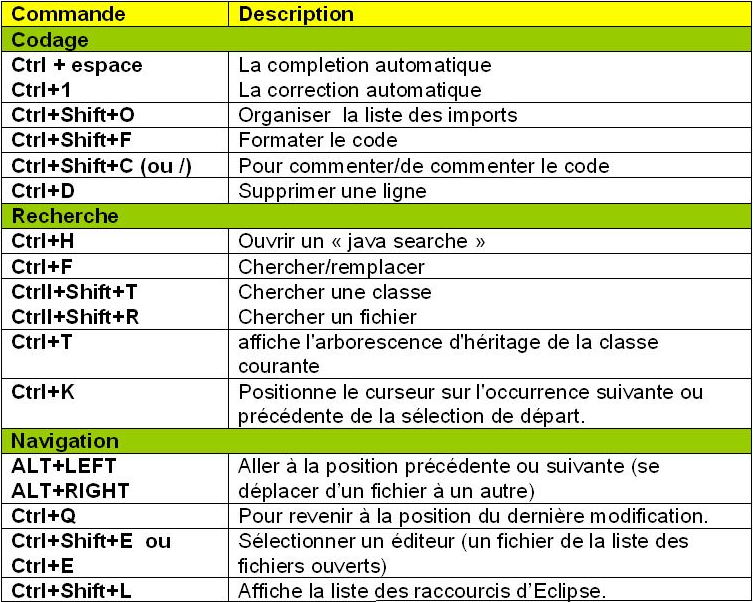
**Cours 5 mars 2018**

**. JAVA EE - Reprise .**

Professeur : Sébastien Péricard

Raccourcis :



ALT MAJ R : Renommer (refactor)  
ALT + fleches : bouger un bloc ou une ligne  
CTRL + MAJ + R : atteindre en tapant  
ALT + MAJ + A : selection par bloc

WINDOWS + E : explorateur de dossiers

F2 : renommer

Type primitif

|  |  |
| --- | --- |
| **Nom** | **Taille (octet)** |
| Boolean | 1 |
| Byte | 1 |
| Short | 2 |
| Char | 2 |
| int | 4 |
| float | 4 |
| long | 8 |
| double | 8 |

String

La manipulation de types primitifs, passe par un passage en copie de la valeur :   
double boucle imbriquée int i = 5; int j = 1 --> j reçoit une copie de la valeur.

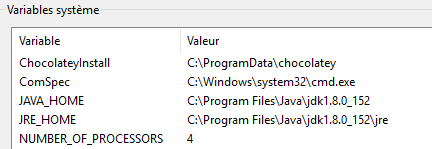
En revanche, pour un type complexe il y a copie de **référence** !   
Le primitif reçoit une copie de l'adresse, pas d'espace mémoire allouée !

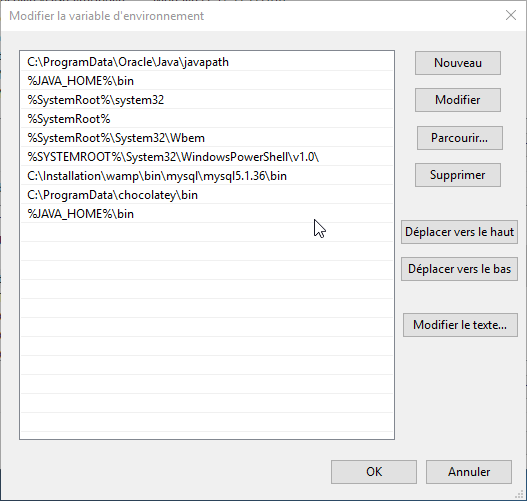
Les string, c'est particulier : si même valeur, elles sont passées en copie d'adresse.   
Ce sont des types primitifs particuliers.   
On peut donc tester avec == sur les nouvelles versions.

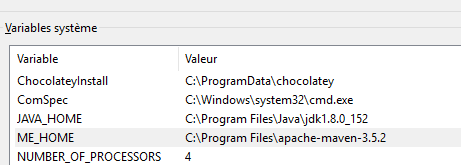
Execution : JRE 🡪 Java JRE\_HOME   
Développement : JDK 🡪 Javac JAVA\_HOME

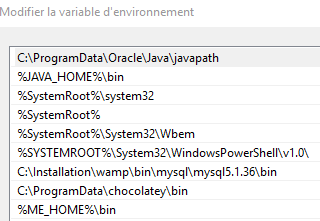
Variables d’environnement

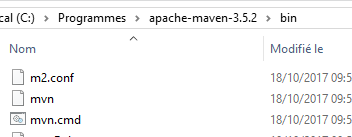
Le path \_HOME est dans tout outil exécutable le chemin d’installation, avant le bin (i.e il y a un \_HOME\bin pour lancer l’exécutable)

On met à jour les variables d’environnement :  


Ajout dans path :  


Idem pour Maven et son path :  




* Chemin maven : mvn  
  En effet, c’est son nom dans le bin :  
  

MAVEN : gestionnaire de Build  
Cycle de projet

Cycle de build [clic openclassroom](https://openclassrooms.com/courses/organisez-et-packagez-une-application-java-avec-apache-maven/familiarisez-vous-avec-le-cycle-de-vie-du-build-maven)

Pour pouvoir gérer la construction d’un projet, il va falloir à chacune des phases de son cycle installer des dépendances

Gestionnaire de dépendances :  
local : (default) /home/user/.m2/repository  
On peut aussi le gérer dans settings.xml, s’il ne trouve pas il télécharge et installe dans le repo control maven.

Compile 🡪 test 🡪 package 🡪 install 🡪 deploy

Plugins : [clic openclassroom](https://openclassrooms.com/courses/organisez-et-packagez-une-application-java-avec-apache-maven/personnalisez-la-construction-avec-les-plugins)

Commande :  
mvn package 🡪 éxecute toutes les phases, jsuqu’à la phase demandée !!  
  
mvn truc :machin 🡪 truc est alors un plugin, et machin le goal  
Ainsi, utilisation des plugins de manière autonome est possible.

Première fois que l’on lance une commande Maven, il va chercher sur le repo local ; s’il ne trouve pas il va chercher sur le réseau local (LAN) éventuellement s’il existe un repo local (Nexus ou Archiva).  
S’il ne trouve pas, il va sur les internets.

Le settings présente par défaut le fichier à utiliser (adresse IP etc).

Nous allons d’abord utiliser internet, puis plus tard on paramètrera Maven pour qu’il aille piocher sur le réseau LAN de l’AFCEPF.

Maven et Windows, petit problème : windows formate les périphériques avec Eclipse, Windows s’affole lors des accès concurents à un fichier 🡪 problèmes.

Eclipse par défaut, est en build auto (paramètre).  
Soucis car dans le contexte d’un projet Maven, CTRL + S lance un maven compile 🡪 remplissage du maven repository en téléchargeant toutes les dépendances et création du target.

Si le projet est hébergé sur un média USB (clef, pas disc car lui est physique pas mémoire flash), plein de fichiers sont susceptibles d’être purement formatés.

**NE JAMAIS avoir son workspace sur USB ou en local !!**

Toujours faire un clean, avant de faire un build 🡪 très souvent on fera donc **mvn clean compile** pour avoir un résultat propre.

Le fichier pom est la base du projet,  
Il définit toujours l’identité du projet (groupId, artifactId, Version)  
C’est cette identité qui permet à Maven de gérer les dépendances.

**Le packaging**  
jar, ejb, war, ear, pom..  
Un jar peut être un ejb, une bibliothèque de classe ou un éxecutable.

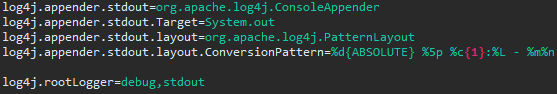
**Les properties** [clic](https://fr.wikipedia.org/wiki/.properties)  
Peuvent être utilisées comme des variables du document maven, utilisées ensuite par l’expression language de maven $nom ou des variables prédéfinies pour le comportement du projet.  
Properties prédéfinies :  
project.build.sourceEncoding :UTF-8  
maven.compiler.source :1.8  
maven.compiler.target : 1.8 (comment vont réellement être compilées les classes)

**Les dépendances**Possède toujours une identité (groupId, artifactId, version) et un scope pour savoir dans quel context va servir la dépendance. Par défaut : compile. Peut aussi être provided (need pour compiler mais pas pour l’exécution), ou test (utilisée uniquement pour les tests), system (dépendance qui n’existe pas dans le repo maven, mais dans un autre dossier, arrive rarement).

**Plugins spécifiques pour le cycle de build du projet :**  
Hors les natifs (que Maven installe lui-même). Pour les ejb et les ear, les jar … on peut ainsi définir comment les construire spécifiquement.

Après le build, on retrouve aussi la section **reporting** :  
maven arrive avec **site**. On préfère utiliser [SonarQube](https://fr.wikipedia.org/wiki/SonarQube).

Premier Projet :

Ajout du log4j  


log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

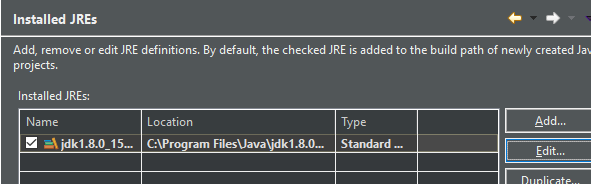
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c**{1}**:%L - %m%n

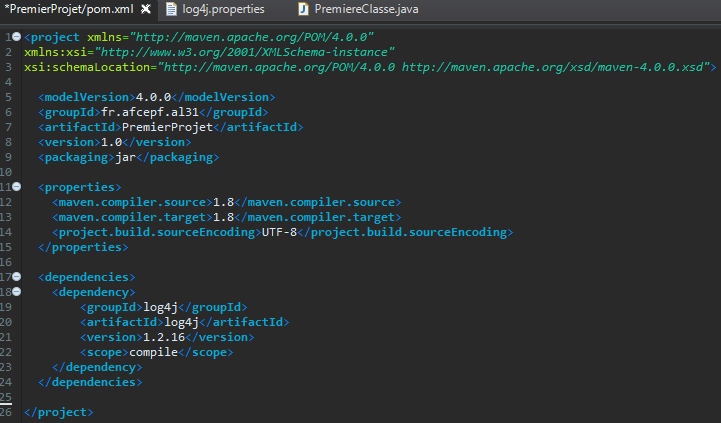
log4j.rootLogger=debug,stdout

Java ARchive  
Web ARchive  
Entreprise ARchive  
  
Ces trois archives ont dans le META-INF le Manifest.mf contenant par exemple author, version, main-class surtout 🡪 gérable avec un plugin de Maven, qui permet au build du projet d’éditer le manifest.

Le mvn deploy va déployer, mais il ne faut pas le faire sur du repo

Ajout de la jdk.



Le pom.xml :  


Si l’on tape (type primitif) int i ; il ne se passe rien, car simple déclaration !  
C’est à l’affectation que tas et pile seront créés  
(type primitif dans un objet complexe, c’est différent attention)  
  
En passant à i = 5, là affectation dans l a table de référence et la mémoire

**ATTENTION le plus grand entier +1, stocké en entier, va donner le plus petit entier !**

Tableau : voir TP.

**Classe**

Définition d’une structure de données  
variables de classe (static) ou d’instance  
constantes (static final)  
(constructeurs)  
méthodes

**Variables & constantes :**  
  
Visibilité :

public tout le monde le voit  
private : pas visible hors de la classe  
protected : visible dans le même package

Type

Nom

La signature d’une méthode : uniquement son nom et ses arguments (nombre et types).  
Cela permet dans une classe de pouvoir surcharger.

Un constructeur : il porte le nom de la classe !  
Il possède, ou pas, des arguments. Constructeur vide.

Une classe est constituée d’un fichier (.java) qui a le même nom que la classe.

Les interfaces définissent des méthodes abstraites (et avant les énumérations, aussi des constantes).

Une classe peut être abstraite en Java : abstract class, elle ne peut alors **pas être instanciée** mais peut posséder des méthodes abstraites.

Dans une interface, les méthodes sont public abstract et donc comme c’est obligatoire, on ne le déclare pas (c’est sous-entendu) !!

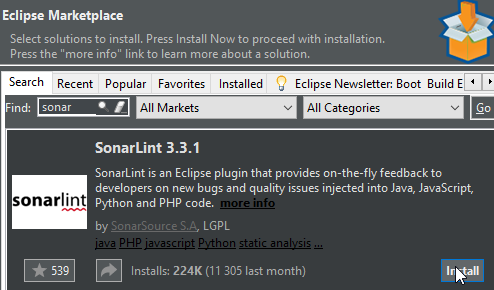
**Objets**

Pattern EXPERT (Encapsulation)  
GRASP : General responsibility assignment software patterns  
getNom() et setNom() qui permettent de manipuler ses attributs, encapsulation.

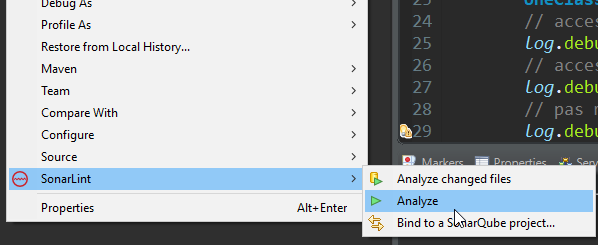
*En C# c’est une propriété avec à l’intérieur les get / set.*

On peut redéfinir la méthode hashCode() mais c’est très dangereux, il faut savoir où l’on met les pieds car c’est utilisé par l’ordinateur pour pointer.

Nb : un StringBuilder, on ne va pas bouger la chaine créée ensuite !  
Un StringBuffer est voué à être ensuite à nouveau concaténé.

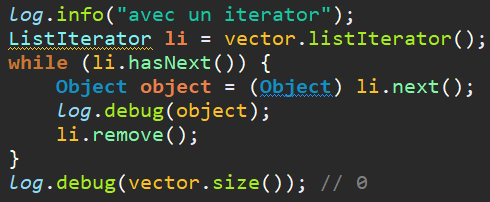
Installation de Sonar, mieux que CheckStyle car propose des solutions.  


On le lance par clic droit sur le projet :



Nb : le parent ‘est instancié’ par eclipse à l’appel d’un enfant, donc s’il y a des debugs on les voit (même si n’a instancié que les enfants).

ListIterator

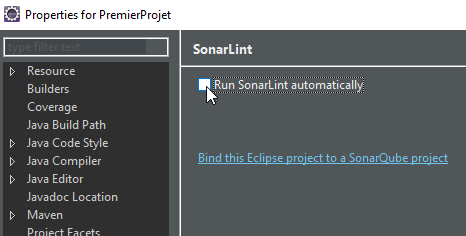


Voir le Class.forName

**File**

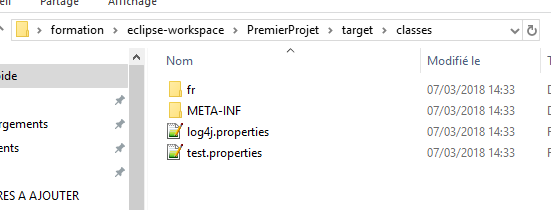
|  |
| --- |
| Manipulation système de fichier mkdir, mkdirs create new File length, name  File Reader : read() 🡪 un **caractère** … jusqu’à end of file. File writer : write(char[] .. String)  Random Access File : lecture / écriture directement les octets du fichier (binaire)  Les Properties |

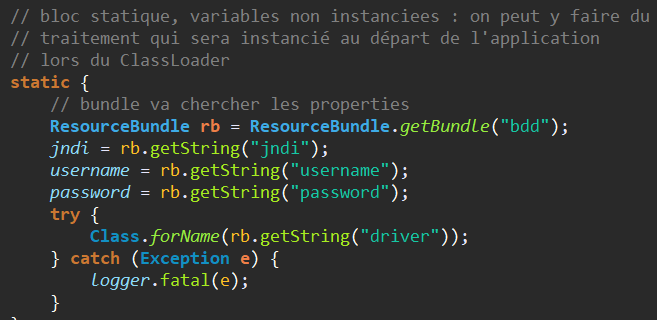
Nb : désactiver Sonar





Nb : attention, les build maven sont créés en dossier TARGET !!





**. Le JDBC .**

Ensemble de classes pour accéder à la base de données ;  
N’est donné dans le java que le JDBC

JDBC ne contient que des interfaces.

Puisque le JDBC est capable de se connecter à tout server de bdd (SQL, MariaDb, PostGreSQL), c’est l’éditeur de la bdd qui donne l’implémentation.

On need des bibliothèques de classes.

On utilisera MariaDb qui est plus performant, opensource 🡪 mieux que MySQL.

On aura toujours besoin d’un **connecteur** i.e une bibliothèque de classe implémentant l’interface pour le JDBC.

Interface Connection qui a besoin pour être créée de plein d’informations :un urlJNDI, un nom d’utilisateur, un password, un driver (de la classe .Driver).  
Ces informations sont données à la classe qui permet de créer la connexion **DriverManager** qui possède .getConnection(url, login, pwd).

Le DriverManager a obligatoirement besoin d’un objet Driver, chargé dans le classLoader (dans les instructions, on a donc un class.forName).

La connection permet de créer différents types d’objets, tels que les **statements** ou des **prepareStatment** (méthodes create).

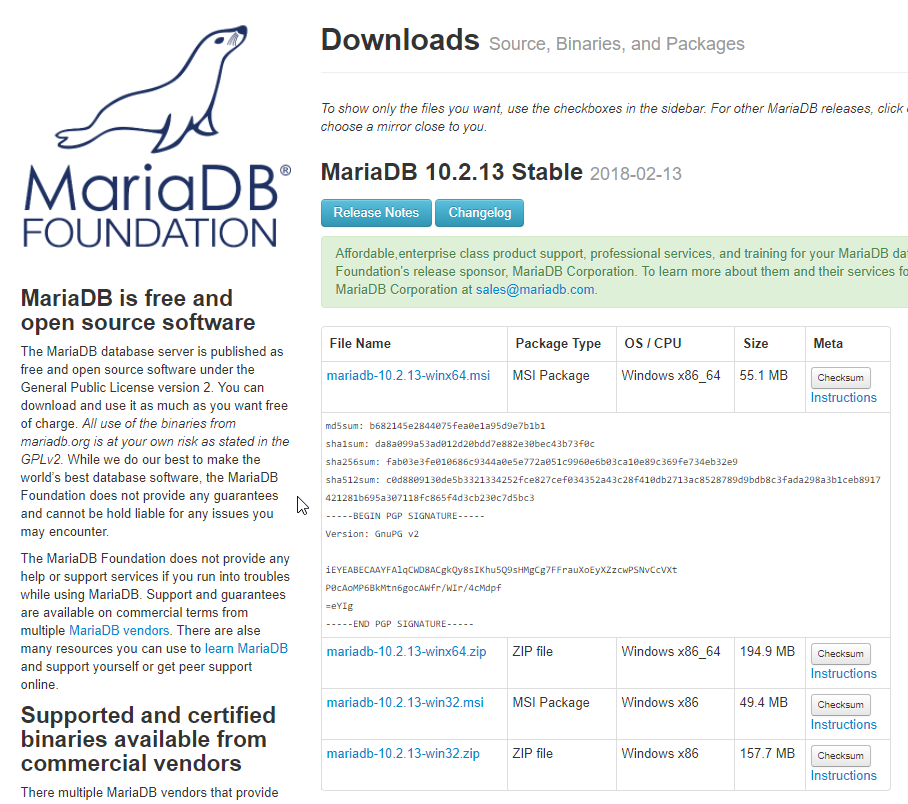
Lorsque l’on éxécute des requêtes SQL, produisent des ResultSet (même fonctionnement que les TableAdapter en .net).

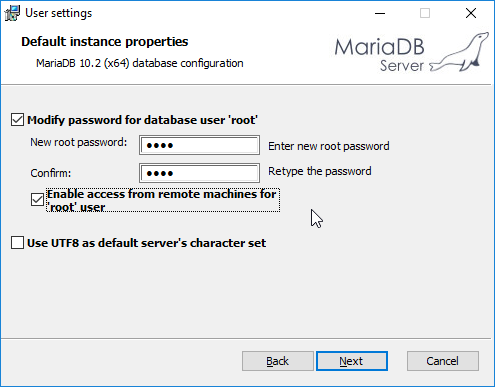
Le ResultSet est un tableau en mémoire, récupérant le résultat de la requête ; on peut dans le ResultSet soit faire une simple lecture, soit se positionner sur une ligne, supprimer des infos, ajouter des lignes, etc 🡪 le ResultSet va ensuite automatiquement faire les requêtes à la base !!

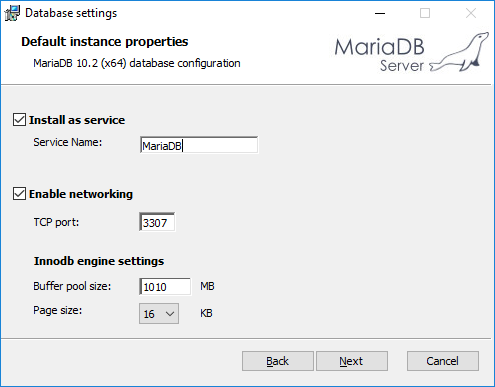
Pour gérer les connections, fourniture d’un élément qui vient de JEE : javax.sql

**DataSource permettent de construire et gérer les connections.**

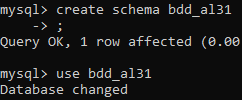
**Configuration de la bdd MariaDB**

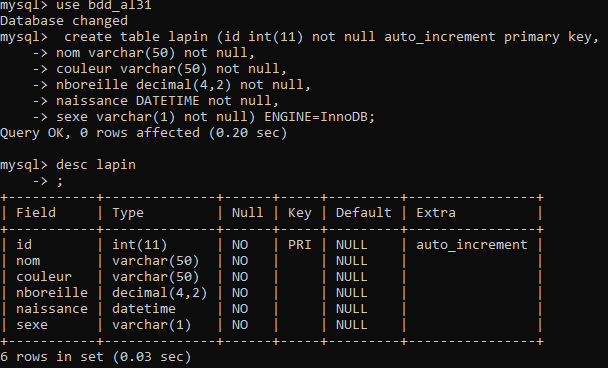
****

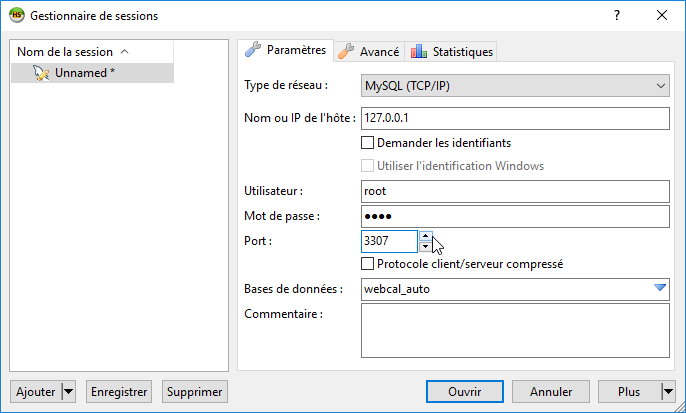
****

****

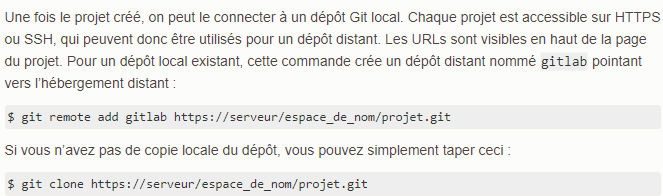
****

****

****

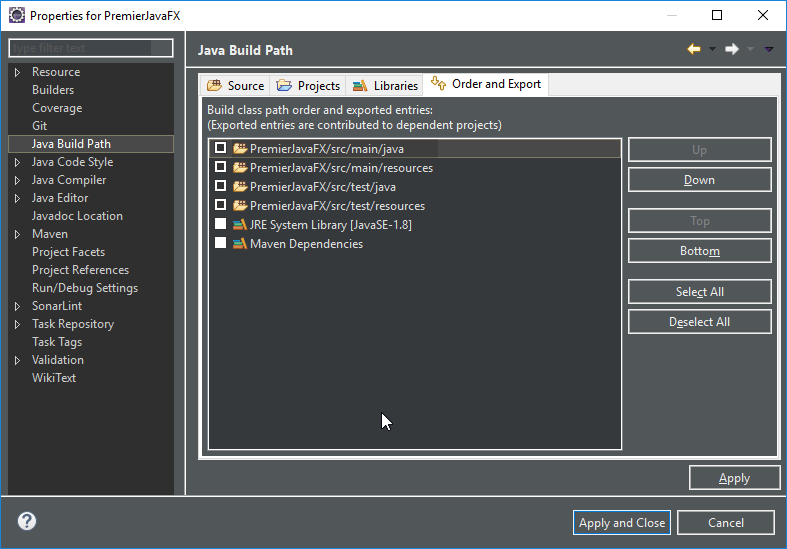
****

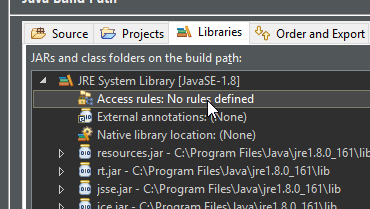
**Git**

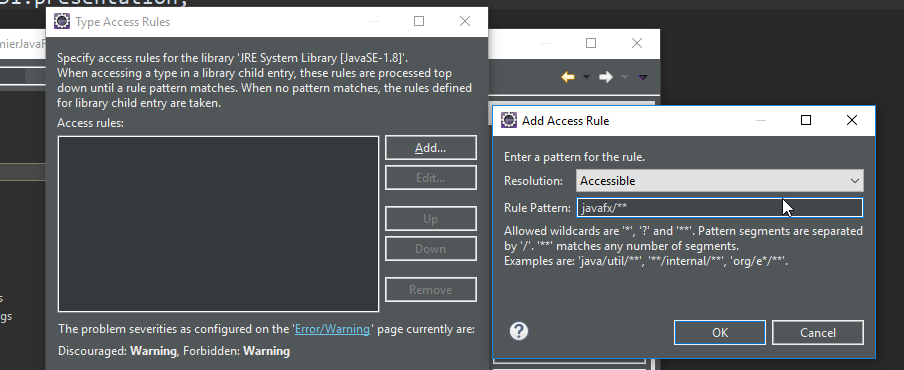
****

**JAVA FX**

Ajout du buildpath – pas là par défaut !!

****

****

****

Voir TP pour les interfaces

Evenements

Trois manières de les gérer

* Anonyme : btn.setOnAction(new EventHandler() {}) ;
* Container : btn.setOnAction(this) ;
* Classe : btn.setOnAction(new MaClasse(…)) ;