

Langage JAVA

Initiation et Approfondissement

Woodson JUSTE
wjuste@dawan.fr

Plus d'informations sur <http://www.dawan.fr>
Contactez notre service commercial au **0810.001.917** (prix d'un appel local)

DAWAN Paris, 11, rue Antoine Bourdelle, 75015 PARIS

DAWAN Nantes, 28, rue de Strasbourg, 44000 NANTES

DAWAN Lyon, Batiment de la banque Rhône Alpes, 2ème étage, montée B - 235, cours Lafayette, 69006 LYON

DAWAN Lille, 16, place du Générale de Gaulle, 6ème étage, 59800 LILLE

formation@dawan.fr

Programme



- ☐ Découvrir la plateforme Java
- ☐ Découvrir l'environnement de développement
- ☐ Maîtriser les bases
- ☐ Apprendre l'objet
- ☐ Gérer les exceptions/les erreurs
- ☐ Utiliser des collections
- ☐ Manipuler des fichiers
- ☐ Utiliser les fonctions de base de Git dans Eclipse
- ☐ Accéder à des base de données
- ☐ Généricité

Historique, versions



- ☐ 90th : Sun Microsystems & James Gosling
- ☐ Première version : langage OAK
- ☐ 1995 : Lancement public de Java
- ☐ 2000 : version 1.2 avec **JavaSE** et **JavaEE**
- ☐ 2006 : Java devient open source
- ☐ 2009 : Java est racheté par Oracle
- ☐ 2014 : Java 8
- ☐ 2018: Java 11
- ☐ 2021 : **Java 17**
- ☐ 2023 : Java 19

Technologies Frameworks Java et positionnement



Ce Framework est utilisé pour développer des applications complexes et qui doivent être performantes. Ainsi, les développeurs sont capables de créer facilement des applications complètes pour de grosses entreprises.

Il est open-source léger et modulaire, peut être utilisé pour développer n'importe quelle couche du projet.

Avec Spring, les développeurs peuvent créer des modules faiblement couplés dans lesquels les dépendances sont gérées par le Framework, plutôt que de dépendre des bibliothèques du code.



Vaadin est un Framework puissant spécialement conçu pour le développement d'applications Web de composants personnalisables.

C'est aussi un cadre léger qui se concentre sur les performances, l'expérience utilisateur (l'UX) et l'accessibilité.



Hibernate facilite le travail de mapping objet relationnel (ORM : Object-Relational Mapping). Grâce à ce Framework, il est beaucoup plus facile de convertir les bases de données. Sa capacité de pouvoir gérer plusieurs bases de données facilite son évolution. Il est performant, facile à mettre à l'échelle, à modifier et à configurer.

Technologies

Frameworks Java et positionnement



Cette infrastructure permet aux développeurs de créer des applications de niveau professionnelles, faciles à entretenir. Hibernate et Spring sont souvent utilisés pour le mapping objet relationnel et pour l'injection de dépendance. Le développement d'applications reliées à cette infrastructure Java auront un temps de traitement réduit.



Développé par Oracle, Java Server Faces est un Framework MVC reposant sur les composants; l'état des composants est enregistré lors de la génération du rendu de la page et est ensuite restauré au retour de la requête.

JSF est notamment utile pour le développement d'applications serveurs complexes, puisqu'il permet de tester l'UI en direct, sans avoir à ajouter d'autres Frameworks ou bibliothèques. Dans le Framework il y'a du HTML, CSS, JavaScript permettant aux développeurs de drag and drop les différents composants UI sans code supplémentaire. Avec ce Framework, vous pourrez donc vous concentrer sur l'essentiel.



Apache Tomcat est une implémentation open source d'un conteneur web qui permet donc d'exécuter des applications web reposant sur les technologies servlets et JSP.

Tomcat est diffusé en open source sous une licence Apache

Dans une architecture d'application web, on a besoin d'un serveur d'application. En associant Tomcat et Spring on peut créer un serveur d'application. Il va gérer le cycle de l'application. Il va se charger de l'instanciation, de la sérialisation, désérialisation, l'inversion de contrôle.

Plateforme Java



- **Java SE** : Java Standard Edition. Il est disponible sous deux formes :
 - Le **JRE** (Java Runtime Environment) : Il s'agit de environnement d'exécution.
 - Le **JDK** (java development kit) : Il contient le JRE, et d'un certain nombre d'outils de développement (compilateur notamment)
- **Java EE** : Java Entreprise Édition
 - Pour développer les applications qui vont s'exécuter dans un serveur d'application JEE (Web Sphere Web Logic, JBOSS, Glassfish)
- **Java ME** : Java Micro Édition
 - Destiné aux applications pour systèmes embarqués et mobiles (microcontrôleur, capteur, passerelle, smartphone, assistant personnel numérique, décodeur TV, imprimante)
- L'évolution de java est gérée par le JCP (Java Community Process) qui émet des JSR (Java Specification Requests). elles décrivent les spécifications et les technologies proposées pour un ajout à la plateforme Java

<https://docs.oracle.com/en/java/javase/17/docs/api/>

■ La machine virtuelle (JVM) ?

La machine virtuelle développée par Sun représente le conteneur dans lequel le code Java est exécuté.

la machine virtuelle définit une architecture d'exécution complète :

- un jeu d'instructions précis ;
- des registres ;
- une pile.

La machine virtuelle permet notamment :

- l'interprétation du bytecode
- l'interaction avec le système d'exploitation
- La gestion de sa mémoire grâce au ramasse-miettes (Garbage Collector)

■ La machine virtuelle (JVM) ?

Plusieurs zones de mémoire sont utilisées par la JVM :

- les **registres** : ces zones de mémoires sont utilisées par la JVM exclusivement lors de l'exécution des instructions du byte code.
- une ou plusieurs **pires** (stack)
- un **tas** (heap)
- une zone de méthodes (method area)

La Pile (Stack)

- Seules des données de type **primitif** et des **références** à des objets peuvent être stockées dans la pile. La pile ne peut pas contenir d'objets.
- Si la taille d'une pile est trop petite pour les besoins des traitements d'un thread alors une exception de type **StackOverflowError** est levée.
- Si la mémoire de la JVM ne permet pas l'allocation de la pile d'un nouveau thread alors une exception de type **OutOfMemoryError** est levée.

■ La machine virtuelle (JVM) ?

Le tas (Heap)

- Tous les objets créés sont obligatoirement stockés dans le tas (heap)
- La libération de cet espace mémoire est effectuée grâce à un mécanisme automatique implémenté dans la JVM : le ramasse-miettes (Garbage Collector).

La zone de mémoire "Method area"

- Cette zone de la mémoire, partagée par tous les threads, stocke la définition des classes et interfaces, le code des constructeurs et des méthodes, les constantes, les variables de classe (variables static) ...
- Comme pour la pile, seules des données de type primitif ou des références à des objets peuvent être stockées dans cette zone de mémoire. La différence est que cette zone de mémoire est accessible à tous les threads

■ Qu'est-ce que JRE ?

JRE (Java Runtime Environment) est un progiciel qui fournit des **bibliothèques** de classes Java, ainsi que **JVM** (Java Virtual Machine), ainsi que d'autres composants permettant d'exécuter des applications écrites en programmation Java.

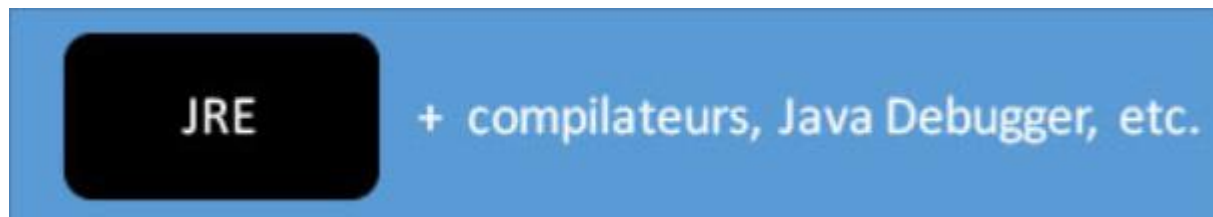
JRE est le sur-ensemble de JVM.



Si vous devez exécuter des programmes Java, JRE est ce dont vous avez besoin.

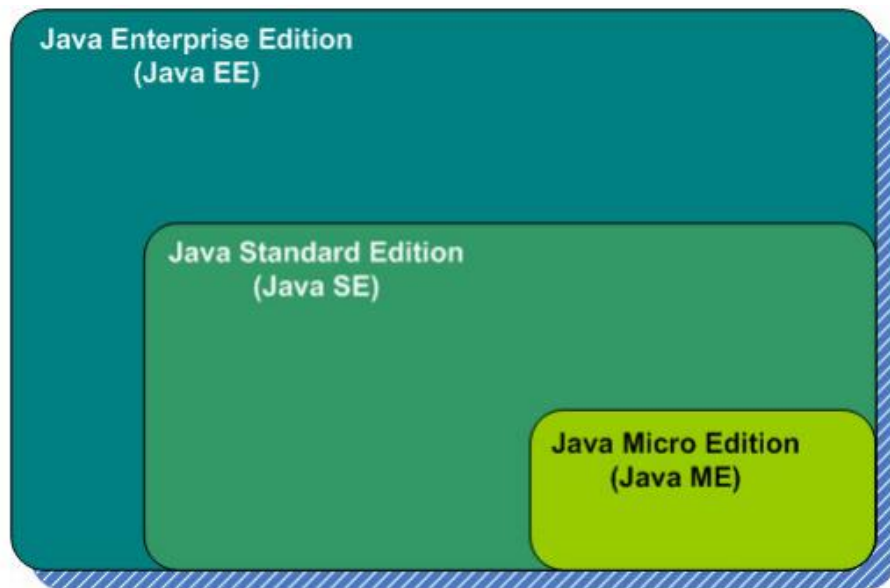
■ Qu'est-ce que JDK ?

JDK (Java Development Kit) est un kit de développement logiciel permettant de développer des applications en Java. Lorsque vous téléchargez JDK, JRE est également téléchargé et vous n'avez pas besoin de le télécharger séparément. En plus de JRE, JDK contient également plusieurs outils de développement (compilateurs, JavaDoc, Java Debugger, etc.).



Plateforme Java

■ Vue d'ensemble



Plateforme Java



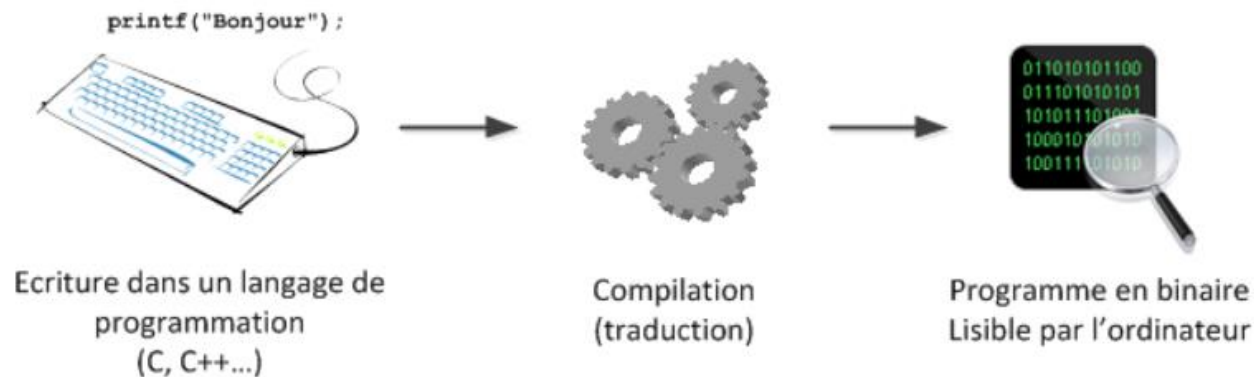
Architecture logicielle de Java SE.

<https://docs.oracle.com/en/java/javase/17/>

Compilation et interprétation par la JVM

- Les langages traditionnelles : C et C++

La **compilation** est le processus par lequel un programme dit au compilateur de traduire le code source en code binaire exécutable



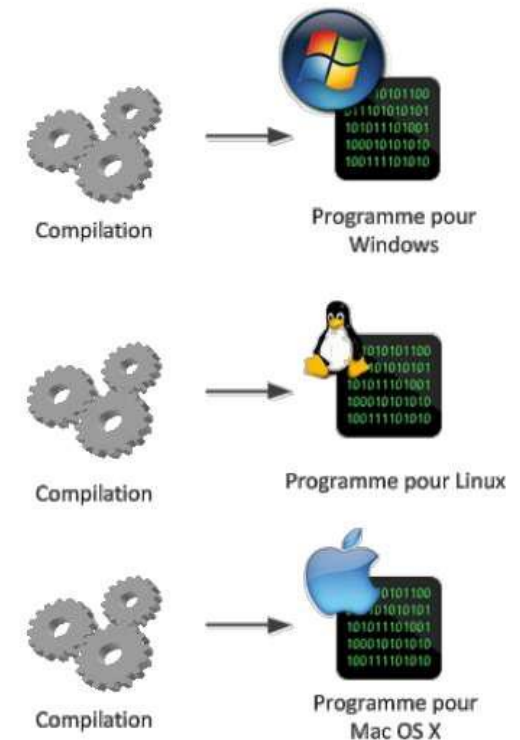
- Les langages traditionnelles : C et C++

Inconvénients

Notre programme tourne sur un Système d'exploitation et un processeur précis.

Ex: *pour faire tourner un programme écrit sous une machine Windows avec un cpu x86 sous une machine linux avec un cpu x64, nous devons recompiler le code suivant les spécificité du systèmes voire faire une modification dans le code pour qu'il s'adapte à ce dernier*

=> Problème d'incompatibilité entre systèmes d'exploitation



Compilation et interprétation par la JVM

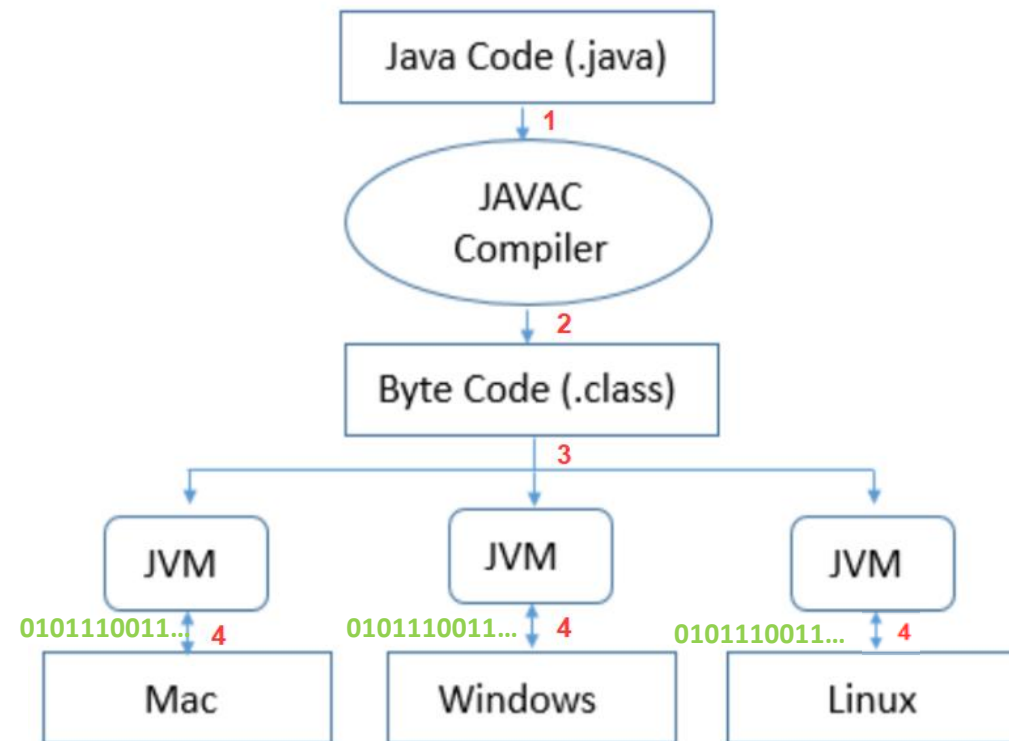
■ Les langages récents Java: la compilation

Étape 1) Le code enregistré dans un fichier **.java**.

Étape 2) En utilisant le compilateur java, le code est converti en un code intermédiaire appelé **bytecode**.
La sortie est un fichier **.class**.

Étape 3) Ce code n'est compris par aucune plate-forme, mais uniquement par une plate-forme virtuelle appelée Java Virtual Machine (**JVM**).

Étape 4) Cette machine virtuelle réside dans la RAM de votre système d'exploitation. Lorsque la machine virtuelle reçoit ce bytecode, elle identifie la plate-forme sur laquelle elle travaille et convertit le bytecode en code machine natif



- Les langages récents Java: la compilation

Avantages : Un langage indépendant de toute plate-forme

- Problème d'incompatibilité résolu
- Le code intermédiaire le ByteCode peut être distribué à tout le monde
- Multiplate-forme (Linux, Windows, MacOS)

- Les différents outils pour programmer en java
 - Éditeur de texte
 - NotePad++
 - IDE (Integrated Development Environment)
 - Netbeans
 - support officiel ORACLE, en général en avance sur les maj JAVA et outils
 - IntelliJ
 - Ergonomie, fonctionnalités, no FOSS (*Free and Open Source Software*)
 - Eclipse
 - Historiquement l'IDE le plus utilisé, communauté / entreprise, structure basé sur un système de module, facile pour customiser (d'autres IDEs se basent sur Eclipse)
 - JDK (Java Development Kit)
 - Minimum conseillé
 - Debugger
 - Compiler
 - Syntaxe, auto complétion, javadoc

■ Installation de java JDK

- Le Kit de développement java JDK peut être téléchargé gratuitement à partir du site d'oracle, son éditeur principal (www.oracle.com)
- Ce kit de développement comprend de nombreux outils situé dans le répertoire *C:\Program Files\Java\jdk1.8.0_162\bin* à savoir :
 - le compilateur java: **javac**
 - l'interpréteur du bytecode : **java**
 - le débogueur java : **jdb**
 - le générateur de documentation de nos programmes java : **javadoc**
 - le désassembleur du bytecode: **javap**

Lien de téléchargement :

<https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html>

- Configuration de l'environnement
 - La configuration de l'environnement comporte deux aspects :
 - Définir la variable d'environnement path qui indique le chemin d'accès aux programmes exécutables
 - Cette variable path doit contenir le chemin du JDK utilisé:
 - *path= C:\Program Files\Java\jdk1.8.0_162\bin*
 - Quand elle exécute une application java, la JVM consulte la variable d'environnement classpath qui contient le chemin d'accès aux classes java utilisées par cette application.
classpath= .; c:\monProjet\lib; c:\programmation ; C:\Program Files\Java\jdk1.8.0_162\bin
 - Paramétrages des variables d'environnement
 - *JAVA_HOME= C:\Program Files\Java\jdk1.8.0_162*
 - *PATH=%JAVA_HOME %\bin*

■ Premier programme

```
1 // Définition d'une classe
2 class HelloWorld
3 {
4     /* Votre programme commence par un appel à main().
5     Affiche "Hello, World" dans la fenêtre du terminal. */
6
7     public static void main(String args[])
8     {
9         System.out.println("Hello, World");
10    }
11 }
```

- Le processus de programmation Java peut être simplifié en trois étapes:
 - Création du programme dans un éditeur de texte et l'enregistrer dans un fichier [First.java](#).
 - Compiler le programme en écrivant «[javac First.java](#)» dans la fenêtre du terminal.
Après compilation on obtient un fichier [First.class](#) (le Bytecode)
 - Exécuter le ByteCode en tapant «[java First](#)» dans la fenêtre du terminal.
Après exécution par la JVM (machine virtuelle Java) on obtient le code binaire (011100...)

■ Environnement de développement (IDE) ECLIPSE

- Eclipse peut être téléchargé gratuitement à partir du site <https://www.eclipse.org/downloads/>
- Eclipse va nous permettre de regrouper plusieurs étapes dont :
 - L'éditeur de texte pour écrire nos codes sources
 - Le compilateur permettant au programmeur de vérifier les erreurs de syntaxes, et de transformer le code source en bytecode
 - Le débogueur permettant de détecter et de diagnostiquer les erreurs dans vos programmes.

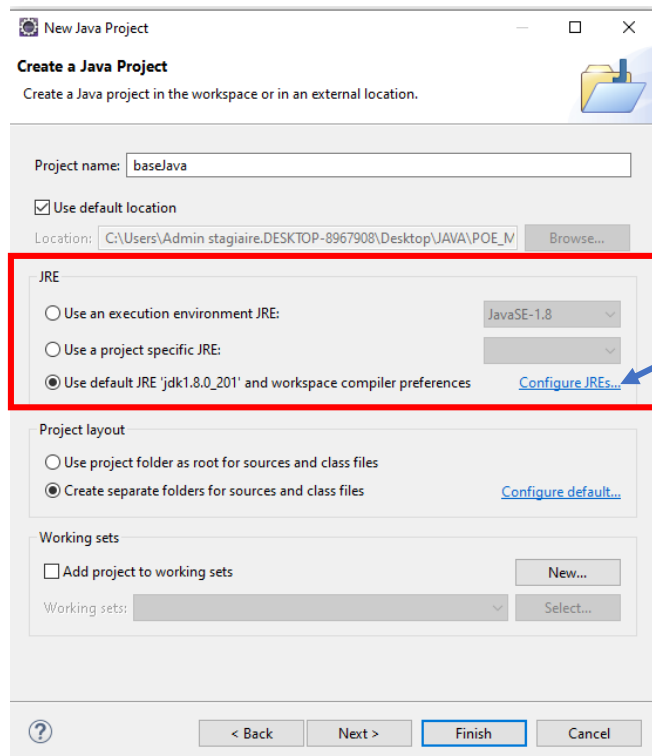
Le debugger vous permet de contrôler l'exécution de votre programme en spécifiant des points d'arrêt (breakpoints), en suspendant des programmes en cours d'exécution, en avançant pas-à-pas dans le code et en examinant l'état des variables.

- Il va nous permettre aussi d'exécuter nos applications

■ Configuration de base d'un projet dans Eclipse(IDE) ECLIPSE

• Créer un nouveau projet sur Eclipse

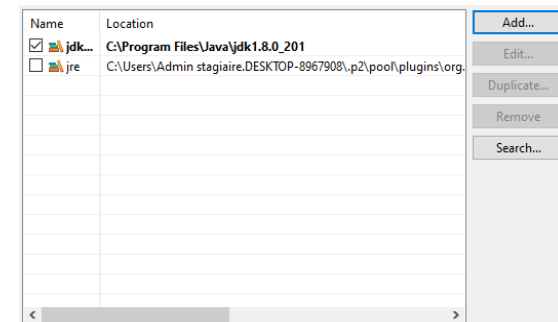
- Cliquez sur **File -> New -> Java Project** ou **File -> New -> Other -> Java -> Java Project**



Configurer le JDK

Cliquer sur **Configure JREs -> Add -> Standard VM -> Next -> Directory**
Indiquer le chemin du JDK `C:\Program Files\Java\jdk1.8.0_201`
Cliquez sur Finish

Selectionner le JDK que vous venez d'ajouter
Cliquez sur Apply And Close

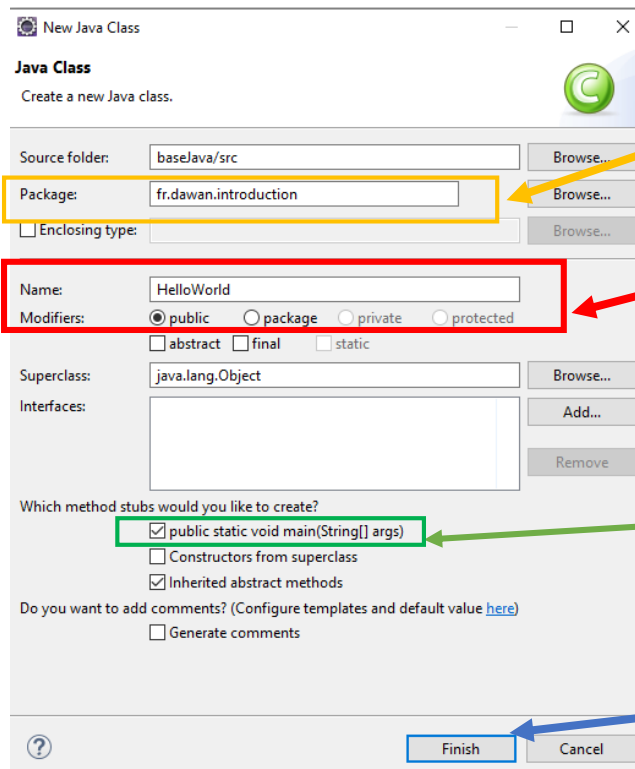


- Après configuration du JDK cliquer sur finish

■ Configuration de base d'un projet dans Eclipse(IDE) ECLIPSE

• Créer une classe

- Cliquez droit sur le dossier src -> New -> Class



○ Utilisations des paquetages

- Les paquetages permettent de classer vos fichiers un peu comme on classe des documents dans des répertoires.

○ Choisir le nom de classe

○ Cochez public static void main(String[] args)

- Cela permettra de générer la méthode main lors de la création de la classe

○ Cliquez sur Finish

■ Raccourcis utiles

• Les indispensables

Cette section regroupe les raccourcis totalement indispensables pour utiliser l'éditeur Java simplement. Si vous devez en retenir que quelques uns, ce sont ceux là:

- **Ctrl + espace**: Active l'auto-complétion, tout simplement indispensable.
- **Ctrl + O**: Liste toutes les méthodes de la classe en cours d'édition.
 - Une nouvelle combinaison **Ctrl + O** liste les méthodes pour la hiérarchie complète de la classe. Vous pouvez filtrer les résultats en tapant le début du nom d'une méthode.
- **Ctrl + E**: Liste l'ensemble des fichiers ouverts.

Les fichiers en gras sont ceux ouverts mais dont l'onglet est caché (plus de place dans la fenêtre). Vous pouvez filtrer les résultats en tapant le début du nom d'un fichier.
- **Ctrl + Shift + R**: Permet de rechercher une ressource (quelque soit son type) présente dans votre workspace. Vous pouvez filtrer les noms avec des *. Par exemple « bouton*.jpg » vous listera toutes les images jpg dont le nom commence par « bouton ».
- **Ctrl + Shift + T**: Même principe que ci dessous mais cette fois uniquement pour les classes Java présentent dans un projet Java.

■ Raccourcis utiles

• Les très utiles mais qui ne servent pas tous les jours

- **Ctrl + T**: En fonction de la position de votre curseur, Ctrl + T va lister différentes choses:
 - Curseur sur le nom d'une classe: Ctrl + T liste la hiérarchie de la classe
 - Curseur sur le nom d'une méthode: Ctrl + T liste les classes implémentant la méthode
- **Ctrl + Alt + H**: A partir de la position du curseur, ouvre un onglet qui présente tous les endroits où est appelé l'élément dans le workspace. Par exemple, si votre curseur est positionné sur le nom d'une méthode, tous les endroits où cette méthode est appelé seront listés.
- **Ctrl + H**: Ouvre la popup de recherche complexe. Dans cette popup, vous pourrez effectuer des recherches simples dans tous les fichiers du workspace ou des recherches sur des serveurs distants ou d'autres type de recherche.
- **Ctrl + Shift + F**: Reformate l'ensemble du fichier ou seulement la zone sélectionnée.
- **Ctrl + Shift + O**: Réorganise les imports d'une classe. Supprime ceux non nécessaire et insère ceux non déclaré mais qu'Eclipse peut reconnaître.

■ Raccourcis utiles

- Les très utiles mais qui ne servent pas tous les jours
 - **Ctrl + Alt + R**: En fonction de la position de votre curseur, Ctrl + Alt + R va renommer différentes choses:
 - Curseur sur le nom d'une méthode de la classe: Renomme la méthode et impact le code de la classe
 - Curseur sur le nom d'une variable d'une méthode: Renomme la variable et impact le code de la méthode

Liens utiles :

<https://thierry-leriche-dessirier.developpez.com/tutoriels/eclipse/raccourcis/>

Empaquetage et déploiement d'une application Java



- Il existe différents type d'empaquetage selon l'application créée.
 - Empaqueter des programmes dans des **fichiers JAR**
 - Un fichier JAR (Java archive) est un fichier ZIP utilisé pour distribuer un ensemble de classes Java. Ce format est utilisé pour stocker les définitions des classes, ainsi que des métadonnées, constituant l'ensemble d'un programme.
 - Les fichiers JAR sont créés et extraits à l'aide de la commande jar incluse dans le JDK.
 - Un fichier JAR peut contenir un **fichier manifest** (en), situé dans le chemin **META-INF/MANIFEST.MF**. Les données du fichier manifest spécifient comment le fichier JAR sera utilisé. Les fichiers JAR sont destinés à être exécutés comme des programmes indépendants, dont une des classes est la classe principale. Le fichier manifest peut comporter la déclaration suivante :

Main-Class: fr.dawan.HelloWorld

Liens utile :

<https://docs.oracle.com/javase/tutorial/deployment/jar/index.html>

<https://docs.oracle.com/javase/tutorial/deployment/jar/build.html>

<https://www.baeldung.com/java-create-jar>

Empaquetage et déploiement d'une application Java



Utilisation des lignes de commandes (fichier JAR) :

- Les différents étapes pour empaqueter et déployer notre application HelloWorld
 - `javac HelloWorld.java` crée le bytecode HelloWorld.class
 - Pour créer le fichier jar, nous allons utiliser la commande `jar cf HelloWorld.jar HelloWorld.class`
 - Il est utile que le manifeste du fichier jar inclue la classe principale.
`jar cfe HelloWorld.jar HelloWorld HelloWorld.class`
 - Nous avons apporté une modification à notre classe et nous l'avons recompilée.
 - Mise à jour de notre fichier jar
`jar uf HelloWorld.jar HelloWorld.class`
 - Pour exécuter le fichier JAR, il faut entrer la ligne de commande suivante
`Java -jar HelloWorld.jar`

Liens utile :

<https://docs.oracle.com/javase/tutorial/deployment/jar/index.html>

<https://docs.oracle.com/javase/tutorial/deployment/jar/build.html>

<https://www.baeldung.com/java-create-jar>

Empaquetage et déploiement d'une application Java



Avec Eclipse(fichier JAR) :

- Export le projet en tant que JAR exécutable (**Runnable Jar**)
 - Clic-droit sur le projet, et sélectionnez **Export**
 - Choisissez **Java > Runnable Jar file > Next**
 - Sur la page suivante, spécifiez le nom et le chemin du fichier JAR à créer et sélectionnez la **configuration de lancement (Launch Configuration)** qui inclut le nom du projet et le nom de la classe HelloWorld
 - Assurez-vous également de sélectionner **Package required libraries into generated JAR** pour intégrer d'autres librairies du projet dans votre fichier JAR.
 - Cliquez sur Finish pour créer le fichier JAR
- Pour confirmer que le fichier JAR a été correctement empaqueté, ouvrez l'invite de commande et le terminal d'Eclipse et exécutez cette commande :
 - `Java -jar <chemin>\filename.jar`
 - `java -jar HelloWorld.jar`

Liens utile : <https://support.smartbear.com/alertsite/docs/monitors/web/selenium/export-eclipse-java-project-as-runnable-jar.html>

Empaquetage et déploiement d'une application Java



Formats connexes:

- Les fichiers [WAR](#) (**W**eb **A**pplication **A**rchive) sont des archives JAR dont l'extension a été changée en [.war](#). Il contient des fichiers XML, des [classes Java](#), des [JSP](#) et d'autres objets pour les applications Web ;
- Les fichiers [RAR](#) (**R**esource **A**dapter **A**rchive) sont des archives JAR contenant des fichiers XML, des [classes Java](#) et d'autres objets pour l'architecture de connecteurs ([JCA](#)) de la plateforme [J2EE](#) ;
- Les fichiers [EAR](#) (**E**nterprise **A**pplication **A**rchive) sont des archives JAR dont l'extension a été changée en [.ear](#). Il contient des fichiers XML, des [classes Java](#) et d'autres objets pour les applications d'entreprise ;
 - Différents éléments peuvent être contenus dans un fichier EAR, pour être déployés sur le serveur :
 - Une archive d'application web, avec une extension [.war](#).
 - Des classes Java, groupées dans des archives [.jar](#).
 - Un module Enterprise JavaBean dans une archive [.jar](#), qui contient dans son propre répertoire META-INF son descripteur de déploiement spécifique. Une fois déployés, ces beans sont visibles aux autres composants.

Utiliser les fonctions de base de Git dans Eclipse

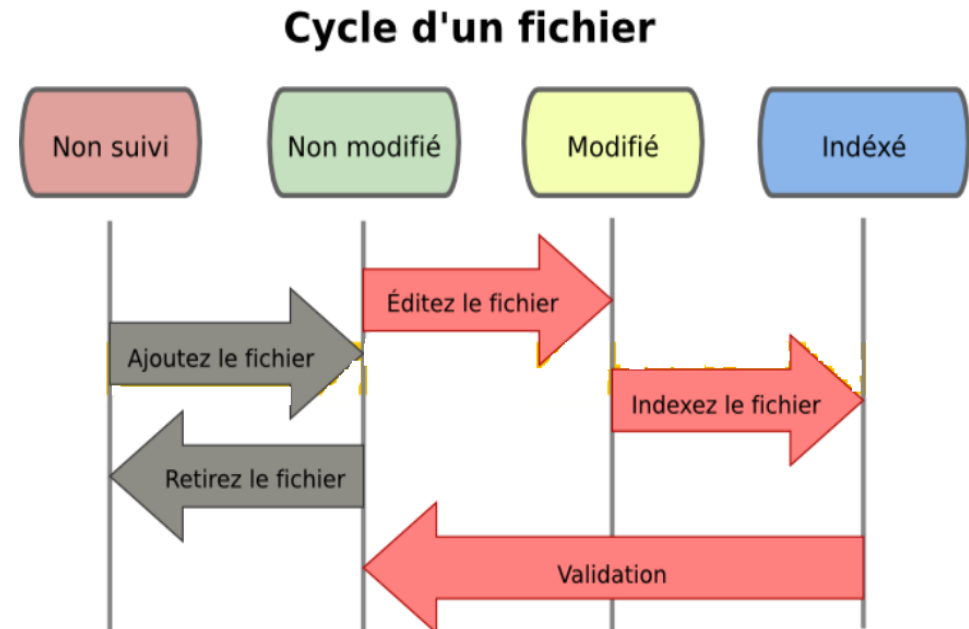
<https://www.atlassian.com/fr/git/tutorials/learn-git-with-bitbucket-cloud>

Utiliser les fonctions de base de Git dans Eclipse

Les commandes de base de Git

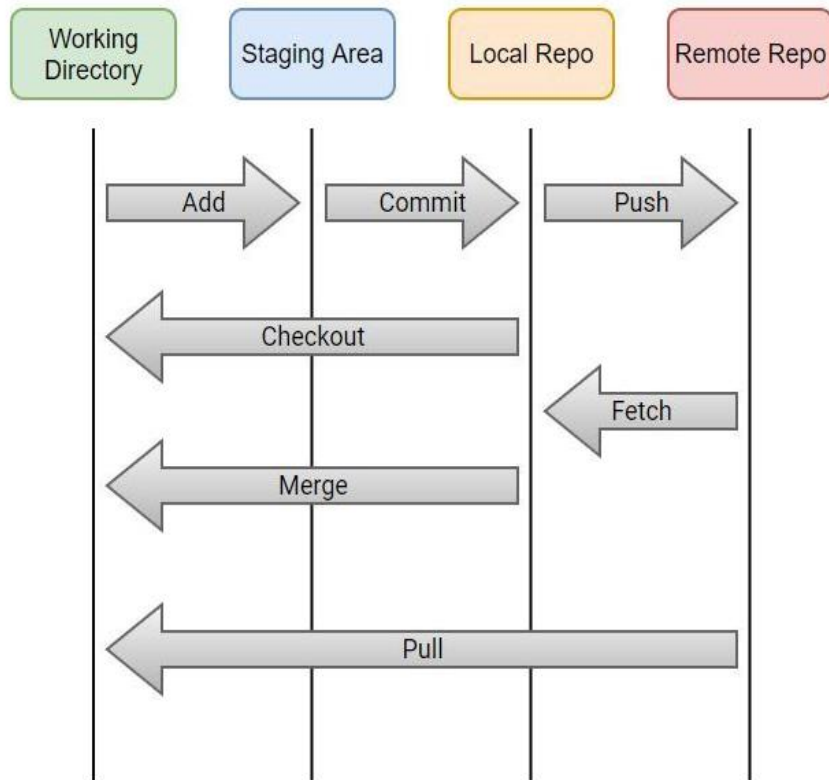
Quatre états d'un projet Git:

- ❖ **Non suivi:** fichier n'étant (n'appartenant) pas ou plus géré par Git;
- ❖ **Non modifié:** fichier sauvegardé de manière sûre dans sa version courante dans la base de données du dépôt;
- ❖ **Modifié:** fichier ayant subi des modifications depuis la dernière fois qu'il a été soumis;
- ❖ **Indexé:** idem pour modifié, sauf qu'il sera pris instantané dans sa version courante de la prochaine soumission (commit)



Utiliser les fonctions de base de Git dans Eclipse

Les commandes de base de Git



git init : Cette commande est utilisée pour créer un nouveau dépôt GIT

Git add . :

La commande **git add** peut être utilisée pour ajouter des fichiers à l'index (Staging Area : zone de transit).

git add addProduct.java

Par exemple, la commande suivante ajoutera un fichier nommé *add addProduct.java* dans l'index

Git commit :

La commande **git commit** permet de valider les modifications.

git commit -m "Ajout de la classe product"

Avant le JDK 5.0, la programmation équivalente à la programmation générique s'obtenait à chaque fois par le mécanisme d'héritage.

Ainsi, la classe ArrayList conservait simplement un tableau de références Object (la classe Object est l'ancêtre de toutes les autres classes) :

```
public class ArrayList { // Avant le JDK 5.0
    public Object get(int indice) { ... }
    public void add(Object élément) { ... }
    ...
    private Object[] tableauElémentsStockés;
}
```

- Cette approche présente deux problèmes. D'une part, il faut avoir recours au transtypage lorsque vous récupérez une valeur :

```
ArrayList fichier = new ArrayList();  
...  
String nomFichier = (String) fichier.get(0);
```

- D'autre part, il n'existe aucune procédure de vérification des erreurs. Vous pouvez ajouter des valeurs de n'importe quelle classe :

```
fichier.add(new File("..."));
```

Depuis la version 5.0, Java autorise la définition de classes et d'interfaces contenant un (des) paramètre(s) représentant un (des) type(s)

Cela permet d'écrire une structure qui pourra être personnalisée au moment de l'instanciation à tout type d'objet

Motivation : Homogénéité garantie

Inférence de type :

- principe d'erasure à la compilation
- pas de duplication de code

Aucune incidence sur la JVM : les casts restent en interne mais deviennent sûrs (sans levée d'exceptions)

```
ArrayList<String> fichier = new ArrayList<String>();
```

```
String nomFichier = fichier.get(0);
```

```
fichier.add(new File("...")); // ne peut ajouter que des objets String à un ArrayList<String>
```

Classe Générique

```
class name<T1, T2, ... , Tn> { /* ... */ }
```

```
public class Box<T> {  
    private T t;  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}  
  
public static void main(String[] args){  
    Box<Integer> intBox = new Box<>();  
    intBox.set(42);  
    Box<String> strBox = new Box<>() ;  
    strBox.set("Test");  
}
```

Conventions de nommage des paramètres de type

E : Element

K : Key

N : Number

T : Type

V : Value

S,U,V ... : 2^{ème}, 3^{ème}, 4^{ème} type

Méthode Générique

Les méthodes génériques sont des méthodes qui introduisent leurs propres paramètres de type

Les méthodes d'instance, de classe et les constructeurs peuvent être des méthodes génériques

```
public class GenMethod {  
  
    public static <T> boolean isEqual(T v1, T v2){  
        return v1.equals(v2);  
    }  
  
    public static void main(String args[]){  
        boolean t1=GenMethod.<String>isEqual("test","hello");  
        boolean t2=GenMethod.<Integer>isEqual(42,30+12);  
    }  
}
```

Contraintes sur les types génériques

Les génériques permettent d'imposer qu'un type T étend un autre (classe ou interface)

```
public class MyClass<T extends Comparable>{...}
```

- Contraintes multiples sur un type générique → &

```
public class MyClass<T extends Comparable & Cloneable>{...}
```

La liste des types

- ne peut comporter qu'une unique classe, qui doit être déclarée en premier
- peut comporter plusieurs interfaces

Implémentation des génériques

Construction d'une instance

On ne peut pas construire une nouvelle instance d'un type T avec : **new** T() ou T.**class**.newInstance()

Il faut utiliser :

```
public static <T> T newInstance(Class<T> clazz) {  
    return clazz.newInstance();  
}
```

Membres statiques

On ne peut pas référencer un type générique déclaré au niveau de la classe dans des membres statiques

Type <?>

- Une classe générique ne peut étendre aucune version d'elle-même
- List<T> n'étend jamais List<U>, quelle que soit la relation entre T et U

```
List<String> ls = new ArrayList<String>();  
List<Object> lo = ls; // Ne compile pas
```

- Pour résoudre ce problème, on peut utiliser ? (wildcard)
- List<?> correspond à une liste de type inconnue

```
List<String> ls = new ArrayList<String>();  
List<?> lo = ls;
```

Pour éviter d'ajouter d'autre éléments que des String dans la List<?>, On interdit l'utilisation des méthodes qui prennent ? en paramètre

Contraintes sur les type<?>

- **Type ? extension d'un type**
- On peut imposer que le type ? en étende un autre

```
List<Integer> lstInt = Arrays.asList(5, 2, 10) ;  
List<? extends Number> lstNum = lstInt ;  
//lstNum.add((Integer)2) ; // Erreur ne compile pas  
int val = (int) lstNum.get(1) ; // 2
```

- **Type ? super-type d'un type**
- On peut imposer d'être le super-type d'un type donné, avec
? **super** T (tous les types dont le type T est un type dérivé)

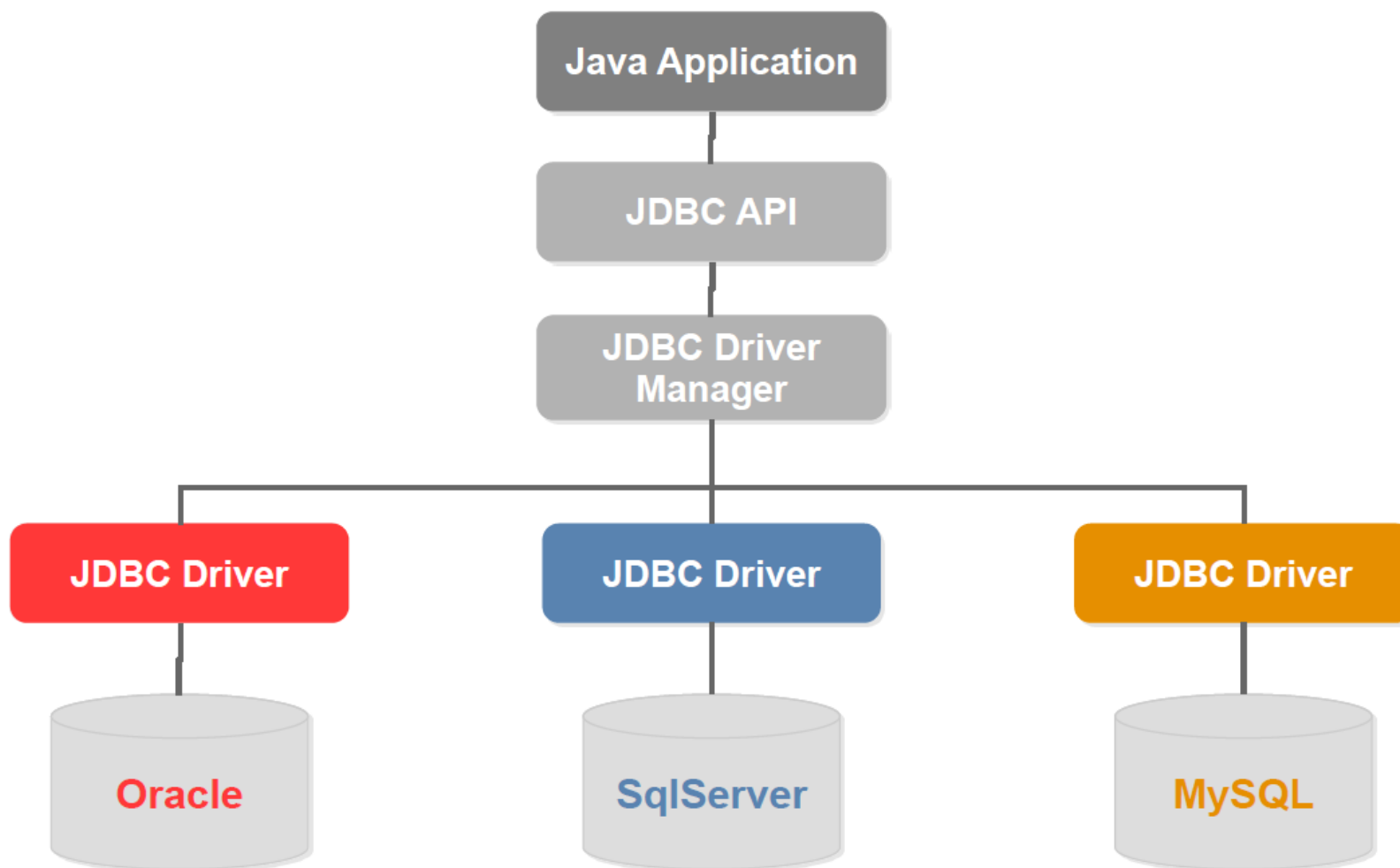
```
List<Integer> lstInt = Arrays.asList(1, 2, 3) ;  
List<? super Integer> lstNum = lstInt ;  
lstNum.add(2) ; //ok  
// Compile, mais n'est pas sûr(le type de retour peut être Object)  
int i = (Integer)lstNum.get(0);
```

JDBC

- JDBC (**J**ava **D**ata **B**ase **C**onnectivity) est l'API Java pour accéder à des bases de données relationnelles avec le langage SQL
- Les interfaces et les classes se trouvent dans le paquetage **java.sql**
- JDBC ne fournit pas les classes qui implémentent les interfaces
- C'est le **driver** JDBC qui implémente ces interfaces
- Les drivers dépendent du system de gestion de base de données auquel ils permettent d'accéder
- Ils sont fournis par les éditeurs des SGBD ([mysql](#), [mariadb](#), [oracle database](#), [postgresql](#), [SQL server](#) ...)

Accéder à des bases de données

JDBC



Principe d'accès à une base de données avec JDBC

1. Chargement du pilote de la base de données
2. Ouverture d'une connexion (**Connection**) à la base de données
3. Création d'un objet (**Statement** ou **PreparedStatement**) qui va permettre d'exécuter la requête SQL sur la connexion
4. S'il y en a, récupération du résultat de la requête dans un objet (**ResultSet**), et exploitation de ce résultat.
5. Fermeture de la connexion

Chargement du pilote de la base de donnée

Le driver d'une base de données est une classe Java qui implémente de l'interface **java.sql.Driver**

Il faut ajouter le chemin de la classe du driver dans le classpath (option -classpath de la commande java)

```
java -classpath CheminDuDriver.jar
```

Avant JDBC 4.0

Il faut charger dynamiquement la classe du driver par appel de la méthode `Class.forName("nom classe");`

Le nom de la classe varie suivant la base de données

Chargement du pilote de la base de donnée

Le driver d'une base de données est une classe Java qui implémente de l'interface **java.sql.Driver**

Il faut ajouter le chemin de la classe du driver dans le classpath (option -classpath de la commande java)

```
java -classpath CheminDuDriver.jar
```

Avant JDBC 4.0

Il faut charger dynamiquement la classe du driver par appel de la méthode `Class.forName("nom classe");`

Le nom de la classe varie suivant la base de données

```
Class.forName("com.mysql.jdbc.Driver");
```

Chargement du pilote de la base de donnée

MySQL	<code>com.mysql.jdbc.Driver</code>
MariaDb	<code>org.mariadb.jdbc.Driver</code>
PostgreSQL	<code>org.postgresql.Driver</code>
Oracle	<code>oracle.jdbc.driver.OracleDriver</code>

À partir de JDBC 4.0 (Java 6)

Lors du chargement d'un JAR, Java examine le contenu du répertoire META-INF du fichier

S'il y trouve un fichier `services/java.sql.Driver`, alors il charge les classes définies dans ce fichier et les enregistre en tant que pilotes JDBC

Il n'est plus nécessaire de charger le driver avec `Class.forName()`

Connection

- La méthode **getConnection** de **DriverManager** permet de créer la Connexion
 - **getConnection**(String url)
 - **getConnection**(String url, Properties prop)
 - **getConnection**(String url, String user, String password)
- L'url est composée de 3 parties séparée par :
 - Protocole
 - nom du SGDB
 - détails de connexion (spécifiques à la base de données)
 - pour MySql → //url du serveur:port/nom de la base

Statement

- **Création**

Un objet de type **Statement** s'obtient en appelant la méthode **createStatement()** de l'interface **Connection**

```
Statement smt = connection.createStatement();
```

- **Exécution**

- **executeQuery(String)**
retourne un objet de type **ResultSet** (SELECT

```
ResultSet rs = smt.executeQuery("SELECT nom, prix FROM articles");
```

- **executeUpdate(String)**
retourne un nombre d'objets modifiés (INSERT, DELETE, UPDATE)

```
int count = smt.executeUpdate("DELETE FROM articles");
```

PreparedStatement

- L'interface **PreparedStatement** étend **Statement**
- Il ajoute la possibilité de paramétrer des requêtes SQL
- Les instances de **PreparedStatement** s'utilisent quand une même requête doit être exécutée plusieurs fois, avec des paramètres différents

```
PreparedStatement ps = connection.prepareStatement("INSERT  
INTO Articles (nom, prix) values (?, ?)");
```

- ? → paramètres, dont la valeur est donnée avec les méthodes setType(int numParam, Type valeur)
- L'indexation des paramètres commence à 1

```
ps.setString(1, "batterie") ;  
ps.setDouble(2, 76.0) ;
```

Clés générées

- Pour récupérer la valeur de la clé primaire générée par la base de donnée:
 - On ajoute en paramètre à executeUpdate après la requête :
`Statement.RETURN_GENERATED_KEYS`
 - Après l'exécution de la requête, on récupère la clé primaire générée avec la méthode `getGeneratedKeys()`

```
Statement stmt = connection.createStatement() ;
int rowCount = smt.executeUpdate("INSERT INTO articles (nom, prix) VALUES ('Stylo', 1.7)",Statement.RETURN_GENERATED_KEYS);

ResultSet rs = stmt.getGeneratedKeys();
if (rs.next()) {
    // récupération de la clé primaire
    int idStylo = rs.getInt(1) ;
}
```

JDBC Exemple

```
try {  
    Class.forName("com.mysql.jdbc.Driver" );  
    cnx = DriverManager.getConnection("jdbc:mysql://localhost/test","root", "");  
    Statement st = cnx.createStatement();  
    ResultSet rs = st.executeQuery("SELECT * FROM articles");  
    while(rs.next()) {  
        // ...  
        rs.getString(1)  
        // ...  
    }  
    cnx.close();  
}catch( ) {  
    // ...  
}
```

Transaction

Une transaction est un ensemble d'une ou plusieurs requêtes exécutées en tant qu'unité, de sorte que soit toutes les requêtes sont exécutées, soit aucune des requêtes n'est exécutée

Désactivation du mode auto-commit

- Par défaut une connexion est en mode auto-commit, Chaque instruction SQL est automatiquement validée juste après son exécution
- Pour grouper plusieurs requêtes dans une transaction il faut désactiver le mode auto-commit

```
cnx.setAutoCommit (false);
```

Transaction (commit et rollback)

- La méthode **commit()** de Connexion permet de valider les requêtes
- Sinon la méthode **rollback()** va permettre d'annuler les requêtes

```
try{
    cnx.setAutoCommit(false);
    Statement stmt = cnx.createStatement();
    String SQL = "INSERT INTO Employees VALUES (2000, 'John', 'Doe')";
    stmt.executeUpdate(SQL);
    // ... autre requête
    cnx.commit();
} catch(SQLException se){
    cnx.rollback();
}
```

DAO

- Le pattern **DAO** (Data Access Object) permet d'isoler la couche métier de la couche de persistance
 - Permet de centraliser les requête SQL dans un seul objet
 - Permet de changer facilement de système de stockage de données (Bdd, XML ...)
- Avec l'objet DAO, on va réaliser les opérations CRUD
 - Créer l'objet en base (INSERT)
 - Rechercher l'objet en base pour le recréer (FIND, RETRIEVE)
 - Mettre à jour l'objet en base (UPDATE)
 - Supprimer l'objet en base (DELETE)
- On aura un DAO par objet Métier (Entité: Java Bean)

Object Relational Mapping

Concept permettant de connecter un modèle objet à un modèle relationnel

Couche qui va interagir entre l'application et la base de données

Avantages

- Gain de temps au niveau du développement d'une application
- Abstraction de toute la partie SQL
- La portabilité de l'application d'un point de vue SGBD

Inconvénients

- L'optimisation des frameworks/outils proposés
- La difficulté à maîtriser les frameworks/outils

ORM: JPA

- .Une API (Java Persistence API)
- .Des implémentations



- .Permet de définir le mapping entre des objets Java et des tables en base de données
- . Remplace les appels à la base de données via JDBC

ORM: Exemple Entité

```
@Entity
@Table(name="products")
public class Product {
    @Id

    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String name;

    private double price;

    @ManyToMany(cascade=CascadeType.ALL)
    @JoinTable(name="products_suppliers")
    private Set<Supplier> suppliers;

    ...
}
```

ORM: Exemple Entité

```
@Entity
@Table(name="suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @Column(nullable=false, unique=true, length=200)
    private String name;
    @ManyToMany(mappedBy="suppliers",
cascade=CascadeType.ALL)
    private Set<Product> products;
    ...
}
```