


Formation

Java perfectionnement



Réalisé par Mr Jacquot David

Fait le mardi 7 mai 2019

Tables des matières

1	Application Java	7
1.1	Création d'un projet Java.....	7
2	LES TYPES.....	11
1.	Les Booléens	11
2.1	Les variables de type élémentaire.....	12
2.1.1	Identificateur (nom d'une variable)	12
2.1.2	Déclaration d'une variable	12
2.1.3	Affectation d'une variable	13
2.1.4	Le format d'une variable	13
2.1.5	Le type String.....	14
2.1.6	La concaténation	14
3	La syntaxe élémentaire de Java	15
3.1	Les commentaires	15
3.1.1	Commentaire sur une ligne	15
3.1.2	Commentaire sur plusieurs lignes	15
3.2	Commentaires de documentation automatique (Javadoc).....	15
3.2.1	Générer la Javadoc	17
4	Les opérateurs	20
4.1	Les opérateurs de calcul	20
4.2	Les Opérateurs d'assignation	20
4.3	Les opérateurs d'incrémentation	21
4.4	Les opérateurs de comparaison	22
4.5	Les opérateurs logiques	22
5	Les tableaux	23
5.1	Déclaration d'un tableau	23
5.2	Initialiser un tableau.....	24
5.3	Tableau multi dimensions.....	24
5.4	Afficher un élément du tableau.....	25
5.4.1	Parcourir un tableau séquentiel.....	25
5.5	L'objet Arrays.....	26
5.5.1	Remplir un tableau.....	26
5.5.2	Copier un tableau	27
5.5.3	Comparer des tableaux	27
6	Les collections d'objets	28
6.1.1	L'interface Iterator.....	28
6.2	Les objets List.....	29
6.2.1	Utilisation d'un ArrayList	30
6.3	Les objets map	31

6.4	Les objets Set	33
7	Le package	34
7.1	Le package par défaut	34
8	Les bibliothèques de classes standard	37
8.1.1	java.lang	37
8.2	La classe Math.....	38
8.2.1	Fonctions trigonométriques (classe Math).....	39
8.3	Les conversions de types(cast)	40
8.3.1	Conversion implicite.....	40
8.3.2	Conversion explicite.....	40
8.4	Classes wrapper	41
8.5	Classe Integer.....	42
8.6	Classe Double	42
8.7	Classe Character	43
8.8	Les méthodes de la classe String.....	44
8.9	Classe String	44
8.9.1	La classe StringBuffer ou StringBuilder.....	46
8.10	La classe BigDecimal	48
8.11	Les calculs mathématiques en Java.....	49
8.12	Formater les nombres pour l’affichage.....	50
9	Gestion des erreurs (exceptions)	51
9.1.1	Quelques exceptions prédéfinies	51
9.1.2	Schéma des exceptions	52
9.1.3	L’instruction try.....	52
9.1.4	Définir une exception	54
10	Le système de fichier.....	56
10.1	La classe File.....	56
10.1.1	Instanciation.....	56
10.1.2	Méthodes principales de la classe File.....	56
10.2	Lister les fichiers d'un dossier.....	57
10.3	Déplacer un fichier.....	57
10.3.1	Sur un même Système de fichier	57
10.3.2	Sur un système de fichier différent.....	57
10.4	Copier un fichier.....	58
10.5	Suppression d'un fichier	58
10.6	Manipulation des fichiers texte	59
10.6.1	Lire un fichier	59
10.6.2	Ecrire un fichier.....	59
10.6.3	Ajouter une ligne à un fichier	60
10.7	Les fichiers propriétés	60

10.7.1	La classe Properties	60
10.7.2	Ecrire dans un fichier properties	61
11	Les annotations	62
11.1	Qu'est-ce qu'Annotation ?	62
11.1.1	Instructions pour la documentation.....	62
11.1.2	Instructions pour le compilateur.....	62
11.1.3	Instruction en temps de construction (Build-time)	62
11.1.4	Instructions d'exécution (Runtime)	63
11.2	Les annotations standards	63
11.2.1	@Deprecated.....	63
11.2.2	@Override	63
11.2.3	@SuppressWarnings.....	64
11.3	Les annotations communes.....	64
11.3.1	L'annotation @Generated.....	64
11.3.2	Les annotations @Resource et @Resources	64
11.3.3	Les annotations @PostConstruct et @PreDestroy	65
11.4	Écrire Votre Annotation	65
11.5	Votre première Annotation.....	65
12	JDBC (Java DataBase Connectivity).....	67
12.1	Ajout de la dépendance dans le pom.xml	67
12.2	Connexion à la base de données	67
12.3	Exécuter une requête SQL	69
12.3.1	Méthodes de la classe Connection pour créer des objets de requêtage	69
	Les objets pour exécuter des requêtes	69
12.3.2	Principales méthodes de l'objet Statement	70
12.3.3	Principales méthodes d'un ResultSet.....	71
12.4	PreparedStatement (Requête préparée)	72
12.4.1	Principales méthodes de l'objet PreparedStatement	72
12.5	Design pattern (modèle de conception) DAO.....	75
12.6	Le Design Pattern Singleton.....	76
12.7	Design pattern Factory	79

1 Application Java

Une application Java s'exécute sous le contrôle de l'interpréteur de code Java et a accès à toutes les ressources du système.

Une application est composée d'au moins une classe qui doit contenir une méthode appelée **main()**, c'est le point de démarrage du programme.

Le nom du fichier Java doit correspondre au nom de la classe (identique) qui contient la méthode **main**.

Une application standard

```
public class PremierProgramme {  
    public static void main(String[] args){  
        System.out.print("Bonjour tout le monde !");  
    }  
}
```

Quelques détails du programme

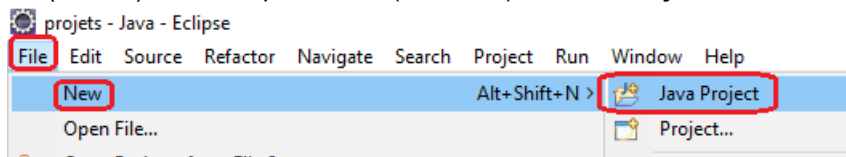
public static void main()

main() est la méthode qui est appelée lors de l'exécution d'une application, cette méthode peut recevoir des arguments (venant de la ligne de commande) que l'on récupère dans le tableau **args**.

System.out.print() → **System** est la **classe** qui permet d'utiliser l'entrée et la sortie standard (saisie clavier et l'affichage à l'écran). **out** est un **objet** de la classe **System** qui gère la sortie standard. **print** est une **méthode** qui permet d'écrire un texte sur la sortie standard (l'écran).

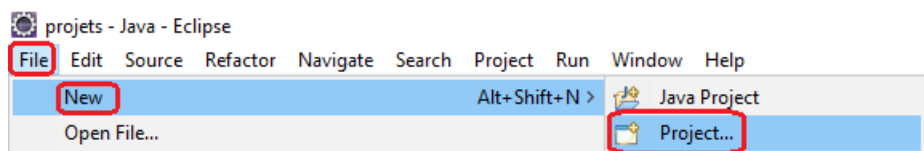
1.1 Création d'un projet Java

Dans le Menu **file** (Fichier) choisir l'option **New** (Nouveau) et **Java Project**.

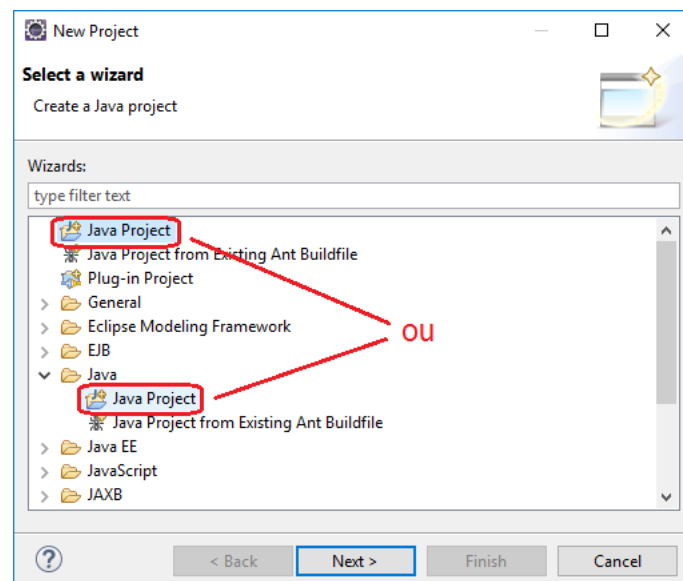


Ou

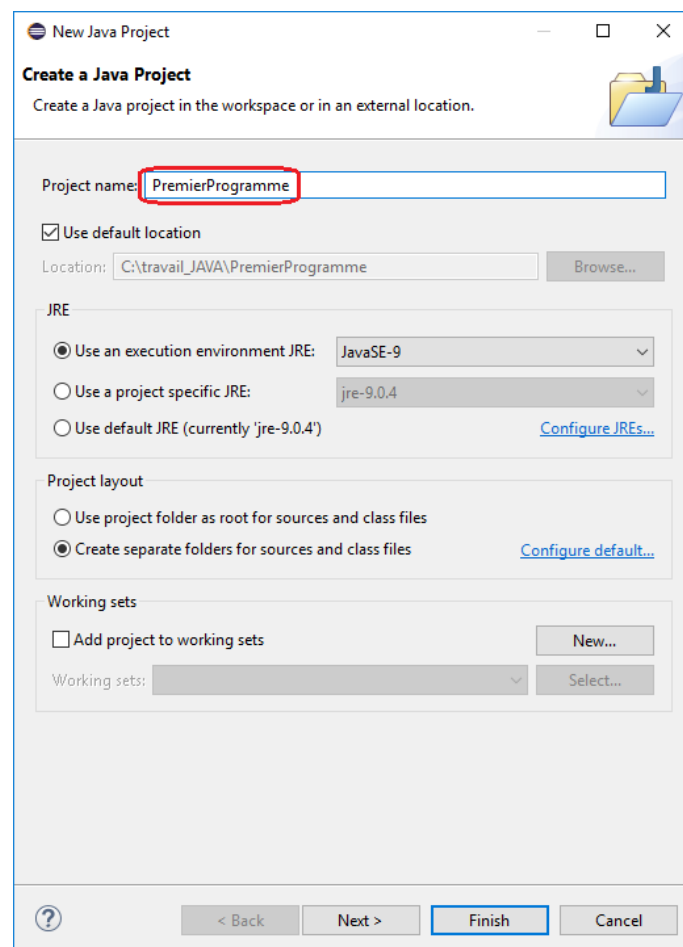
Dans le Menu **file** (Fichier) choisir l'option **New** (Nouveau) et enfin **Project...** (Projet).



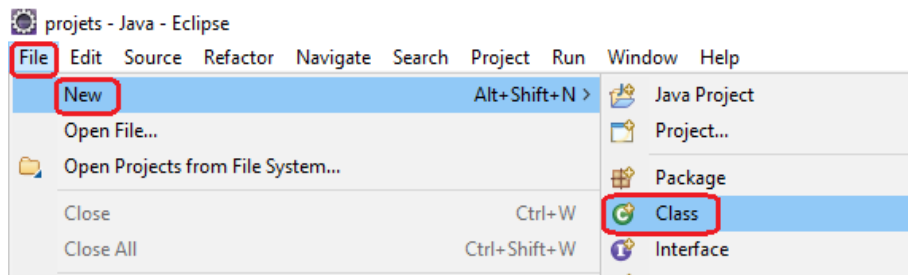
Choisir Java Project (Projet Java).



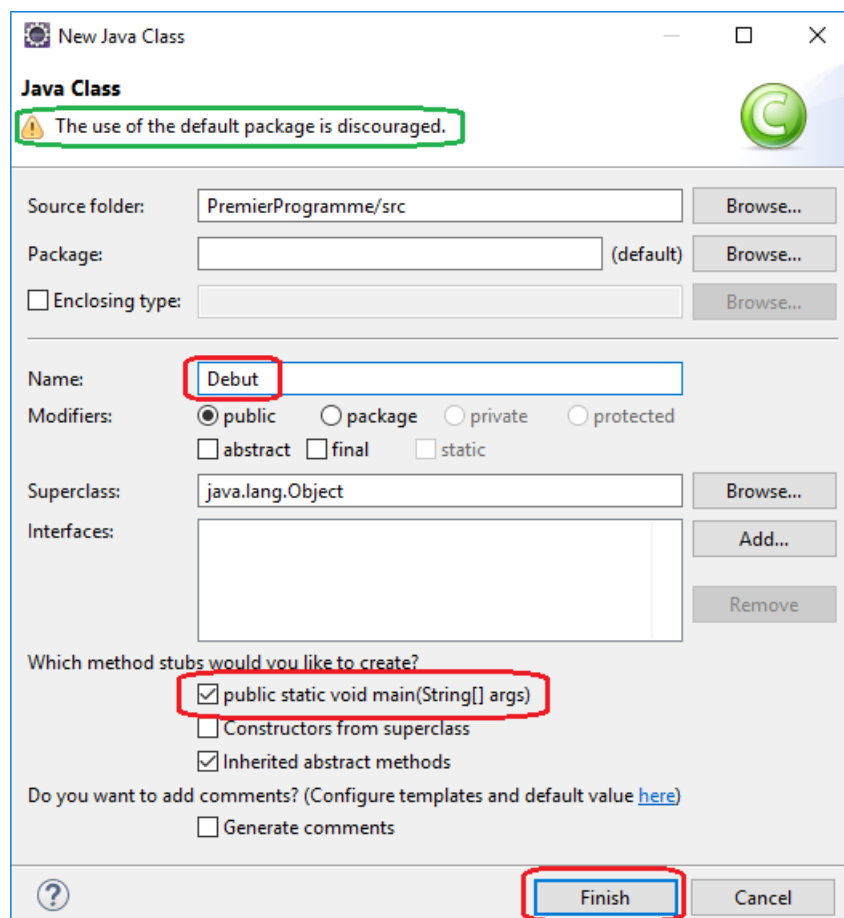
Définir le nom du projet, éventuellement l'emplacement, la version du JRE.



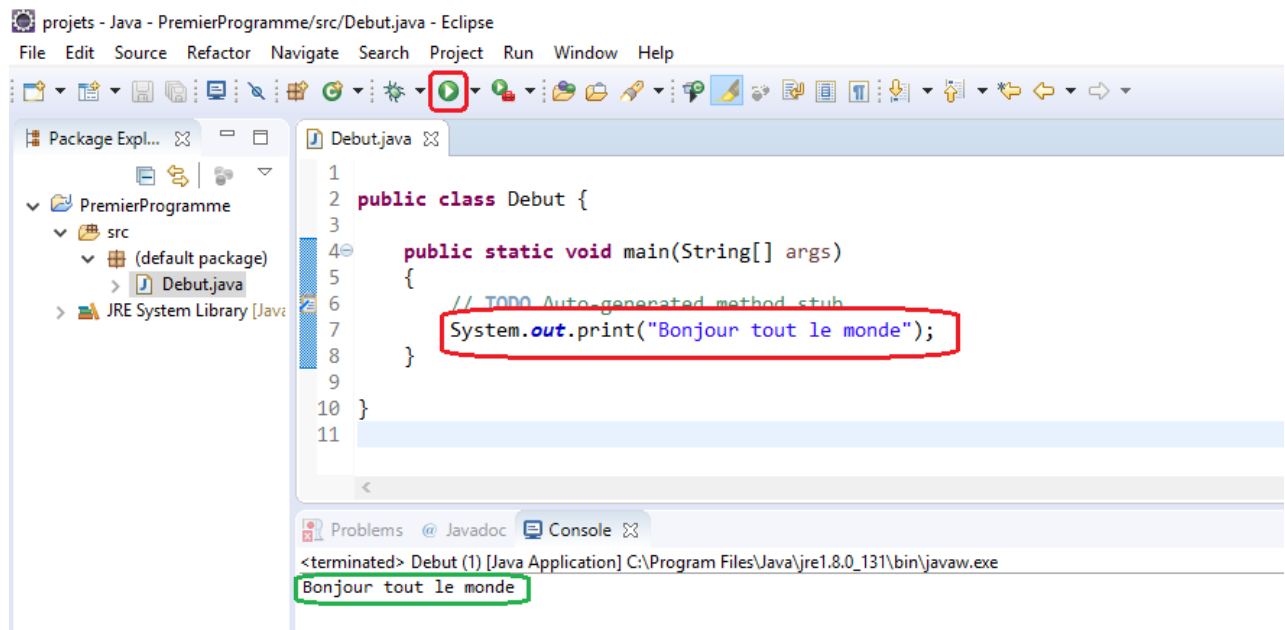
Clic sur File, puis New puis Class ou (Clic droit sur le nom du projet puis New/Class (Nouvelle/Classe))



On ne définit pas de package (c'est tout de même recommandé, on verra par la suite).
On donne un nom à notre classe et l'on coche l'option pour générer le code de la méthode main().



Saisir le code et lancer  le programme dans eclipse.



Ou en ligne de commande (dans le dossier contenant la classe).

```
C:\travail\projets\PremierProgramme\bin>Java Debut
Bonjour tout le monde !
C:\travail\projets\PremierProgramme\bin>
```

```
Windows PowerShell
PS C:\travail_JAVA\PremierProgramme\bin> java Debut
Bonjour tout le monde
PS C:\travail_JAVA\PremierProgramme\bin>
```

On peut utiliser le caractère d'échappement `\n` pour passer à la ligne ou la méthode `println()`.

```
public class PremierProgramme {
    public static void main(String[] args) {
        System.out.print("Bonjour tout \n le monde !");
    }
}
```

Ou

```
public class PremierProgramme {
    public static void main(String[] args) {
        System.out.println("Bonjour tout ");
        System.out.println("le monde !");
    }
}
```

Résultat :

```
<terminated> PremierProgramme [Java Ap
Bonjour tout
le monde !
```

2 LES TYPES

Java est un langage typé, c'est-à-dire que chaque variable possède un type.

Les types élémentaires ont une taille identique quel que soit la plate-forme d'exécution, c'est un des éléments qui permet à Java d'être indépendant de la plate-forme sur lequel le code s'exécute.

1. Les Booléens (Valeur logique)

Type	Taille en bit	Valeurs possibles	Commentaires
boolean	1	true (vrai) ou false (faux).	Pas de conversion possible vers un autre type.

2. Les Entiers

Type	Taille en bit	Valeurs possibles	Commentaires
byte	8	De -128 à 127.	Valeur sur un octet signé.
short	16	De -32 768 à 32 767.	Entier court signé.
int	32	De -2 147 483 648 à 2 147 483 647.	Entier signé.
long	64	De -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807.	Entier long signé.

3. Les réels (nombres à virgules)

Type	Taille en bit	Valeurs possibles	Commentaires
float	32	1.40239846e-045 (Float.MIN_VALUE) à 3.4028235e+038 (Float.MAX_VALUE)	Nombres en virgule flottante simple précision (IEEE754). <ul style="list-style-type: none">• 23 bits pour la mantisse.• 8 bits pour l'exposant.• 1 bit pour le signe.
double	64	4.9406564584124654e-324 (Double.MIN_VALUE) à 1.797693134862316e+308 (Double.MAX_VALUE)	Nombres en virgule flottante double précision (IEEE754) <ul style="list-style-type: none">• 52 bits pour la mantisse• 11 bits pour l'exposant• 1 bit pour le signe

L'existence de constantes prédéfinies de la forme *Float.MAX_VALUE* fournissent les différentes limites.

4. Le type char

Type	Taille en bit	Valeurs possibles	Commentaires
char	16	\u0000 à \uFFFF	Caractère Unicode, caractère international universel. Entouré de cotes simples 'A' dans du code Java.

Les types élémentaires commencent tous par une minuscule.

2.1 Les variables de type élémentaire.

2.1.1 Identificateur (nom d'une variable)

- Le premier caractère doit être une lettre (majuscule ou minuscule), le caractère de soulignement _ ou le signe dollar \$.
- Un nom de variable peut être composé de tous les caractères alphanumériques (**sans espace**) et des caractères _ et \$.
- Les noms des variables doivent être différents des mots réservés du langage.

2.1.2 Déclaration d'une variable

Indique au système d'exploitation, de réserver un espace mémoire pour une variable, l'espace sera fonction du type de la variable et de donner un nom à cet espace (le nom de la variable → identificateur).

Syntaxe

```
type    nom_variable;
```

Il est possible de définir plusieurs variables de même type en séparant chacune d'elles par une virgule.

Exemple

```
int  abc;           // On déclare un entier abc.
float bateau;       // On déclare un réel (float) bateau.
double voiture;     // On déclare un réel de grande taille (double) voiture.
boolean trouve;     // On déclare un booléen trouve.
char reponse;       // On déclare un caractère reponse.
int  jour, mois, annee; // On déclare 3 variables (jour, mois et annee) de type int.
```

2.1.3 Affectation d'une variable

C'est l'action de donner une valeur à une variable, si c'est une première valeur, on parle d'initialisation.

Exemple

```
abc=152;           // On met 152 dans la variable abc.
voiture=15.62;     // On met 15.62 dans la variable voiture.
trouve=true;       // On met true dans la variable trouve.
reponse='n';       // On met le caractère n dans la variable reponse.
```

Il est possible en une seule instruction de faire la déclaration et l'affectation d'une ou de plusieurs variable (en séparant chacune d'elles par une virgule).

Exemple

```
int    abc=12, def=56; // On déclare abc et def, on affecte la valeur 12 à abc
        // et on affecte la valeur 56 à def.
```

2.1.4 Le format d'une variable

Il permet d'indiquer le type d'une valeur qui serait ambigu sans cette précision.

Un nombre est considéré comme un double (par défaut, donc si non précisé par le format) si le nombre possède un point et/ou un exposant, ce qui peut provoquer des erreurs lors de la compilation.

Préfixe (Placé avant la valeur)	Description
0x (zéro x)	Indique que le nombre est codé en hexadécimal .
0 (zéro)	Indique que le nombre est codé en octal .

Suffixe (Placé après la valeur)	Description
L ou l	Indique que le nombre est un entier long .
F ou f	Indique que le nombre est un float .
D ou d	Indique que le nombre est double .

Exemple

```
double voiture = 3d; // On affecte 3 à voiture (3 est considéré comme un
double).
float bateau = 8.56f; // On affecte la valeur 8.56 à bateau.
float nb1 = +.5f;      // On affecte la valeur 0.5 à nb1.
float nb2 = 2e10f;     // On affecte la valeur 2 * 1010 à nb2.
```

Valeurs par défaut

Quand une donnée d'un type élémentaire est membre d'une classe (Cf Les classes) on est assuré qu'elle a une valeur par défaut si on ne l'initialise pas.

Type	Valeur par défaut
boolean	false
char	'\u0000' (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

2.1.5 Le type String

Le type **String** (s majuscule) est un **objet**, ce n'est pas comme dans beaucoup de langages un tableau de caractères, il correspond à un type chaîne de caractère (caractères Unicode codés sur 2 octets).

Les chaînes de caractères ne sont pas vraiment limitées en java. On notera cependant que les caractères de la chaîne sont indexés par des 'int', ce qui nous fixe une limite à 2³² caractères (soit une chaîne de plus de 4Go en mémoire).

On utilise les guillemets "..." pour délimiter une chaîne de caractères.

Il se déclare comme une variable élémentaire.

Exemple

```
String chaine1; // On déclare chaine1.  
chaine1="bonjour"; // On affecte la bonjour à chaine1.  
String chaine1="bonjour"; // On déclaration et affectation de chaine1.
```

2.1.6 La concaténation

Le signe (+) permet de concaténer des chaînes de caractères, c'est à dire de regrouper des parties de chaînes de caractères.

Exemple

```
String chaine1="bonjour";  
String resultat=chaine1 + " à tous ! ";  
  
➔ chaine1 contient la chaîne "bonjour à tous"
```

3 La syntaxe élémentaire de Java

3.1 Les commentaires

Ils sont utilisés par le programmeur pour décrire des parties critiques de son programme, ils sont **indispensables** pour **maintenir** facilement ses programmes et pour **travailler en équipe**.

3.1.1 Commentaire sur une ligne

Les caractères double slashes (//) indiquent que tout ce qui suit sur la ligne est un commentaire.

Syntaxe

```
// ceci est un commentaire sur une ligne
```

3.1.2 Commentaire sur plusieurs lignes

Les caractères /* */ indiquent respectivement le début et la fin du commentaire.

Syntaxe

```
/* le commentaire  
est sur  
plusieurs lignes */
```

3.2 Commentaires de documentation automatique (Javadoc)

Ces commentaires seront utilisés par javadoc (le générateur de documentation), ils peuvent contenir des sections spéciales permettant d'ajouter des informations supplémentaires sur les paramètres des méthodes, les valeurs de retour, les exceptions ... à l'aide de tags préfinis.

Nous utiliserons pour cela le caractère @ pour commencer notre ligne de commentaire.

Déclaration des commentaires de documentation automatique

Caractères	Description
/**	Indiquent le début du commentaire.
*	Indique une ligne de commentaire.
*/	Indiquent la fin du commentaire.

On fait précéder la classe, l'interface, la méthode ... par un commentaire, qui permet de décrire l'élément, ses fonctions, ses paramètres ... ce qui va permettre la génération de la Javadoc.

Syntaxe

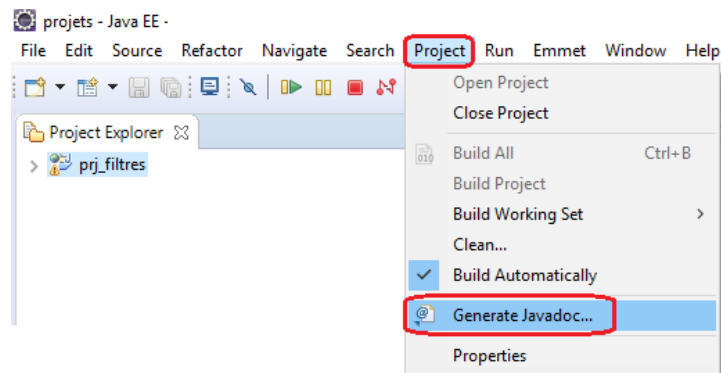
```
/**  
 * Description de l'élément  
 *  
 * @tag1 Commentaire pour le tag1  
 * @tag2 Commentaire pour le tag2  
 * ...  
 */
```

Les Principaux tags

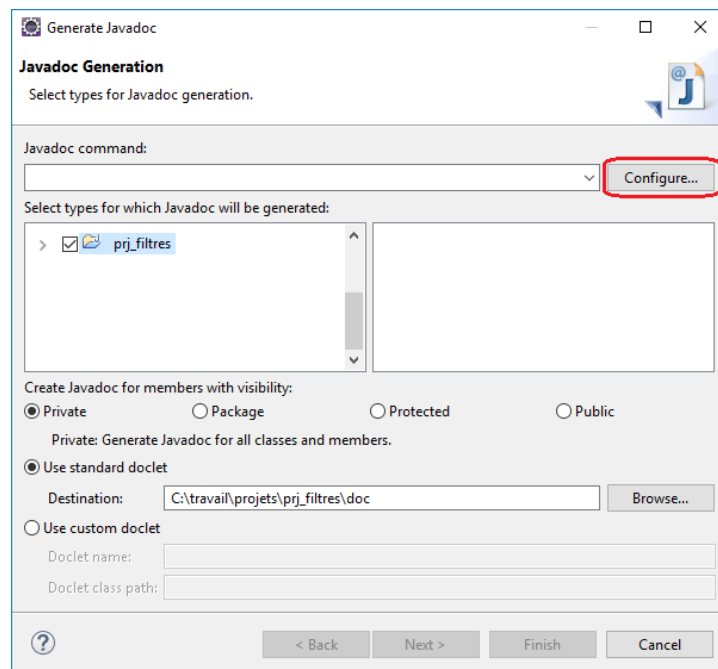
Tag	Description
@author	Permet d'indiquer le ou les auteurs de l'élément (méthode, classe ou interface).
@deprecated	Permet de spécifier si un élément est déprécié.
@exception @throws	Permet de spécifier qu'une exception peut être levée par l'élément. <code>@exception IOException</code> Sur une erreur de saisie.
{@link}	Permet de spécifier un lien vers une autre partie de la documentation. Suite est une sous classe de {@link Info}
@param	Permet de spécifier un paramètre de l'élément pour les méthodes et les constructeurs. <code>@param num</code> un entier
@return	Permet de spécifier la valeur de retour d'une méthode.
@see	<p>Permet de spécifier une liaison (liens hypertextes) avec un autre élément documenté.</p> <p>La syntaxe de cette instruction est simple : elle prend un paramètre qui correspond à l'élément lié. Cet élément est constitué du nom de la classe puis éventuellement de caractère # suivi du nom d'une méthode. Comme le langage Java supporte la surcharge de méthode, on peut donner la signature de la méthode à lier, histoire de lever toute ambiguïté.</p> <p><code>@see section</code> <code>@see # méthode(Type argument, Type argument,...)</code> <code>@see Classe</code> <code>@see Classe # méthode (Type argument, Type argument,...)</code></p>
@since	Permet de spécifier depuis quelle version l'élément a été ajouté. <code>@since 2012-09-11</code>
@version	Permet de spécifier le numéro de la version courante. <code>@version 2.1</code>

3.2.1 Générer la Javadoc

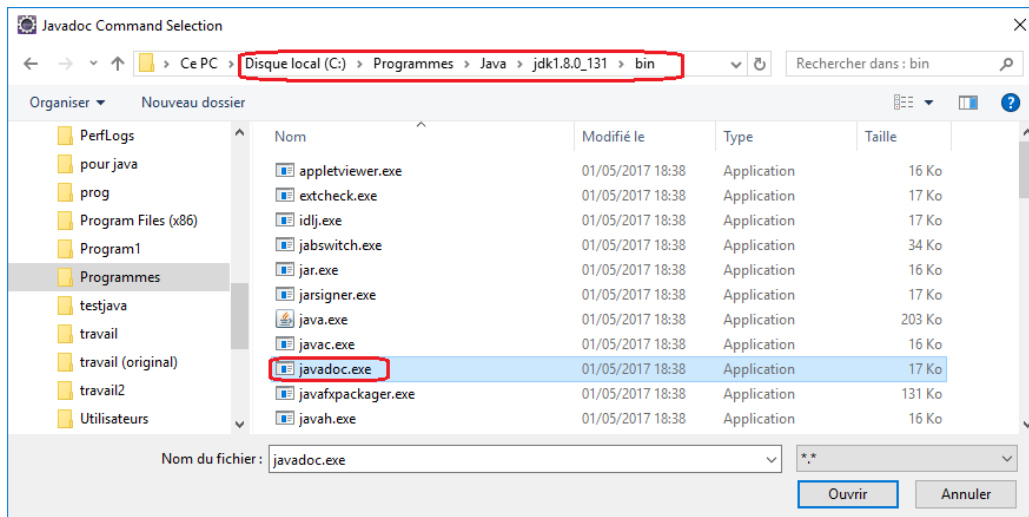
Dans le menu Project / Generate Javadoc ...



On clique sur le bouton Configure...



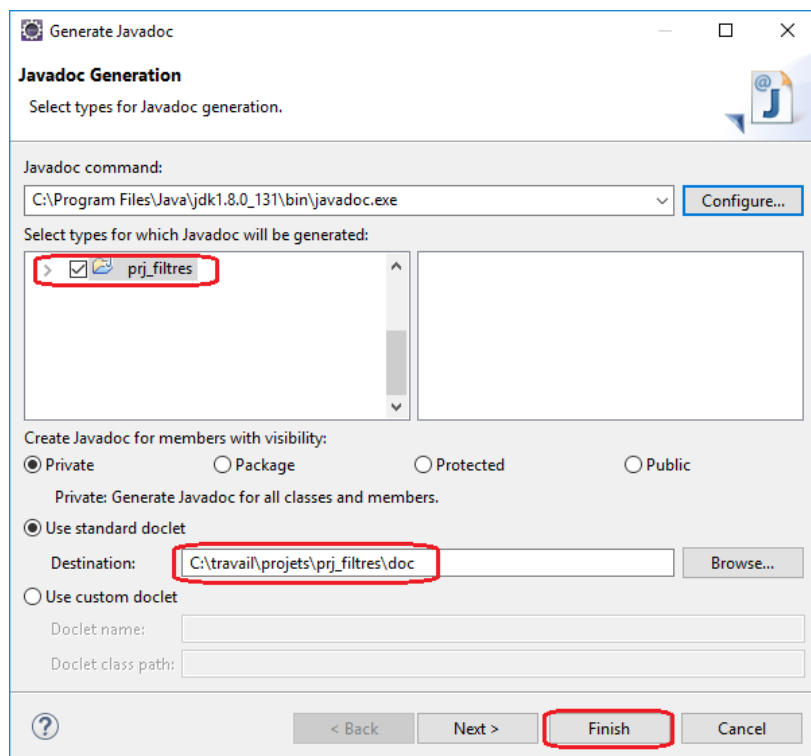
On indique à Eclipse où il va trouver l'exécutable qui permet de générer les javadocs (le fichier javadoc.exe dans le dossier bin du JDK).



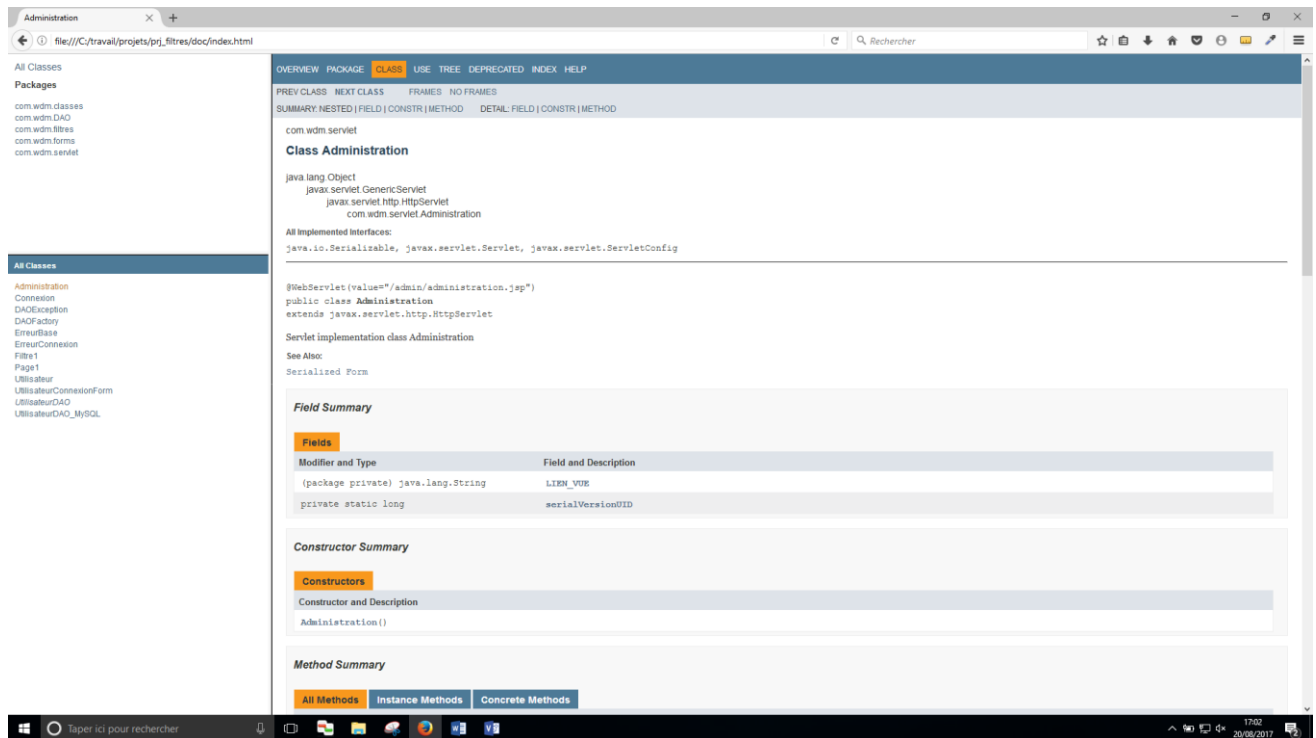
On sélectionne le projet et l'emplacement de la documentation.

La visibilité :

- **Private** : Documentation de **tous les attributs et toutes les méthodes**.
- **Package** : Documentation des attributs et méthodes **public**, **protected** et **package**.
- **Protected** : Documentation des attributs et méthodes **public** et **protected**.
- **Public** : Documentation des attributs et méthodes **public**.



Dans le dossier doc, on retrouve la documentation au format HTML.



Exemple

```

/**      Ce programme permet l'affichage de formes géométriques
 * Ces formes seront le rectangle, le cercle, le triangle ...
 */
...
/**      Cette classe est la classe de base de toute figure
géométrique
 * de mon package. <B>Cette classe est abstraite</B>, et ne peut
 * donc être utilisée pour instancier des objets.
 * @see Rectangle
 * @see Cercle
 * @see Triangle
 * @version 4.1
 * @since 1.0
 * @author David
 */
public abstract class Forme {
    ...

    /**      Cette méthode permet de calculer la surface de la forme
géométrique.
 * @return renvoie la surface de la forme
 * @see Rectangle#surface
 * @see Cercle#surface
 * @see Triangle#surface
 */
    public abstract double surface();
}

```

4 Les opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...

4.1 Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Rôle	Exemple
+	addition	Ajoute deux valeurs.	
-	soustraction	Soustrait deux valeurs.	
*	multiplication	Multiplie deux valeurs.	
/	division	Divise deux valeurs et renvoi un nombre entier si les deux nombres sont entiers (ou bien des chaînes converties en entier). Un nombre à virgule flottante sera retourné, si l'une des opérands est un nombre à virgule flottante.	int a=10, b=3, res1; double c=10, d=3, res2; res1=a/b; ➔ res1=3 res2=c/d; ➔ res2=3.3333 ... res2=c/b; ➔ res2=3.3333 ...
%	modulo	Reste de la division entière.	int a=10, b=3, res; res=a%b; ➔ res=1
=	affectation	Affecte une valeur à une variable.	

4.2 Les Opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles, qu'ajouter une valeur dans une variable et stocker le résultat dans la variable (**incrément**). Une telle opération s'écrirait habituellement de la façon suivante : $x=x+5$;

Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme : $x+=5$;
Ainsi, si la valeur de x était 3 avant opération, elle sera de 8 après l'assignation.

Les autres opérateurs de même type sont les suivants :

Opérateur	Description	Utilisation	Equivalence
+=	Additionne la valeur de la variable droite à la valeur de la variable de gauche et stocke le résultat dans la variable de gauche.	$a+=b;$	$a=a+b;$
-=	Soustrait la valeur de la variable de droite à la valeur de la variable de gauche et stocke le résultat dans la variable de gauche.	$a-=b;$	$a=a-b;$

=	Multiplie la valeur de la variable de gauche par la valeur de la variable de droite et stocke le résultat dans la variable de gauche.	<code>a=b;</code>	<code>a=a*b;</code>
/=	Divise la valeur de la variable de gauche par la valeur de la variable de droite et stocke le résultat dans la variable de gauche. Cf opérateur de division.	<code>a/=b;</code>	<code>a=a/b;</code>
%=	Divise la valeur de la variable de gauche par la valeur de la variable de droite et stocke le reste (de la division entière) dans la variable de gauche.	<code>a%=b ;</code>	<code>a=a%b;</code>
=	Elève la valeur de la variable de gauche à la puissance de la variable de droite et stocke le résultat, dans la variable de gauche.	<code>a^=b</code>	<code>a = a ^ b</code>

4.3 Les opérateurs d'incrément

Ce type d'opérateur permet d'augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente d'un en un).

Un opérateur de type `x++` permet de remplacer des notations lourdes telles que `x=x+1` ou bien `x+=1`.

Opérateur	Dénomination	Description	Utilisation	Equivalence
++var	Pré-incrémente	Incrémente la variable d'une unité, puis retourne sa valeur.	<code>a=5;</code> <code>b=++a;</code> résultat : a=6, b=6	<code>a=5;</code> <code>a=a+1;</code> <code>b=a;</code>
var++	Post-incrémente	Retourne la valeur de la variable, puis incrémente la variable d'une unité	<code>a=5;</code> <code>b=a++;</code> résultat : a=6, b=5	<code>a=5;</code> <code>b=a;</code> <code>a=a+1;</code>
--var	Pré-décrémente	Décrémente la variable d'une unité, puis retourne sa valeur.	<code>a=5;</code> <code>b=--a;</code> résultat : a=4, b=4	<code>a=5;</code> <code>a=a-1;</code> <code>b=a;</code>
var--	Post-décrémente	Retourne la valeur de la variable, puis décrémente la variable d'une unité	<code>a=5;</code> <code>b=a--;</code> résultat : a=4, b=5	<code>a=5;</code> <code>b=a;</code> <code>a=a-1;</code>

NB : `a++` et `++a` sont équivalents lorsqu'il n'y a pas d'affectation.

`a ++ → a=a+1`

`++a → a=a+1`

4.4 Les opérateurs de comparaison

Le résultat d'une comparaison retourne **true** si la condition est vraie, **false** si elle est fausse.

Opérateur	Dénomination	Effet
==	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité.
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur.
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une autre variable ou valeur.
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une autre variable ou valeur.
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une autre variable ou valeur.
!=	opérateur de différence	Vérifie qu'une variable est différente d'une autre variable ou valeur.

4.5 Les opérateurs logiques

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies.

Opérateur	Dénomination	Effet	Syntaxe
 	OU logique	Renvoie true si une des conditions est réalisée.	(condition1) (condition2)
&&	ET logique	Renvoie true si toutes les conditions sont réalisées.	(condition1)&&(condition2)
^	OU exclusif	Renvoie true si une des conditions est réalisée, mais pas les deux en même temps.	(condition1) ^ (condition2)

5 Les tableaux

Un tableau est un conteneur renfermant plusieurs valeurs (d'un type choisi), auxquelles on accède par l'intermédiaire d'un indice.

Un tableau se caractérise par les crochets [].

Il ne faut pas oublier que les tableaux, par défaut, commencent toujours à l'indice 0.

Exemple :

	pays				
Valeurs :	be	ca	ch	fr	uk
Indices :	0	1	2	3	4

Les tableaux dérivent de la classe Object, il faut utiliser des méthodes pour y accéder dont font partie des messages de la classe Object tel que equals() ou getClass().

5.1 Déclaration d'un tableau

Syntaxe

```
type[ ] nom_tableau; // Déclaration (syntaxe Java)
nom_tableau = new type [taille]; // Allocation
```

```
type nom_tableau[ ]; // Déclaration (syntaxe du c)
nom_tableau = new type [taille]; // Allocation
```

Ou

```
type [ ] nom_tableau = new type [taille]; // Déclaration et allocation
```

Ou

```
type nom_tableau[] = new type [taille]; // Déclaration et allocation
// Forme la plus utilisée
```

La **taille** : nombre de « cases » du tableau.

La syntaxe avec les crochets juste après le type, est la plus utilisée en Java.

Exemple

```
String [ ] pays;
pays = new String [5] ;
String [ ] pays = new String [5] ;
```

5.2 Initialiser un tableau

Il existe plusieurs méthodes pour initialiser (donner des valeurs aux cases de notre tableau) un tableau.

- En utilisant des indices

Exemple

```
String [ ] pays = new String [5] ;  
pays[0]="be";  
pays[1]="ca";  
pays[2]="ch";  
pays[3]="fr";  
pays[4]="uk";
```

- Lors de la déclaration
La taille du tableau sera déterminée lors de l'assignation.

Exemple

```
String [ ] pays = {"be", "ca", "ch", "fr", "uk"};
```

La propriété **length** d'un tableau permet de connaître sa taille (nombre d'éléments → de cases).

Exemple

```
String [ ] pays = {"be", "ca", "ch", "fr", "uk"};  
int taille=pays.length; // Taille est égal à 5
```

5.3 Tableau multi dimensions

Pour déclarer un tableau à plusieurs dimensions, il faut déclarer un tableau de tableau.

Syntaxe

```
type[ ][ ] nom_tableau =new type[taille1][taille2]
```

Exemple

```
int [ ][ ] mon_tableau = new int[10][10];
```

On peut déclarer des tableaux, dont la seconde dimension, n'est pas identique pour chaque occurrence.

Exemple

```
int [ ][ ] mon_tableau = new int[2][];  
mon_tableau [0] = new int[7];  
mon_tableau [1] = new int[5];  
  
int [ ][ ] resultats = {{7,4,9,1},  
                        {8,3,1},  
                        {12,7,1,3,9,4,6,10}};
```

Chaque élément du tableau est initialisé selon son type par l'instruction new : 0 pour les numériques, '\0' pour les caractères, false pour les booléens et nil pour les chaînes de caractères et les autres objets.

5.4 Afficher un élément du tableau

Exemple

```
String [ ] pays = {"be","ca","ch","fr","uk"};  
String a = pays[1];    // On récupère l'élément ayant l'indice 1 dans le tableau.
```

5.4.1 Parcourir un tableau séquentiel

Avec une boucle for simple

Exemple

```
int i;  
String [ ] pays = {"be","ca","ch","fr","uk"};  
int nb= pays.length;    // Nombre d'éléments du tableau  
for (i=0;i < nb; i++){  
    System.out.println("pays[" + i + "] =" + pays[i]);  
}
```

Avec une boucle for (for each)

Exemple

```
String [ ] pays = {"be","ca","ch","fr","uk"};  
  
int i =0;  
  
for(String p: pays)  
{  
    System.out.println("pays[" + i + "] =" + p);  
    i++,  
}
```

5.5 L'objet Arrays

Il permet de manipuler les tableaux

Méthode	Description
toString(tab)	La méthode retourne une chaîne de caractères contenant les éléments du tableau (convertis en chaîne de caractères), séparés par des virgules, et entre crochets.
equals(tab, tab2)	Pour un type primitif ou un Object retourne true si les deux tableaux sont égaux et false sinon. Deux tableaux sont égaux s'ils contiennent le même nombre d'éléments, et si les éléments de même rang sont égaux. Si des éléments du tableau peuvent être des tableaux, il faut utiliser la méthode <code>deepEquals(tab, tab2)</code> .
fill(tab, val)	Affecte la valeur val à tous les éléments du tableau <code>tab</code> .
binarySearch(tab, val)	La méthode effectue une recherche de la valeur val dans le tableau trié (par ordre croissant) <code>tab</code> . La méthode retourne l'indice de l'élément s'il existe, et <code>- pointDInsertion-1</code> si la valeur <code>val</code> ne se trouve pas dans le tableau (si le tableau compte 8 cases, il renvoie -9). La valeur <code>pointDInsertion</code> est le rang où la clé serait ajoutée, si on l'ajoutait au tableau. C'est le rang du premier élément plus grand que l'élément cherché, ou la taille du tableau.

5.5.1 Remplir un tableau

La bibliothèque standard Java Arrays propose aussi une méthode `fill()`.

Elle permet de dupliquer une certaine valeur dans chaque cellule, ou dans le cas d'objets.

Exemple

```
long[] tab = new long[6];
Arrays.fill(tab, 0); // Rempli le tableau (les six cases) avec la valeur 0
// Manipulation de plages d'index (index de 2 à 5 non compris) avec
// la valeur 1
Arrays.fill(tab, 2, 5, 1);
// Affiche le contenu d'un tableau
System.out.println(Arrays.toString(tab));
```

5.5.2 Copier un tableau

La fonction `System.arraycopy()`, réalise des copies de tableau bien plus rapidement qu'une boucle `for`.

Exemple

```
long[] tab1 = new long[25];
long []tab2 = new long[25];
Arrays.fill(tab1, 8);           // Rempli le tableau tab1 avec la valeur 8

// Copie le tableau 1 dans le tableau 2
System.arraycopy(tab1, 0, tab2, 0, tab2.length);
```

5.5.3 Comparer des tableaux

`Arrays` fournit la méthode surchargée `equals()` pour comparer des tableaux entiers. Encore une fois, ces méthodes sont surchargées pour chacun des types de base, ainsi que pour les `Objects`. Pour être égaux, les tableaux doivent avoir la même taille et chaque élément doit être équivalent (au sens de la méthode `equals()`) à l'élément correspondant dans l'autre tableau (pour les types scalaires, la méthode `equals()` de la classe d'encapsulation du type concerné est utilisé ; par exemple, `Integer.equals()` est utilisé pour les `int`).

Exemple

```
long[] tab1 = new long[25];
long []tab2 = new long[25];
Arrays.fill(tab1, 8);           // Rempli le tableau tab1 avec la valeur 8
Arrays.fill(tab2, 8);

// Compare le tableau 1 et le tableau 2
if(Arrays.equals(tab1, tab2))
{
    System.out.println("les tableaux sont égaux");
}
```

6 Les collections d'objets

Les collections d'objets permettent de stocker des données de façon dynamique (pas de taille prédéfinie), on ne peut donc pas dépasser leur capacité.

6.1.1 L'interface **Iterator**

Un itérateur est un objet qui a pour rôle de parcourir une collection.

L'Iterator peut être utilisé pour les objets **List**, les objets **Map** et les Objets **Set**.

Les méthodes de l'Interface **Iterator** (super classe de **ListIterator**)

Méthode	Description
hasNext()	Renvoie true s'il y a des éléments suivants. Sinon, retourne false.
next()	Renvoie l'élément suivant. Lève une exception <code>NoSuchElementException</code> s'il n'y a pas d'élément suivant.
remove()	Supprime l'élément courant. Lève une exception <code>IllegalStateException</code> si l'on appelle <code>remove ()</code> sans avoir fait un <code>next ()</code> auparavant.

Un **ListIterator** peut être utilisé pour les objets **List**.

Les méthodes de l'Interface **ListIterator**

Méthode	Description
add (Object obj)	Insère un élément dans la liste, à la place de l'élément qui sera renvoyée par le prochain appel à <code>next ()</code> .
hasNext()	Renvoie true s'il y a des éléments suivants. Sinon, retourne false.
hasPrevious ()	Renvoie true s'il y a au moins un élément précédent. Sinon, retourne false.
next()	Renvoie l'élément suivant. Lève une exception <code>NoSuchElementException</code> s'il n'y a pas d'élément suivant.
nextIndex ()	Renvoie l'index de l'élément suivant. S'il n'y a pas d'élément suivant, renvoie la taille de la liste.
previous ()	Renvoie l'élément précédent. Lève une exception <code>NoSuchElementException</code> est levée s'il n'y a pas d'élément précédent.
previousIndex ()	Renvoie l'index de l'élément précédent. S'il n'y a pas d'élément précédent, renvoie -1.
remove()	Supprime l'élément courant. Lève une exception <code>IllegalStateException</code> si l'on appelle <code>remove ()</code> sans avoir fait un <code>next ()</code> ou un <code>previous ()</code> auparavant.
set()	Affecte un élément à la place de l'élément courant. C'est le dernier élément retourné par un appel soit <code>next()</code> ou <code>previous()</code> .

6.2 Les objets List

Ces listes acceptent n'importe quel type de données, y compris **null**.

Objet	Description
ArrayList	L'ArrayList est rapide en lecture, même avec un gros volume de données, il fournit un accès aux éléments par leur indice très performant et est optimisé pour des opérations d'ajout/suppression d'éléments en fin de liste . il est plus lent (que les autres listes) si vous devez ajouter ou supprimer des données en milieu de liste.
LinkedList	Une LinkedList (liste chaînée) est une liste dont chaque élément est lié aux éléments adjacents par une référence vers l'élément précédent et à l'élément suivant (sauf le premier dont la première référence est égale à null et le dernier dont la dernière référence est égale à null). La LinkedList est utile lorsqu'il faut beaucoup manipuler une collection en supprimant ou en ajoutant des objets en milieu de liste , mais attention au ralentissement lorsqu'elle devient trop grande. L'ajout et la suppression d'éléments est aussi rapide quelle que soit la position , mais l'accès aux valeurs par leur indice est très lente.
Vector	Vector permet de stocker des objets dans un tableau dont la taille évolue au fil du programme. Dans tous les cas, il est préférable d'utiliser un ArrayList, car elle n'est conservée dans l'API actuelle que pour des raisons de compatibilité ascendante.

Ces listes implémentent l'interface **Iterator**, un itérateur est un objet qui a pour rôle de parcourir une collection.

Les principales méthodes d'un **ArrayList**

Méthode	Description
add(élément)	Ajoute l'élément à la fin de la liste.
add(indice, élément)	Ajoute l'élément à l'indice passé en paramètre.
addAll(Collection)	Permet d'insérer une collection à la liste.
clear()	Vide la liste.
contains(Object)	Renvoie un booléen qui indique si l'objet est contenu dans la liste.
get(int)	Renvoie l'élément dont l'indice est passé en paramètre.
indexOf(Object)	Renvoie l'indice de l'objet est passé en paramètre.
isEmpty()	Renvoie un booléen qui indique si la liste est vide
iterator()	Renvoie un itérateur qui permet de parcourir la liste.
lastElement()	Renvoie le dernier élément
lastIndexOf(Object)	Retourne l'index de la dernière occurrence de l'objet.
remove (int)	Supprime l'objet dont l'indice est passé en paramètre.
remove (élément)	Supprime le premier élément trouvé correspondant au paramètre.
size()	Renvoie le nombre d'éléments
toArray()	Retourne un tableau classique contenant les éléments de la liste.

6.2.1 Utilisation d'un ArrayList

Syntaxe

```
ArrayList <Type> nomListe = new ArrayList<Type>();
```

Exemple

```
import java.util.ArrayList;
import java.util.ListIterator;

...

// Déclaration de l'ArrayList
ArrayList <String> monArray = new ArrayList<String>();

// On ajoute des éléments à notre liste (à la fin)
monArray.add("Football");
monArray.add("Handball");
monArray.add("Basket-ball");
monArray.add("Rugby à XV");

// On affiche l'élément de la liste à la position 2
System.out.println("Élément à la position 2 : " + monArray.get(2));

// On ajoute Tennis à la position 2
monArray.add(2, "Tennis");

// On affiche l'élément de la liste à la position 2
System.out.println("Élément à la position 2 : " + monArray.get(2));

// L'élément a été décalé par l'insertion précédente
System.out.println("Élément à la position 3 : " + monArray.get(3) +
"\n");

// On déclare un itérateur pour parcourir l'ArrayList
ListIterator li = monArray.listIterator();

// On affiche le contenu de l'ArrayList
while(li.hasNext())
{
    System.out.println(li.next());
}

// On affiche le contenu de l'ArrayList avec un for-each
System.out.println();
for(String sp: monArray)
{
    System.out.println(sp);
}
```

6.3 Les objets map

Une map permet de créer un tableau associatif, un ensemble de couples clé/valeur.

La clé permettant de retrouver rapidement la valeur qui lui a été associée.

Les clés sont des objets uniques pouvant être **null**; Les valeurs peuvent être multiples et égale à null.

Objet	Description
HashMap	C'est la map standard, elle est adaptée à la plupart des situations .
Hashtable	Conservée pour des raisons de compatibilité ascendante et elle ne devrait pas être utilisée dans les nouveaux programmes.
IdentityHashMap	Cette map contrairement aux autres implémentations utilise l'opérateur l'opérateur == pour savoir si deux clés sont identiques, les autres utilisent la méthode equals().
LinkedHashMap	Elle conserve l'ordre d'ajout des clés. Si la clé ajoutée est déjà présente, l'ordre ne change pas.
TreeMap	Cette map possède une fonction de tri des clés de la map. L'ordre des clés peut être choisi en donnant une instance de Comparator sinon c'est l'ordre naturel des clés qui sera utilisé (elles doivent donc implémenter java.lang.Comparable). Si vous avez une grande quantité de données à ajouter dans la collection et que l'ordre clés n'est utile qu'après l'ajout, il est plus efficace de créer un HashMap pour ajouter les éléments et de construire la TreeMap à partir de la HashMap
WeakHashMap	Elle conserve les couples clé/valeur en utilisant des références faibles, donc si la clé n'est plus référencée ailleurs dans le programme, le couple est automatiquement supprimé de la collection

Les principales méthodes d'un HashMap

Méthode	Description
clear()	Vide le tableau.
containsKey(clé)	Renvoie true , si le tableau contient un élément correspondant à la clé spécifiée.
containsValue(valeur)	Renvoie true , si le tableau contient un élément correspondant à la valeur spécifiée.
get(clé)	Renvoie la valeur à laquelle la clé spécifiée est associée, ou null si ce tableau ne contient aucune correspondance pour la clé.
isEmpty()	Renvoie true , si le tableau n'a aucun couple clé/valeur.
keySet()	Renvoie l'ensemble des clés contenues dans cette map.
put(clé, valeur)	Permet d'insérer un élément dans le tableau, si celui-ci contient déjà cette clé, l'ancienne valeur est remplacée.
remove(clé)	Permet de supprimer un élément dans le tableau, renvoie la clé de l'élément supprimé ou null s'il n'a pas été trouvé.
size()	Renvoie le nombre d'éléments dans le tableau.
values()	Renvoie une collection des valeurs contenues dans la map.

Un exemple d'utilisation de HashMap

Exemple

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.ListIterator;

import java.util.Iterator;
import java.util.Map;

...

// Déclaration du HashMap
HashMap<Integer, String> sportMap = new HashMap<Integer, String>();

// Insertion des données
sportMap.put(11, "Football");
sportMap.put(7, "Handball");
sportMap.put(5, "Basket-ball");
sportMap.put(15, "Rugby à XV");

System.out.println("Affiche le nombre d'éléments du tableau : " + sportMap.size());

// Recherche la présence d'un élément
System.out.println("Handball présent ? " + sportMap.containsValue("Handball"));

// Affiche la valeur dont la clé est 5
System.out.println("clé 5 = " + sportMap.get(5));

System.out.println("Affiche le tableau : " + sportMap);

// Déclaration d'un ArrayList
ArrayList<String> monArray = new ArrayList<String>();

// On transfère le contenu de la map dans un ArrayList
monArray.addAll(sportMap.values());

// On déclare un itérateur pour parcourir l'ArrayList
ListIterator li = monArray.listIterator();

// On affiche le contenu de l'ArrayList
while(li.hasNext())
{
    System.out.println(li.next());
}

// En utilisant un Iterator
Iterator iter = sportMap.entrySet().iterator();
while (iter.hasNext()) {
    Map.Entry contenu = (Map.Entry) iter.next();
    System.out.println(contenu.getKey() + " -> " + contenu.getValue());
    //iter.remove();// Pour éviter les ConcurrentModificationException
}
```


Avec une boucle (style Foreach)

Exemple

```
// Déclaration du HashMap
HashMap<Integer, String> sportMap = new HashMap<Integer, String>();

// Insertion des données
sportMap.put(11, "Football");
sportMap.put(7, "Handball");
sportMap.put(5, "Basket-ball");
sportMap.put(15, "Rugby à XV");

for (Map.Entry< Integer, String> sport : sportMap.entrySet()) {
    System.out.println("Clé : " + sport.getKey() + " Valeur : " +
sport.getValue());
}
```

6.4 Les objets Set

Une collection set est équivalente à une collection map ou List mais elle **n'accepte pas les doublons**.

Les objets **HashSet**, **TreeSet**, **LinkedHashSet** sont particulièrement adaptés pour manipuler une grande quantité de données (sauf en insertion ou elles sont lentes).

Il faut redéfinir les méthodes equals() et hashCode() des objets comparés.

7 Le package

Le package permet d'utiliser les classes standard du langage Java ou vos propres classes, sans avoir à faire des copiés-collés dans votre programme source pour les inclure.

Un package est une unité (un fichier) regroupant des classes voisines, c'est-à-dire qui ont quelque chose en commun. Le mot clé package doit être la première instruction dans un fichier source, une classe ne peut pas appartenir à plusieurs packages.

Le nom d'un package correspond au nom d'un dossier de l'arborescence (il contient le dossier et les sous-dossiers).

Syntaxe

```
package nomPackage;
```

Utilisation d'un package

<pre>import nomPackage.*;</pre>	Toutes les classes du package sont importées.
<pre>import nomPackage.nomClasse;</pre>	Seule la classe citée est importée (plus rapide lors de la compilation).

`import java.*` **n'est pas une écriture valide**, car l'astérisque **n'importe pas les sous paquetages**.

7.1 Le package par défaut

Les fichiers présents dans le dossier **du projet** sont considérés comme faisant partie du **package par défaut**.

Les principaux paquetages Java
(<http://docs.oracle.com/javase/7/docs/>)

Package	Description
java.applet	Fournit les classes nécessaires pour créer une applet et les classes pour communiquer avec ses applets.
java.awt	Contient toutes les méthodes pour la création d'interfaces utilisateur et des images graphiques (paint).
java.awt.color	Fournit des classes pour les espaces de couleurs.
java.awt.datatransfer	Offre des interfaces et de classes pour transférer des données entre et au sein des applications.
java.awt.dnd	Pour la gestion des Drag and Drop.
java.awt.event	Offre des interfaces et de classes pour traiter avec différents types d'événements venant des composants AWT.
java.awt.font	Fournit des classes et d'interface relatives aux polices.
java.awt.image	Fournit des classes pour créer et modifier les images.
java.awt.print	Fournit des classes et interfaces pour une API d'impression.
java.lang	Fournit des classes qui sont fondamentales pour la conception du langage de programmation Java.
java.math	Fournit des classes pour l'exécution de calcul de précision entier arithmétique (BigInteger) et de précision arithmétique décimale (BigDecimal).
java.net	Fournit les classes pour la mise en œuvre des applications en réseau.
java.security	Fournit les classes et interfaces pour la sécurité.
java.sql	Fournit l'API d'accès et de traitement des données stockées dans une source de données (en général une base de données relationnelle) en utilisant le langage de programmation Java TM.
java.text	Fournit des classes et des interfaces pour le traitement de texte, des dates, des chiffres et des messages d'une manière indépendante des langues naturelles.
java.util	Contient les collections cadre, héritage de collecte des classes, un événement modèle, la date et l'heure, l'internationalisation, et diverses classes utiles (un string tokenizer, un générateur aléatoire, les tableaux).
java.util.jar	Fournit des classes pour la lecture et l'écriture des JAR (Java Archive) format de fichier, qui est basée sur la norme de format de fichier ZIP avec un fichier manifeste facultatif.
java.util.regex	Les classes pour faire correspondre les séquences de caractères avec les comportements spécifiés par des expressions régulières.

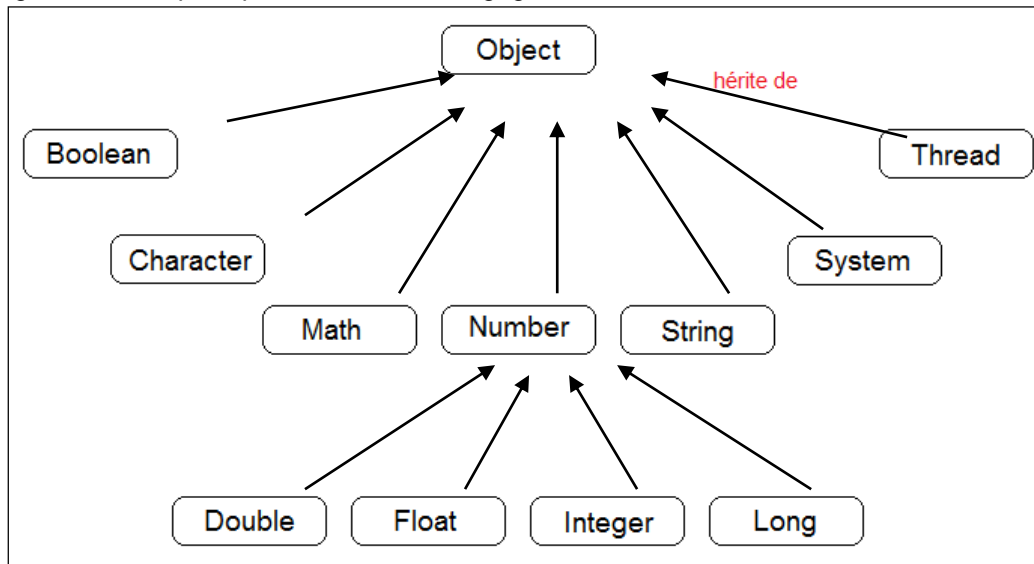
javax.imageio	Le paquet principal de Java Image I / O API.
javax.net	Fournit des classes pour la mise en réseau des applications.
javax.net.ssl	Fournit des classes pour le Secure Socket paquet.
javax.print	Fournit les principales classes et interfaces pour Java TM Print Service API.
javax.print.event	Fournit les classes et interfaces auditeur.
javax.security.auth.callback	Ce package fournit les classes nécessaires pour les services d'interaction avec des applications afin de récupérer des informations (données d'authentification y compris les noms d'utilisateur ou mots de passe, par exemple) ou d'afficher des informations (erreur et messages d'avertissement, par exemple).
javax.sql	Fournit l'API côté serveur pour la source de données l'accès et le traitement du langage de programmation Java TM.
javax.swing	Fournit un ensemble de "léger" (tout langage Java) des composants qui, dans la mesure du possible, fournit des travaux identiques sur toutes les plates-formes.
javax.swing.border	Fournit des classes et d'interface spécialisées pour l'élaboration des frontières autour d'un composant Swing.
javax.swing.colorchooser	Contient les classes et interfaces utilisées par le composant JColorChooser.
javax.swing.event	Fournit des événements tirés par les composants Swing.
javax.swing.plaf	Fournit une interface et de nombreuses classes abstraites que Swing utilise pour fournir à ses pluggable look-and-feel capacités.
javax.swing.plaf.basic	Offre à l'utilisateur l'interface des objets construits conformément à la base look and feel.
javax.swing.plaf.metal	Offre à l'utilisateur l'interface des objets construits conformément à Java look and feel (une fois de code Metal), qui est l'option par défaut look and feel.
javax.swing.table	Fournit des classes et interfaces pour faire face à javax.swing.JTable.
javax.swing.text	Fournit des classes et interfaces qui traitent avec éditables et noneditable texte composants.
javax.swing.tree	Fournit des classes et interfaces pour faire face à javax.swing.JTree.

8 Les bibliothèques de classes standard

Java contient un ensemble très complet de bibliothèques.

8.1.1 java.lang

java.lang contient les principales classes du langage.



Classe	Description
Boolean	Encapsule le type élémentaire boolean
Character	Encapsule le type élémentaire char .
Double	Encapsule le type élémentaire double fournit des méthodes de conversions.
Integer	Encapsule le type élémentaire int fournit des méthodes de conversions.
Float	Encapsule le type élémentaire float fournit des méthodes de conversions.
Long	Encapsule le type élémentaire long fournit des méthodes de conversions.
Math	Contient des fonctions et des constantes mathématiques.
String	Permet de mémoriser et utiliser les chaînes de caractères.
System	Contient les méthodes permettant d'interagir avec le système qui héberge l'application. Elle n'est pas instanciable.
Thread	Permet d'exécuter des tâches parallèles.

8.2 La classe Math

Cette classe contient des méthodes de calcul sur les entiers et les réels conformes à la norme IEEE 754 ce qui assure leur portabilité sur les clients.

Toutes les méthodes doivent être préfixées avec Math.

Les méthodes les plus utilisées (sauf indication contraire, les arguments sont de type double)

Méthode	Description
abs()	Renvoie la valeur absolue d'un nombre passé en argument. les types int, long, float et double sont admis comme arguments.
ceil()	Arrondie à l'entier supérieur (Renvoie un double sans partie décimale et non pas un entier).
exp ()	Renvoie la valeur exponentielle d'un nombre passé en argument.
floor ()	Arrondie à l'entier inférieur (Renvoie un double sans partie décimale et non pas un entier).
log ()	Renvoie le logarithme népérien d'un nombre passé en argument.
min(x,y)	Renvoie la valeur la plus petite de deux nombres passés en argument. les types int, long, float et double sont admis comme arguments.
max(x,y)	Renvoie la valeur la plus grande de deux nombres passés en argument. les types int, long, float et double sont admis comme arguments.
pow()	Elève le premier argument à la puissance (second argument). Si le premier argument est nul, le second doit être strictement positif. Si le premier argument est strictement négatif, le second doit être une valeur entière.
random ()	Génère un nombre pseudo-aléatoire compris entre 0 et 1.0. Il existe également dans le package java.util une classe Random qui permet une gestion plus complète des nombres aléatoires.
round ()	Arrondi à 0,5 près (Renvoie un double sans partie décimale et non pas un entier). round admet aussi des arguments de type "float" auquel cas la valeur retournée est du type "int".
sqrt ()	Renvoie la racine carrée d'un nombre passé en argument.

Exemple

```
double d=4.86, d2;  
d2=Math.round(d);      → d2 est égale à 5.0  
  
double d3 = 5.236145;  
double d3_arrondi = (Math.round(nb*100)/100.0); → d5 est égale à 5.24  
  
double d4=4, d5;  
d5=Math.pow (d,3.0);    → d5 est égale à 64.0  
  
double d6;  
d6=Math.random();      → d6 est égale à 0.5845182474323014
```

8.2.1 Fonctions trigonométriques (classe Math)

Ces fonctions travaillent toujours sur des valeurs en radians (les arguments sont de type double)

Méthode	Description
acos()	Renvoie l'arc cosinus d'un nombre passé en argument (valeur dans l'intervalle $[0, +\pi]$).
asin()	Renvoie l'arc sinus d'un nombre passé en argument (valeur dans l'intervalle $[-\pi/2, +\pi/2]$).
atan()	Renvoie l'arc tangente d'un nombre passé en argument (valeur dans l'intervalle $[-\pi/2, +\pi/2]$).
atan2()	Renvoie la valeur (en radians) de l'angle θ des coordonnées polaires (r et θ) du point correspondant aux coordonnées cartésiennes qui sont passées comme arguments.
cos()	Renvoie le cosinus d'un nombre passé en argument.
sin()	Renvoie le sinus d'un nombre passé en argument.
tan()	Renvoie la tangente d'un nombre passé en argument.

8.3 Les conversions de types(cast)

8.3.1 Conversion implicite

Une conversion implicite consiste en une modification du type de donnée effectuée automatiquement par le compilateur. Cela signifie que lorsque l'on va stocker un type de donnée dans une variable déclarée avec un autre type, le compilateur ne retournera pas d'erreur mais effectuera une conversion implicite de la donnée avant de l'affecter à la variable.

Une conversion de type implicite sans perte d'informations est réalisée d'un type primitif vers un type plus grand (élargissement) avec l'ordre croissant suivant sur les types :

1	byte	short	int	long	float	double
2	char	int				

La conversion de types, n'est pas le point fort de java, il faut faire très attention aux données que l'on manipule.

Une conversion peut entraîner une perte d'informations, mais le compilateur vous le signale (normalement).

8.3.2 Conversion explicite

Une conversion explicite (appelée aussi opération de cast) consiste en une modification du type de donnée forcée. Cela signifie que l'on utilise un opérateur dit de cast pour spécifier la conversion. L'opérateur de cast est tout simplement le type de donnée, dans lequel on désire convertir une variable, entre des parenthèses précédant la variable.

Une conversion **avec un cast** peut entraîner une perte d'informations, mais le compilateur **ne vous le signale pas**.

Syntaxe

```
variable = (type) valeur;  
ou  
variable = (type) variable;
```

Exemple

```
int i;  
i= (int) 2.9;  
  
→ i est égal à 2    (attention, attention)  
  
float f;  
f= (float) 2.9;  
→ f est égal à 2.9
```

Il n'existe pas en Java de fonction de conversions, pour réaliser une conversion, on utilise des **méthodes contenues dans des classes (wrapper** mot anglais signifiant *enveloppeur*). La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion des types élémentaires. Les enveloppeurs sont donc des objets pouvant contenir une primitive et auxquels sont associés des méthodes permettant de les manipuler, ils ont le même nom que le type élémentaire sur lequel ils reposent avec la première lettre en majuscule.

8.4 Classes wrapper

Ces classes wrapper sont des objets qui encapsulent les primitives de type correspondants, elles ajoutent les méthodes de la classe **Object** à vos types primitifs, ainsi que des méthodes permettant de caster leurs valeurs, **l'autoboxing**, une fonctionnalité qui permet de transformer automatiquement un type primitif en classe wrapper (**le boxing**) et transformer une classe wrapper en un type primitif (**l'unboxing**).

Classes	Description
Character	Pour convertir un char.
Double	Pour les nombres à virgule flottante en double précision (double).
Float	Pour les nombres à virgules flottante (float).
Integer	Pour les valeurs entières (integer).
Long	Pour les entiers longs signés (long).
String	Pour les chaines de caractères Unicode.

Ces classes contiennent généralement plusieurs constructeurs. Pour les utiliser, il faut les instancier puisque ce sont des objets.

Exemple

```
String ch = "3.14";  
float f = Float.parseFloat(ch);
```

f → 3.14

8.5 Classe Integer

Le constructeur convertit un int en objet Integer.

Les méthodes les plus utilisées

Méthode	Description
doubleValue()	Méthode d'instance qui permet de convertir un objet Integer en double.
floatValue()	Méthode d'instance qui permet de convertir un objet Integer en float.
intValue()	Méthode d'instance qui permet de convertir un objet Integer en int.
parseInt()	Méthode de classe qui convertit une chaîne en entier de type int.
toBinaryString ()	Méthode de classe qui convertit un objet Integer en une chaîne dans la base 2. [analogue à toString(l , 2)]
toString ()	Méthode de classe qui convertit le premier argument (objet Integer) en une chaîne dans la base de numération donnée par le second argument. Si la base est omise la conversion est faite en base 10.
valueOf ()	Méthode de classe qui convertit le premier argument (chaîne) en un objet Integer dans la base de numération donnée par le second argument. Si la base est omise la conversion est faite en base 10.

Il existe des méthodes similaires pour la classe Long.

Exemple

```
int entier=10;  
String chaine = Integer.toString(entier);    // Convertie mon int en String
```

8.6 Classe Double

Le constructeur convertit un double en objet Double.

Les méthodes les plus utilisées

Méthode	Description
intValue()	Méthodes d'instance qui permet de convertir un objet Double en int.
floatValue()	Méthodes d'instance qui permet de convertir un objet Double en float.
longValue()	Méthodes d'instance qui permet de convertir un objet Double en long.
doubleToLongBits ()	Méthode de classe qui convertit un double en sa représentation binaire au format IEEE. longBitsToDouble() assure la conversion inverse.
doubleValue()	Méthodes d'instance qui permet de convertir un objet Double en double.

toString()	Méthode de classe qui convertit un objet Double en une chaîne.
valueOf()	Méthode de classe qui convertit une chaîne en un objet Double. Cette méthode peut être combinée avec la méthode doubleValue() pour convertir une chaîne en double. d = Double.valueOf(s).doubleValue(); la partie soulignée assure la conversion de la chaîne en Double, la seconde partie de l'instruction assure la conversion du Double en double.

Il existe des méthodes similaires pour la classe Float.

Exemple

```
int d=10;
String chaine = Double.toString(d); // Convertie mon double en String
```

8.7 Classe Character

Le constructeur convertit un char en objet Character.

Les méthodes les plus utilisées

Méthode	Description
charValue()	Méthode d'instance qui convertit un Character en char.
digit()	Méthode de classe qui retourne la valeur numérique du Character passé comme premier argument en utilisant le second argument comme base de numération [bases de 2 (MIN_RADIX) à 36 (MAX_RADIX)]. Si la conversion est impossible, la méthode retourne - 1.
forDigit()	Méthode de classe qui retourne le char correspondant à l'entier passé comme premier argument en utilisant le second argument comme base de numération (bases de 2 à 36). Si la conversion est impossible, la méthode retourne le char null.
isDigit()	Méthode de classe booléenne qui teste si le char passé en argument est un chiffre.
isLowerCase()	Méthode de classe booléenne qui teste si le char passé en argument est une minuscule.
isUpperCase()	Méthode de classe booléenne qui teste si le char passé en argument est une majuscule.
isSpace()	Méthode de classe booléenne qui teste si le char passé en argument est un espace.
toLowerCase()	Méthodes de classe qui modifie si nécessaire la casse du char passé en argument (en minuscule).
toString()	Méthode de classe qui convertit un objet Character en chaîne.
toUpperCase()	Méthodes de classe qui modifie si nécessaire la casse du char passé en argument (en majuscule).

8.8 Les méthodes de la classe String

En fait Java propose 3 classes apparentées aux chaînes de caractères.

- La classe String (chaîne de caractères non modifiables (**immutable**))
- La classe StringBuffer (chaîne de caractères modifiables à volonté).
Attention, si vous manipulez des chaînes (modification, concaténation ...) et à plus forte raison des chaînes de grande taille, vous devez utiliser des StringBuffer.
- La classe StringTokenizer (séparation d'une chaîne en plusieurs entités)

Dans la plupart de cas il convient d'utiliser String pour créer, stocker et gérer des chaînes de caractères.

On n'utilise pas les caractères égal (==) pour comparer deux chaînes de caractères, mais la méthode equals.

8.9 Classe String

Il faut inclure la classe java.lang.String.

Dans les méthodes :

- pos : indique une position (entier).
- ch : une chaîne de caractères.
- d : position de début (entier).
- f : position de fin (entier).

Dans les exemples :

```
String ch1="Bonjour"; // Chaîne de référence
String ch2=""; // Chaîne vide
String ch3="Une chaîne de caractères"; // Chaîne avec plusieurs mots
char car; // Une variable caractère
int res; // Une variable de type int
boolean verif; // Une variable booléenne
```

Les principales méthodes applicables à une instance de la classe String.

Méthode	Description	Exemple
charAt(pos)	Renvoie le caractère de la position pos passée en paramètre. Le premier caractère est à la position 0.	<pre>car=ch1.charAt(3);</pre> <p>→ car est égale à j</p>
compareTo(ch)	Renvoie : <ul style="list-style-type: none">• une valeur positive si la chaîne de référence est supérieure à la chaîne passée en paramètre ch (selon l'ordre des caractères Unicode),• une valeur négative si la chaîne est inférieure à la chaîne passée en paramètre ch (selon l'ordre des caractères Unicode),• 0 si les deux chaînes sont identiques. Attention les majuscules sont placées avant les minuscules.	<pre>res=ch1.compareTo("autre");</pre> <p>→ res est égale à 1</p> <pre>res=ch1.compareTo("Bonjour");</pre> <p>→ res est égale à 0</p>
compareToIgnoreCase()	Identique à compareTo(ch) , mais ne tient pas compte de la casse.	
endsWith(ch)	Renvoie true , si la chaîne finit par le bout de chaîne passé en paramètre, sinon renvoie false .	<pre>verif=ch3.endsWith("ères");</pre> <p>→ verif est égale à true</p>

equals(ch)	Renvoie true si les deux chaines sont égales caractère à caractère.	<pre>verif=ch1.equals("Bonjour");</pre> → verif est égale à true
equalsIgnoreCase()	Renvoie true si les deux chaines sont égales caractère à caractère, quelque soit la casse.	<pre>verif=ch1.equals("bONjoUr");</pre> → verif est égale à true
indexOf(pos)	Renvoie la position de la chaine passée en paramètre, dans la chaine de référence et -1 si elle n'est pas trouvée. On peut indiquer un second paramètre pour dire à partir de quelle position commence la recherche. Le premier caractère est à la position 0.	<pre>res=ch1.indexOf("j");</pre> → res est égale à 3
isEmpty()	Renvoie true , si la longueur de la chaine vaut 0 sinon renvoie false .	<pre>verif=ch2.isEmpty();</pre> → verif est égale à true
lastIndexOf(ch)	Renvoie la position de la chaine passée en paramètre, dans la chaine de référence en partant de la droite et -1 si elle n'est pas trouvée. Le premier caractère est à la position 0.	<pre>res=ch1.lastIndexOf("o");</pre> → res est égale à 4
length()	Renvoie le nombre de caractères présents dans la chaine.	<pre>res=ch1.length();</pre> → res est égale à 7
split()	Permet de découper une chaine en plusieurs parties. On doit définir en paramètres ce qui détermine la coupure.	<pre>String d="25/07/2012"; String[] tab=d.split("/"); tab={"25","07","2012"};</pre>
startsWith(ch)	Renvoie true , si la chaine commence par le bout de chaine passé en paramètre, sinon renvoie false .	<pre>ch3.startsWith("Une");</pre> → verif est égale à true
substring(d,f)	Renvoie une chaîne qui contient les caractères de la position d , à la position f-1 . Le premier caractère est à la position 0.	<pre>String chx=ch1.substring(1,4);</pre> → chx est égale à onj
toLowerCase()	Convertit une chaine de caractères en minuscules.	<pre>String chx=ch1.toLowerCase();</pre> → chx est égale à bonjour
toUpperCase()	Convertit une chaine de caractères en majuscules.	<pre>String chx=ch1.toUpperCase();</pre> → chx est égale à BONJOUR
trim()	Renvoie une chaîne sans les espaces en trop (en début et/ou en fin de chaîne).	<pre>String chx=" abc "; chx=chx.trim();</pre> → chx est égale à "abc"

Attention aux String qui sont des chaînes de caractères non modifiables (immutable)

Exemple

```
String ch1="BONJOUR";

ch1.toLowerCase();           → ch1 est égale à
BONJOUR

String chx=ch1.toLowerCase(); → chx est égale à bonjour
```

8.9.1 La classe StringBuffer ou StringBuilder

Si l'on manipule des chaînes (**modification**, **concaténation** ...) et à plus forte raison des chaînes de **grande taille**, vous devez utiliser des StringBuffer ou StringBuilder.

Ces deux classes possèdent les mêmes méthodes, mais StringBuffer est fait pour les programmes multi-thread alors que StringBuilder est fait pour les programmes mono-thread.

Les StringBuffer ou StringBuilder sont des chaînes de caractères modifiables à volonté (mutable).

Exemple

```
StringBuffer ch1= new StringBuffer("Bonjour");    // Chaîne de
référence
StringBuffer ch2= new StringBuffer(); // Chaîne vide
// Chaîne avec plusieurs mots
StringBuffer ch3= new StringBuffer("Une chaîne de caractères");
char car;    // Une variable caractère
int res; // Une variable de type int
boolean verif; // Une variable booléenne
```

Méthode	Description	Exemple
append()	Permet d'ajouter la chaîne passée en paramètre à la chaîne.	<pre>ch2.append ("Bonjour "); ch2.append ("tout le monde.");</pre> → Bonjour tout le monde
charAt(pos)	Renvoie le caractère de la position pos passée en paramètre. Le premier caractère est à la position 0.	<pre>car=ch1.charAt(3);</pre> → car est égale à j
equals(ch)	Renvoie true si les deux chaînes sont égales caractère à caractère.	<pre>verif=ch1.equals("Bonjour");</pre> → verif est égale à true
delete(d,f)	Permet de supprimer les caractères entre la position d et f - 1.	<pre>ch2.append("Bonjour tout le monde"); ch2.delete(8,13);</pre> → Bonjour le monde
deleteCharAt(d)	Permet d'effacer un caractère en spécifiant sa position.	<pre>ch2.append("Bonjour toutZ le monde"); ch2.deleteCharAt(12);</pre> → Bonjour le monde
indexOf(pos)	Renvoie la position de la chaîne passée en paramètre, dans la chaîne de référence et -1 si elle n'est pas trouvée. On peut indiquer un second paramètre pour dire à partir de quelle position commence la recherche. Le premier caractère est à la position 0.	<pre>res=ch1.indexOf("j");</pre> → res est égale à 3
insert(d,ch)	Permet d'insérer la chaîne passée en paramètre à la position d.	<pre>ch2.append ("Bonjour le monde"); ch2.insert (8, "tout");</pre> → Bonjour tout le monde
lastIndexOf(ch)	Renvoie la position de la chaîne passée en paramètre, dans la chaîne de référence en partant de la droite et -1 si elle n'est pas trouvée. Le premier caractère est à la position 0.	<pre>res=ch1.lastIndexOf("o");</pre> → res est égale à 4

<code>length()</code>	Renvoie le nombre de caractères présents dans la chaîne.	<code>res=ch1.length();</code> → res est égale à 7
<code>reverse()</code>	Inverse la chaîne de caractères.	<code>ch1.reverse();</code> → ch1 est égale <code>ruojnoB</code>
<code>replace(d,f,ch)</code>	Remplace les caractères d'une StringBuffer par les caractères passés en paramètre. La sous-chaîne est placée à la position de début spécifié jusqu'à la position de fin - 1 ou à la fin de StringBuffer.	<code>ch3.replace(4, 10, "liste");</code> → ch3 est égale à Une liste de caractères
<code>setCharAt(d, ch);</code>	Remplace un caractère d'une StringBuffer par le caractère passé en paramètre à la position d.	<code>ch2.append("Bonjour touS le monde");</code> <code>ch2.setCharAt(11, 't');</code> → Bonjour tout le monde
<code>substring(d,f)</code>	Renvoie une chaîne qui contient les caractères de la position d , à la position f-1 . Le premier caractère est à la position 0.	<code>String chx=ch1.substring(1,4);</code> → chx est égale à onj

8.10 La classe BigDecimal

Ces objets permettent de manipuler des nombres de très grande taille, en évitant de perdre de la précision dans les calculs.

Il faut inclure la classe **java.math.BigDecimal** (sous-classe java.lang.Number).

Les calculs sont **plus lents** qu'avec les types de bases, mais sont **plus précis**.

Méthode	Description
abs()	Renvoie la valeur absolue de l'objet BigDecimal.
add()	Renvoie la somme de l'objet BigDecimal et de celui qui est passé en paramètre à la méthode.
compareTo()	Compare l'objet BigDecimal avec celui qui est passé en paramètre à la méthode.
divide()	Renvoie le résultat de la division de l'objet BigDecimal par celui qui est passé en paramètre à la méthode.
doubleValue()	Convertit un BigDecimal en un double.
equals()	Compare l'objet BigDecimal avec celui qui est passé en paramètre à la méthode. Renvoie true, s'ils sont égaux.
floatValue()	Convertit un BigDecimal en un float.
hashCode()	Retourne le code de hachage de l'objet BigDecimal.
intValue()	Convertit un BigDecimal en un int.
longValue()	Convertit un BigDecimal en un long.
max()	Compare l'objet BigDecimal avec celui qui est passé en paramètre à la méthode. Renvoie le plus grand des deux.
min ()	Compare l'objet BigDecimal avec celui qui est passé en paramètre à la méthode. Renvoie le plus petit des deux.
movePointLeft()	Renvoie un BigDecimal qui est équivalent à celui-ci avec la virgule déplacée de x (nombre) positions vers la gauche.
movePointRight ()	Renvoie un BigDecimal qui est équivalent à celui-ci avec la virgule déplacée de x (nombre) positions vers la droite.
multiply()	Renvoie le résultat de la multiplication de l'objet BigDecimal et de celui qui est passé en paramètre à la méthode.
negate()	Renvoie la valeur négative de l'objet (on multiplie par -1).
scale()	Renvoie la taille du BigDecimal.
setScale()	Définie la taille du BigDecimal (l'échelle est la valeur spécifiée).
signum()	Renvoie le signe du BigDecimal.
subtract()	Renvoie le résultat de la soustraction de l'objet BigDecimal par celui qui est passé en paramètre à la méthode.

toBigInteger()	Convertit un BigDecimal en un BigInteger.
unscaledValue()	Renvoie un BigInteger dont la valeur est la valeur sans virgule du BigDecimal.
valueOf ()	Renvoie une valeur de type long du BigDecimal avec une échelle de zéro. Si l'on passe un second paramètre, on peut spécifier l'échelle.

8.11 Les calculs mathématiques en Java

Attention, Java fait ses calculs en fonction des types des variables que vous utilisez et des types par défaut qu'il considère.

Exemple

```
x=30*(24*3600*1000);    // Calcul fait en int

x=30.0*(24.0*3600.0*1000.0);    // Calcul fait en double

x=30L*(24L*3600L*1000L);    // Calcul fait en long

x=30f*(24f*3600f*1000f);    // Calcul fait en float
```

De nombreuses erreurs de calcul, sont dues au fait qu'il n'est pas possible de représenter exactement un nombre réel en float ou en double.

Exemple

```
float f1 =3.14f;
float f2=1.0f;
f1= f1 + f2;    → f1 est égale à 4.1400003
```

La méthode BigDecimal permettra de résoudre le problème.

Exemple

```
float f1 =3.14f;
float f2=1.0f;

BigDecimal bd1 = new BigDecimal(Float.toString(f1));
BigDecimal bd2 = new BigDecimal(Float.toString(f2));

bd1= bd1.add(bd2);
→ bd1 est égale à 4.14
```



Attention, les chiffres que vous voulez représenter doivent être passés comme des Strings dans le constructeur

8.12 Formater les nombres pour l’affichage

Pour formater une valeur numérique dans un format de son choix, il faut utiliser un pattern.

Il faut inclure la classe **java.text.DecimalFormat**.

Pattern les plus utilisés

Pattern	Description
#	Permet de représenter un chiffre en ignorant les zéros inutiles.
0	Permet de représenter un chiffre qui devra obligatoirement être présent, même s'il s'agit d'un zéro inutile.
.	Le point (.), permet de représenter le séparateur décimal.
,	La virgule (,) permet de représenter le séparateur des groupes (milliers, millions, milliards).

Exemple

```
double pi= 3.14159265;  
DecimalFormat df = new DecimalFormat("0.00");  
g.drawString(df.format(pi), 25,150);  
→ affichage : 3.14
```

Exemples de pattern

pattern	Nombre à afficher et résultat de l’affichage				
	0	0,02	0,8	12,9	1546,858
#	0	0	1	13	1547
###	0	0	1	13	1547
0	0	0	1	13	1547
000	000	000	001	013	1547
#.##	0	0,02	0,8	12,9	1546,86
0.##	0	0,02	0,8	12,9	1546,86
0.00	0,00	0,02	0,80	12,90	1546,86
#.00	,00	,02	,80	12,90	1546,86
#,##0.00	0,00	0,02	0,80	12,90	1 546,86

Par défaut, la partie entière sera toujours agrandie même si le pattern précise un nombre d’éléments moins important.

Toutes les caractéristiques du pattern peuvent être modifiées par les méthodes de la classe **DecimalFormat**. La classe **NumberFormat** permet d'obtenir un certain nombre de formatage standard via des méthodes statiques.

9 Gestion des erreurs (exceptions)

Java, comme de nombreux langages de programmation de haut niveau possèdent un mécanisme permettant de gérer **les erreurs** (une exception représente une erreur) qui peuvent intervenir lors de l'exécution d'un programme.

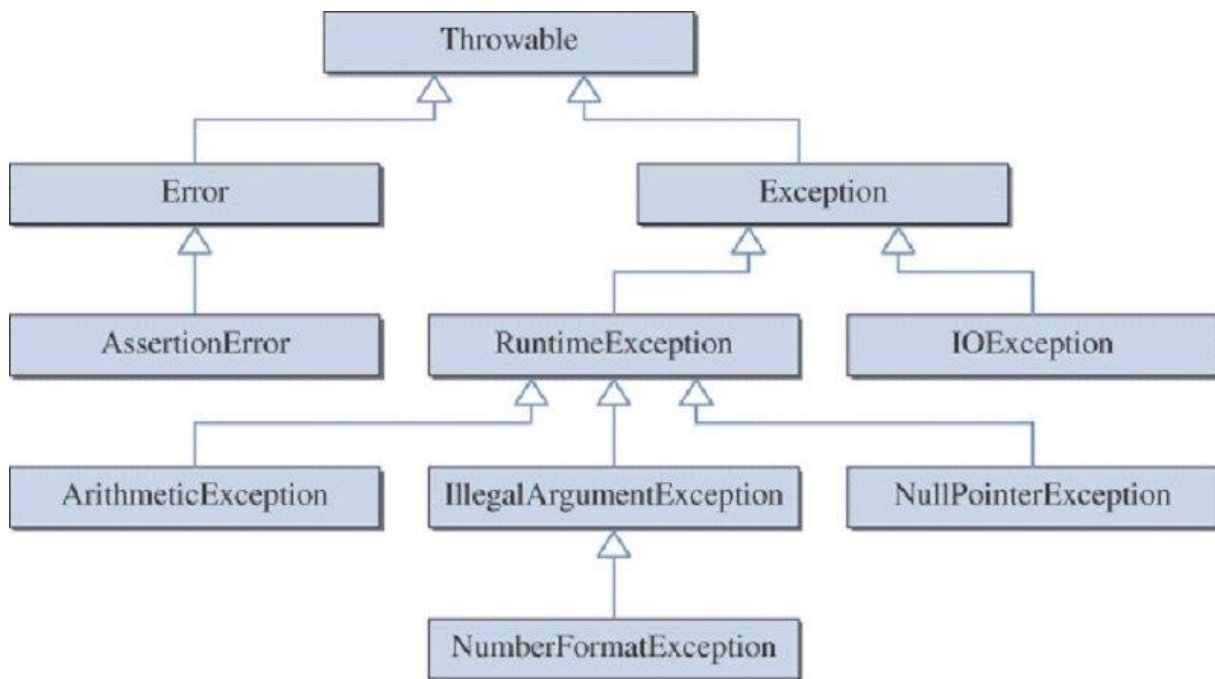
Il existe de nombreuses classes d'exceptions prédéfinies en Java, que l'on peut classer en trois catégories :

- Celles définies en étendant la classe **Error** : elles représentent des **erreurs critiques** qui ne sont pas censées être gérées en temps normal.
Exemple : l'exception `OutOfMemoryError` est levée lorsqu'il n'y a plus assez de mémoire disponible pour le programme.
- Celles définies en étendant la classe **Exception** : elles représentent les **erreurs qui doivent normalement être gérées par le programme**.
Exemple : l'exception `IOException` est levée si une erreur survient lors de la lecture d'un fichier qui est introuvable ou illisible.
- Celles définies en étendant la classe **RuntimeException** : elles représentent des erreurs pouvant éventuellement être gérées par le programme.
Exemple : l'exception `NullPointerException` est levée si l'on tente d'accéder au contenu d'un objet qui vaut null.
La division par zéro est une `ArithmeticException`.

9.1.1 Quelques exceptions prédéfinies

Exception	Description
ArithmeticException	Une division par zéro provoque une exception de type.
ArrayIndexOutOfBoundsException	Accès à une case inexistante dans un tableau. Accès au N ^{ième} caractère d'une chaîne de caractères de taille inférieure à N. Création d'un tableau de taille négative.
IOException	Erreur d'entré/sortie, ne trouve pas un fichier ...
NullPointerException	Accès à un champ ou appel de méthode non statique sur un objet valant null. Utilisation de <code>length</code> ou accès à une case d'un tableau valant null.
NumberFormatException	erreur lors de la conversion d'une chaîne de caractères en nombre.

9.1.2 Schéma des exceptions



9.1.3 L'instruction try

Lorsqu'une méthode est susceptible de déclencher une erreur (exception), il faut utiliser un bloc **try** pour la tester et **un ou plusieurs** blocs **catch** pour capturer l'exception qui vient d'être levée (on dit aussi intercepter), afin de signaler l'erreur, faire un autre traitement.

On peut ajouter un bloc **finally** pour indiquer un traitement final qui **sera exécuté dans tous les cas**.

Syntaxe

```
try
{
    // Code ou fonction risquant provoquer une erreur
}
catch (typeException1 [| typeException2] variableException1)
{
    // Mon traitement si cette erreur intervient
}
[catch (typeException2 variableException2)
{
    // Mon traitement si cette erreur intervient
}]
[finally
{
    // Code final exécuté dans tous les cas
}]
```

Les crochets [] indiquent des options facultatives. (Il ne faut pas les saisir, même si vous utilisez l'option).

La liste de toutes les exceptions est consultable à l'adresse suivante :

- <https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html>

Exemple : Utilisation simple

```
import java.net.URL;
...
URL monUrl;
...
try
{
    // Définition de l'Url d'un site à atteindre
    monUrl = new URL("http://www.louvre.fr");

    // On change la page courante du navigateur avec notre URL
    getAppletContext().showDocument(monUrl, "_self");
}
catch (Exception ex)
{
    // On affiche un message d'erreur
    System.out.println("Erreur : " + ex.toString());
}
```

Exemple : Utilisation un peu plus complexe (avec deux blocs catch).

```
import java.net.URL;
import java.net.MalformedURLException;

...

URL monUrl;
...
try
{
    // Définition de l'Url d'un site à atteindre
    monUrl = new URL("http://www.louvre.fr");

    // On change la page courante du navigateur avec notre URL
    getAppletContext().showDocument(monUrl, "_self");
}
catch (MalformedURLException ex)
{
    // On affiche un message d'erreur pour l'Url mal formée
    System.out.println("Url mal formée : " + ex.toString());
}
catch (Exception ex)
{
    // On affiche un message d'erreur
    System.out.println("Erreur : " + ex.toString());
}
```

9.1.4 Définir une exception

Dans certains cas particuliers, il peut être utile de déclarer sa propre exception.

Syntaxe

```
public class NomClasseException extends Exception {
    public NomClasseException(String message)
    {
        super(message);
    }
}
```

Déclarer qu'une méthode peut déclencher (lever) une exception et (éventuellement) lancer une exception.

Syntaxe

```
public NomMéthode throws Exception {
    ...
    throw new Exception("Message de l'erreur" );
}
```

Dans le cas où une méthode peut lever une exception, il est intéressant de le déclarer, cela permet de mettre à jour la Javadoc et de l'indiquer à quelqu'un qui utilisera cette méthode.

Exemple

```
/**
 * Méthode qui calcule la moyenne
 * @throws ArithmeticException si nombre est égal à zéro
 */
public double moyenne(int total, int nombre) throws
ArithmeticException {
    return total / nombre;
}
```

Dans certains cas, il peut être intéressant de déclencher sa propre exception, pour gérer une erreur.

Exemple

```
public double baisserStock(int nombre) throws Exception {
    stock = stock - nombre;
    if (stock < 0)
    {
        throw new Exception("Stock négatif");
    }
}
```

10 Le système de fichier

10.1 La classe File

La classe File permet de manipuler le système de fichier (déplacement, copie, etc.) mais ne permet pas d'accéder au contenu des fichiers.

L'objet *File* permet de manipuler le système de fichier. File peut être soit un dossier soit un fichier, aucune distinction n'est faite entre ces deux éléments. La documentation de la classe *File* est ici :

<http://docs.oracle.com/javase/7/docs/api/java/io/File.html>

10.1.1 Instanciation

Pour un système de fichier Windows

Le caractère \ étant un caractère réservé, il faut le banaliser en le doublant, on remplace donc les \ par \\

```
File fichier = new File("D:\\dossier\\fichier.txt");
File dossier = new File("D:\\dossier");
```

Pour un système de fichier unix/linux

```
File fichier = new File("/home/dossier/fichier.txt");
File dossier = new File("/home");
```

10.1.2 Méthodes principales de la classe File

Méthode	Description
delete()	Supprime le fichier sur le système de fichiers
deleteOnExit()	Supprime le fichiers sur le système de fichiers lors de la fermeture de la JVM
exists()	Retourne vrai si le fichier ou le dossier existe sur le système de fichiers
getName()	Retourne le nom du fichier ou du dossier
getParent()	Retourne le nom du dossier parent ou null s'il n'existe pas de parent
isDirectory()	Retourne vrai si le path associé à l'objet file est un dossier
isFile()	Retourne vrai si le path associé à l'objet file est un fichier
list()	Retourne un tableau de String contenant les noms des dossiers et des fichiers dans le répertoire désigné par le path
listFiles()	Retourne un tableau de File contenant les noms des dossiers et des fichiers dans le répertoire désigné par le path
mkdir()	Crée le répertoire désigné par le path
mkdirs()	Crée le répertoire désigné par le path avec les répertoires parents s'ils n'existent pas
renameTo(File destination)	Renomme ou déplace le fichier sur le système de fichier

10.2 Lister les fichiers d'un dossier

Pour lister les fichiers d'un répertoire, on utilise la méthode `list()` ou `listFiles`

Exemple d'utilisation de `list()`

Les fichiers et les dossiers de `D:\dossier1` sont affichés dans la console :

```
File dossier = new File("D:\\dossier1");
String[] contenu = dossier.list();
System.out.println(Arrays.toString(contenu));
```

Exemple d'utilisation de `listFiles()`

Les fichiers et les dossiers de `D:\dossier1` sont listés dans la console, les fichiers sont précédés de "Fichier :." et les dossiers de "Dossier :."

```
File dossier = new File("D:\\dossier1");
File[] contenu = dossier.listFiles();
for (File f : contenu){
    if(f.isDirectory()){
        System.out.println("Dossier : "+f.getName());
    }
    else {
        System.out.println("Fichier : "+f.getName());
    }
}
```

10.3 Déplacer un fichier

10.3.1 Sur un même Système de fichier

```
File fichier = new File("D:\\dossier1\\fichier.txt");
fichier.renameTo(new File("D:\\dossier1\\dossier\\fichier.txt"));
```

10.3.2 Sur un système de fichier différent

Si le système de fichier est différent, il faut déplacer le contenu du fichier sur l'autre système de fichier. Dans ce cas, l'objet `File` n'est pas suffisant.

Dans ce cas, il faudra copier le fichier vers la destination et supprimer le fichier initial. (Pour la copie, voir la section suivante)

10.4 Copier un fichier

La copie d'un fichier nécessite un accès aux données du FileSystem, la classe File ne gère pas ce fonctionnement.

```
FileChannel source=null;
FileChannel destination=null;
try {
    source = new FileInputStream("D:\\dossier1\\fichier.txt").getChannel();
    destination = new FileOutputStream("C:\\dossier2\\f.txt").getChannel();
    source.transferTo(0, source.size(), destination);
}
catch (FileNotFoundException e1) {
    e1.printStackTrace();
}
catch (IOException e) {
    e.printStackTrace();
}
finally {
    if(source != null) {
        try {
            source.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
    if(destination != null) {
        try {
            destination.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

10.5 Suppression d'un fichier

```
File fichier = new File("D:\\dossier1\\fichier.txt");
fichier.delete();
```

10.6 Manipulation des fichiers texte

10.6.1 Lire un fichier

```
/* Le chemin vers le fichier à lire */
String pathFichier="D:\\fichier.txt";

BufferedReader fluxEntree=null;
try {
    /* Création du flux vers le fichier texte */
    fluxEntree = new BufferedReader(new FileReader(pathFichier));

    /* Lecture d'une ligne */
    String ligneLue = fluxEntree.readLine();

    while(ligneLue!=null){
        System.out.println(ligneLue);
        ligneLue = fluxEntree.readLine();
    }
} catch(IOException exc){
    exc.printStackTrace();
} finally{
    try{
        if(fluxEntree!=null){
            /* Fermeture du flux vers le fichier */
            fluxEntree.close();
        }
    } catch(IOException e){
        e.printStackTrace();
    }
}
```

10.6.2 Ecrire un fichier

```
PrintWriter out=null;
try{
    /* Path vers le fichier à créer */
    String pathFichier = "D:\\fichier.txt";

    /* Ouverture du fichier en écriture */
    out = new PrintWriter(new BufferedWriter(new FileWriter(pathFichier)));

    /* Ecriture d'une ligne puis saut de ligne */
    out.println("Ligne 1");

    /* Ecriture d'une ligne sans saut de ligne */
    out.print("Ligne 2");
    out.println(" suite de la ligne 2");
} catch(IOException exc){
    exc.printStackTrace();
} finally {
    if(out!=null){
        /* Fermeture du flux */
        out.close();
    }
}
```

10.6.3 Ajouter une ligne à un fichier

```
/* Chemin vers le fichier à modifier */
String pathFichier = "D:\\fichier.txt";

/* Texte à ajouter */
String aAjouter = "Texte a ajouter";
FileWriter writer = null;
try {
    /* Ouverture du fichier en écriture */
    writer = new FileWriter(pathFichier, true);

    /* Ajout du contenu */
    writer.write(aAjouter, 0, aAjouter.length());
}
catch (IOException ex) {
    ex.printStackTrace();
}
finally {
    if (writer != null) {
        try {
            /* Fermeture du flux */
            writer.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

10.7 Les fichiers properties

Les fichiers .properties sont des fichiers dans lesquelles on retrouve des propriétés. Comme par exemple des propriétés de connexion à une base de données, à un serveur distant, etc ...

Exemple de fichier properties

```
exercice=exo java;
anneeEnCours=2018;
db.url=jdbc:mysql://localhost:3306/nomDB
db.username=user
db.password=password
```

10.7.1 La classe Properties

Cette classe permet de récupérer chaque propriété du fichier lu. Il faut utiliser la méthode "load" en lui passant en paramètre le inputStream du fichier.

La méthode getProperty("nom paramètre",null) permet de lire le paramètre.

exemple

```
Properties prop = new Properties();
try {
    InputStream inputStream = new FileInputStream("c:\\data\\data.properties");
    prop.load(inputStream);
    String exercice = prop.getProperty("exercice", null);
    System.out.println(exercice);
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Si la propriété exercice n'existe pas alors la méthode getProperty retournera null par défaut.

10.7.2 Ecrire dans un fichier properties

```
final Properties prop = new Properties();
OutputStream output = null;

try {
    output = new FileOutputStream("c:\\data\\data.properties");
    // set the properties value
    prop.setProperty("db.database", "localhost");
    prop.setProperty("db.buser", "david");
    prop.setProperty("db.password", "123");

    // save properties to project root folder
    prop.store(output, null);
} catch (final IOException io) {
    io.printStackTrace();
} finally {
    if (output != null) {
        try {
            output.close();
        } catch (final IOException e) {
            e.printStackTrace();
        }
    }
}
```

11 Les annotations

11.1 Qu'est-ce qu'Annotation ?

Les annotations de Java sont utilisées pour fournir des métadonnées pour votre code Java. Étant des méta-données, les Annotations n'influent pas directement sur l'exécution de votre code, bien que certains types d'annotations puissent effectivement être utilisés à cette fin.

Les Annotations de Java ont été ajoutées à Java à partir de Java 5.

Les annotations de Java sont typiquement utilisées aux buts suivants :

- ✓ Utilisation par le compilateur pour détecter des erreurs ou ignorer des avertissements
- ✓ Documentation
- ✓ Génération de code
- ✓ Génération de fichiers

La version 5 de Java EE fait un important usage des annotations dans le but de simplifier les développements de certains composants notamment les EJB, les entités et les services web. Pour cela, l'utilisation de descripteurs est remplacée par l'utilisation d'annotations ce qui rend le code plus facile à développer et plus clair.

11.1.1 Instructions pour la documentation

Les annotations peuvent être mises en oeuvre pour permettre la génération de documentations indépendantes de JavaDoc : listes de choses à faire, de services ou de composants, ...

Il peut par exemple être pratique de rassembler certaines informations mises sous la forme de commentaires dans des annotations pour permettre leur traitement.

Par exemple, il est possible de définir une annotation qui va contenir les métadonnées relatives aux informations sur une classe. Traditionnellement, une classe débute par un commentaire d'en-tête qui contient des informations sur l'auteur, la date de création, les modifications, ... L'idée est de fournir ces informations dans une annotation dédiée. L'avantage est de facilement extraire et manipuler ces données qui ne seraient qu'informatives sous leur forme de commentaires.

11.1.2 Instructions pour le compilateur

Java possède 3 Annotations intégrées que vous pouvez utiliser pour donner des instructions au compilateur Java.

- @Deprecated
- @Override
- @SuppressWarnings

11.1.3 Instruction en temps de construction (Build-time)

Les annotations Java peuvent être utilisées au moment de la construction (Build-time), lorsque vous créez votre projet logiciel.

Le processus de construction comprend :

- ✓ La génération du code source
- ✓ La compilation de la source
- ✓ La génération de fichiers XML (par exemple, des descripteurs de déploiement)
- ✓ Le packaging du code compilé et des fichiers dans un fichier JAR, etc.

La construction du logiciel est généralement effectuée par un outil de construction automatique comme Apache Ant ou Apache Maven.

Les outils de construction peuvent scanner votre code Java pour des annotations spécifiques et générer du code source ou d'autres fichiers basés sur ces annotations.

11.1.4 Instructions d'exécution (Runtime)

Normalement, les annotations de Java ne sont pas présentes dans votre code Java après la compilation. Il est toutefois possible de définir vos propres annotations disponibles au moment de l'exécution. Ces annotations peuvent ensuite être consultées via Java Reflection et utilisées pour donner des instructions à votre programme ou à une API tierce.

11.2 Les annotations standards

Il existe 3 annotations importantes de Java

- ✓ @Deprecated
- ✓ @Override
- ✓ @SuppressWarnings

11.2.1 @Deprecated

Il s'agit d'une annotation utilisée pour annoter quelque chose obsolète comme classe ou méthode. Et pour le mieux, nous ne devrions plus l'utiliser. Si vous utilisez quelque chose obsolète, le compilateur vous informera que vous devriez utiliser une autre manière. Ou avec la programmation IDE comme Eclipse, cela vous montre également des notifications visuelles.

Les entités marquées avec l'annotation @Deprecated devraient être documentées avec le tag @deprecated de Javadoc en lui fournissant la raison de l'obsolescence et éventuellement l'entité de substitution.

Exemple

```
/**
 * @deprecated remplacée par {@link #nouvelleMethode(String,String)}
 */
@Deprecated
public void ancienneMethode(String nom) {
    System.out.println("nom " + nom);
}

public void nouvelleMethode(String nom, String prenom) {
    System.out.println("nom " + nom + " prenom " + prenom);
}
```

11.2.2 @Override

L'annotation @Override est utilisée sur des méthodes qui remplacent les méthodes dans une superclasse (superclass). Si la méthode ne correspond pas à une méthode dans la superclasse, le compilateur vous donnera une erreur.

L'annotation @Override n'est pas nécessaire pour remplacer une méthode dans une superclasse. C'est pourtant une bonne idée de l'utiliser encore. Dans le cas où quelqu'un a changé le nom de la méthode surchargée dans la superclasse, votre méthode de sous-classe ne la remplacerait plus. Sans l'annotation @Override, vous ne le verrez pas. Avec l'annotation @Override, le compilateur vous dirait que la méthode dans la sous-classe ne remplace aucune méthode dans la superclasse.

11.2.3 @SuppressWarnings

L'annotation @SuppressWarnings fait que le compilateur supprime les avertissements pour une méthode donnée. Par exemple, si une méthode appelle une méthode obsolète, ou crée un type non sécurisé, le compilateur peut générer un avertissement. Vous pouvez supprimer ces avertissements en annotant la méthode contenant le code avec l'annotation @SuppressWarnings.

Exemples

```
@SuppressWarnings("deprecation")
@SuppressWarnings({ "deprecation", "unused", "unchecked" })
```

Le compilateur fourni avec le JDK supporte les avertissements suivants :

Nom	Rôle
deprecation	Vérification de l'utilisation d'entités déclarées deprecated
unchecked	Vérification de l'utilisation des generics
fallthrough	Vérification de l'utilisation de l'instruction break dans les cases des instructions switch
path	Vérification des chemins fournis en paramètre du compilateur
serial	Vérification de la définition de la variable serialVersionUID dans les beans
finally	Vérification de l'absence d'instruction return dans une clause finally

11.3 Les annotations communes

Les annotations communes sont intégrées dans Java 6.

11.3.1 L'annotation @Generated

De plus en plus d'outils ou de frameworks génèrent du code source pour faciliter la tâche des développeurs notamment pour des portions de code répétitives ayant peu de valeur ajoutée. Le code ainsi généré peut-être marqué avec l'annotation @Generated.

11.3.2 Les annotations @Resource et @Resources

L'annotation @Resource définit une ressource requise par une classe. Typiquement, une ressource est par exemple un composant Java EE de type EJB ou JMS.

L'annotation @Resource possède plusieurs attributs :

Attribut	Description
authenticationType	Type d'authentification pour utiliser la ressource (Resource.AuthenticationType.CONTAINER ou Resource.AuthenticationType.APPLICATION)
description	Description de la ressource
mappedName	Nom de la ressource spécifique au serveur utilisé (non portable)
name	Nom JNDI de la ressource
shareable	Booléen qui précise si la ressource est partagée
type	Le type pleinement qualifié de la ressource

11.3.3 Les annotations @PostConstruct et @PreDestroy

Les annotations @PostConstruct et @PreDestroy permettent respectivement de désigner des méthodes qui seront exécutées après l'instanciation d'un objet et avant la destruction d'une instance.

Une telle méthode doit respecter certaines règles :

- ✓ Ne pas avoir de paramètres sauf dans des cas précis
- ✓ Ne pas avoir de valeur de retour
- ✓ Ne doit pas lever d'exceptions vérifiées
- ✓ Ne doit pas être statique

11.4 Écrire Votre Annotation

L'utilisation de @interface est un mot-clé de déclaration d'une Annotation, et cette annotation est assez similaire à une interface. Une annotation peut avoir ou non des éléments (element).

Les caractéristiques des éléments (element) d'une annotation :

- ✓ Il n'y a pas de corps de fonction
- ✓ Il n'y a pas de paramètre fonctionnel
- ✓ La déclaration de retour doit être un type spécifique :
 - Type primitif (boolean, int, float, ...)
 - Enum
 - Annotation
 - Classe (Par exemple String.class)
- ✓ L'élément peut avoir des valeurs par défaut

11.5 Votre première Annotation

[MyFirstAnnotation.java](#)

```
public @interface MyFirstAnnotation {  
  
    // L'élément 'name'.  
    public String name();  
  
    // L'élément 'description', sa valeur par défaut est "".  
    public String description() default "";  
  
}
```

L'annotation peut être utilisée sur :

1. TYPE - Classe, interface (y compris le type d'annotation) ou déclaration d'énumération.
2. FIELD - Champ de déclaration (field), inclut des constantes d'énumération.
3. METHOD - Déclaration de méthode.
4. PARAMETER - Déclaration de paramètre
5. CONSTRUCTOR - Déclaration de constructeur
6. LOCAL_VARIABLE - Déclaration de variable locale.
7. ANNOTATION_TYPE - Déclaration d'Annotation
8. PACKAGE - Déclaration de package.

Exemple d'utilisation de l'annotation

```
@MyFirstAnnotation(name = "Some name", description = "Some description")
public class UsingMyFirstAnnotation {

    // L'annotation est annotée sur un Constructeur.
    // La valeur de l'élément 'name' est "John"
    // La valeur de l'élément 'description' est "Write by John".
    @MyFirstAnnotation(name = "John", description = "Write by John")
    public UsingMyFirstAnnotation() {

    }

    // L'annotation est annotée sur une méthode.
    // La valeur de l'élément 'name' est "Tom"
    // L'élément 'description' n'est pas déclaré, on lui attribuera une valeur par défaut.
    @MyFirstAnnotation(name = "Tom")
    public void someMethod() {

    }

    // Une annotation est annotée sur le paramètre de méthode.
    public void todo(@MyFirstAnnotation(name = "none") String job) {

        // Une annotation est annotée sur une variable locale.
        @MyFirstAnnotation(name = "Some name")
        int localVariable = 0;

    }

}
```

12 JDBC (Java DataBase Connectivity)

JDBC est une interface de programmation (API) qui permet aux applications Java d'accéder à des bases de données en utilisant une interface commune (les requêtes sont adaptées en fonction de la base de données visée) quel que soit la base de données stockées dans un SGBD (en utilisant le pilote adapté).

JDBC permet :

- La connexion à la base de données.
- L'envoi de requêtes SQL à la base de données depuis une application Java.
- La gestion des données, des erreurs ...


12.1 Ajout de la dépendance dans le pom.xml

Le fichier est à récupérer sur le site officiel d'oracle et le copier-coller dans le répertoire `.m2/repository/com/oracle/version`

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc14</artifactId>
  <version>10.2.0.4.0</version>
</dependency>
```

Pour récupérer la dépendance maven pour mysql

- Taper jdbc mysql maven repo sur google
- Puis sur le site sélectionner la version de mysql correspondante

**MySQL Connector/J**
JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
Tags	mysql database connector driver
Used By	3,238 artifacts

Central (70) | Jahia (1)

	Version	Repository	Usages	Date
8.0.x	8.0.13	Central	28	Sep, 2018
	8.0.12	Central	39	Jun, 2018
	8.0.11	Central	74	Apr, 2018
	8.0.9-rc	Central	0	Jan, 2018
	8.0.8-dmr	Central	14	Sep, 2017
	8.0.7-dmr	Central	6	Jun, 2017
6.0.x	6.0.6	Central	157	Feb, 2017
	6.0.5	Central	59	Oct, 2016
	6.0.4	Central	18	Aug, 2016

12.2 Connexion à la base de données

Pour se connecter à la base de données, on utilise l'interface DriverManager, elle s'occupe du chargement des pilotes et permet de créer de nouvelles connexions à des bases de données. Elle tient à jour, la liste des pilotes JDBC recensés du système.

Méthode	Description
getConnection()	Permet de créer une connexion vers la base de données, on passe à la méthode l'URL de connexion, le login et le password,
registerDriver()	Permet d'enregistrer le pilote (passé en argument) auprès du gestionnaire de pilotes JDBC de la machine Java, s'il est déjà enregistré, il n'est pas ajouté.

Exemple

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

    Connection conn;

    String DOMAINE = "jdbc:oracle:thin:@localhost:1521:xe";
    String LOGIN = "basetest";
    String PASSWORD = "123";

    try {
        Log.info("open connection ");
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(DOMAINE, LOGIN, PASSWORD);
        if(conn.isClosed()) {
            System.out.println("La connexion est fermée");
        }else {
            System.out.println("La connexion est ouverte");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e1) {
        e1.printStackTrace();
    }
    finally {
        Log.info("connection ok");
        System.out.println("connection ok");
    }

}
```

Si une erreur se produit, elle déclenche une exception :

- **SQLException: Connection refused**
- **Connection timed out**
- **CommunicationsException: Communications link failure.**

Solutions
Vérifier que le serveur MySQL est démarré
Vérifier le fichier de configuration my.cnf du serveur MySQL (vérifiez le numéro du port).
Test de la connectivité (ping adressehote).
Vérifier que le pare-feu et/ou proxy ne bloquent pas le port MySQL.
Utiliser l'adresse IP à la place du nom d'hôte.

12.3 Exécuter une requête SQL

Il faut utiliser certaines méthodes de la classe Connection pour créer des d'objets qui permettent l'exécution de requêtes SQL.

12.3.1 Méthodes de la classe Connection pour créer des objets de requêtage

Méthode	Description
createStatement()	Cette méthode permet d'obtenir un objet implémentant l'interface Statement .
prepareStatement()	Cette méthode permet d'obtenir un objet implémentant l'interface PreparedStatement .
prepareCall()	Cette méthode permet d'obtenir un objet implémentant l'interface CallableStatement .

Les objets pour exécuter des requêtes

Objet	Description
Statement	Cet objet permet d'exécuter des requêtes prédéfinies (qui ne contient aucune partie variable).
PreparedStatement	Cet objet permet d'exécuter des requêtes contenant des paramètres (variable) correspondant à la saisie de l'utilisateur.
CallableStatement	Cet objet permet d'exécuter des procédures stockées (requête stockée dans la base de données).

12.3.2 Principales méthodes de l'objet Statement

Objet	Description
execute()	<p>Méthode qui permet l'exécution d'une requête prédéfinie passée en paramètre. Retourne true si la requête a renvoyée un premier enregistrement de type ResultSet (un jeu de résultat), sinon retourne false.</p> <p>Elle est intéressante pour les requêtes d'insertion (INSERT). Le deuxième paramètre autoGeneratedKeys permet d'obtenir en retour la clé autogénérée (typiquement un auto-incrément de la clé primaire). La méthode retourne toujours un booléen. Si la réponse est true, alors le premier résultat est un jeu de résultats.</p> <p>Si la requête exécutée est simplement une instruction INSERT, ce jeu de résultats contient la clé autogénérée. Le ResultSet est accessible au travers de la méthode getGeneratedKeys(). Il contient une unique ligne et une unique colonne avec la valeur de la clé.</p>
executeQuery()	<p>Cette méthode est adaptée pour exécuter les requêtes SQL de type SELECT. Elle retourne directement le résultat sous la forme d'un ResultSet.</p>
executeUpdate()	<p>Cette méthode destinée à exécuter les requêtes SQL de types INSERT, UPDATE, DELETE. Elle retourne un entier correspondant au nombre de lignes affectées par l'instruction.</p> <p>Cette méthode est rarement utilisée car les insertions, mises à jour et suppressions sont généralement des requêtes dynamiques.</p> <ul style="list-style-type: none">• ObjStatement.executeUpdate("requete") <p>ou</p> <ul style="list-style-type: none">• ObjStatement.executeUpdate("requete", paramètre) <p>Le paramètre permet de demander à la méthode de renvoyer la valeur de l'index créé ou pas.</p> <p>Paramètre :</p> <ul style="list-style-type: none">○ Statement.RETURN_GENERATED_KEY○ SetStatement.NO_GENERATED_KEYS
getResultSet()	<p>Retourne un objet ResultSet (contenant les données retournées par la requête sous forme de lignes et de colonnes et un curseur pour se déplacer dans le ResultSet) ou null s'il n'y a pas de données.</p>
getUpdateCount()	<p>Retourne le nombre de lignes mise à jour.</p>
getMoreResults()	<p>Retourne true s'il s'agit d'un objet ResultSet et ferme implicitement tout objet ResultSet actuel obtenu avec la méthode getResultSet.</p>

12.3.3 Principales méthodes d'un ResultSet

En fonction de l'objet sur lequel on crée notre ResultSet (Statement, prepareStatement ou Call), le ResultSet aura un type défini par **trois caractéristiques**, qui définissent les **possibilités de déplacement**, les **possibilités de mise à jour** et le **maintien des curseurs** lors des transactions.

Méthode	Description
afterLast()	Place le curseur après le dernier élément.
beforeFirst()	Place le curseur avant le premier élément (position par défaut du curseur).
close()	Permet de vider/supprimer un ResultSet.
first()	Place le curseur sur le premier élément.
get<Type>()	Il existe de nombreuses méthodes pour récupérer la valeur d'un champ en spécifiant le type. <ul style="list-style-type: none">• <code>NomResultSet.getInt("nom_champ")</code>• <code>NomResultSet.getDouble("nom_champ")</code>• <code>NomResultSet.getLong("nom_champ")</code>• <code>NomResultSet.getString("nom_champ")</code>• ...
getRow()	Retourne l'index du curseur.
isAfterLast()	Retourne true , si le curseur est après la dernière ligne, sinon retourne false .
isBeforeFirst()	Retourne true , si le curseur est avant la première ligne, sinon retourne false .
isFirst()	Retourne true , si le curseur est sur la première ligne, sinon retourne false .
isLast()	Retourne true , si le curseur est sur la dernière ligne, sinon retourne false .
last()	Place le curseur sur le dernier élément.
next()	Déplace le curseur sur l'élément suivant .
previous()	Déplace le curseur sur l'élément précédent .

12.3.3.1 Récupérer des données de la base de données (SELECT)

Exemple

```
try {
    Statement stat = conn.createStatement();
    ResultSet res = stat.executeQuery("select * from articles");
    System.out.println(res.toString());
    while (res.next())
    {
        String nom = res.getString("nom_art");
        System.out.println(nom);
    }
} catch (SQLException e) {
    e.printStackTrace();
}finally{
    conn.close();
}
```

12.3.3.2 Modifier des données de la base de données (UPDATE)

En modifiant la requête, on peut facilement ajouter (INSERT) ou supprimer (DELETE) des données.

Exemple

```
...
// On crée notre Statement
Statement stat = cnx.createStatement();

// On exécute une requête de type UPDATE et on stocke le résultat dans un Entier
int flag = stat.executeUpdate("update clients set ville_cli = 'MONACO' where n_cli = 5");

// Si l'entier supérieur ou égal à 1, la modification a eu lieu
// (Nombres de lignes affectées par la requête SQL)
if (flag >= 1)
{
    System.out.println("La ligne a été modifiée");
}
else
{
    System.out.println("Pas de modification");
}

// On ferme de Statement
stat.close();
...
```

12.4 PreparedStatement (Requête préparée)

L'interface PreparedStatement permet de **protéger** le système contre les **injections SQL**, d'utiliser les paramètres pour passer les valeurs à la requête en **contrôlant leur type** et enfin notre requête sera **pré-compiler** ce qui diminuera le temps d'exécution.

12.4.1 Principales méthodes de l'objet PreparedStatement

Une instance de PreparedStatement contient une instruction SQL déjà compilée. qui améliore les performances, si cette instruction doit être appelée de nombreuses fois.

Objet	Description
execute()	Méthode qui permet l'exécution d'une requête prédéfinie passée en paramètre. Retourne true si la requête a renvoyée un premier enregistrement de type ResultSet (un jeu de résultat), sinon retourne false.
executeQuery()	Cette méthode est adaptée pour exécuter les requêtes SQL de type SELECT . Elle retourne directement le résultat sous la forme d'un ResultSet .
executeUpdate()	Cette méthode destinée à exécuter les requêtes SQL de types INSERT , UPDATE , DELETE . Elle retourne un entier correspondant au nombre de lignes affectées par l'instruction ou null s'il n'y a pas eu de modification. <ul style="list-style-type: none">ObjPreparedStatement.executeUpdate("requete") ou <ul style="list-style-type: none">ObjPreparedStatement.executeUpdate("requete", paramètre) Le paramètre permet de demander à la méthode de renvoyer la valeur de l'index créé ou pas. Paramètre :<ul style="list-style-type: none">Statement.RETURN_GENERATED_KEYSetStatement.NO_GENERATED_KEYS

set<Type>()	<p>Il existe de nombreuses méthodes pour spécifier la valeur d'un champ en spécifiant le type.</p> <ul style="list-style-type: none"> • <code>NomResultSet.setInt(Numéro, valeur)</code> • <code>NomResultSet.setDouble(Numéro, valeur)</code> • <code>NomResultSet.setLong(Numéro, valeur)</code> • <code>NomResultSet.setString(Numéro, "valeur")</code> • <code>NomResultSet.setNull(Numéro, java.sql.Types.INTEGER) ;</code> • ...
-------------	--

Les instructions SQL des instances de PreparedStatement contiennent un ou plusieurs paramètres d'entrée, non spécifiés lors de la création de l'instruction. Ces paramètres sont représentés par des points d'interrogation (?) remplacés par la valeur du paramètre.

Ces paramètres doivent être spécifiés avant l'exécution, ils sont numérotés de 1 à n (nombre de paramètres).

Syntaxe

```
PreparedStatement ps = objConnexion.prepareStatement( "requete SQL... ? ... ? ... ?" ) ;

ps.setType(1, valeur1) ;
ps.setType(2, valeur2) ;
ps.setType(3, valeur3) ;
```

Exemple : Lecture des données de La base de données avec une requête préparée (SELECT)

```
String rechClient = "D%";
// Création de notre PreparedStatement pour gérer la requête préparée
String requete = "select * from clients where nom_cli like ?";
PreparedStatement pstat = (PreparedStatement) conn.prepareStatement(requete);

// On définit le paramètre pour le passer à la requête
pstat.setString(1, rechClient);

// On exécute une requête de type SELECT et on stocke le résultat dans un ResultSet
ResultSet res = pstat.executeQuery();
while (res.next())
{
    long id = res.getLong("n_cli");
    String nom = res.getString("nom_cli");
    String ville = res.getString("ville_cli");
    System.out.println(id + " " + nom + " " + ville);
}
```

Exemple : Insertion des données avec une requête préparée (INSERT)

En modifiant la requête, on peut facilement modifier (UPDATE) ou supprimer (DELETE) des données.

```
// Ici on initialise directement les variables avec une valeur, mais dans un
// cas réel ces valeurs viendraient d'un formulaire rempli par l'utilisateur
String nom = "mon article java";
String des = "ma designation java";
Double prix = 123.50;
int stock = 5;

String requete = "INSERT INTO articles VALUES(article_seq.nextval, ?,?,?,?)";
// Création de notre PreparedStatement pour gérer la requête préparée avec
// Statement.RETURN_GENERATED_KEYS pour récupérer l'id créé automatiquement
PreparedStatement pstmt = (PreparedStatement) conn.prepareStatement( requete);

// On définit les paramètres pour les passer à la requête
pstmt.setString(1, nom);
pstmt.setString(2, des);
pstmt.setDouble(3, prix);
pstmt.setInt(4, stock);

// On exécute une requête de type INSERT et on stocke le résultat dans un Entier
// (Nombres de lignes affectées par la requête SQL)
int flag = pstmt.executeUpdate();

// Si l'entier supérieur ou égal à 1, la modification a eu lieu
if (flag >= 1){
    System.out.println("La ligne a été ajoutée");
    conn.commit();
}
else System.out.println("Pas d'ajout");
```

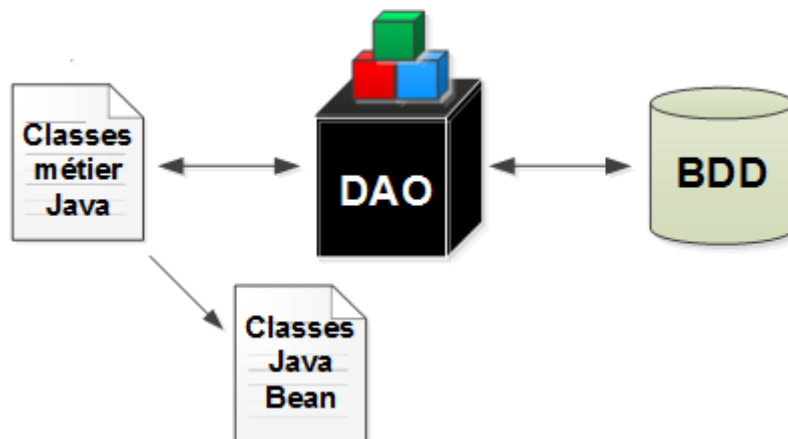
12.5 Design pattern (modèle de conception) DAO

Le modèle DAO (**D**ata **A**ccess **O**bject) permet de faire le lien entre la couche métier et le système de stockage des données.

Dans la solution précédente, le code des **traitements métier** et le code qui gère le **stockage des données** sont mélangés, ce qui pose quelques problèmes.

On ne peut pas :

- Changer de type de stockage (Autre système de base de données), sans une réécriture complète de tout le modèle.
- Le code devient difficile à maintenir et à faire évoluer (autres demandes dans une nouvelle version).
- Impossible de faire des tests unitaires :
 - Faire des tests sur le code métier sans faire intervenir la gestion de la base de données.
 - Faire des tests sur le stockage des données sans faire intervenir le code métier.



Schémas du DAO

Le DAO va agir comme une boîte noire, la classe métier va demander au DAO, de modifier, lire, ajouter ... des données et le DAO se charge de ces opérations et retourne une information indiquant si l'opération s'est bien déroulé ou pas, la classe métier pourra modifier ou récupérer les informations dans les Beans.

12.6 Le Design Pattern Singleton

Le **singleton** a pour objectif de restreindre l'instanciation d'une classe à un seul objet (parfois à quelques objets). Cette solution permet de réduire la quantité de mémoire utilisée et d'accélérer le système puisque l'on utilise un seul objet au lieu de plusieurs.

La classe **ConnexionOracle** va nous permettre de créer une connexion unique à notre base de données.

Exemple

```
public class ConnexionOracle {

    private static final Logger Log = Logger.getLogger(ConnexionOracle.class);

    private static Connection conn;

    private final static String DOMAINE = "jdbc:oracle:thin:@localhost:1521:xe";
    private final static String LOGIN = "basetest";
    private final static String PASSWORD = "123";

    public static Connection getConnection(){

        if (conn == null) {
            try {
                Log.info("open connection ");
                Class.forName ("oracle.jdbc.driver.OracleDriver");
                conn = DriverManager.getConnection(DOMAINE,LOGIN,PASSWORD);
                conn.setAutoCommit(false);
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ClassNotFoundException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            finally {
                Log.info("connection ok");
                System.out.println("connection ok");
            }
        }
        return conn;
    }

    public static void closeConnection() {
        try {
            conn.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

Cette classe permet de lancer des exceptions, pour **masquer aux utilisateurs** le type de bases de données en cas **d'erreur de connexion**.

Exemple

```
public class ErreurBase extends RuntimeException {  
  
    // Exception qui envoie un message  
    public ErreurBase(String message) {  
        super(message);  
    }  
  
    // Exception qui envoie un message et l'erreur  
    public ErreurBase(String message, Throwable erreur) {  
        super(message, erreur);  
    }  
  
    // Exception qui envoie l'erreur  
    public ErreurBase(Throwable erreur)  
    {  
        super(erreur);  
    }  
}
```

Cette classe permet de lancer des exceptions, pour masquer aux utilisateurs les **erreurs d'exécution en SQL**.

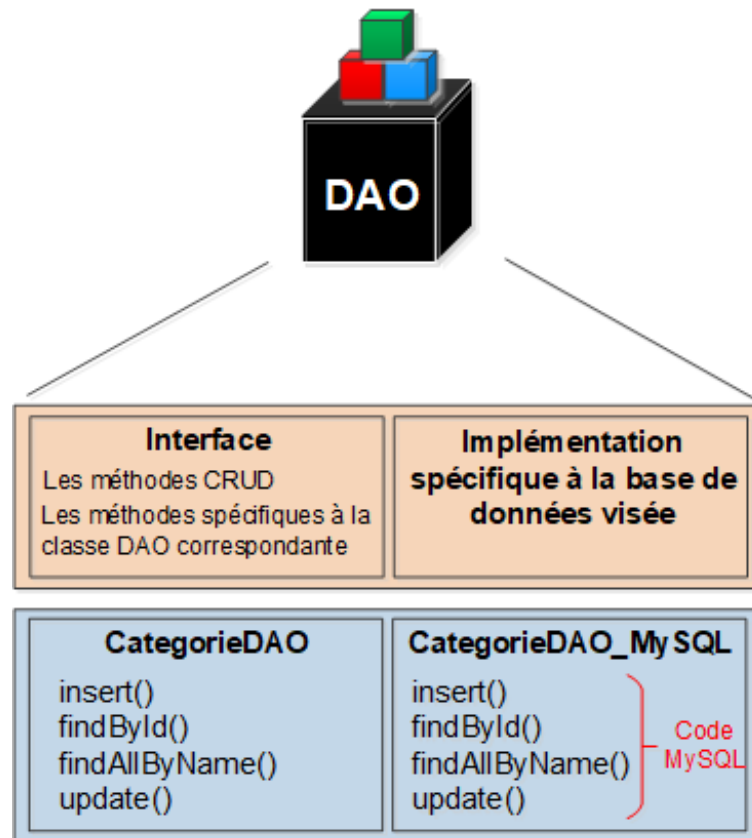
Exemple

```
public class DAOException extends RuntimeException {  
  
    // Exception qui envoie un message  
    public DAOException(String message) {  
        super(message);  
    }  
  
    // Exception qui envoie un message et l'erreur  
    public DAOException(String message, Throwable erreur) {  
        super(message, erreur);  
    }  
  
    // Exception qui envoie l'erreur  
    public DAOException(Throwable erreur) {  
        super(erreur);  
    }  
}
```

L'interface DAO va nous permettre de définir les **méthodes** que devront obligatoirement définir les classes qui implémenteront la classe DAO, c'est une interface car on ne créera aucun objet à partir de cette classe.

Cette classe met en œuvre (le plus souvent) les fameuses méthodes CRUD (Create → création, Read → Lecture, Update → modification, Delete → suppression) et des méthodes spécifiques propre à la classe.

On créera une interface par JavaBean, car les besoins de chaque Bean sont différents.



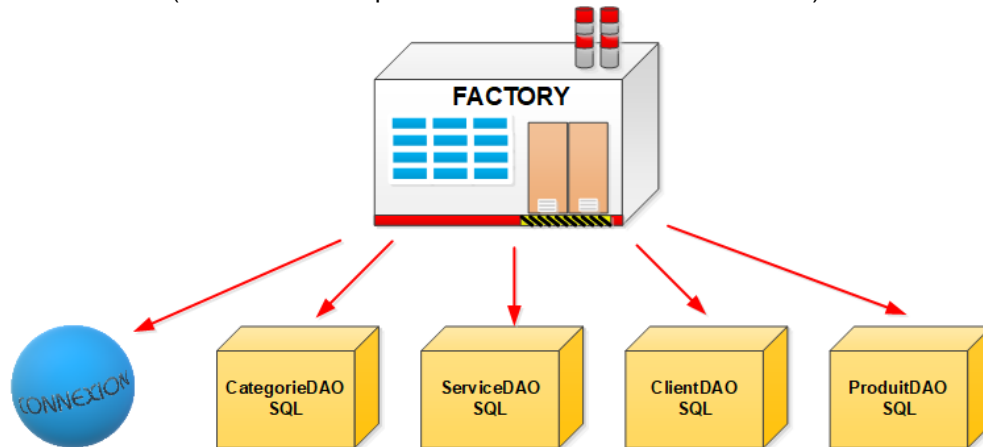
Exemple

```
public interface ArticlesDAO {  
    Long insert(Article art) throws DAOException;  
    Article findById(Long id) throws DAOException;  
    List<Article> findAllByName(String ch) throws DAOException;  
    Boolean update(Article articles) throws DAOException;  
}
```

Si la base de données venait à changer, il suffirait de créer une nouvelle classe DAO contenant les méthodes déclarées dans l'interface avec les commandes adaptées à cette base de données.

12.7 Design pattern Factory

La **Factory** va avoir pour rôle d'encapsuler l'**instanciation** de tous les objets **DAO** de notre application (ce qui permet de se préparer sur les éventuelles futures modifications sur la façon de créer les objets car l'instanciation de nos DAO sera centralisée dans un seul objet) et **fournir une connexion UNIQUE à la base de données** (elle va donc remplacer notre classe ConnexionOracle).



Notre classe **DAOFactory.java**

Exemple

```
package fr.epsi.poe.exempleJDBC.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import org.apache.log4j.Logger;

public class DaoFactory {

    private static final Logger log = Logger.getLogger(DaoFactory.class);

    private static Connection conn;

    private final static String DOMAINE = "jdbc:oracle:thin:@localhost:1521:xe";
    private final static String LOGIN = "basetest";
    private final static String PASSWORD = "123";

    public static DaoFactory getInstance()
    {
        DaoFactory fact = new DaoFactory();
        return fact;
    }

    public static Connection getConnection(){

        if (conn == null) {
            try {
                log.info("open connection ");
                Class.forName ("oracle.jdbc.driver.OracleDriver");
                conn = DriverManager.getConnection(DOMAINE,LOGIN,PASSWORD);
                conn.setAutoCommit(false);
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ClassNotFoundException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
    }
}
```

```

        finally {
            log.info("connection ok");
            System.out.println("connection ok");
        }
    }
    return conn;
}

public static void closeConnection() {
    try {
        conn.close();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

/*
 * Méthodes pour l'implémentation des différents DAO
 */

// On définit quel DAO On utilise (si un changement doit avoir lieu, il se fera ici uniquement)
public ArticleDao getArticleDao()
{
    return new ArticleDaoImpl();
}
}

```

La classe ConnexionOracle.java ne sert plus à rien.

On remplace

Dans le fichier ArticleDaoImpl.java , on remplace
Connection conn = ConnexionOracle.getConnection();
par
Connection conn = DAOFactory.getConnection();

On n'instancie plus la classe DAO dans le programme principal

Dans le programme principal , on remplace
ArticleDao DarticleDao = new ArticleDaoImpl();
par
ArticleDao DarticleDao = DAOFactory.getInstance().getArticleDao();