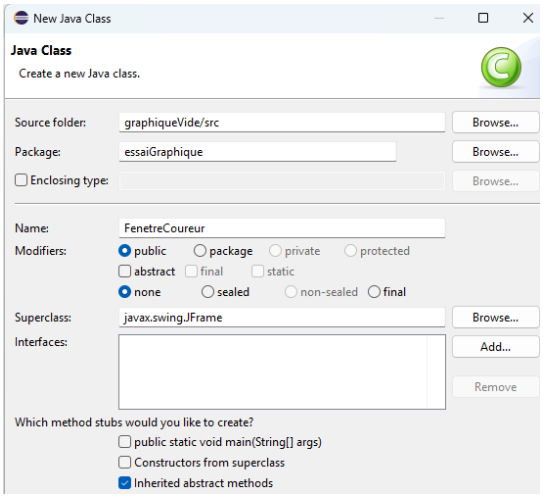
Pour réaliser cette partie, il faut que WindowBuilder soit installé (déjà fait en 2236)



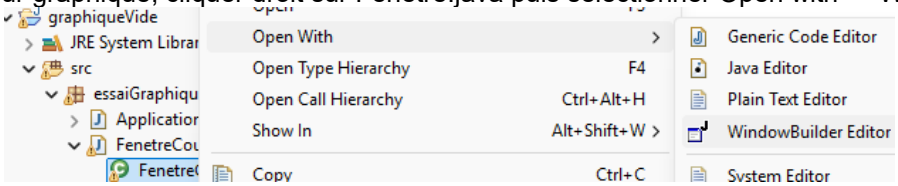


4.2 Préparation de l'interface

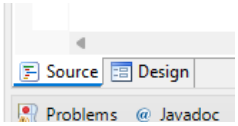
- Créer une application Java sous Eclipse
  - décocher module.java
- Ajouter une classe " Application " avec un main() : (package essaiGraphique)
- Ajouter une classe " FenetreCoureur "
  - cliquer sur Browse pour faire hériter la classe de
    - javax.swing.JFrame



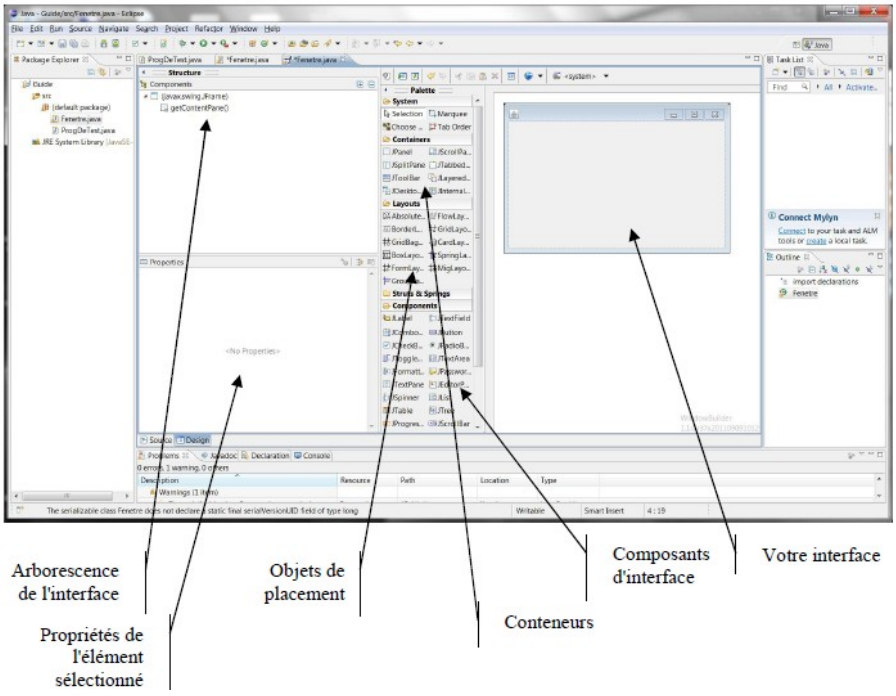
- Pour ouvrir l'éditeur graphique, cliquer droit sur Fenetre.java puis sélectionner Open with -->WindowBuilder Editor



- Cliquer sur Design



- Voici les éléments accessibles sur l'interface



- Nommer La boite : " Les coureurs"
- Modifier la propriété " defaultCloseOperation " en " EXIT ON CLOSE"

- Choisir la taille de la boite à 800,420

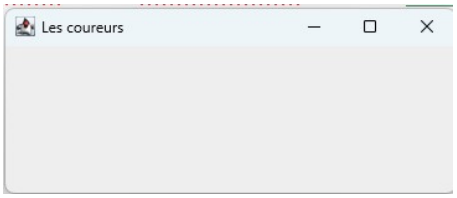
- Dans le main(), ajouter la ligne de code
  - `FenetreCoureur fen = new FenetreCoureur();`
- Dans le constructeur de FenetreCoureur (tout en bas), ajouter le code
  - pack();
  - setVisible(true);
- Ajouter une méthode permettant de fermer proprement l'application

```
private class fermerFenetre extends WindowAdapter
{
        @Override
        public void windowClosing(WindowEvent e) {
                System.exit(0);
        }
}
```

- Dans le constructeur de Fenêtre, ajoute le code suivant

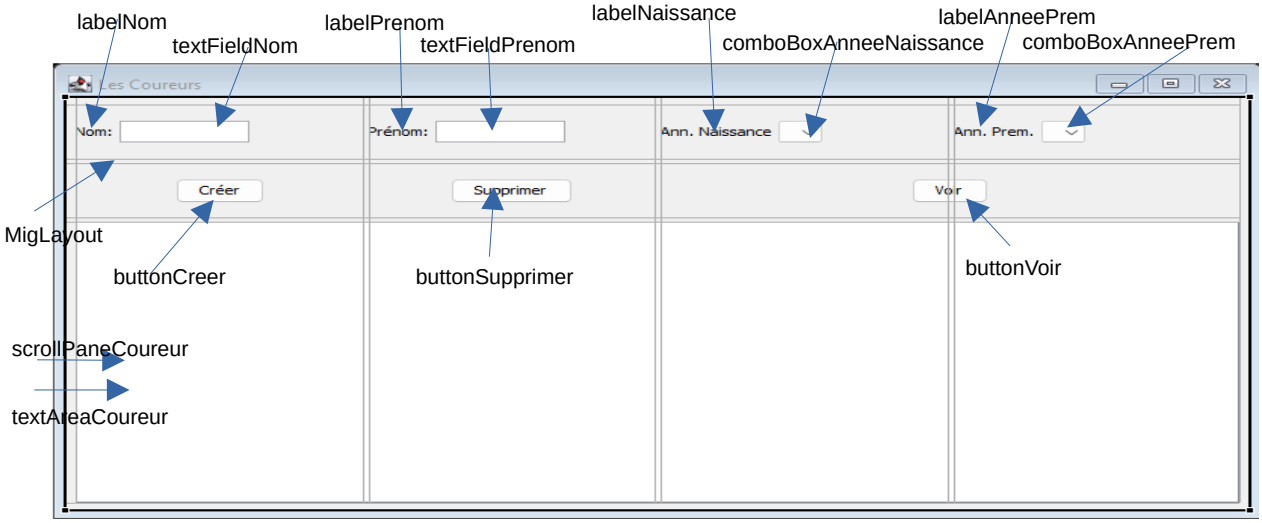
```
addWindowListener(new fermerFenetre()); // fermeture application
```

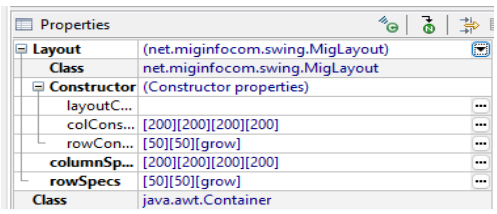
- Tester l'application

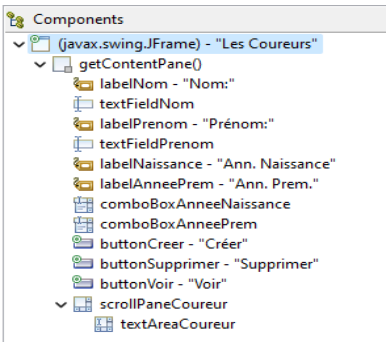


4.3 Interface complète sans traitement

- Compléter l'interface graphique avec ces composants
  - Le MigLayout sert à simplifier le placement des composants



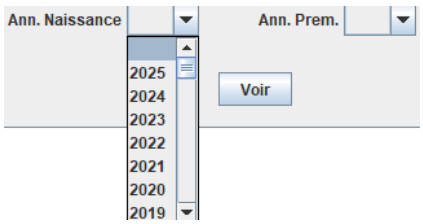




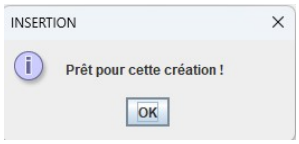
  - Le JscrolPane permettra au JtextArea de disposer d'un ascenceur
- Tester l'application

4.4 Interface complète avec traitement basique

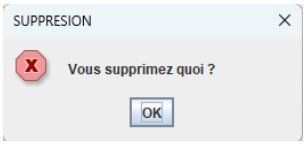
- Compléter le code de l'application pour que les comboBox proposent des années comprises entre 1950 et 2025



- Ajouter une méthode **quandCreer** (placer le code en dehors du constructeur) sur l'événement bouton " Créer ". Cette méthode affiche un message comme ci-dessous
  - Instancier **dynamiquement** un JOptionPane



- Ajouter une méthode **quandSupprimer** sur l'événement bouton "Supprimer". Cette méthode affiche un message comme ci-dessous



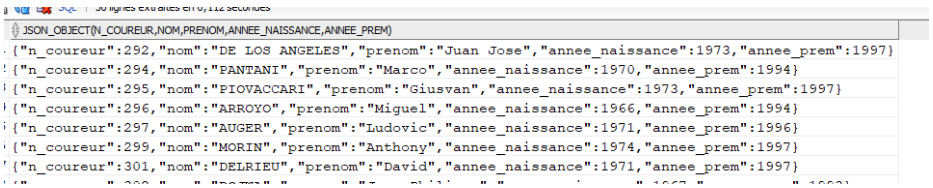
- Ajouter une méthode **quandVoir** sur l'événement bouton "Voir". Cette méthode affiche les informations saisies dans les TextField et les ComboBox après les avoir formatées
  - nom en majuscule
  - prénom en minuscule avec la première lettre en majuscule

5 Application complète

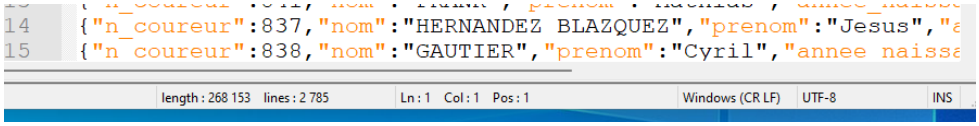
5.1 Création de la collection coureurs

- Lancer SQLDeveloper est se connecter sur son compte Oracle
- Exécuter cette requête

```
select json_object(
     n_coureur, nom,prenom,  annee_naissance, annee_prem
)
from prof.vt_coureur;
```



- Sauvegarder le résultat dans un fichier coureurs.json (sans le modifier et vérifier type UTF8)

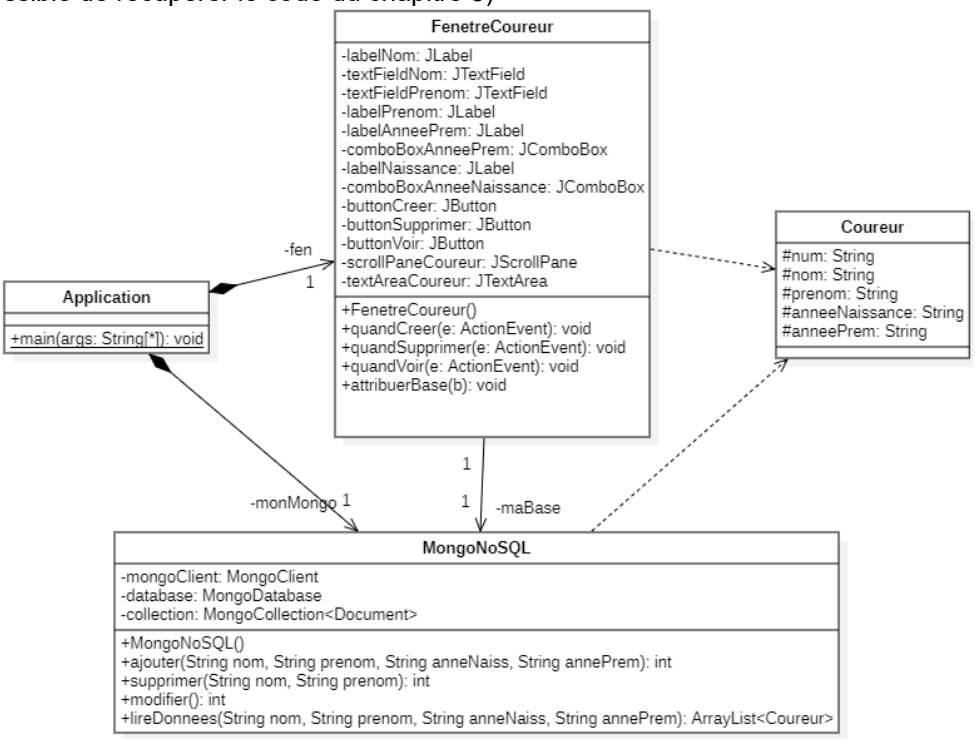


- Créer une collection mescoureurs et insérer les données coureurs
- Vérifier que les coureurs ont été correctement importés

```
{ n_coureur: 1,  nom: 'JOACHIM',    prenom: 'Benoit',    annee_naissance: 1974,    annee_prem: 2000  },
{ n_coureur: 2,  nom: 'KJAERGAARD', prenom: 'Steffen',   annee_naissance: 1973,    annee_prem: 2000  },
{ n_coureur: 3,  nom: 'LEON',       prenom: 'Francisco', annee_naissance: 1973,    annee_prem: 2000  }
…
```

5.2 Diagramme de classe

- Faire une copie de l'application précédente et FenetreCoureur
- Créer toutes les classes et méthodes manquantes (même vides)
  - il est possible de récupérer le code du chapitre 3)



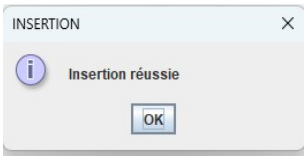
5.3 Réalisation de l'application

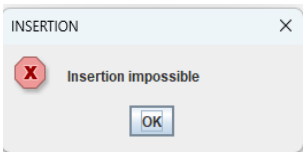
5.3.1 Classe Application

- La méthode **main**() :
  - étudier le diagramme de classe et créer les éléments de code nécessaires
  - Vérifier que l'application fonctionne (affichage de la fenêtre)

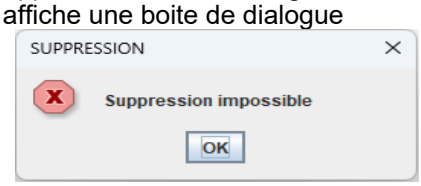
5.3.2 Classe FenetreCoureur

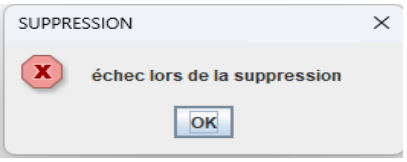
- La méthode **attribuerBase**()
  - Le problème est réglé, non ?
- La méthode **quandCreer**()
  - vérifie que les 4 zones de saisies ont été remplies. Sinon erreur !
  - formate nom en majuscule, prénom en première majuscule
  - appelle la méthode MongoNoSQL::ajouter (ne pas la développer pour le moment)
  - affiche une boite de dialogue



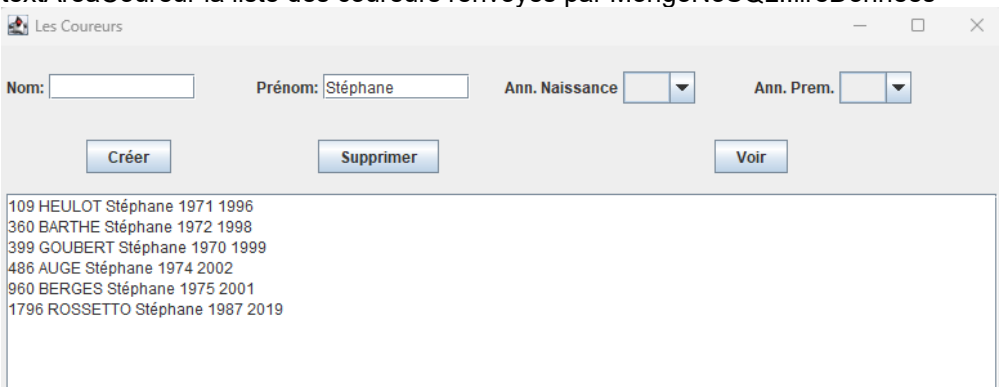


- La méthode **quandSupprimer**()
  - vérifie que nom et prénom ne sont pas vides (sinon suppression impossible)
  - appelle la méthode MongoNoSQL::**supprimer** (ne pas la développer pour le moment)
  - affiche une boite de dialogue





- La méthode **quandVoir**()
  - récupère les données des 4 zones de saisies
  - formate nom en majuscule, prénom en première majuscule
  - appelle la méthode MongoNoSQL::lire (ne pas la développer pour le moment)
  - affiche dans textAreaCoureur la liste des coureurs renvoyés par MongoNoSQL::lireDonnees

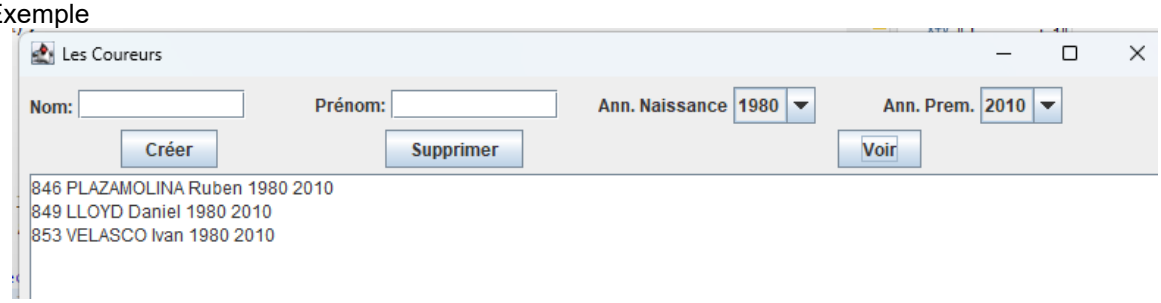


5.3.3 Classe Coureur

- Réaliser cette classe

5.3.4 Classe MongoNoSQL

- Le constructeur **MongoNoSQL**() :
  - initialise mongoClient, database et collection
- La méthode **lireDonnees**() :
  - lit le document correspondant aux critères non obligatoires (nom, prenom, anneNaiss, annePrem)
  - ne pas lire l' id.
  - trier pas n_coureur
  - boucle sur les documents pour compléter une liste de coureurs
  - retourne la liste de coureurs
  - compléter si nécessaire **quandVoir** pour obtenir un affichage satisfaisant.
  - Exemple



- La méthode **supprimer**() :
  - essaie de supprimer les documents portant ce couple nom,prénom
  - retourne -1 en cas d 'échec et 0 si réussite

- La méthode **ajouter**() :
  - extrait le plus grand numéro de coureur de la collection
    il est conseillé de réaliser l'équivalent de la requête ci-dessous (voir 3.2)
    db.mescoureurs.find({},{"_id":0,n_coureur:1}).sort({n_coureur : 1})