INITIATION JAVA

Historique

- Java est un langage de programmation orienté objet
- Créé par James Gosling et Patrick Naughton, employés de Sun Microsystems
- 1ère version sorti en 1995
- Langage compilés vers une représentation binaire intermédiaire qui peut être exécutée dans une machine virtuelle Java (JVM) en faisant abstraction du système d'exploitation.

1

ENVIRONNEMENT DE TRAVAIL

Outils nécessaires

- **Java Development Kit (JDK) :** Ensemble de librairie de base pour la programmation Java
  - https://www.oracle.com/java/technologies/javase-downloads.html
  - Test java ? java -version
  - Ajouter la variable d'environnement :

    C:\Program Files\Java\jdk-xxxx\bin

- **Eclipse (Version Oxygen) :** IDE JAVA gratuit
  - https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-developers

2

LES BASES DU JAVA

Comment fonctionne Java ?

- Pour créer un fichier en Java, ajouter l'extension **.java**
- Pour compiler un fichier, exécuter la commande: **javac <*class_name*>.java**
- Pour exécuter un fichier compilé, exécuter la commande: **java <class_name>**
- La méthode **main()** est le point d'entrée d'un programme Java :

```
public static void main(String[] args)
```

- La méthode **System.out.println()** permet d'afficher des lignes de texte

```
System.out.println("message");
```

- Description par un exemple !

Informations de bases

- Règle de base : Une instruction se termine toujours pas un **point virgule** !
- Les commentaires : `// mon commentaire` `/* mon commentaire sur plusieurs lignes */`
- Instanciation d'une variable : **type nom;** / **type nom = valeur;**
  - Chaine de caractère : `String test = "Test";`
  - Nombre: `int test = 6;` / `int x = 5, y = 8, z = 22;`
  - Nombre flottant: `float test = 6.22f;` **Toujours terminer par le f !**
  - Booléen: `boolean test = true;`
- Et plus…
  - Concaténation : `String test1 = "Je "; String test2 = "suis"; String result = test1 + test2;` **result = "Je suis"**
  - Casting (**parfois automatique !**) : `float x = 3.1f; int y = (int) x;` **y = 3**
  - Transformer une String en Int : `String x = "50 "; int y = Integer.parseInt(x);` **y = 50**
  - Transformer une String en float : `String x = "50 "; float y = Float.parseFloat(x);` **y = 50.0**

Exercices

EXERCICE 1 :

- Concevoir un programme qui crée 3 variables:
  - firstname
  - name
  - age
- Affecter les valeurs respectives “Jean”, “Dupont” et 25
- Le programme doit afficher dans le terminal le message “Bonjour Jean Dupont, vous avez 25 ans” en s’appuyant sur les variables.

Les opérateurs mathématiques

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

- Opérations d'affectation : int a = 3; / a += 2; □ **a = 5**

  += / -= / *= **…**

Exercices

EXERCICE 2 :

1. Concevoir un programme qui crée 3 variables et qui affecte les valeurs **15**, **8** et **9,5.**
   Afficher dans le terminal la moyenne des 3 notes.

2. Garder les mêmes valeurs, mais cette fois-ci calculer la moyenne en arrondissant à l'entier supérieur. Afficher à l'écran les 2 résultats (initial + arrondi)

Les recuperations clavier

- La classe **Scanner** permet de récupérer les textes tapés au clavier sous forme de **String**

```
import java.util.Scanner;
```

- Pour récupérer du texte saisie, il faut :

1. Créer l'objet Scanner :

```
Scanner sc = new Scanner(System.in);
```

2. Sauvegarder l'information dans une variable :

```
System.out.println("Veuillez saisir un mot :");
String str = sc.nextLine();
```

Les tableaux

- Les tableaux sont utilisés pour stocker des valeurs dans une variable.
- Méthodes liées au tableau :
  - Déclarer un tableau : `String[ ] tab;` / `String[ ] tab = {"val1", "val2", "valN"};` / `int[ ] tab = {0, 1, 5};`
  - Accéder aux éléments: `tab[index];`
  - Modifier un élément: `tab[0] = 25;`
  - Connaître la taille: `tab.length;`
  - Tableau à plusieurs dimensions: `int[ ][ ] tab = { {0, 1, 5} , {2, 0, 6} };`
- Parcourir un tableau (exemple) :

```
int[] tab = {1, 5, 8, 455};
for (int i: tab) {
  System.out.println(i);
}
```

EXERCICE 3 :

1. Reprendre l’exo 1 :
   - Afficher un message “Quel est votre nom ?”.
   - Récupérer la valeur saisie.
   - Faire pareil pour le prénom et l'âge.
   - Afficher le message initial avec nom, prénom et âge saisi dans le terminal.

Les opérateurs logiques

- Opérateurs de comparaisons

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

- Opérateurs logiques

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

La comparaison d'objet

- Exception pour la comparaison d'objet !
- Pour comparer 2 objets, il faut utiliser la syntaxe suivante :

```
obj1.equals(objet2)
```

- Attention : Les variables de types String sont des objets !!

La condition IF

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Les conditions WHILE/FOR

```
while (condition) {
  // code block to be executed
}
```

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

Exercices

EXERCICE 4 :

- Créer un programme qui calcul la moyenne des notes des élèves en prenant en compte les spécificités suivantes :
  - Les notes sont saisies au clavier par un opérateur.
  - L'opérateur peut saisir autant de note qu'il souhaite mais au minimum 1 note doit être saisie.
  - Les notes peuvent avoir des décimales.
  - La moyenne doit être affichée seulement à la fin du programme.

EXERCICE 5 :

- Ecrire un programme qui lit un nombre saisi au clavier et qui indique si le nombre est positif, négatif ou s'il vaut zéro **ET** s'il est pair ou impair.

Exercice Bonus

Exo Bonus 1 :

- Écrire un programme permettant de faire la somme des N premiers entiers. On initialise un nombre entier N, et renvoie un autre nombre entier, la somme demandée.

Exo Bonus 2.1 :

- Écrire un programme qui permet de connaître le minimum et le maximum d'un tableau d'entiers.

int[] tab = {8, 0, 9, 1, 14, 5, 17, 2, 7};

Exo Bonus 2.2 :

- Écrire un programme qui permet de connaître le minimum et le maximum d'un tableau d'entiers à 2 dimensions.

int[][] tab = {{8, 9, 1, 14, 5, 17, 2, 99}, {25, 0, 30, 3, 6, 4, 19, 7}};

Exercice Bonus

Exo Bonus 4 :

- Écrire une fonction nommé moyenne permettant de renvoyer la moyenne d'un tableaux de nombre décimal passé en paramètres

Afficher en dehors de la fonction (dans le main) : "La moyenne est X"

TP - Jeu du Plus ou Moins

- Créer un programme avec une variable randomNum qui est instancié avec une valeur aléatoire entre MIN et MAX, le but du jeu étant de trouver le numéro généré aléatoirement :
    - Au début de la partie, le programme doit demander à l'utilisateur le nombre MIN et MAX
        - ✔ *Si MIN >= MAX, le programme doit indiquer une erreur et redemander l'intervalle*
    - Tant que le numéro n'est pas trouvé, le programme doit poser la question "Quel est le numéro mystère ?"
    - Si le nombre renseigné est inférieur au numéro mystère, indiquez "C'est plus"
    - Si le nombre renseigné est supérieur au numéro mystère, indiquez "C'est moins"
    - Si le nombre renseigné est égal au numéro mystère, indiquez " Félicitation vous avez trouvé !"
    - A la fin du jeu, le programme devra proposé de rejouer une partie

- Pour générer un nombre aléatoire compris dans un intervalle, utilisé la fonction :

Math.random()

TP - Jeu du Plus ou Moins - Bonus

- Ajouter la possibilité de faire un replay de la partie
  - A la fin du jeu, le programme devra proposer de rejouer une partie ou de réafficher le déroulement de la partie et le nombre de coups utilisé pour gagner.

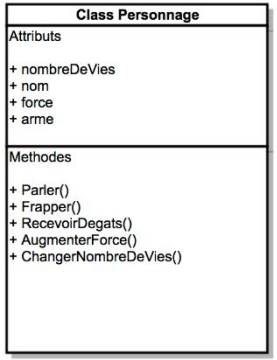
→ Ne pas utiliser de collections !

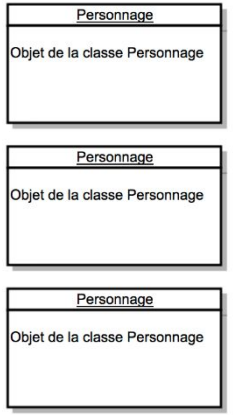
3

LES CLASSES

Qu’est ce qu’une classe ?

- Une classe est un “moule” qui permet de créer plusieurs objets à partir de ce moule.
- Un objet est un élément qui possède des caractéristiques (**propriétés**) et qui peut exécuter certaines fonctions (**méthodes**).
- Un objet est une **instance** de classe
- Exemple : Plusieurs personnages d’un jeux vidéo. On parle alors d’instances de la classe Personnage.





Comment créer une classe ?

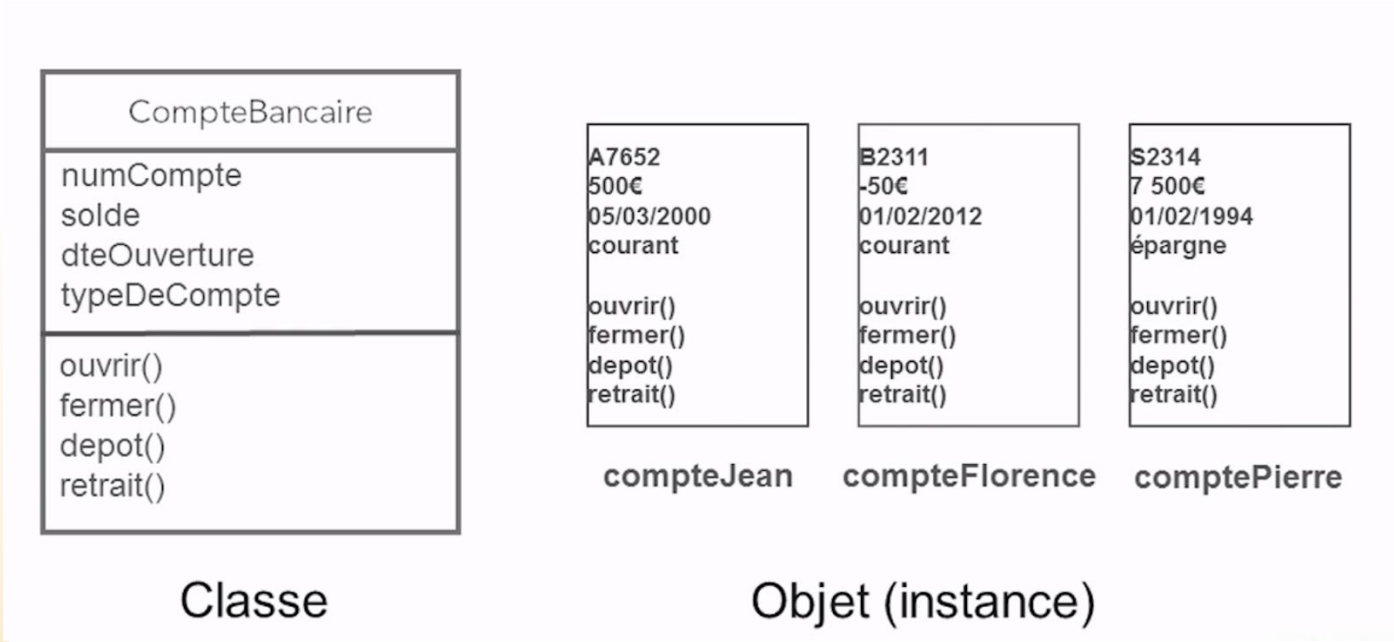
- Une classe est composé d'un **nom**, **d'attributs** et de **comportements**.

- Nom : c'est quoi ?
  - Employé, compte bancaire, joueur, document, album…

- Attributs : ce qui la décrit.
  - Largeur, hauteur, couleur, type de fichier, score…
  - On les appelle aussi des propriétés

- Comportements : que peut elle faire ?
  - Jouer, ouvrir, chercher, enregistrer, imprimer…
  - On les appelle le plus souvent des méthodes

Classe et objets



POO en Java

- Un programme Java est défini **sous forme de classes**
- Pas de code Java sans classe !
- Les instructions Java sont positionnées dans des **méthodes** et toutes les méthodes sont implémentées dans des **classes**
- Beaucoup de classe préconçues : **String, Date, Array …**

Visibilité

- Dans une classe en Java, il existe plusieurs type de protection (visibilités) pour une variable, méthode, …

  - **Public :** Accès partout dans le programme

```
public int attribute = 10;
```

  - **Protected :** Accès dans la classe, les classes qui en dérivent et les classes du même package

```
protected Point center = new Point();
```

  - **Non précisé (défault) :** Accès par les classes du même package

```
int attr3 = 30;
```

  - **Private :** Accès uniquement dans la classe actuelle

```
private static int privateAttribute = 10;
```

Getters et setters

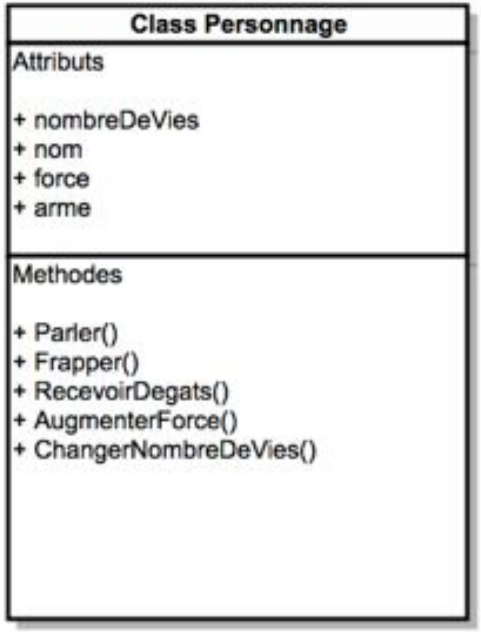
- Les **getters** et **setters** permettent respectivement de **récupérer** et **mettre à jour** une information (un attribut par exemple).
- La syntaxe d'un getter est la suivante :

  public <*type* > <*getAttributName* > () { return this.attributName }
- La syntaxe d'un setter est la suivante :

  public void<*setAttrName* > (<*type* > <*attrName* >) {this. attrName = attrName }

Exemple !

- Création de la classe Personnage + son constructeur + ses attributs + méthodes
- Implémentation de la méthode « Parler » avec un paramètre
- Création d'un personnage depuis une autre classe (main)
- Ajout des getters / setters
- Appel de la méthode « Parler »



Exercices

EXERCICE 6 :

1. Créer un programme qui permet de créer 3 personnages. Ces personnages doivent avoir les 4 attributs (String nom, int nombreDeVie, int force, String arme) définis dès leur instanciation.

2. Implémenter la méthode ‘Frapper()’ qui à pour but de retirer de la force au personnage attaqué.

3. Si sa force est inférieur ou égal à zéro, alors 1 vie doit lui être retiré et sa force est égale à 25.

4. Si son nombre de vie est égal à zéro, alors vous devez afficher un message : “Le personnage <name_perso> est mort”

5. Tester votre programme et vos méthodes !

Exercices

EXERCICE 6b :

- Reprendre le programme précédent et compléter :
  a. Créer une classe Arme avec :
     - *3 attributs: nom, degatsInfliges et niveau (niveau toujours égal à 1 à l'instanciation)*
     - *1 constructeur avec tous les attributs*
     - *1 méthode augmenterNiveau() qui permet d'augmenter de 5 les dégâts infligés*
  b. L'attribut 'arme' de la classe Personnage s'appuie sur cette nouvelle classe
  c. Dans le main, créer 2 armes et attribuez les aux personnages
  d. La méthode frapper() doit maintenant retirer les dégâts en fonction de la puissance de l'arme
  e. Testez

Exercices

EXERCICE 6c :

- Reprendre le programme précédent et compléter :
  a. Créer une classe Armure avec :
     - *3 attributs: nom, force et niveau (niveau toujours égal à 1 à l'instanciation)*
     - *1 constructeur avec tous les attributs*
     - *1 méthode augmenterNiveau() qui permet d'augmenter de 5 la force*
  b. L'attribut 'force' de la classe Personnage s'appuie sur cette nouvelle classe
  c. Dans le main, créer 2 armures et attribuez les aux personnages
  d. La méthode frapper() doit maintenant retirer les dégâts à la force de l'armure
  e. Testez

4

LES COLLECTIONS

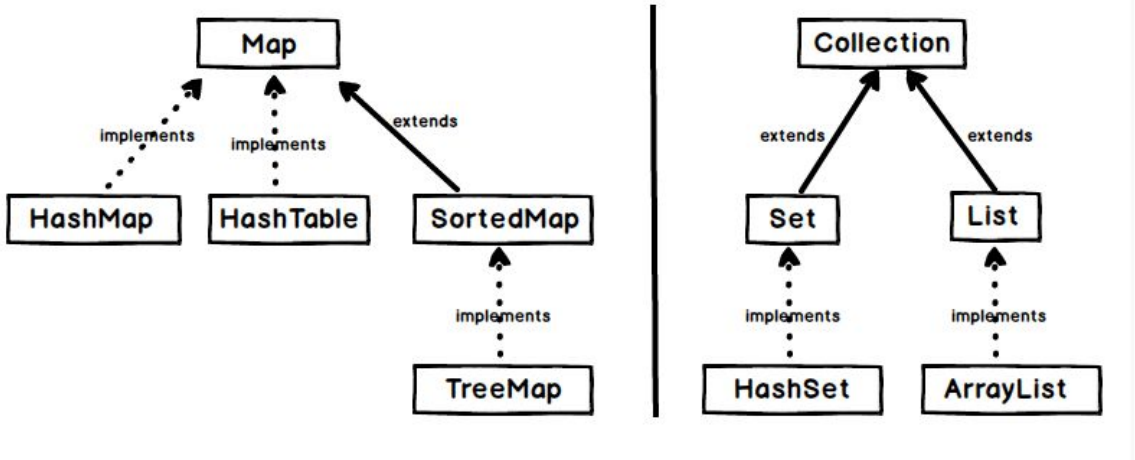
Les Collections

- Une collection en Java permet de gérer des **ensembles d'objet**.

- Il existe 4 types de collections :
  - List (ArrayList) : collection d'éléments ordonnées qui accepte les doublons
  - Set: collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons.
  - Map: collection qui fonctionne avec une paire clé/valeur.
  - Queue/Deque: collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement

Set et Map

- List et Set sont des interfaces qui héritent de la classe Collection
- Map est organisé et stocké sous la forme clé/valeur.



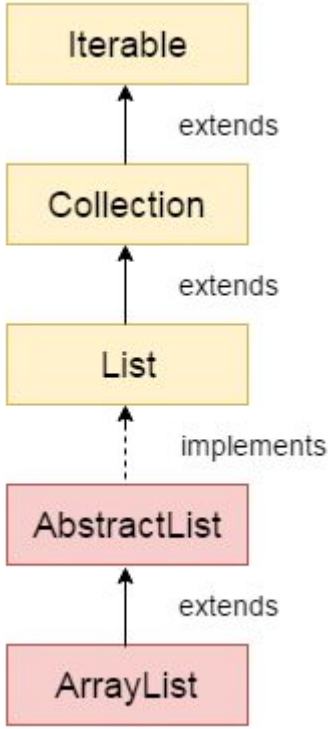
ArrayList et List

- La collection ArrayList correspond au schéma suivant :
- L'ArrayList est en réalité un **tableau dynamique.**
- L'ArrayList peut spécifier le **type d'élément** attendu.
- L'ArrayList doit être **instancié**.
- La syntaxe pour créer une List :

```
List<type> nom = new ArrayList<type>();
```

- Exemple :

```
List<String> list = new ArrayList<String>();
```



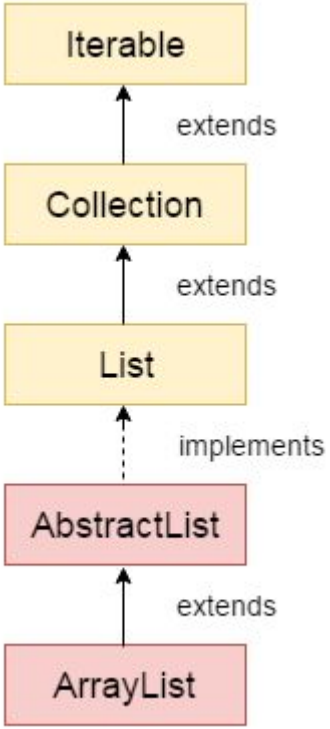
ArrayList – méthodes

- Les ArrayList admettent différentes méthodes :
  - Création : `List<Integer> list = new ArrayList<Integer>();`
  - Ajout élément : `list.add(15);`
  - Taille de la liste: `list.size()` / `list.isEmpty();`

- Les ArrayList permettent de manipuler les éléments :
  - Elément contenu: `list..contains(15);` → boolean
  - Connaitre position: `list.indexOf(15):` → return index
  - Récupérer valeur: `list.get(0);` → return 15
  - Suppression élément: `list.remove(15):` (1ère occ. supprimée !)

- Parcourir une liste avec un Iterator:

```
Iterator i = list.iterator();
while (i.hasNext() {System.out.println(i.next()); }
```



Exercices

EXERCICE 7 :

1. Créer un programme qui demande à l'utilisateur d'entrer des nombres jusqu'à ce que l'utilisateur tape 'q' (comme quitter).

2. A ce moment la, le programme doit :
   a. Afficher la liste des nombres.
   b. Afficher « Vous avez saisi <x> nombres. »
   c. Refuser les doublons à la liste.
   d. Si le nombre est déjà dans la liste: Affichée « Chiffre déjà ajouté à la saisie <x>. »

HashMap

- L'HashMap est en réalité un **tableau clé/valeur.**
- L'HashMap peut spécifier le **type d'élément** attendu.
- L'HashMap doit être **instancié**.
- La syntaxe pour créer une HashMap :

```
HashMap<type, type> nom = new HashMap<type, type>();
```

- Exemple :

```
HashMap<String, int> tab = new HashMap<String, int>();
```

HashMap - Méthodes

Méthode	Description
`void clear()`	Supprimer tous les mappages de cette map.
`boolean containsKey(Object key)`	Renvoyer true si cette map contient un mappage pour la clé spécifiée.
`boolean containsValue(Object value)`	Renvoyer true si cette map mappe une ou plusieurs clés sur la valeur spécifiée.
`Object get(Object key)`	Renvoyer la valeur à laquelle la clé spécifiée est mappée dans cette map, ou null si la carte ne contient aucune correspondance pour cette clé.
`Object put(Object key, Object value)`	Associer la valeur spécifiée à la clé spécifiée dans cette map.
`boolean isEmpty()`	Renvoyer true si cette map ne contient aucune correspondance clé-valeur.
`putAll(Map m)`	Copie tous les mappages de la map spécifiée vers cette map. Ces mappages remplaceront tous les mappages que cette map avait pour toutes les clés actuellement dans la map spécifiée.
`Object remove(Object key)`	Supprimer le mappage de cette clé de cette map si elle est présente..
`int size()`	Renvoyer le nombre de mappages clé-valeur dans cette map.
`Collection values()`	Renvoyer une vue des valeurs contenues dans cette map.

Exercices

EXERCICE 7.2 :

1. Créer un relevé de note qui demande à l'utilisateur d'entrer le nom de l'élève et la note qu'il a eu jusqu'à ce que l'utilisateur tape 'q' (comme quitter) en utilisant une HashMap.

2. A ce moment la, le programme doit :
   a. Afficher la liste des notes "nom élève/note".
   b. Afficher « Vous avez saisi <x> notes. »
   c. Refuser les doublons d'élève.
   d. Si l'élève est déjà dans la HashMap: Affichée « Élève déjà ajouté à la saisie <x>. »

Les enumérations

- Les classes de type **enum** permettent de définir une liste de String réutilisable.

```
public enum Name {
ENUM1, ENUM2, ENUM3
}
```

- L'utilisation d'un énum se fait sans instanciation:

```
Name.ENUM1;
```

5

L'HÉRITAGE

Héritage et interfaces

- **L'héritage** en Java permet de faire des classes réutilisables.
- Pour étendre les propriétés d'une classe, on utilise le mot clé extends
- Une interface est un **modèle de classe**.
- Pour utiliser une interface, on utilise le mot clé implements
- Le mot clé @Override permet de surcharger une méthode

Mots clés

- Certains mots sont **réservés** en Java (for, while, if ....)
- Le mot clé final permet de bloquer la surcharge de méthode
- Le mot clé abstact permet de rendre une **classe/méthode abstraite**
- Le mot clé static permet :
  - Devant une variable : utilisation sans création d'instance (ex. lib java.lang.Math)
  - Devant un bloc de code : Exécution une seule fois

Exemples

- Exemple d'implémentation de l'héritage et d’une interface

Exercices

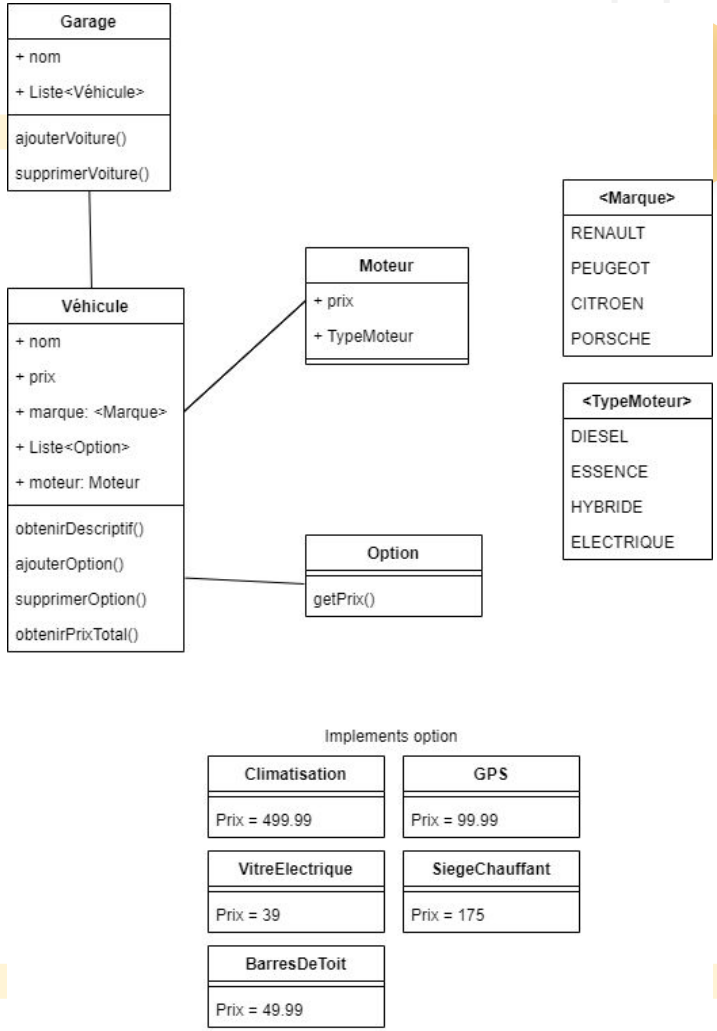
EXERCICE 10:

- Créer une classe véhicule avec les propriétés : nom et vitesseMax et leurs getters et setters respectifs
- Créer une interface actionVehicule avec les signatures de méthodes : peutVoler(), peutRouler(), peutNaviguer() renvoyant un boolean
- Créer une classe voiture et bateau qui étendent la classe véhicule et qui implémente chacun l'interface actionVehicule
- Overider les méthodes de l'interface actionVéhicule et la méthode setName de la classe étendu
- Dans un main, créer un véhicule de chaque type, est afficher son nom ainsi que les actions qu'il peut faire (voler, rouler, naviguer)

Exercices

EXERCICE 10b:

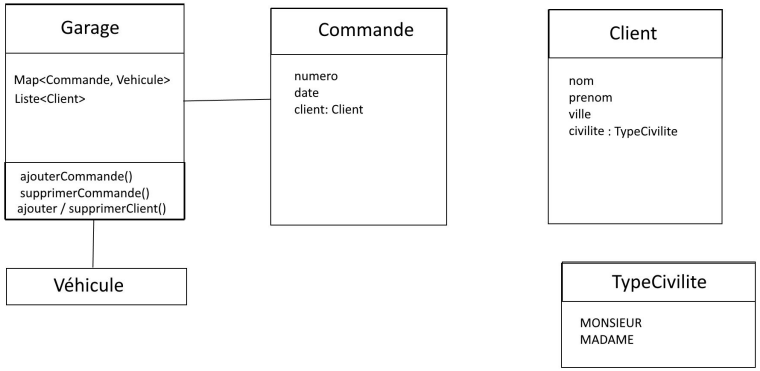
- Créer un programme qui réponds au schéma suivant :
- Créer un garage et plusieurs véhicules qui s'ajoutent au garage.
- Ajouter/retirer des options en actualisant le prix. Faire pareil avec les motorisations.
- Pensez à faire un main pour tout testez



Exercices

EXERCICE 10c - Bonus :

- Reprendre l'exercice précédent et ajouter les éléments suivants manquant à votre application :



- Ajouter un fichier Launcher.java (avec un main) et proposer une interface permettant d'interagir avec un utilisateur gérant un garage en vous appuyant sur toutes les fonctionnalitées développées

6

LES THREADS

Les Threads (1/2)

- Un Thread permet de gérer parallèlement **plusieurs objets** simultanément
- Les Threads permettent de faire des **exécutions simultanéments**
- Pour utiliser un Thread, il faut **étendre** la classe :

```
public class exemple extends Thread {
```

- Les Threads ont leur propres méthodes, pour la **surcharger** il faut utiliser le **décorateur** @Override

```
@Override
public void run(){
```

Les Threads (2/2)

- Pour **instancier** un Thread, il faut le déclarer comme n'importe quel :

```
MaClasseThread x = new MaClasseThread();
```

- Pour **exécuter** un Thread, il faut appeler la méthode start() qui correspond à la méthode run() surchargé :

```
x.start();
```

Les Runnables

- Une classe Java ne peut étendre (extend) qu'une seule classe
- L'implémentation d'un Thread bloque des éventuels extends.
- Il existe un seconde méthode pour implémenter un Thread : **Le Runnable**
- Un Runnable s'utilise par **implémentation**
- Un Runnable n'expose qu'une seule méthode: **run()**
- L'instanciation est différente :

```
RunnableCompteur x = new RunnableCompteur();
Thread t = new Thread(x);
```

Exemples

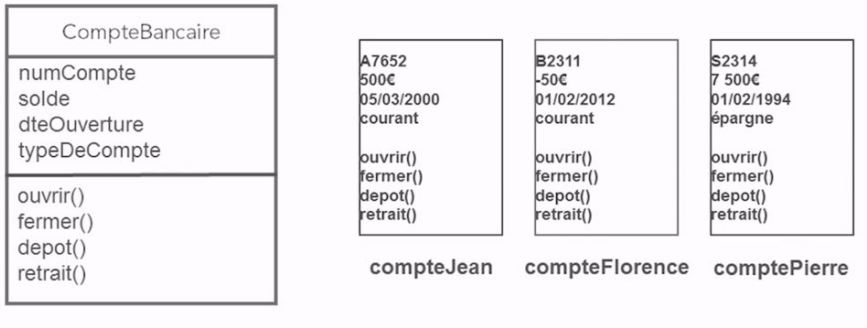
- Exemple d'implémentation d’un Thread avec un compteur :

1. Extend Thread
2. Implement Runnable

Exercices

EXERCICE 8 (1/2) :

- Créer un programme qui instancie 3 comptes bancaire dans un main. L'objet CompteBancaire correspond au schéma ci-dessous et s'appuie sur un Thread



- L'appel de la méthode start() doit afficher le message suivant :

```
Le compte A7652 a été crée le 05/03/2020 et a un solde initial de 500.0€. Il s'agit d'un compte courant.
```

Exercices

EXERCICE 8 (2/2) :

- Ajouter l'attribut *estOuvert* de type booléen, défini à False par défaut et qui devient True à la création d'un compte.
- Implémenter les 5 méthodes :
  - ouvrir() permet d'indiquer que le compte bancaire est ouvert
  - fermer() permet d'indiquer que le compte bancaire est fermé
  - depot() permet d'ajouter une somme définie au solde du compte
  - retrait() permet d'enlever une somme définie au solde du compte
  - virement() qui permet de transférer de l'argent vers un autre compte
- Les méthodes depot() et retrait() doivent afficher un message avec le nouveau solde.
- Ajouter une classe "Banque" respectant :
  - Une banque contient une liste de comptes bancaire. (ArrayList)
  - Une banque à une méthode "ajouterCompte" qui ajoute un compte à la liste.
- Dans le main, créer une banque et ajouter les 3 comptes bancaires.

Exercices

EXERCICE 9 :

- Reprendre le programme précédent, et ajouter :
  - Une interaction avec un utilisateur lui permettant de créer un compte en récupérant toutes les informations nécessaires.
  - Ajouter ce nouveau compte à la banque.
  - Laisser le choix à l'utilisateur de réaliser des opérations de dépôt, retrait, clôture, et virement (méthode à implémenter) vers un autre compte.

Utiliser l'interface Runnable à la place de l'objet Thread

7

GESTION DES EXCEPTIONS

Les exceptions

- Les exceptions représentent le mécanisme de **gestion des erreurs** intégré au langage Java.
- Elles se composent **d'objets** de représentation des erreurs (condition anormale).
- Il existe des **mots clés** pour traiter ces erreurs:
  - try: instructions où des exceptions peuvent être levées.
  - catch: traitement de l'exception (avec le nom de l'exception à traiter)
  - finally: instruction toujours exécutée.

Les exceptions - Exemple

```
1     try {
2       operation_risquée1;
3       opération_risquée2;
4     } catch (ExceptionInteressante e) {
5       traitements
6     } catch (ExceptionParticulière e) {
7       traitements
8     } catch (Exception e) {
9       traitements
10    } finally {
11      traitement_pour_terminer_proprement;
12    }
```

Exercices

EXERCICE 12:

- Reprenez l’exercice 7.1
- Faites en sorte de gérer les différentes erreur qui peuvent être causé à cause de l'utilisateur et de prévoir les cas ou l’entrée utilisateur devrait être refuser

Exemple : Si l’entrée utilisateur n’est ni ‘q’ ni un nombre, etc...

8

TESTS UNITAIRES EN JAVA

Définition et utilité d’un test unitaire

- En programmation informatique, le test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme.

- . On écrit un test pour confronter une réalisation à sa spécification.

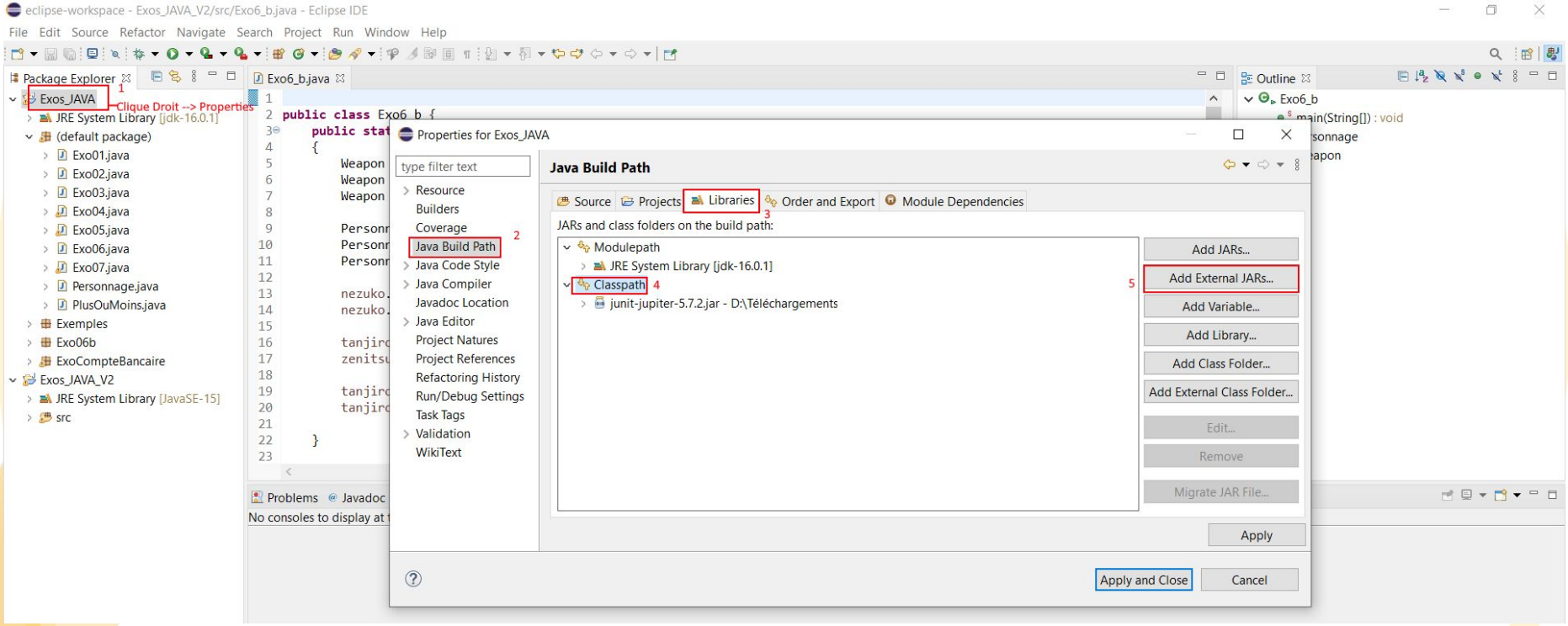
- Le test définit un critère d'arrêt (état ou sorties à l'issue de l'exécution) et permet de statuer sur le succès ou sur l'échec d'une vérification.

- Le test permet de vérifier que la relation d'entrée / sortie donnée par la spécification est bel et bien réalisée.

JUnit - Présentation

- JUnit est un framework open source pour le développement et l'exécution de tests unitaires automatisables.
- Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression.
- Pour tester les différentes fonctions, on utilise des **assertions**
- Docs : https://junit.org/junit5/
- Pour pouvoir utiliser JUnit, il faut ajouter le fichier junit.jar au classpath.

JUnit - Ajouter au classpath



JUnit - Exemple 1/2

L'exemple utilisé dans cette section est la classe suivante :

```java
public class MaClasse {

  public static int calculer(int a, int b) {
    int res = a + b;

    if (a == 0) {
      res = b * 2;
    }

    if (b == 0) {
      res = a * a;
    }
    return res;
  }
}
```

1 - Compiler la classe MaClasse

- **javac *MaClasse*.java**

JUnit - Exemple 2/2

2 - Ecrire une classe qui va contenir les différents tests à réaliser par JUnit

```
1  import junit.framework.*;
2
3  public class MaClasseTest extends TestCase {
4
5    public void testCalculer() throws Exception {
6
7      assertEquals(2,MaClasse.calculer(1,1));
8    }
9  }
```

3 - Enfin, appeler JUnit pour qu'il exécute la séquence de tests.

```
1  java -cp junit.jar;. junit.textui.TestRunner MaClasseTest
2  C:\java\testjunit>java -cp junit.jar;. junit.textui.TestRunner
3    MaClasseTest
4    .
5    Time: 0,01
6  OK (1 test)
```

Attention : le respect de la casse dans le nommage des méthodes de tests est très important. Les méthodes de tests doivent obligatoirement commencer par test en minuscule car JUnit utilise l'introspection pour déterminer les méthodes à exécuter.

Exercices

EXERCICE 11 :

- Reprenez l’exercice 5 et transformer le traitement en fonction

Rappel du sujet : Ecrire un programme qui lit un nombre saisi au clavier et qui indique si le nombre est positif, négatif ou s'il vaut zéro **ET** s'il est pair ou impair.

- Écrivez des tests pour les différents cas

- Exécutez les tests avec JUnit

9

BASE DE DONNÉES

JDBC ?

- Java DataBase Connectivity
  - API Java (Application Programming Interface)
  - Accès aux Bases de Données Relationnelles
  - Fonctionnalités :
    - Ouvrir une connexion avec un SGBD
    - Envoyer des requêtes SQL au SGBD
    - Récupérer des données résultantes des requêtes
    - Traiter ces données (tables)
    - Gérer les erreurs associées aux requêtes

- Similaire à l'API ODBC du langage C

Ressources associées au JDBC

- L'historique de JDBC
  - JDK 1.1 → JDBC 1.0 (1997)
  - JDK 1.2 → JDBC 2.0 (1998)
  - JDK 1.4 → JDBC 3.0 (2002)
  - JDK 1.6 → JDBC 4.0 (2007)

- Les API du JDK
  - java.sql
    - Toutes les classes de base pour manipuler les BDD relationnelles
  - javax.sql
    - Classes complémentaires (introduites avec le JDK 1.4)

- Les Drivers JDBC
  - Postgres : https://jdbc.postgresql.org/download.html
    https://jdbc.postgresql.org/documentation/head/connect.html

Schéma classique

BDD ➔ Données

1. Choix du Driver de connexion
2. Connexion à la base
3. Création de la requête SQL
4. Exécution de la requête
   - Récupération du résultat
   - Traitement des erreurs éventuelles
5. Boucle de traitement des données
6. Fermeture de la connexion

Le Driver

Le Driver JDBC

- Permet d'établir la connexion et la communication entre

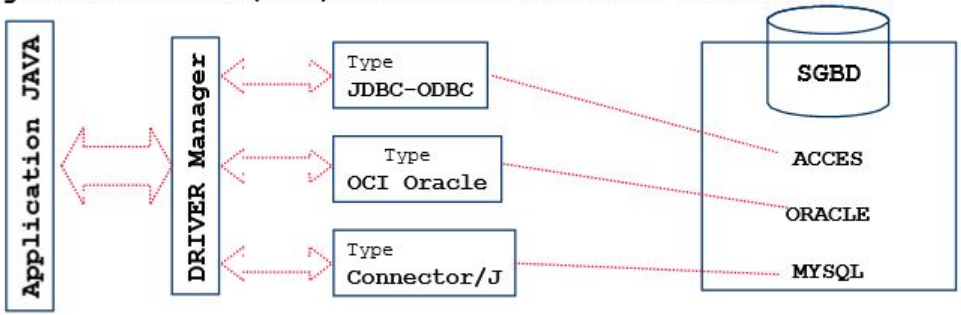
  Le programme Java ←→ Le système de gestion de bases de données

- **Chaque SGBDR possède donc son (ou ses) driver(s) spécifique(s)**

- Les classes liées aux drivers sont externes au JDK
- Elles doivent être liées à l'environnement de travail par le CLASSPATH
- Les drivers sont disponibles sur les sites des constructeurs ou sur le site de Sun

La classe **`DriverManager`** gère les drivers

- **`getConnection(xxx)`** retourne une connexion via le bon Driver



Chargement du driver

Une méthode (courante) consiste à utiliser la méthode `Class.forName`, qui aura pour effet d'enregistrer le Driver auprès du `DriverManager`.
N'oubliez pas de vérifier que le jar contenant le driver est bien dans le classpath

```
String nomDriver =
"nom_du_driver";  try{
    Class.forName(nomDriver);
}catch(ClassNotFoundException cnfe){
    System.out.println("La classe "+nomDriver+" n'a pas été trouvée");
    cnfe.printStackTrace();
}
```

En pratique, à cause d'implémentations imparfaites des spécifications, il sera parfois nécessaire d'utiliser cette syntaxe :

```
Class.forName(nomDriver).newInstance();
```

Exemple :

```
//pour le pont JDBC-ODBC
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

//pour MySQL et ConnectorJ
Class.forName("com.mysql.jdbc.Driver");
```

java.sql.Connection

class `Connection`

- Objet représentant une connexion

- **`DriverManager.getConnection(url,login,passwd)`**
  - url : « jdbc:<subprotocol>:<subname> »
    - <subprotocol> ➔ nom du driver
    - <subname> ➔ base de données (syntaxe liée au sous-protocole)
    - Exemple :
      - jdbc:mysql://monserveur.fr/mabase
      - jdbc:oracle:thin@//localhost:8000:base
      - jdbc:oracle:oci8@:base
- **`close()`**
  - ferme la connexion

- **`createStatement()`**
  - crée un objet « **`Statement`** » (requête SQL)

java.sql.Statement 1/2

interface `Statement`

- Représente une requête SQL
- Propose plusieurs méthodes d'exécution
  - **`executeQuery(String query)`**
    - Pour exécuter une requête SELECT
    - Le résultat est un objet **`ResultSet`** (contenant les données)
  - **`execute/executeUpdate/executeBatch()`**
    - Pour modifier la base (INSERT, UPDATE, DELETE, CREATE)
    - Pour faire des transactions
    - Le résultat des méthodes diffère selon l'action
- **`close()`**
  - Libère la mémoire du Statement
    - *Programme la libération par le garbage collector*

java.sql.ResultSet 1/2

interface `ResultSet`

- Contient les résultats d'une requête SQL

- **`getString(String nomDeColonne)`**
  - Renvoie la valeur contenue dans la colonne nommée.
  - La colonne doit être de type chaîne de caractère
    - (`VARCHAR` par exemple en MySQL)
  - Le résultat est un objet de type String

- **`getInt(String nomDeColonne)`**
  - Renvoie la valeur contenue dans la colonne nommée.
  - La colonne doit être de type numérique entier
    - (`INT` par exemple en MySQL)
  - Le résultat est une variable de type int

- **`next()`**
  - Permet de déplacer le curseur à la ligne suivante
  - Retourne `true` si l'opération est possible (`false` sinon)

1ère connexion

```
import java.sql.*;
public class Connect1 {
  public static void main(String args[]) {
    try {
      Class.forName("com.mysql.jdbc.Driver");
    } catch(ClassNotFoundException e) {
      System.err.println("Classe Driver MySQL non trouvée: "+e.getMessage());
    }
    try {
      Connection con =
         DriverManager.getConnection("jdbc:mysql://localhost/mabase","user","pass");
      Statement st = con.createStatement();
      String query = "SELECT * FROM livres";
      System.out.println("query : " +query);
      ResultSet rs = st.executeQuery(query);
      while ( rs.next() ) {
        String auteur   = rs.getString("auteur");
        String titre    = rs.getString("titre");
        int  nbPages    = rs.getInt("pages");
        System.out.println(titre+ ":" + auteur + "(" +nbPages+ " pages)");
      }
      rs.close();
      st.close();
      con.close();
    } catch(SQLException ex) {
      System.err.println("SQLException: " + ex.getMessage());
    }
} }
```

java.sql.ResultSet 2/2

java.sql.ResultSet *(suite)*

Un objet ResultSet contient les tuples (lignes) issus d'une requête. Sa structure est similaire à celle d'une table de la base.

- Des lignes
  - row, tuples ou n-uplets
  - La numérotation des lignes commence à 1
- Des colonnes
  - column, attribut d'un enregistrements
  - Chaque colonne possède
    - un nom
    - un type
    - un numéro
  - La numérotation des colonnes commence à 1
- `getXXX(String nomDeColonne)`
- `getXXX(int numeroDeColonne)`

Un curseur

Pointe sur une ligne

Permet le déplacement dans le ResultSet

A la création du ResultSet → position 0 (beforeFirst)

A la fin du parcours → position N+1 (afterLast)

La manipulation du curseur en dehors des limites du ResultSet lève des exceptions (SQLException)

`next()`

Permet de déplacer le curseur à la ligne suivante
Retourne `true` si l'opération est possible (`false` sinon)

SQL	méthode	JAVA
BIT	getBoolean	boolean
DATE	getDate	Java.sql.Date
DECIMAL	getBigDecimal	Java.math.BigDecimal
DOUBLE	getDouble	double
FLOAT	getDouble	double
REAL	getFloat	float
INTEGER	getInt	int
VARCHAR	getString	String

java.sql.SQLException

```
try {
     Connection con = DriverManager.getConnection("jdbc:mysql://localhost/java_iform","root","");
     Statement st = con.createStatement();
     String query = "SELECT mauvaischamp FROM livre";
     System.out.println("query : " +query);
     ResultSet rs = st.executeQuery(query);
     rs.close();
     con.close();
} catch ( SQLException ex ) {
     System.out.println("Exception SQL :
     ");  while (ex != null) {
       System.out.println("Message = "+ ex.getMessage()      +
                          "\nSQLState = " + ex.getSQLState() +
                          "\nErrorCode = "+ ex.getErrorCode() );
        ex.printStackTrace();
        ex = ex.getNextException();
}    }
```

Une `SQLException` contient :

- Un message
- Un statut SQL
- Un code d'erreur

La plupart des méthodes de l'API JDBC peuvent lever ce type d'exception

```
query : SELECT mauvaischamp FROM livre
Exception SQL :
Message = Champ 'mauvaischamp' inconnu dans field list
SQLState = 42S22
ErrorCode = 1054
com.mysql.jdbc.exceptions.MySQLSyntaxErrorException: Champ 'mauvaischamp' inconnu dans field list  at
         com.mysql.jdbc.SQLError.createSQLException(SQLError.java:936)
         at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2941)  at
         com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1623)
         at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1715)  at
         com.mysql.jdbc.Connection.execSQL(Connection.java:3243)  at
         com.mysql.jdbc.Connection.execSQL(Connection.java:3172)
         at com.mysql.jdbc.Statement.executeQuery(Statement.java:1197)
         at Connect2.main(Connect2.java:16)
```

Arborescence des Exceptions

SQLTransientException

- **SQLTransientConnectionException**, **SQLTimeoutException**, **SQLTransactionRollbackException**
- Erreurs passagères
  - *(la requête pourrait éventuellement fonctionner si elle était relancée)*

SQLNonTransientException

- **SQLDataException** *(arguments non valides)*, **SQLSyntaxErrorException**, **SQLFeatureNotSupportedException**, **SQLIntegrityConstraintViolationException**, **SQLInvalidAuthorizationSpecException**, **SQLNonTransientConnectionException**
- Erreurs permanentes *(ne fonctionnera jamais tant que le problème n'est pas résolu)*

java.sql.Statement 2/2

interface `Statement`

- A chaque exécution de requête, le `Statement` retourne un `ResultSet`
  - On ne peut utiliser qu'un seul `ResultSet` par `Statement`
  - Pour parcourir en parallèle le résultat de 2 requêtes
    - Il faut 2 `Statement` distincts
- **`execute(query)`**
  - Méthode généraliste
  - Elle retourne un booléen (vrai si la requête retourne un `ResultSet`)
  - La méthode `getResultSet()` pourra être utilisée pour obtenir le `ResultSet`
- **`executeQuery(query)`**
  - Pour exécuter une requête SELECT
  - Le résultat est un objet `ResultSet` (contenant les données)
  - NB: Le résultat n'est jamais `null`
- **`executeUpdate(query)`**
  - Pour exécuter une requête de modification (INSERT, UPDATE, DELETE, CREATE)
  - Le résultat est un entier indiquant le nombre de lignes modifiées
  - Dans le cas d'une requête CREATE, le résultat vaut toujours 0
  - Dans le cas d'une requête INSERT, le statement peut fournir les références de la clé primaire créé Cf. **`getGeneratedKeys()`**

Requêtes de mise à jour (exemple)

```
. . .
Statement st = con.createStatement();
String query = "SELECT COUNT(*) FROM livres";
boolean etat = st.execute(query);
System.out.println("query : " +query+" boolean resultat="+etat);
if (etat) {
   ResultSet resultSet = st.getResultSet();
   // ResultSet resultSet = st.executeQuery(query); etait plus direct !
   resultSet.next();
   System.out.println("count ="+resultSet.getInt(1));
}
query = "INSERT INTO livres VALUES(null,'Prohibition','AlCapone',758)";
int nombre = st.executeUpdate(query);
System.out.println("query : " +query+" nb lignes ajoutees = "+nombre);
query = "DELETE FROM livres WHERE titre='Prohibition' AND auteur='AlCapone'";
nombre = st.executeUpdate(query);
System.out.println("query : " +query+" nb lignes supprimees="+nombre);
. . .
```

```
query : SELECT COUNT(*) FROM personne boolean resultat=true
count =4
query : INSERT INTO personne VALUES(null,'Ita','AlCapone',99) nb lignes ajoutees = 1  query :
DELETE FROM personne WHERE nom='Ita' AND prenom='AlCapone' nb lignes supprimees=1
```

java.sql.PreparedStatement

L'interface `PreparedStatement` permet de gérer des requêtes précompilées

- Utile lorsqu'une requête est renouvelée plusieurs fois avec des paramètres différents
- Chaque paramètre est représenté par un « ? » dans la requête

Un objet `Connection` permet d'obtenir un `PreparedStatement` à partir d'une requête SQL

- *connexion*.**prepareStatement**(query)
- *connexion*.**prepareStatement**(query, rsType, rsConcurrency)
  - *Comme pour la méthode `createstatement`, `rsType` et `rsconcurrency` paramètrent éventuellement le `ResultSet`*

Les principales méthodes d'un `PreparedStatement` sont :

- **setXXX**(indexParam, valeurParam)
  - `indexParam` est le numéro du paramètre variable dans la requête (le 1er porte le n°1)
  - `valeurParam` est la valeur donnée au paramètre
    - Exemple `setString(2,"ma chaine"); setInt(2, 234);`
- **clearParameters**()
  - Réinitialise tous les paramètres
  - Un paramètre en état « initial » ne permet pas d'exécuter la requête
- **execute**()
  - Retourne un booléen (true si le résultat de la requête est un ResultSet)
- **executeQuery**()
  - Dédié aux requêtes de type SELECT
  - Retourne un ResultSet contenant les résultats de la requête
- **executeUpdate**()
  - Dédié aux requêtes de mise à jour
  - Retourne le nombre d'enregistrements modifiés

Exercices

EXERCICE 13 :

- Reprenez le projet de l'exemple (archive zip)
- Préparer votre BDD avec le fichier biblio.sql
- Mettre à jour les infos de connection et tester le launcher
- Ajouter la possibilité de mettre à jour un auteur existant
- Faites la même chose avec les livres que le code présent pour les auteurs (CRUD)
- Mettre à jour le launcher pour tester vos nouvelles fonctionnalitées

Exercices

EXERCICE 14 :

- Récupérez le fichier mediatheque.sql, préparez une BDD et exécutez le contenu du fichier SQL
- Créer un programme faisant la même chose que l'exercice 13 mais avec la nouvelle BDD

TP JAVA (APP CRM)

Le but du TP va être de produire une app permettant d'avoir une V1 de l'application CRM.

L'interface utilisateur sera géré avec la console.

1 . Dans un premier temps, occupez vous seulement de permettre le CRUD sur les clients.
2. Ensuite, mettez en place le CRUD sur les commandes

3. Pour aller plus loin, gérer la persistance en utilisant le JDBC de PostgreSQL.
4. Pour aller encore plus loin, gérer les utilisateurs, création, connection et déconnection

Pensez à respecter et à mettre en application tout ce qu'on a vu ensemble durant les cours de JAVA.