# Les bases du language

Une "cheatsheet" sur la syntaxe du language, et toutes les bases. Ce chapitre ne traite pas l'orienté objet.

# Introduction

Ceci sont mes notes de Java. J'ai essayé de les écrire sous le même format que Rust by example.

Vous pouvez trouver mes solutions et celle d'autres personnes sur mon Git :

- codeberg.org/SnowCode/prb

Pour sélectionner les personnes ou différentes versions du code, changer la "branche" Git.

Bonne lecture, amusez vous bien :)

# Hello World

Pour le cours de Java on va utiliser OpenJDK 18. On peut l'installer depuis le site de java OpenJDK. JDK veut dire "Java Development Kit" et openJDK est une implémentation de cela en open source. On peut aussi installer un éditeur de code (exemple: Notepad++, Atom, Visual Codium).

Une fois cela fait, on peut créer un dossier pour notre projet "premier-projet" dans lequel on va mettre un autre dossier "hello". Dans ce dossier on va créer un fichier `Hello.java`. Attention que les noms, doivent être exacts à l'exception du premier dossier "premier-projet".

Dans le fichier Hello.java, on peut ajouter le code suivant

```
// Le package contient les classes, cette ligne indique que le fichier fait partie du package "prb"
// Le package doit être du même nom que le dossier dans lequel le fichier est
package hello;


// La classe doit être du même nom que le fichier, dans ce cas ci la classe "Bonjour" doit être dans un dossier appellé "Bonjour.java"
// Tout le code en java est écrit dans des classes. La mention "public" signifie que cette classe peut être réutilisée par d'autres classes ailleurs
public class Hello {
    // static veut dire que ce n'est pas un "objet" de la classe mais est associée. Il ne faut pas créer une instance de la classe pour appeller cette fonction/méthode
    // void signifie qu'il n'y a pas de valeur "return" à la méthode.
    // la méthode main est la fonction recherchée par java quand la classe est lançée. C'est ici qu'est le programme principal
    public static void main(String[] args) {
        // System est une classe par défault de Java
        // println est une méthode (ou fonction) du bloc "out" de la classe "System"
        System.out.println("Bonjour !");
    }
}
```

On peut enfin lançer ce programme en ouvrant l'invite de commande dans le dossier `premier-java` et en lançant

```
javac hello\Hello.java
java hello.Hello
```

La commande javac va compiler un fichier .class. Java est un language hybride, il compile le code dans un language intermédiaire (le bytecode) prévu pour être exécuté par la JVM (Java Virtual Machine) pour pouvoir fonctionner sur toutes les plateformes car seul la JVM est différente sur chaque plateforme. Le slogan de Java "Write once, run everywhere" viens de là.

# Variables (types, constantes, cast et String)

## Les types primitifs

```
/* Code à intégrer dans la fonction main d'une classe */

// On peut déclarer une variable avant de l'initier
int entier;
entier = 42;

// Ou on peut faire les deux en même temps
boolean test = true;
char lettre = 'A'; // Attention, le ' est nécessaire et ne peut pas être remplacé par "
byte octet = 127
double nombreAVirgule = 5.2;

// On peut aussi créer une constante, sa valeur ne pourra pas changer
final int reponseALaVie = 42;
```

Il y a 8 types primitifs en Java:

| Type | Taille en bits | Domaine de valeurs | Information supplémentaire |
|---|---|---|---|
| `boolean` | ? | 0 (false), 1 (true) | La taille dépends de la JVM |
| `char` | 16 | N'importe quel caractère unicode | N/A |
| `byte` | 8 | $ -128 $ → $ +127 $ | N/A |
| `short` | 16 | $ -2^{15} $ → $ 2^15 - 1 $ | N/A |
| `int` | 32 | $ -2^{31} $ → $ 2^{31} - 1 $ | N/A |
| `long` | 64 | $ -2^{63} $ → $ 2^{63} -1 $ | N/A |

| Type | Taille en bits | Domaine de valeurs | Information supplémentaire |
|---|---|---|---|
| `float` | 16 | $ -3.4 * 10^{38} $ → $ 3.4 * 10^{38} $ | Nombre à virgule (virgule flotante). La précision est de $ 1.4 * 10^{-45} $ |
| `double` | 64 | $ -1.7 * 10^{308} $ → $ 1.7 * 10^{308} $ | Comme `float` mais avec une précision de $ 4.9 * 10^{-324} $ |

# Convertir entre les types avec cast

```
// Conversion de float en int avec cast
double x = 42.5;
int y = (int)x; // 42
int z = (int)42.5; // 42
int a = (int)(42.0 + 0.5); // 42
```

On peut aussi transformer une valeur en une autre en utilisant un *cast*. Comme vu ci dessus en précisant le type cible en ().

A savoir qu'ici, dans le cas d'une conversion d'un nombre en virgule flotante en entier, on perds les informations des décimales. Cast va toujours arrondir vers le bas dans une conversion comme celle ci. Pour avoir plus de controle sur l'arrondis, on peut utiliser la classe java Math (voir plus tard).

# Introduction aux Strings (chaine de caractères)

```
String maChaine = "Hello, World!";
System.out.println(maChaine);


// Convertir un string en un int (cast est impossible car String est une classe)
String nombreStr = "42"; // "42"
int nombreInt = Integer.parseInt(nombreStr); // 42
```

String n'est pas un type primitif, mais bien une classe (ce qui est pourquoi la première lettre est en majuscule) qui corresponds à une chaine de caractère, dans un chapitre suivant nous allons voir quelques méthodes qui peuvent être appliquer aux objets de la classe "String".

N'étant pas un type primitif nous ne pouvons pas utiliser cast dessus mais on peut utiliser d'autres méthodes d'autres classes comme `Integer.parseInt()`, il existe aussi d'autres méthodes du même

genre comme `Double.parseDouble()`.

# En savoir plus

- Wikiversity FR - Java: Variables et types
- Oracle Docs - String
- Oracle Docs - Integer.parseInt

# Opérations sur les variables (calculs, logiques, binaire)

```
int a = 12;
int b = 20;
int c = 42;

// Opérateurs de calcul
int x = (a + b + 2) / c; // Le résultat sera tronqué car x est un int et non pas un double, donc les décimales ne seront pas prise en compte
int y = c % 3; // Effectue de reste d'une division euclidienne

// Opérateurs lors de l'assignation d'une variable
a = a + 5 // Ajoute 5 à la variable a
a += 5; // Ajoute 5 à la variable a
b++; // Ajoute 1 à la variable b

// Opérateurs logiques / binaires
boolean a = true && false; // false
boolean b = true || false; // true
boolean c = !false; // true

// Ou exclusif, 1 et 1 donne 0 mais 1 et 0 donne 1
boolean d = true ^ true; // false
boolean e = true ^ false; // true
```

Il existe 5 opérateurs de base en Java:

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : modulo (reste d'une division euclidienne, exemple $ 13 = 5 * 2 + 3 $ donc $ 13 \mod 5 = 3 $

Il existe ensuite plusieurs opérateurs logiques et binaires, mais on va se concentrer sur les principaux uniquement

- `&&` : ET (true + true = true)
- `||` : OU inclusif (true + false = true) et (true + true = true)
- `^` : OU exclusif (true + false = true) et (true + true = false)
- `!` : NEGATION (true = false) et (false = true)

On peut ensuite effectuer des modifications lors de l'affectation de la variable directement pour gagner du temps en ajoutant l'opérateur devant le signe `=`

On peut aussi ajouter 1 ou diminuer un rapidement en écrivant `x++` ou `x--` par exemple.

# En savoir plus

- Wikiversity FR - Java: Opérations

# Acquisition des données de l'utilisateur (Console)

Créer un dossier `io` dans `premier-java` et y placer le fichier `Console.java` donné dans la page du cours. Il est possible qu'il soit nécessaire de changer le package du fichier vers `io` si ce n'est pas déjà le cas. Ensuite dans notre Hello.java (ou autre).

```
// io (package) .Console (classe) .lireInt() (méthode)
int variableInput = io.Console.lireInt();
System.out.println(variableInput);
```

La classe Console contient plusieurs méthodes: `lireString`, `lireChar`, `lireInt`, `lireLong`, `lireFloat`, `lireDouble` pour récupérer des informations depuis une saisie de l'utilisateur dans le terminal.

Cette classe Console est une simplification du code Java de base pour aller plus vite.

Pour savoir comment la saisie fonctionne dans le java de base, il suffit de lire le code de Console.java.

# En savoir plus

- Voir le fichier Console.java

# Affichage formatté (print, println, printf)

```
System.out.print("Quel est votre nom ? ");
String nom = io.Console.lireString();
System.out.println("Hello World");
System.out.printf("Hello %s", nom);

// Intégrer un nombre décimal
int age = 17;
System.out.printf("Hello %s, you are %d", name, age);

// Intégrer un nombre à virgule flotante
double temperature = 16.5;
System.out.printf("The temperature today is %.2f\n", temperature); // The temperature today is 16,50
```

Il existe 3 fontions principales de PrintStream:

- `print` qui va afficher une valeur sans retour à la ligne
- `println` qui va afficher une valeur avec un retour à la ligne automatique (ce qui est comme ajouter \n à la fin de la valeur dans print)
- `printf` qui va permettre de faire un "template"` pour afficher des valeurs.

Printf prends une grande variété de conversions, en voici quelques basiques :

- `%c` pour afficher un `char`
- `%s` pour afficher un `String`
- `%d` pour afficher un nombre décimal (`byte`, `short`, `int`, `long`)
- `%f` pour afficher un nombre à virgule flotante (`float`, `double`)
- `%%` pour afficher un '%'
- `%n` pour afficher un retour à la ligne

La syntaxe est `%[longueur][conversion]` par exemple, `%.5s` va afficher un `String` d'une longueur de 5 caractères. Si le `String` est moins long, `printf` va remplacer l'espace manquant par des espaces, si elle est trop courte, la valeur va être coupée.

# En savoir plus

- Oracle Docs - Formatter pour plus d'information sur les conversions avec printf
- Oracle Docs - PrintStream pour voir les différentes fonction de System.out.

# Tableaux

```
// Création d'un tableau d'entiers vide d'une longueur de 5 éléments
int[] puissancesDeux = new int[5];
puissancesDeux[0] = 1;
puissancesDeux[1] = 2;
puissancesDeux[2] = 4;
puissancesDeux[3] = 8;
puissancesDeux[4] = 16;
System.out.println(puissanceDeux[4]); // 16

// Création d'un tableau de Strings avec des valeurs prédéfinies
String[] autreTableau = { "un", "deux", "trois", "quatre", "cinq" };
System.out.println(autreTableau[0]); // "un"

// Création d'un tableau de tableau
String[][] tableauDeTableau = { { "un", "deux", "trois" }, { "quatre", "cinq", "six" } };
System.out.println(tableauDeTableau[1][0];

// Connaitre la longueur d'un tableau
System.out.println(tableauDeTableau.length);

// Ajouter un nouvel élément dans un tableau après qu'il soit défini
String[] nouveauTableau = new String[autreTableau.length + 1];
for (int i = 0; i < autreTableau.length; i++) {
nouveauTableau[i] = autreTableau[i];
}

nouveauTableau[autreTableau.length] = "six";
```

Les tableaux en Java ont déjà une longueur définie, ce qui fait qu'ajouter des nouveaux éléments est plus compliqué. Il faut créer un nouveau tableau avec une longueur plus longue et créer un for loop pour ajouter chaque élément dans le nouveau tableau + l'élement à ajouter.

Les indices (le nombre qui est dans les crochets) qui permet de référencer un élément dans le tableau, commencent par 0. Donc le premier élément de la liste est l'indice 0, le deuxième 1, le troisième 2, etc.

On peut connaitre la longueur d'un tableau en utilisant ".length" sur un tableau.

# En savoir plus

- Wikiversity FR
- Le powerpoint de Mr Comblin sur les Tableaux

# Manipulations de Strings

```
String inputString = "ceci est mon string, 32";
String[] tab = inputString.split(", ");
String str = tab[0].toUpperCase();
int number = Integer.parseInt(tab[1]);
System.out.printf("%s =  %d %n", str, number);

// Tester des strings
boolean finiParTest = inputString.endsWith("test"); // false
boolean commenceParCeci = inputString.startsWith("ceci"); // true

// Avoir la longueur d'un string
System.out.println(inputString.length()); // 23

// Utiliser substring pour diviser un string
String hhmm = "10:30";
System.out.println(hhmm.substring(0, hhmm.length() - 3)); // 10
System.out.println(hhmm.substring(hhmm.length() - 2, hhmm.length())); // 30

// Trouver la position d'un caractère dans un String
int position = hhmm.indexOf(':'); // 2

// Retrouver le caractère à un certain indice de position
char caractere = hhmm.charAt(position); // ':'
System.out.printf("Le caractère à %d est '%c'", position, caractere); // "Le caractère en position 2 est ':'"

// Transformer un int en String
int nombre = 42;
String nombreEnString = String.valueOf(nombre); // "42"
```

String étant une classe, elle contient plusieurs méthodes, en voici 2 qui soit très utiles :

- `.split("regex")` pour diviser un String en un tableau en utilisant un autre String comme délimiteur de la colonne.
- `.toUpperCase()` et `.toLowerCase()` pour convertir un String en majuscule ou minuscule
- `.endsWith()` et `.startsWith()` pour tester si un String termine ou commence par un certain substring.

- `.length()` pour obtenir la longeur de la chaine de caractères.
- `.substring()` pour diviser un string avec 2 positions. Cette méthode prends 2 arguments, la première position (qui est inclusive, donc prise en compte dans le résultat), et la deuxième position qui est exclusive (donc non prise en compte dans le résultat). Donc si on a 0 en premier argument et 5 en deuxième argument, on aura les caractères 1, 2, 3 et 4.
- `.indexOf()` permet de savoir quel est l'indice de position d'un caractère ou un String dans un String.
- `.charAt()` est l'inverse de `indexOf` et retourne un caractère à une position donnée
- `.valueOf()` premet de transformer un autre type en String.

> A noter que l'utilisation des méthodes String indiquées ici ne transforme pas la variable String d'origine, elle ne font que retourner une nouvelle valeur.

# En savoir plus

- Oracle Docs - String

# Un peu de Math

```
// Génération de nombres aléatoires
double nombreAleatoire = Math.random();

// Puissances et racines
double puissance2 = Math.pow(5.0, 2.0); // 5^2
double racineCarree = Math.sqrt(puissance2);

// Arrondir un nombre
double nombre = 5.67;
System.out.println(Math.ceil(nombre));  // 6.0
System.out.println(Math.floor(nombre)); // 5.0
System.out.println(Math.round(nombre)); // 6
System.out.println(Math.rint(nombre));  // 6.0
```

Pour générer une valeur aléatoire entre 0 et 1 on peut utiliser la méthode `Math.random()`.

Pour faire une puissance de 2 on peut utiliser `Math.pow(double a)` et pour faire une racine `Math.sqrt(double a)`. Toutes ces fonctions renvoie des valeurs de type `double`

Pour faire des arrondis on peut utiliser différentes fonctions :

- `Math.ceil(nombre)` pour arrondir un nombre vers le haut (retourne un double)
- `Math.floor(nombre)` pour arrondir un nombre vers le bas (retourne un double)
- `Math.round(nombre)` pour arrondir un nombre en fonction de sa première décimale (retourne un int)
- `Math.rint(nombre)` pour arrondir un nombre en fonction de sa première décimale (retourne un double)

# Pour en savoir plus

- Oracle Docs - Math

# Conditions (if, else, switch)

```
System.out.print("Sélectionnez un nombre: ");
int first = io.Console.lireInt();

if (first == 42) {
    System.out.println("Félicitations, vous avez trouvé la réponse à la vie");
} else if (first > 40 && first < 50) {
    System.out.println("Vous y êtes presque");
} else {
    System.out.println("Vous n'avez rien trouvé du tout");
}

// Comprarer des Strings ou autres objets ne se fait pas de la même manière
System.out.print("Quel est votre nom ? ");
String nom = io.Console.lireString();
if (nom.equals("Roger")) {
    // faire quelque chose
}

// Donne le jour en texte à partir d'un numéro, cas par cas
int jour = 4;
switch (jour) {
  case 1:
    System.out.println("Lundi");
    break;
  case 2:
    System.out.println("Mardi");
    break;
  case 3:
    System.out.println("Mercredi");
    break;
  case 4:
    System.out.println("Jeudi");
    break;
  case 5:
    System.out.println("Vendredi");
```

```
    break;
  case 6:
    System.out.println("Samedi");
    break;
  case 7:
    System.out.println("Dimanche");
    break;
} // "Jeudi"

switch (jour) {
  case 6:
    System.out.println("Samedi");
    break;
  case 7:
    System.out.println("Dimanche");
    break;
  default:
    System.out.println("En attente du weekend");
} // "En attente du weekend"
```

Il y a deux types de conditions en java, les `if else` et les `switch`.

# If, Else if, Else

Les `if` permettent de mettre une condition pour l'exécution d'un code.

Il existe plusieurs mots clés comme vu dans l'exemple du début

- `if (...) {` : si *condition* alors ...
- `else if (...) {` : si la contion précédente du bloc `if` n'a pas réussi alors si *condition*, ...
- `else` : si aucune des conditions précédentes du bloc `if` n'a réussi alors ...

Ces conditions fonctionnent en utilisant un "opérateur conditionnel" entre deux nombres. En voici un exemple :

| Opérateur contionnel | Description |
| --- | --- |
| `a == b` | A est égal à B |
| `a != b` | A est différent de B |
| `a > b` | A est plus grand que B |
| `a < b` | A est plus petit que B |

| Opérateur contionnel | Description |
| --- | --- |
| `a >= b` | A est plus grand ou égal à B |
| `a <= b` | A est plus petit ou égal à B |
| `a` | A est vrai |
| `!a` | A est faux (en vérité ceci est un opérateur binaire et non un opérateur conditionnel) |

On peut aussi mettre plusieurs conditions ensemble avec des opérateurs logiques (ce sont les même qu'au cours de math et d'archi) :

| Opérateur logique | Description |
| --- | --- |
| `&&` | Les deux conditions doivent être remplies |
| `\|\|` | Au moins une des deux conditions doit être remplie |
| `^` | Une seule des deux conditions doit être remplie mais pas plus |

## Cas spéciaux avec les Strings

On ne peut pas utiliser un opérateur tel que `==` sur un String car un String est un objet de la classe String. Si on utilise l'opérateur logique, on va comparer la référence mémoire de la variable et non la valeur en elle même.

Il faut donc utiliser la méthode `.equals(String autreString)` à la place comme vu dans l'exemple.

# switch

Si on veut limiter l'usage des `if else` et que l'on veut tester si une valeur correspond à x, y ou z valeur, on peut utiliser `switch`

On donne à switch la valeur à tester et plusieurs *cas* (`case`) de valeur possible. Pour chacune d'entre elle, quelque chose à faire avec.

On peut utiliser le mot clé `break` pour arrêter le switch sans tester le reste.

Enfin il y a le `default` qui fonctionne comme le `else`, le default, c'est quand aucun autre cas n'est passé.

# En savoir plus

- Wikiversity FR - Boucle et structures conditionnelles

- Wikiversity FR - Opérations

# Méthodes static

```
public class Bonjour {
    public static void main(String[] args) {
        System.out.print("Quel est ton nom ? ");
        String nom = Console.lireString();
        direBonjour(nom);

        int premierNombre = 40;
        int deuxiemeNombre = 2;
        int troisiemeNombre = somme(premierNombre, deuxiemeNombre); // 42
    }

    public static void direBonjour(String nom) {
        System.out.printf("Bonjour %s !\n", nom);
    }

    private static int somme(int a, int b) {
        int c = a + b;
        return c;
    }
}
```

Une méthode (ou fonction) est un bloc de code qui peut être réutiliser plusieurs fois. En fonction de comment elle est définie, une méthode peut renvoyer une certaine valeur ou rien, et peut prendre des paramètres ou pas.

Pour mieux comprendre voici comment on peut décomposer notre méthode `direBonjour` au dessus :

- `public` indique l'accessibilité. Elle signifie que d'autres classes peuvent aussi utiliser cette méthode. Au contraire, si cela serait `private` alors seul la classe `Bonjour` pourrait y accéder
- `static` indique que ce n'est pas un "objet" de la classe `Bonjour`, on vera plus tard ce que cela signifie
- `void` signifie que la méthode ne retourne aucune valeur
- `direBonjour` est le nom de notre méthode
- `(String nom)` indique que la méthode ne prends qu'un argument de type String, qui va être appellé "nom" dans le bloc de la méthode.

Dans le deuxième exemple :

- `private` indique que la méthode ne peut être utilisée que dans la classe `Bonjour`
- `static` même chose que pour `direBonjour`
- `int` indique que la méthode retourne un type "int"
- `somme` est le nom de la méthode
- `(int a, int b)` indique que la méthode prends 2 arguments, tout deux de type "int" qui seront appellé dans la méthode "a" et "b".
- `return` indique une valeur à retourner de la fonction. Une fois que la fonction a retourné quelque chose, tout code écrit après ce `return` ne sera pas exécuté.

Mais il y a aussi la fonction main ! On a l'a tout le temps utilisé mais c'est aussi une méthode. La seule différence est que quand on exécute une classe, Java va chercher cette fonction "main" dans notre classe pour l'exécuter. Si il n'y a pas de fonction main, la classe n'est pas exécutable à elle seule.

# Plusieurs méthodes avec le même nom

```
public static void main(String[] args) {
    System.out.printf("12 + 15 = %d%n", somme(12, 15));
    System.out.printf("12.5 + 13.0 = %f%n", somme(12.5, 13.0));
}

public static int somme(int a, int b) {
    return a + b;
}

public static double somme(double a, double b) {
    return a + b;
}
```

Dans java, contrairement à d'autres language il peut y avoir plusieurs méthodes qui ont le même nom tant qu'elle ne prennent pas les mêmes arguments (paramètres).

Dans ce cas ci, on a deux variables "somme" mais une prends des types "int" et l'autre prends des types "double". Donc elles peuvent toutes les deux exister dans la même classe sans problème.

# Apeller une méthode

Comme vu dans les exemples ci dessus, on peut appeller une méthode quand on est dans la même classe qu'elle. Mais on peut aussi appeller une méthode qui est dans une autre classe.

```
public static void main(String[] args) {
    int input = io.Console.lireInt();
}
```

Dans l'exemple ici on appelle la méthode `lireInt` qui est dans la classe `Console` du package `io` par exemple.

# En savoir plus

- Wikiversity FR - Méthodes

# Boucles (for, for each, while, do...while)

Pour ne pas avoir besoin de repeter un code beaucoup de fois, on peut utiliser des boucles. Il en existe 4 différentes.

- `while` qui exécute un code en boucle tant qu'une certaine condition est remplie
- `do...while` qui exécute une fois le code, puis la répète encore tant que la condition est remplie
- `for` qui permet d'exécuter un code un nombre définis de fois
- `for each` qui permet d'exécuter un code pour chaque élément dans une collection (par exemple un tableau comme vu dans un chapitre précédent)

## La boucle while

```
// Compter jusqu'a 42
int i = 0;
while (i <= 42) {
    System.out.printf("Le compteur i est à %d.\n", i);
    i++;
}

// Ne va rien faire car la condition n'est pas remplie
int j = 0;
while (i == 666) {
    System.out.printf("Le compteur j est à %d.\n", j);
    j++;
}
```

Le premier bloc de code va s'exécuter et on verra 43 lignes allant de 0 à 42. Le deuxième bloc de code ne s'exécutera pas car sa condition que i doit être égal à 666 ne sera pas remplie.

## La boucle do...while

```
// Compter jusqu'a 42
do {
    System.out.printf("Le compteur i est à %d.\n", i);
    i++;
} while (i <= 42);

// Ne va faire qu'une seule itération car la condition ne sera pas remplie
int j = 0;
do {
    System.out.printf("Le compteur j est à %d.\n", j);
    j++;
} while (j == 666);
```

Dans ce cas ci, comme avant, le premier bloc de code s'exécutera 43 fois de 0 à 42. Tandis que le second bloc de code ne s'éxécutera qu'une fois et indiquera "0".

Contrairement à while, la boucle do...while s'exécutera toujours un minimum de une fois. Puis se réexécutera si/tant que la condition est remplie.

# La boucle for

```
// Compter jusqu'a 42
for (int i = 0; i <= 42; i++) {
    System.out.printf("Le compteur est à %d.\n", i);
}
```

Ce code indiquera 43 lignes allant de 0 à 42. C'est comme une version contractée de la boucle while que nous avons déjà vu plus tot.

La différence est que le nombre d'itération est déjà prédéfinis. On définis une boucle for comme ceci :

```
for (initialisation; condition; modification) {
    // code à exécuter dans la boucle
}
```

# La boucle for each

```
// Lire chaque élément d'un tableau
String[] tableau = { "un", "deux", "trois", "quatre", "cinq", "six", "sept", "huit", "neuf", "dix" };

for (String element: tableau) {
    System.out.println(element);
}
```

La boucle for each permet de passer en revue tous les élément d'une *collection*, par exemple dans ce cas ci, d'un tableau (voir dans un chapitre précédent). Dans ce code on assigne pour chaque itération la variable "element" qui correspond à l'élément en cours dans la variable "tableau".

# En savoir plus

- Wikiversity FR - Boucles et structures conditionnelles

# Le RegEx

Voici un petit résumé de la signification des différents caractères :

| Element en regex | Signification |
|---|---|
| `\` | Indique que le caractère qui suit est litéral et qu'il ne faut pas qu'il soit interpreté comme synatxe du regex |
| `^` | Début de la chaine de caractères |
| `$` | Fin de la chaine de caractères |
| `*` | Match le caractère précédent 0 fois ou plus |
| `+` | Match le caractère précédent 1 fois ou plus |
| `?` | Match le caractère précédent 0 ou 1 fois |
| `.` | Match n'importe quel caractère |
| `[abc]` | Match n'importe lequel des caractère dans les crochets |
| `[A-Z]` | Match n'importe quel caractère dans une série (ici allant de A à Z majuscule) |

Les Regex peuvent être appliqué dans diverses fonctions de String et Pattern

```
class Test {
    public static void main(String[] args) {
        String monString = "Hello World";
        // String.matches() peut être utiliser pour vérifier si un String correspond à une certaine expression
        if (monString.matches("[A-z]+ [A-z]+")) {
            System.out.println(monString);
        }
    }
}
```

D'autres méthodes peuvent aussi utiliser des regex, tel que la méthode `String.replaceAll` vue dans le chapitre sur les Strings.

# Tests unitaires avec jUnit 5

Les tests unitaires permettent d'avoir une vue globale de la santé d'un projet en s'assurant que toute ses fonctions se comportent comme elle doivent.

Dans Eclipse il faut reproduire la structure demandée. Le code source est dans le dossier `src`, tandis que les tests sont dans le dossier `tests`. La structure des packages et des classes est conservée. A l'exception des nom de la classe de test qui finissent par `Tests`.

```
.
├── src
│   └── util
│       └── TableauChaines.java
└── tests
    └── util
        └── TableauChainesTests.java
```

Dans cet exemple on va tester une fonction de `TableauChaines.java`

```
// Il faut indiquer le même package pour le test que ce que l'on test
package util;

// Il faut importer Assert et Test de junit
import static org.junit.Assert.*;
import org.junit.jupiter.api.Test;

// Les fichiers de test finissent par "Tests.java" ainsi leur classe fini par "Tests" aussi
class TableauChainesTests {
    // Chaque test est une fonction void sans argument précédée de @Test
    @Test
    public void contientTest() {
        String[] tableau = {"PRENOM", "SEXE", "LONGUEUR DES CHEVEUX", "LUNETTES"};

        // Ce qui vérifie les tests sont les assertions
        // Il y en a de différent types (assertEquals, assertArrayEquals, assertTrue, etc) dépendant de ce que l'on test
        // Le premier paramètre est le résultat attendu, le deuxième est le résultat obtenu à partir de la méthode que l'on test
```

```
        assertTrue(TableauChaines.contient(tableau, "Longueur des cheveux"));
    }
}
```

On peut ensuite exécuter le test en lançant le projet dans Eclipse. Et c'est tout !

# Génération de la documentation avec javadoc

La Javadoc permet d'écrire la documentation des méthodes du programme directement dans le code. Un programme va ensuite générer un site pour afficher toute cette documentation.

```
package labo6;


class JourDeLaSemaine {
	/**
	 * Adapte la longueur d'une chaîne de caractères en la complétant par des
	 * caractères espace ou en réduisant son nombre de caractères.
	 *
	 * @param chaine  la chaîne de caractères à ajuster.
	 * @param largeur La largeur que doit avoir la chaîne de caractères spécifiée.
	 * @return La chaîne de caractères ajustée.
	 */
	public static String ajusterLargeur(String chaine, int largeur) {
		final int LG_CHAINE = chaine.length();
		chaine = (LG_CHAINE > largeur) ? chaine.substring(0, largeur) : chaine;
		return chaine + " ".repeat(largeur - chaine.length());
	}
}
```

Dans l'exemple ci dessus, on a une méthode "ajusterLargeur" que l'on veut documenter. Pour se faire il suffit d'écrire `/**` puis enter dans Eclipse pour que le template soit généré tout seul.

Dans la première partie on écrit la description de la méthode, puis on décrit à quoi correspondent chaque paramètre et ce que la méthode retourne.

Pour générer la Javadoc par la suite on peut aller sur Eclipse dans le menu "Project" puis "Generate javadoc". La javadoc sera ainsi générée dans le dossier `doc` du projet.

Et voici le résultat pour notre méthode :


screenshot de la javadoc pour la méthode