

Langage JAVA

Romain SESSA

Objectifs

- Présentation de JAVA
- Syntaxe du langage
- Programmation Objet
- Accès aux bases de données

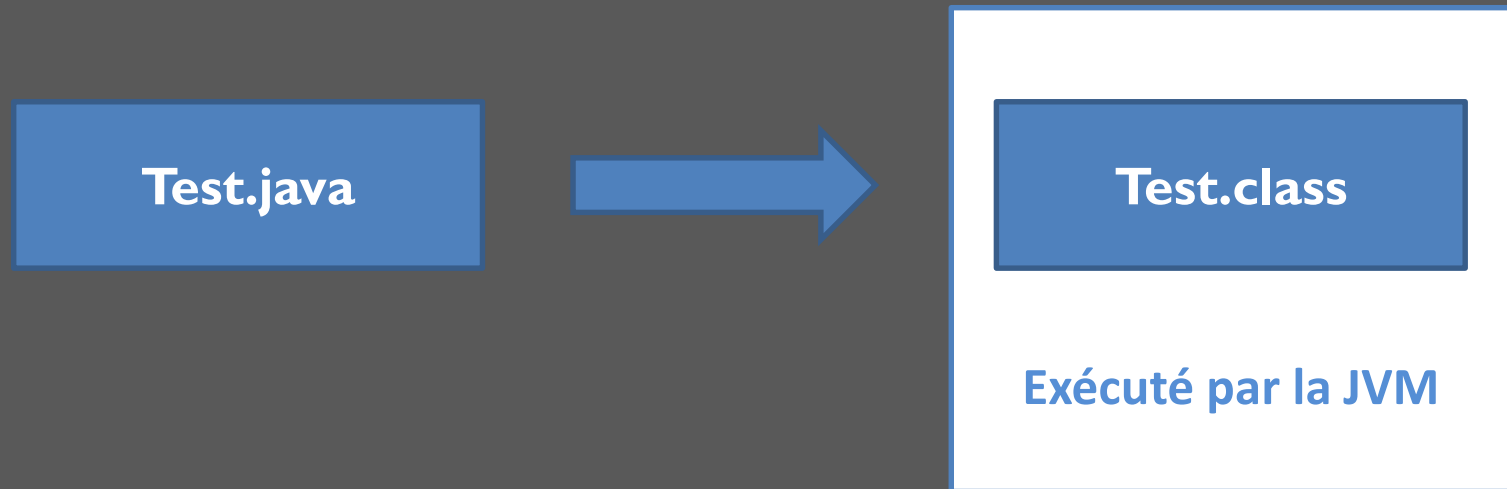
Présentation de JAVA

Caractéristiques du langage

- La JVM exécute le bytecode obtenu après compilation de la source.
- Langage Objet.
- Langage typé.
- Garbage collector : gestion automatique de la mémoire.
- Version : Java 8 depuis Mars 2014.

Compilation et exécution

- Créer un fichier .java valide.
- Compiler **javac Test.java** : on obtient Test.class
- Exécuter avec la commande : **java Test**



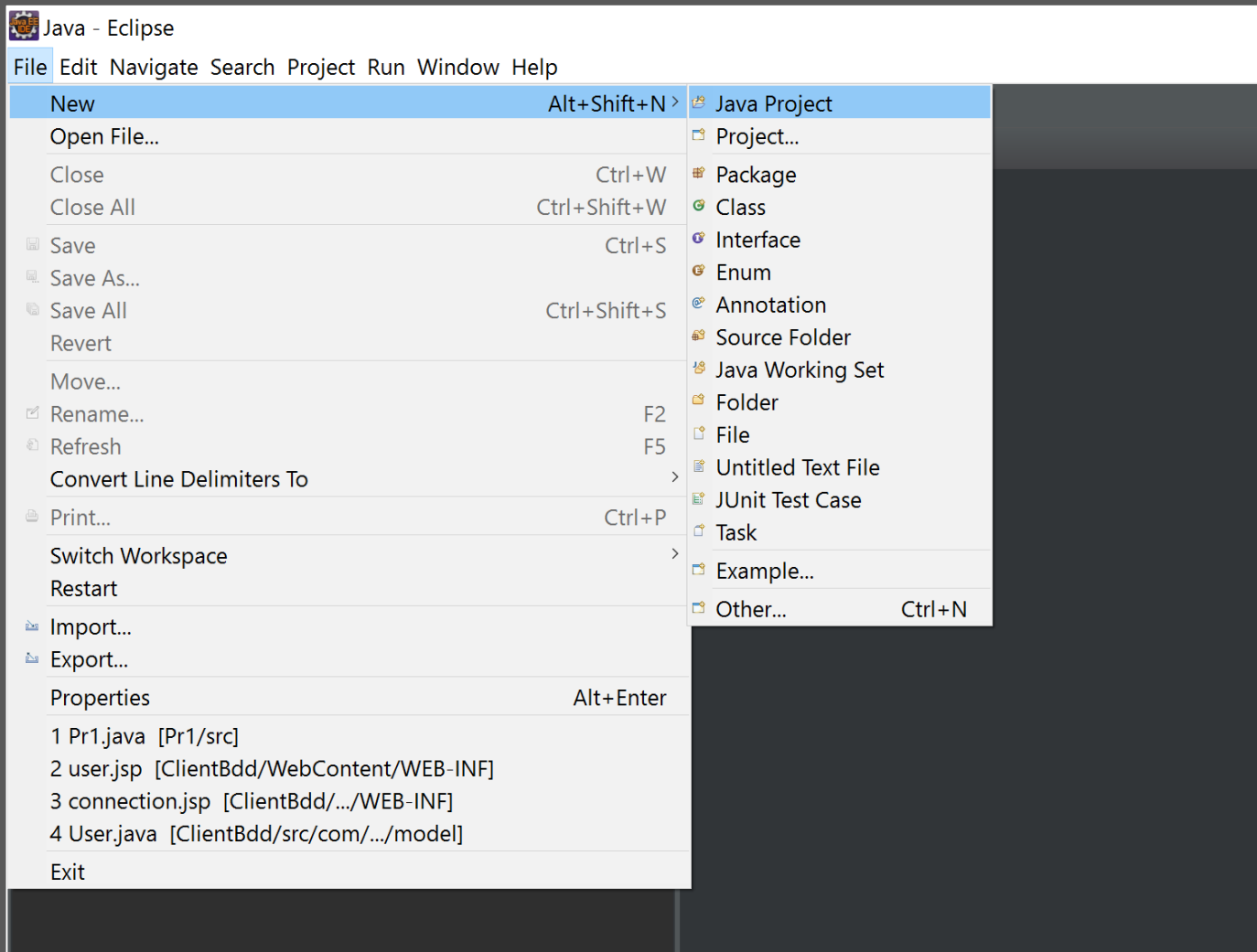
Compilation et exécution

- Classpath : Indique l'endroit où sont les classes.
- Il peut être défini à l'exécution :
`java -classpath D:/Workspace Test`
- Cela implique que le répertoire D:/Workspace contienne un fichier Test.class

Environnement de développement

- IDE : Eclipse, Netbeans, JCreator, etc.
- Télécharger : Eclipse Java EE IDE for Web Developers.
- Possibilité de créer :
 - JAVA Project.
 - Dynamic Web Projet

Créer un JAVA Project - I



Créer un JAVA Project - 2

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location: [Browse...](#)

JRE

☐ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0_73') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☐ Create separate folders for sources and class files [Configure default...](#)

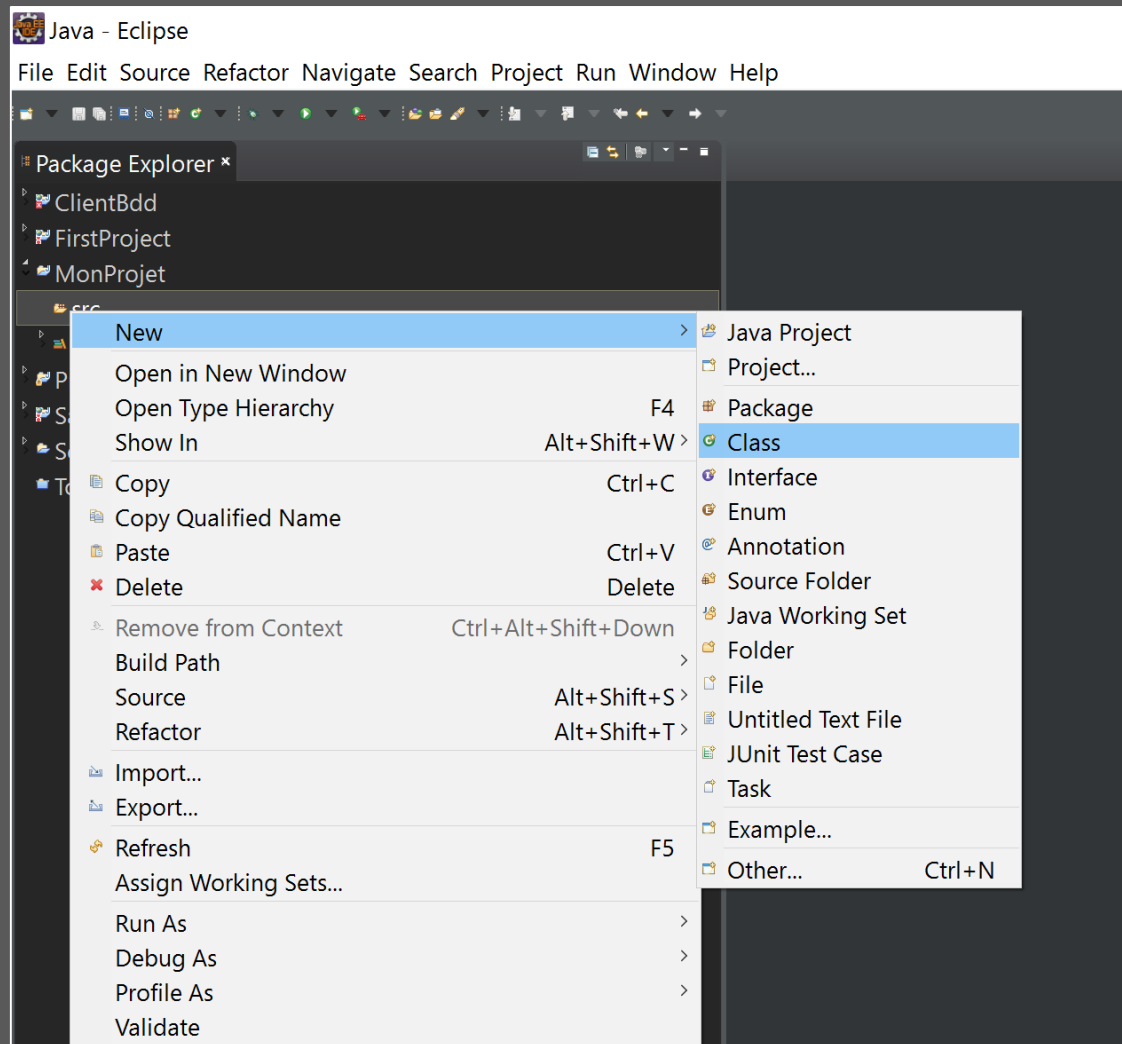
Working sets

☐ Add project to working sets


Working sets: [Select...](#)


[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)


Créer la première classe - I



Créer la première classe - 2

 New Java Class

Java Class 

 The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?


☒ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

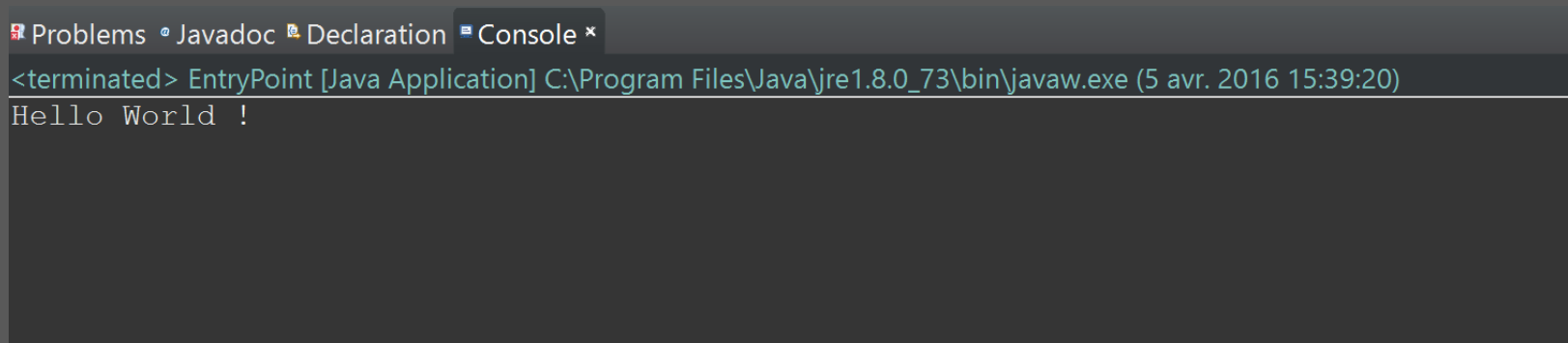
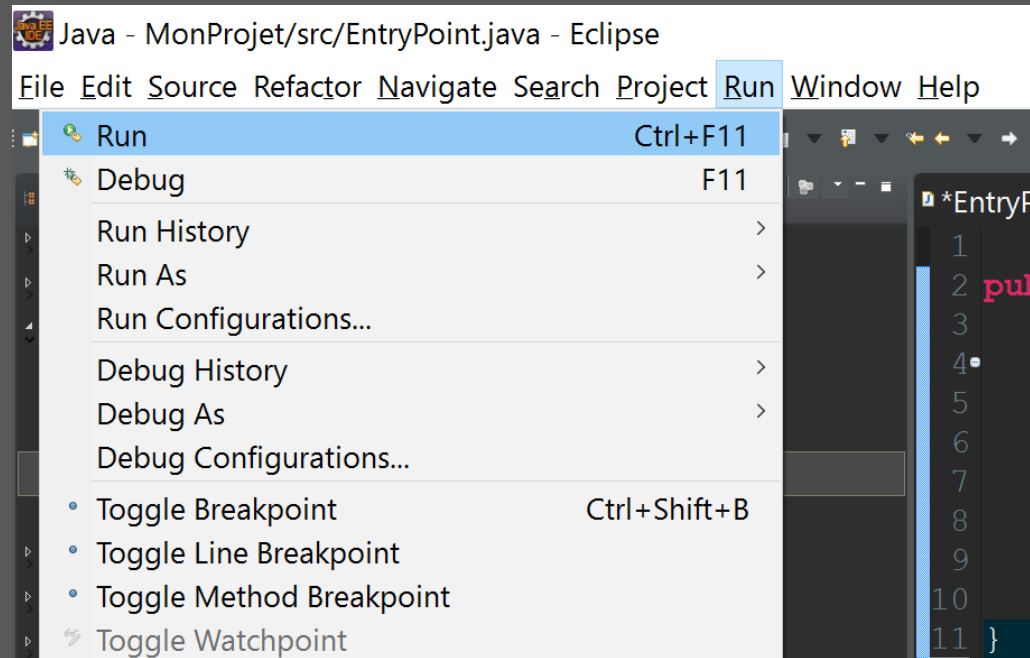


Créer la première classe - 3

Implémenter la méthode main pour afficher un « Hello World ».

```
*EntryPoint.java *  
1  
2 public class EntryPoint {  
3  
4     public static void main(String[] args) {  
5  
6         System.out.println("Hello World !");  
7  
8     }  
9  
10  
11 }
```

Exécuter son projet



Syntaxe du langage

Les variables

- Comme de nombreux langages, pour stocker une information en mémoire, JAVA implémente les variables :
- [Type] [Nom] = [valeur] ;
- Exemple : **int unEntier = 2;**

Cast

- Transformer un type en un autre.

```
int i = 2;
```

```
float j = (float)i;
```

- Afficher i et j puis observer la différence.

Types primitifs, complexes

- Primitif : int, char, float, boolean, etc.
- Complexe : Le type est un objet.
Exemple : **String uneChaine;**
- Note : Les types primitifs possèdent aussi un type complexe associé. Exemple : **Integer;**

Les tableaux

- `int table[] = {0,1,2};`
- `int table[] = new int[6];`
- `int[] table = new int[6];`
- `table[0] = 3;`

Les structures itératives

- `while (boolean) { ... }`
- `do { ... } while (boolean);`
- `for (initialization ; condition ; modification) {
... }`

Les structures conditionnelles

```
if ( boolean ) {  
    ...  
} else if ( boolean ) {  
    ...  
} else {  
    ...  
}
```

Les structures conditionnelles

```
switch (expression) {  
    case valeur1 :  
        ...  
    case valeur2 :  
        ...  
    default :  
        ...  
}
```

Note :

Switch sur une valeur de type String uniquement depuis Java 7.

Application I

1. Initialiser un tableau d'entier à 6 entrées.
 2. Parcourir le tableau et remplir chaque entrée avec un nombre aléatoire entre 0 et 10.
 3. Si le nombre est supérieur à 5 on incrémentera un compteur.
 4. Afficher le tableau et la valeur du compteur.
-
1. Affecter à une variable un nombre aléatoire entre 0 et 10.
 2. Tant que la valeur est différente de 5, recommencer.

Programmation Objet

Définition

- La programmation objet est un paradigme (ou une manière de traiter des problèmes) de programmation.
- Ce paradigme se base sur l'utilisation d'entités nommées « objet ».

Avantages

On obtient un code :

- Organisé
- Évolutif
- Réutilisable
- Robuste

Objet

- Un **objet** permet de modéliser n'importe quel élément nécessaire au programme.
- On peut modéliser des éléments physiques (une voiture) ou virtuels (une connexion à une base de données).

Objet

- Un objet se définit par :
 - des **attributs** ou données : l'état de l'objet.
 - des **méthodes** ou messages : le comportement de l'objet.
- Un objet est décrit par une **classe** ou patron, plan.
- Un objet est une **instance** ou application de la classe qui le définit.

Objet

```
1
2 public class Objet {
3
4     public String attribut = "valeur de l'attribut";
5
6     public void methode() {
7
8     }
9
10 }
```

```
Objet o = new Objet();
o.methode();
```

Encapsulation

- Un objet a pour vocation d'être une boîte noire : exploiter les capacités d'un objet sans en connaître le fonctionnement afin d'en conserver l'intégrité.
- Les données à transmettre ou à récupérer sont définies par le **prototype** des méthodes.
- Techniquement, on définit la **visibilité** d'un attribut ou d'une méthode : **public**, **protected**, **private**.

Encapsulation

```
1
2 public class Objet {
3
4     private String attribut = "valeur de l'attribut";
5
6     public String getAttribut() {
7         return attribut;
8     }
9
10    public void setAttribut(String attribut) {
11        this.attribut = attribut;
12    }
13
14 }
```

Constructeur

- C'est une méthode qui possède les caractéristiques suivantes :
 - Possède le même nom que la classe.
 - Non typée.
 - Appelée automatiquement à l'instanciation.
 - Facultative.
 - Paramétrable.

Constructeur

```
1
2 public class Objet {
3
4     private String attribut = "valeur de l'attribut";
5
6     public String getAttribut() {
7         return attribut;
8     }
9
10    public void setAttribut(String attribut) {
11        this.attribut = attribut;
12    }
13
14    public Objet() {
15        attribut = "valeur initiale";
16    }
17
18    public Objet(String valeur) {
19        attribut = valeur;
20    }
21
22 }
23
```


Destructeur

- Méthode *finalize*

Méthode appelée automatiquement lors la libération de l'espace mémoire occupé par l'objet.

Cette opération est effectuée par la garbage collector.

Héritage

- Un objet A (classe fille) a la possibilité d'utiliser les attributs et les méthodes d'un objet B (classe mère) grâce à l'héritage.



Héritage

```
1
2 public class Personne {
3
4     public void display() {
5         System.out.println("Personne");
6     }
7
8 }
```

```
public class Employe extends Personne {

}
```

```
Employe e = new Employe();
e.display();
```

Héritage

Le mot clé **this** permet d'accès aux informations de l'objet courant.

Le mot clé **super** permet d'accès aux informations de la classe mère.

Polymorphisme

- Le polymorphisme est lié au concept d'héritage.
- Il peut revêtir différentes formes :
 - Redéfinition : le prototype de la méthode reste le même.
 - Surcharge : le prototype de la méthode change.

Polymorphisme

- Chaque classe (Personne, Employe, Directeur) implémente une méthode display avec :
System.out.println(« Nom de la classe »);

```
System.out.println("---");
ArrayList<Personne> listPersonnes = new ArrayList<Personne>();
listPersonnes.add(new Personne());
listPersonnes.add(new Employe());
listPersonnes.add(new Directeur());
for (Personne personne : listPersonnes) {
    personne.display();
}
```

Abstract

- Une classe abstraite ne peut être instanciée. Elle sera héritée et servira de modèle.

```
public class abstract Objet { }
```

- Une méthode abstraite doit obligatoirement être redéfinie par la classe qui hérite de cette méthode et est vide :

```
protected abstract void display();
```

Static

- Ce mot clé s'applique aux attributs et aux méthodes.
- L'attribut ou la méthode ne dépend pas d'une instance mais est commun à toutes les instances de la classe.

Static

```
1
2 public class Objet {
3
4     private static String attribut = "valeur de l'attribut";
5
6     public static void displayAttribut() {
7         System.out.println(attribut);
8     }
9
10 }
```

```
46
47     Objet.displayAttribut();
48
```

Interface

- Mot clé : « Interface ».
- Définit des constantes et des prototypes de méthodes.
- Peut étendre une autre interface.
- Est implémentée par une classe grâce au mot clé « implements ».

Interface

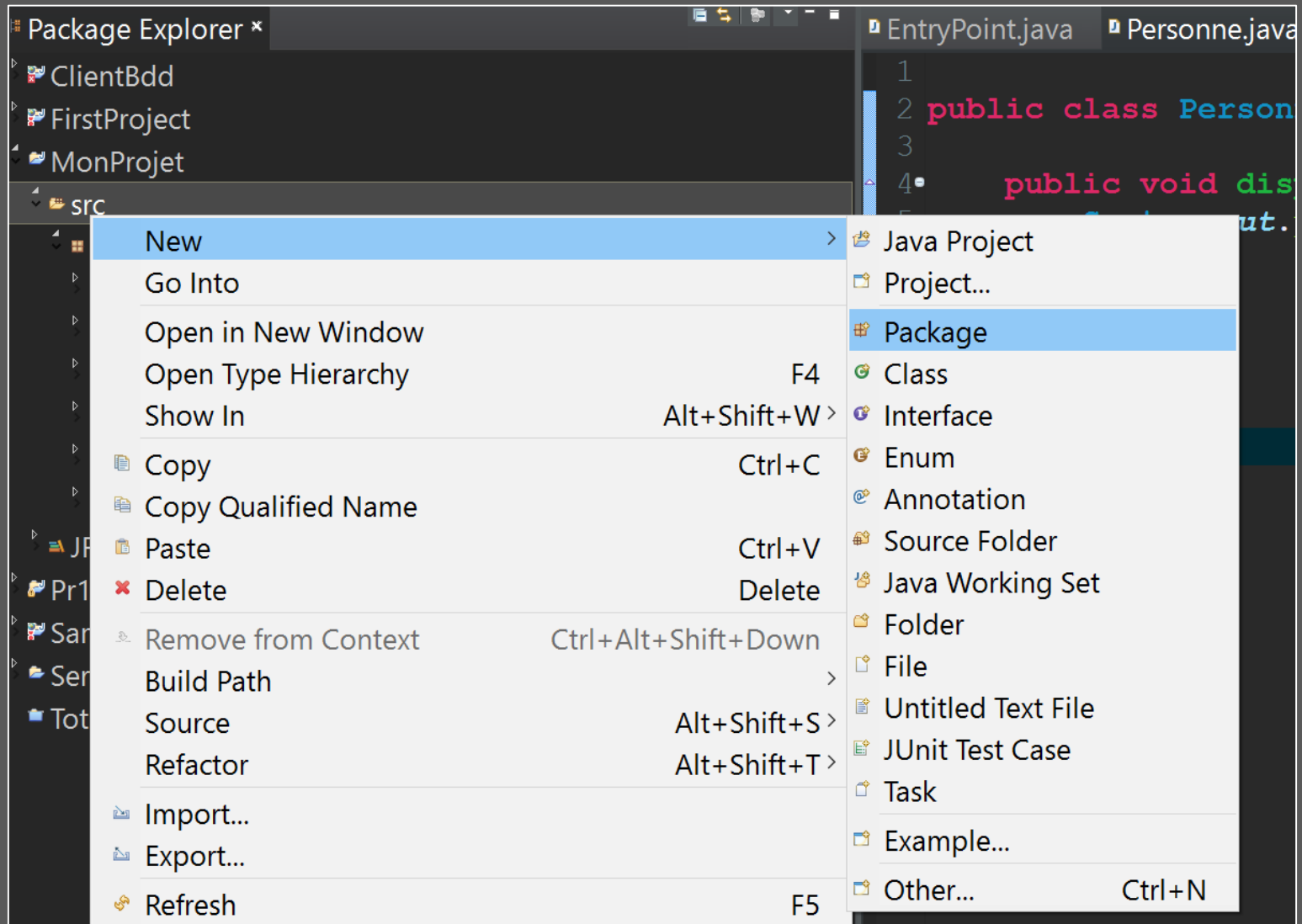
```
1
2 public interface Humain {
3
4     public void display();
5
6 }
```

```
1
2 public class Personne implements Humain {
3
4     public void display() {
5         System.out.println("Personne");
6     }
7
8 }
```

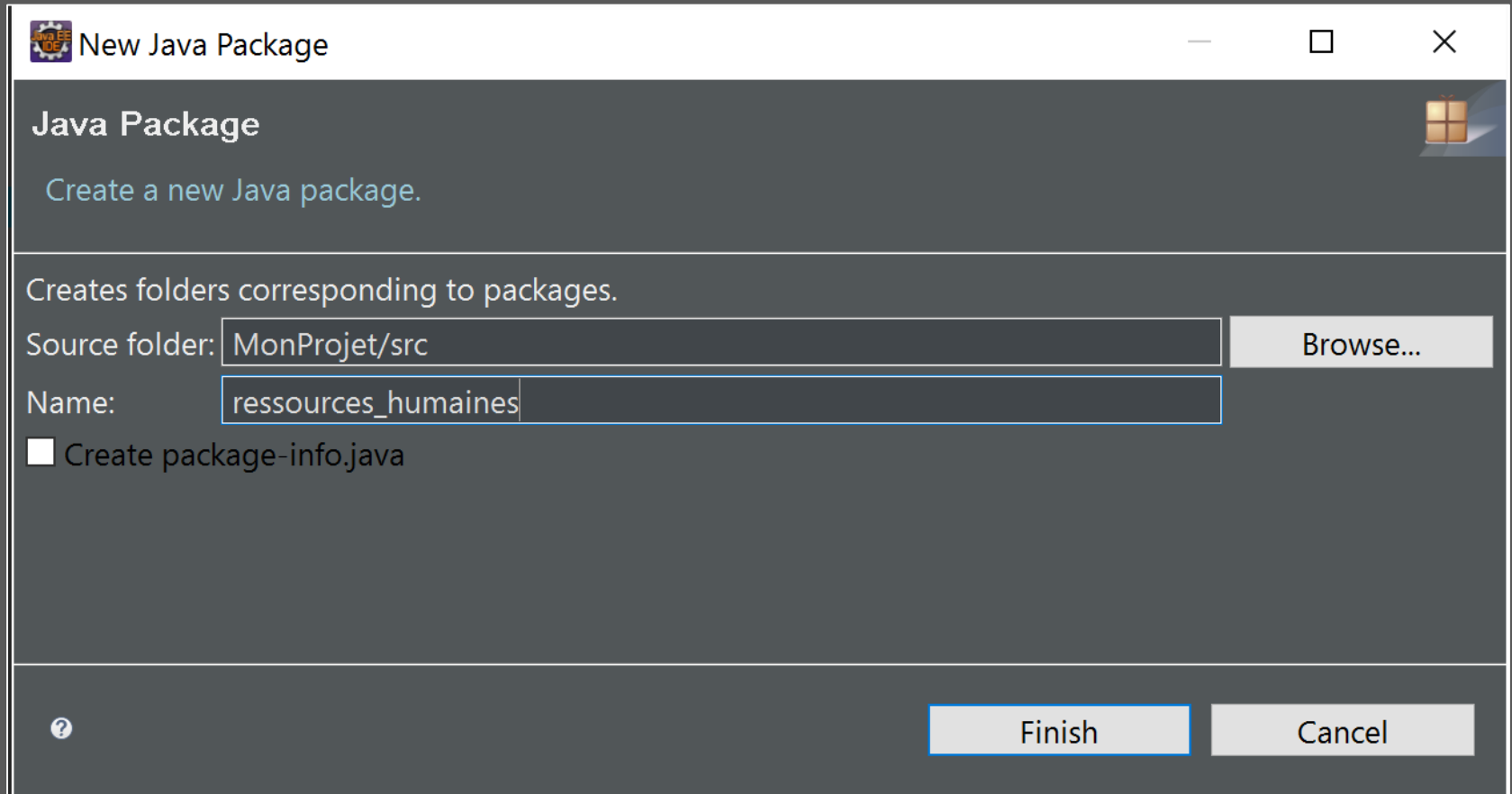
Package

- JAVA fournit un moyen d'organiser les classes : package.
- Un package peut contenir :
 - Un autre package.
 - Une ou plusieurs classes.

Créer un package - I



Créer un package - 2



The image shows a 'New Java Package' dialog box. The title bar includes a Java logo and the text 'New Java Package'. The main area is titled 'Java Package' and contains the instruction 'Create a new Java package.' Below this, it says 'Creates folders corresponding to packages.' There are two input fields: 'Source folder:' with the value 'MonProjet/src' and a 'Browse...' button to its right; and 'Name:' with the value 'ressources_humaines'. At the bottom left, there is a checkbox labeled 'Create package-info.java' which is currently unchecked. At the bottom right, there are 'Finish' and 'Cancel' buttons. A help icon (?) is located at the bottom left of the dialog.

New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

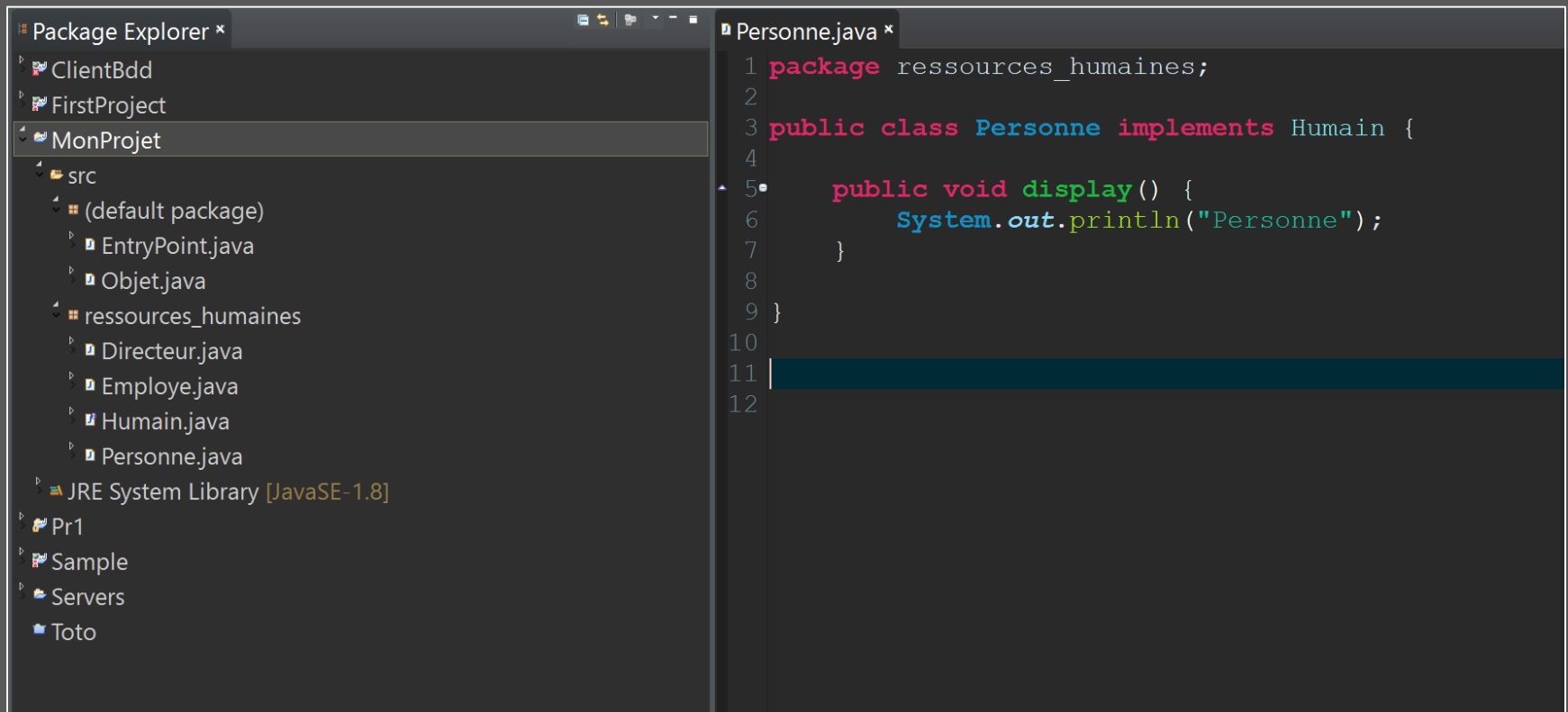
Source folder: MonProjet/src Browse...

Name: ressources_humaines

☐ Create package-info.java

? Finish Cancel

Créer un package - 3



Application 2

Modélisation d'un Jardin virtuel : voir document annexe.

Modélisation

- Les différentes parties du programme peuvent être modélisées selon le paradigme objet, on parle de conception orientée objet ou COO.
- La phase de conception précède la phase de programmation orientée objet ou POO.
- **La clé d'une bonne programmation est donc une bonne conception.**

Design Pattern

- Les développeurs font face aux mêmes problématiques.
- En réponse, des design pattern ou patrons de conception ont été modélisé pour répondre à cette problématique.
- Les design pattern sont basés sur la programmation objet.

Exception

- Mécanisme de gestion des erreurs.
- Objets représentant les erreurs.
- Mot clé : try, catch, finally, throw, throws

Exception

- Pour gérer l'exception :

```
try {  
    // Code pouvant générer l'exception  
} catch(Exception e) {  
    // Traiter l'exception  
}
```

- Pour rediriger l'exception :

```
public void maMethode() throws Exception { }
```

- Pour générer une exception :

```
throw new Exception();
```

Accès aux bases de données

Objectifs

- Une variable a une durée de vie limitée au temps d'exécution de la page web.
- Conserver une information au delà du temps d'exécution de l'application implique la persistance des données permise par les bases de données.
- Une base de données permet **d'organiser** et de **hiérarchiser** les données.

SGBD

Système de Gestion de Base de données :
Logiciel permettant le stockage des données.

- MySQL : Libre et gratuit, très connu.
- PostgreSQL : Libre et gratuit, connu.
- Oracle : Payant, utilisé par les très grosses entreprises.
- Microsoft SQL Server : Payant, propriétaire à Microsoft.

phpMyAdmin

Logiciel permettant l'accès à une base de données MySQL : libre et gratuit.

Interface web :

- Création de bases de données.
- Création de tables.
- Insertion de données.
- Modification de données.
- Suppression de données.
- Import / Export de données.

Prérequis

Driver (connecteur) MySQL

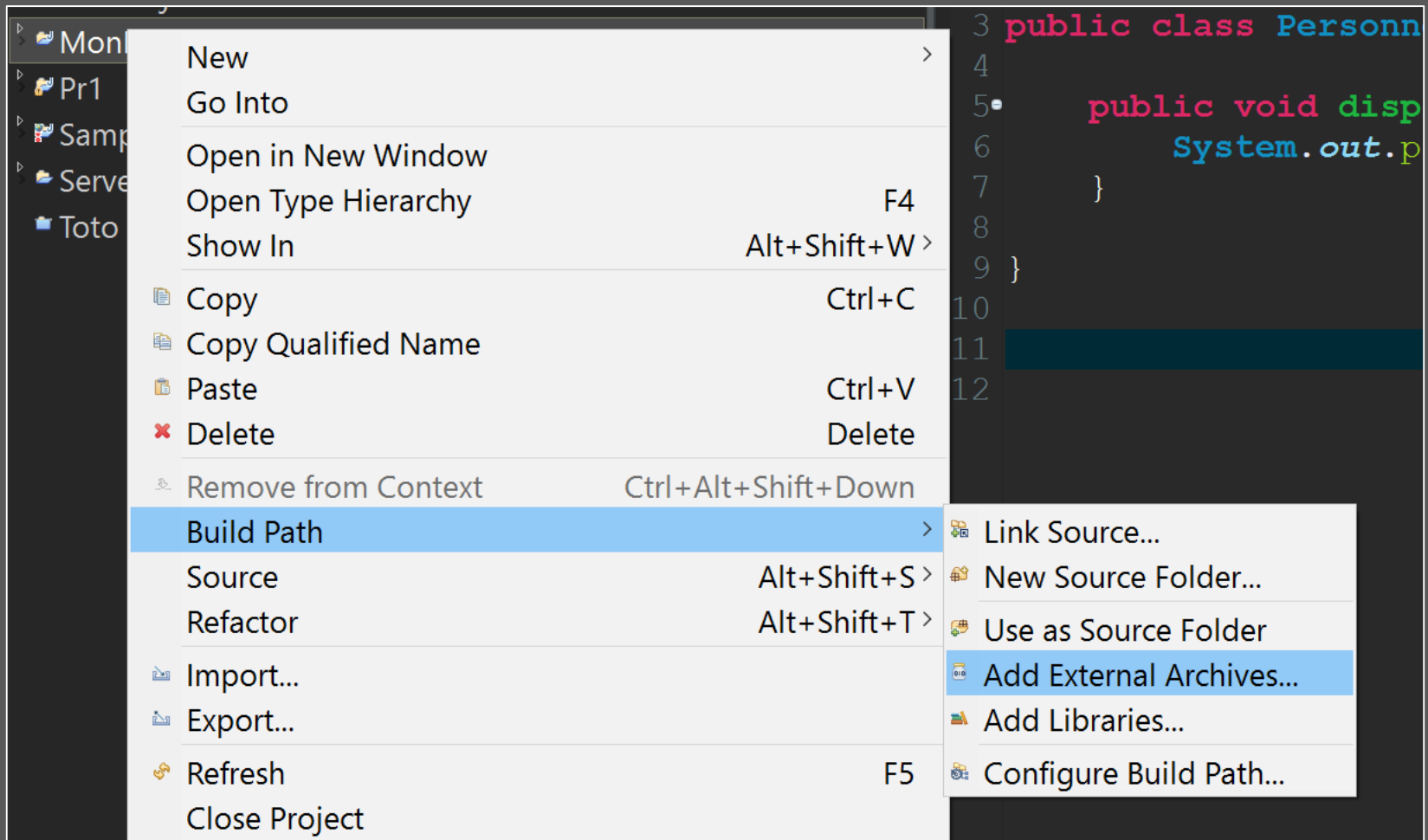
Télécharger :

<http://dev.mysql.com/downloads/connector/j/>

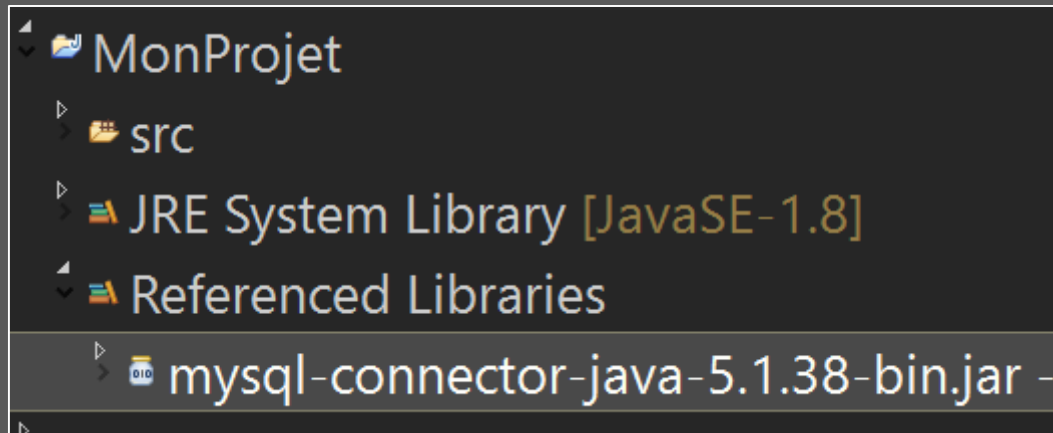
Ressource obtenue :

mysql-connector-java-xxx-bin.jar

Ajouter le driver MySQL - I



Ajouter le driver MySQL - 2



Connexion

```
1 •import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4 import java.sql.Statement;
5
6 public class MySQLManager {
7
8     private Statement statement = null;
9     private Connection connection = null;
10    private String url = "jdbc:mysql://localhost:3306/bdd_java";
11    private String utilisateur = "root";
12    private String motDePasse = "";
13
14    public MySQLManager() {
15        try {
16            Class.forName("com.mysql.jdbc.Driver");
17            this.connection = DriverManager.getConnection( url, utilisateur, motDePasse );
18            this.statement = this.connection.createStatement();
19        } catch (ClassNotFoundException e) {
20            // A gérer.
21        } catch (SQLException e) {
22            // A gérer.
23        } finally {
24            if(connection != null) {
25                try {
26                    connection.close();
27                } catch (SQLException e) {
28                    // Ignore.
29                }
30            }
31        }
32    }
33 }
```

Application

- Mise en place du design pattern Singleton :
- Créer une classe nommé Singleton.
- Ajouter un attribut static typé du nom de la classe nommé instance.
- Ajouter une méthode static et synchronized getInstance qui retourne un objet de type Singleton.
 - Au sein de cette méthode, si instance est null on lui affecte une nouvelle instance.
- Mettre le constructeur en private.
- Tester !

Requête de lecture

- Pour exécuter une requête de lecture :

```
ResultSet resultat = statement.executeQuery(  
    "SELECT id, mail, password FROM User;" );
```

```
while( resultat.next() ) {  
    resultat.getInt(« id »);  
}
```

Requête d'écriture

- Pour exécuter une requête d'écriture :

```
int statut = statement.executeUpdate(  
    "INSERT INTO User (  
        mail, password)  
    VALUES (  
        'toto@toto.com', 'toto')"  
    );
```

- int : succès ou échec ou nombre de lignes impactées

Requête préparée

- Gain de performances.
- Requêtes paramétrables.
- Requêtes protégées.

Requête préparée

```
PreparedStatement preparedStatement =  
connection.prepareStatement(  
"SELECT id, mail, password FROM User;");
```

```
ResultSet rs =  
    preparedStatement.executeQuery();
```

Requête préparée

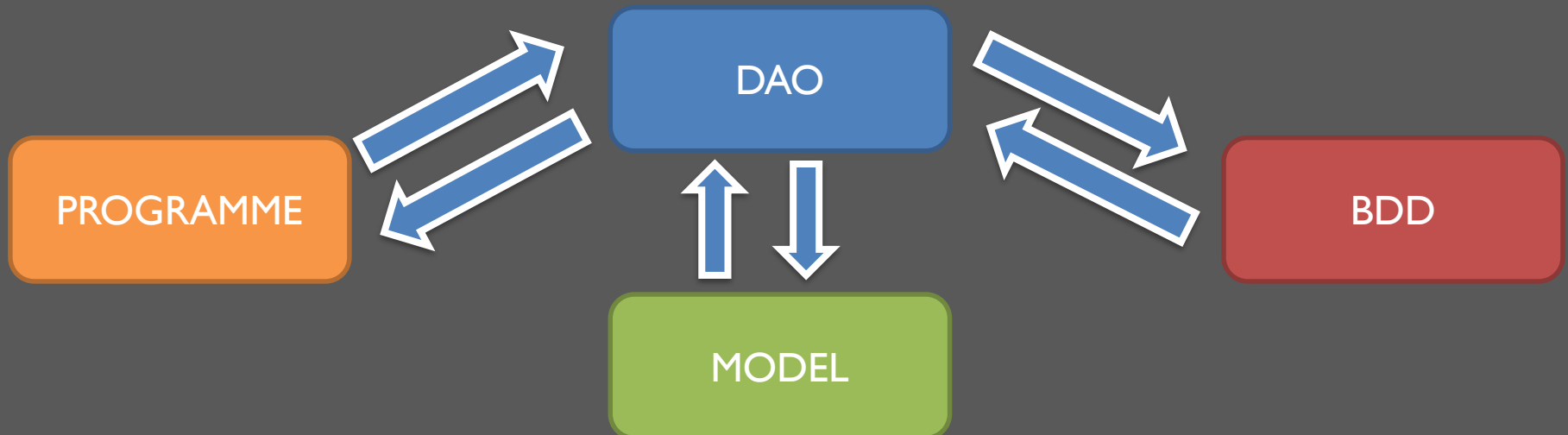
```
PreparedStatement preparedStatement =  
    connection.prepareStatement(  
        "INSERT INTO User (mail, password)  
VALUES(?, ?) ;");
```

```
preparedStatement.setString( 1, paramMail);  
preparedStatement.setString( 2, paramPass);
```

```
int statut = preparedStatement.executeUpdate();
```

DAO

- Séparation de la couche métier et de la couche accès aux données.
- Réponse aux contraintes dues à la gestion de plusieurs bases de données.



Application

- Mise en place du design pattern DAO.

Maven

Gestion de la structure du projet, des dépendances et de la production de l'artefact

1. Télécharger :
<https://maven.apache.org/download.cgi>
2. Ajouter le chemin vers le repertoire bin à la variable d'environnement Path
3. Tester dans la console avec `mvn -version`

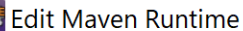
masteri



- General
 - Ant
 - Code Recommenders
 - Data Management
 - Help
 - Install/Update
 - Java
 - Java EE
 - Java Persistence
 - JavaScript
 - Maven
 - Archetypes
 - Discovery
 - Errors/Warnings
 - Installations
 - Java EE Integration
 - Lifecycle Mappings
 - Templates
 - User Interface
 - User Settings
 - Mylyn
 - Oomph

Select the installation used to launch Maven:

Note: Embed



Installation type: ☐ External ☒ Workspace

```
Installation home: D:\Workspace\_maven\apache-maven-3.5.2
```

```
Installation name: apache-maven-3.5.2
```

Additional extension libraries:

Project...

Remove

Up

Down

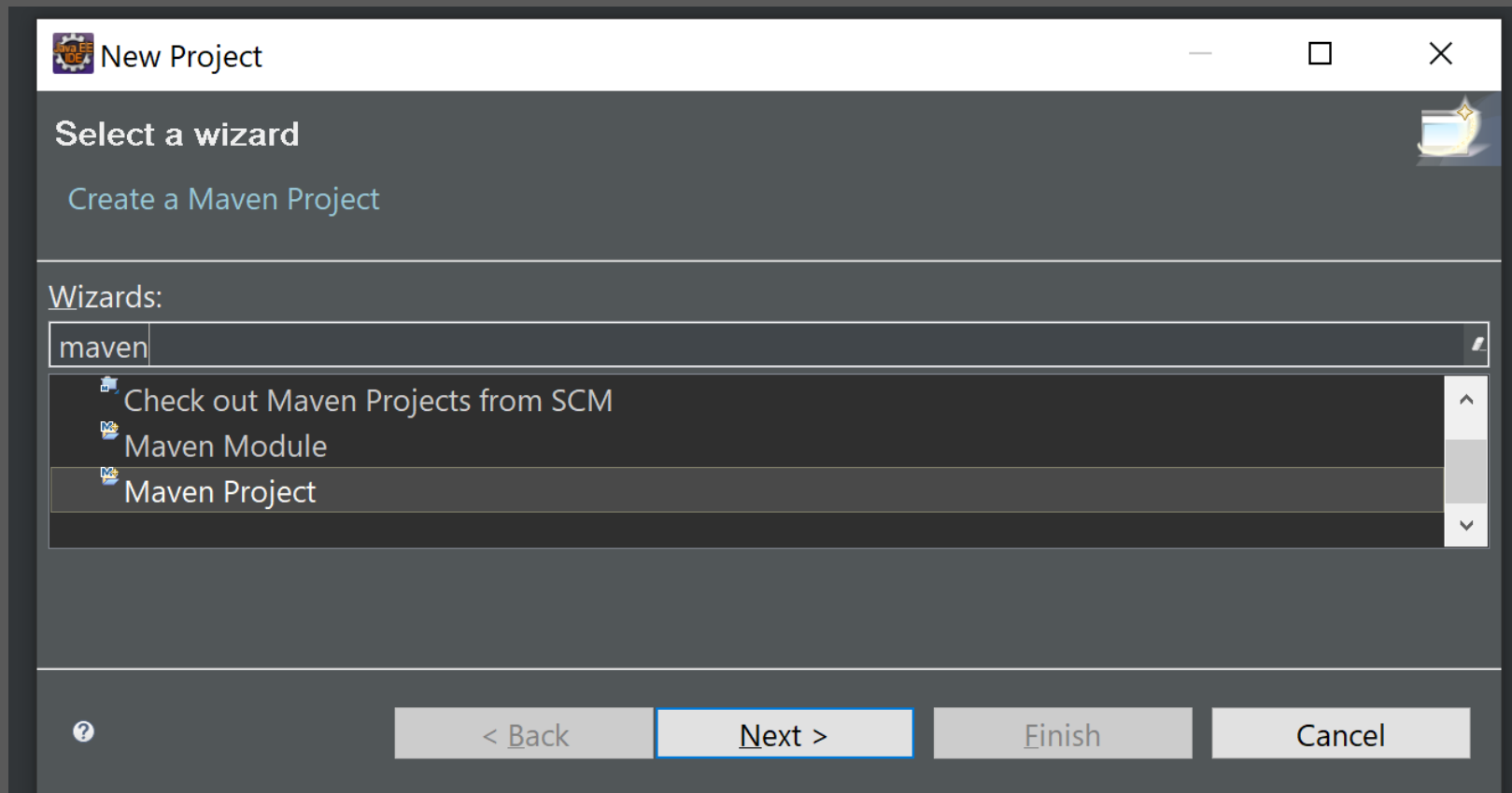
Restore Default

Finish

Cancel

Créer un Maven project

File > New > Project...



Créer un Maven project

Cocher « Create a simple project... »

New Maven Project

New Maven project

[Configure project](#)

Artifact

Group Id: ynov.romain

Artifact Id: firstMavenProject

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name: First Maven Project

Description:

Parent Project

Group Id:

Artifact Id:

Version: Browse... Clear

Advanced

< Back Next > Finish Cancel

pom.xml

Fichier principal d'un projet Maven, il permet :

- 1) Définir les caractéristiques du projet (groupId, artifactId, etc.)
- 2) Définir les dépendances du projet
- 3) Définir les plugins du projet

Build

Un projet maven est construit grâce à la commande : mvn

Exemple :

```
mvn clean package
```

Dépendances

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.9</version>
  </dependency>
</dependencies>
```

Plugin

Traitements supplémentaires lors du build.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <archive>
              <manifest>
                <mainClass>
                  ynov.romain.Bootstrap
                </mainClass>
              </manifest>
            </archive>
            <descriptorRefs>
              <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Application

Créer une application Java qui reprendra le code précédemment créer pour communiquer avec une base de données.

Ce projet sera un 'Maven project'.

Définir les bonnes dépendances et le plugin pour la génération du jar.

Tester le bon fonctionnement avec la commande `java -jar [projectName]-[version]-jar-with-dependencies.jar`

Questions

?