Introduction

CogniTIC

Building Together Better IT Solutions

Plan du cours

Historique et concepts

CHAPITRE 1

Historique

- Le nom « **Java** » n'est pas un acronyme, il a été choisi lors d'un brainstorming en remplacement du nom d'origine « **Oak** », à cause d'un conflit avec une marque existante, parce que le café (« java » en argot américain) est la boisson favorite de nombreux programmeurs.

wiki/Java

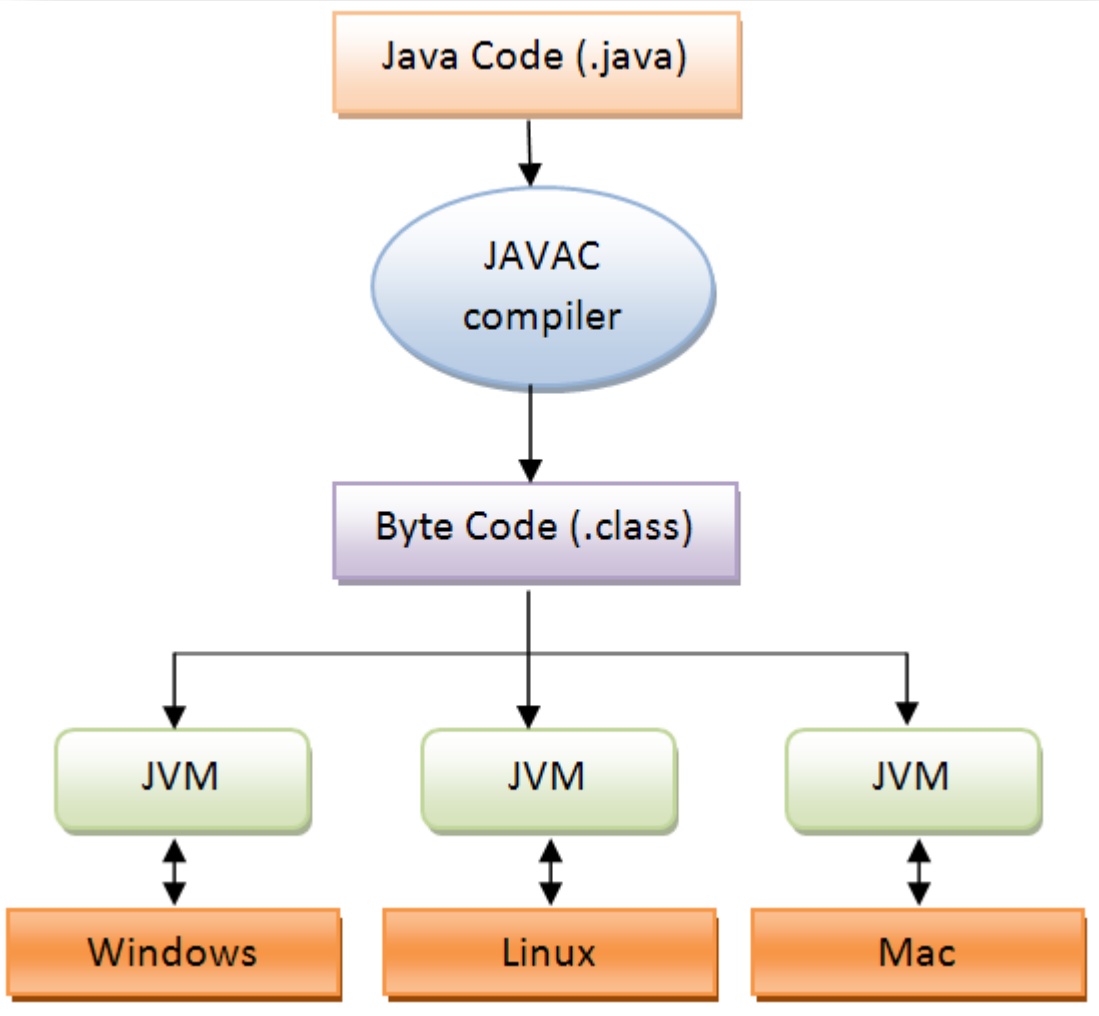
Historique

- Naissance du projet Java en 1991
  - Créé par des employés de Sun Microsystems
  - But au départ : code embarqué (destiné à de petits appareils électroniques)
  - Syntaxe proche du C++, tout comme le C#
  - Présenté officiellement en 1995
  - Racheté par Oracle en 2009

Concepts

- Portabilité, Machine virtuelle et Bytecode
  - Un langage multiplateforme
    - Le Java a été pensé afin que les programmes écrits dans ce langage puissent fonctionner de manière similaire sur différentes architectures matérielles
  - Le bytecode Java
    - Le bytecode Java est le résultat de la compilation du code source
  - La machine virtuelle Java
    - La « JVM » est un appareil informatique fictif qui exécute le bytecode ainsi compilé

Concepts


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

Concepts

- La JVM
  - L'appareil (machine/OS) est simulé par un **logiciel spécifique à chaque plate-forme**
  - La JVM permet aux applications Java compilées en bytecode de **produire les mêmes résultats** quelle que soit la plate-forme

Concepts

- La JVM
  - La machine virtuelle Java effectue les tâches principales suivantes:
    - Charge le code
    - Vérifie le code
    - Exécute le code
    - Fournit l'environnement d'exécution

Concepts

- La JVM
  - JVMs propriétaires
    - J9 (IBM) pour AIX, Linux, MVS, OS/400, Pocket PC, z/OS
    - PERC (Aonix/Atego), JVM temps réel pour système embarqué
    - SAPJVM (SAP), JVM Sun modifiée pour SAP
    - …
  - JVMs open source
    - ART (Android) remplaçant de Dalvik dans Androïd
    - HotSpot, la 1ère référence d'implémentation de JVM
    - JamVM, développée pour être extrêmement petite
    - …

wiki/Liste_des_machines_virtuelles_Java

Concepts

- La JVM
  - Le ramasse-miettes
    - Un ramasse-miettes (en anglais ***garbage collector***) est un sous-système de gestion automatique de la mémoire
    - Il est responsable du recyclage de la mémoire préalablement allouée puis inutilisée
    - La règle principale pour déterminer qu'un objet n'est plus utilisé est de vérifier qu'il n'existe plus aucun autre objet qui lui fait référence

wiki/ramasse-miettes

Concepts

- La JVM
  - Le ramase-miettes
    - Lorsque le ramasse-miettes va libérer la mémoire d'un objet, il a l'obligation d'exécuter un éventuel finalizer définit dans la classe de l'objet
    - Attention, l'exécution complète de ce finalizer n'est pas garantie
    - Si une exception survient durant son exécution, les traitements sont interrompus et la mémoire de l'objet est libérée sans que le finalizer soit entièrement exécuté

Concepts

- Java et la Programmation Orienté Objet
  - Presque un pur langage de P.O.O.

Utilisations du java

CHAPITRE 2

Utilisations du Java

- Langage très utilisé dans le milieu professionnel

- Possibilités d'utilisation
  - Applications en console
  - Applications avec interfaces graphiques
  - Applets (Programmes incorporés aux pages Web)
  - Applications mobiles (avec Java ME et Android)
  - Sites Web dynamiques (avec Java EE)

Environnement

CHAPITRE 3

Environnement

- Les JDKs
  - Définition

    Le Java Development Kit (JDK) désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java.

Environnement

- L' Oracle JDK
  - Depuis le 6/04/2019, la licence d'Oracle JDK a changé. Dorénavant la licence commerciale et le support sont payant. La licence pour un usage personnel et le développement reste gratuit

    oracle now requires a subscription to use Java SE

  - Version stable : 11
  - Dernière version : 12 (19/03/2019)

Environnement

- L'OpenJDK
  - L'OpenJDK constitue l'implémentation de référence officielle et libre de Java SE, tel que défini par le Java Community Process et ce, depuis sa version 7. Il est le résultat de l'effort de l'entreprise Sun Microsystems (propriétaire de Java jusqu'à son rachat par Oracle) à vouloir rendre Java SE open source

    wiki/OpenJDK

  - Version stable : 11
  - Dernière version : 12 (19 mars 2019)

Environnement

- Installation du Java Development Kit (JDK)
  - JDK

https://adoptopenjdk.net

- Installation d'un Integrated Development Environment (IDE)
  - *IntelliJ*

https://www.jetbrains.com/idea/download

  - Eclipse

https://eclipse.org/downloads

  - *Netbeans*

https://netbeans.org/downloads/index.html

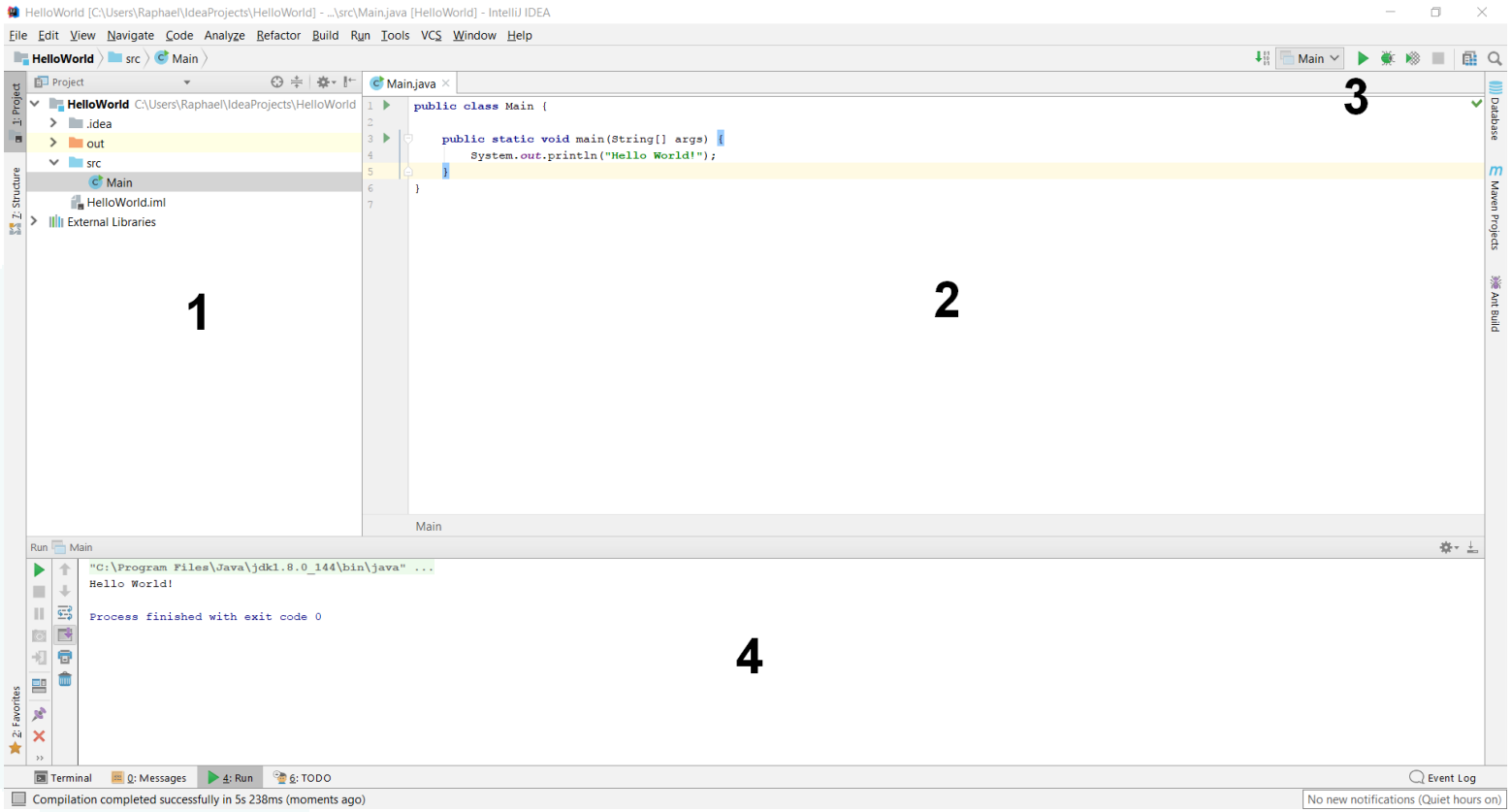
Environnement

- Configuration des installations
  - La variable système PATH
    - En cas d'erreur lors de la tentative de compilation, ou d'exécution, il se peut qu'il faille configurer manuellement la variable système PATH :
      - Dans Windows, aller dans Panneau de configuration\Système et sécurité\Système puis dans les Paramètres système avancés.
      - Dans Variables d'environnement, ajouter ou modifier la variable système PATH en ajoutant le chemin du dossier bin de l'installation Java, par exemple :

        C:\Program Files\AdoptOpenJDK\jdk-11.0.3.7-hotspot\bin

Environnement

- L'interface d'IntelliJ



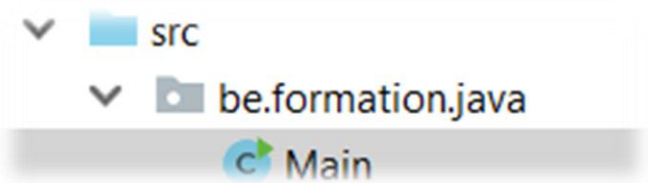
Environnement

1. Project : Votre projet et ses fichiers s'affichent ici

2. La zone principale, où vos fichiers s'ouvrent et où vous pouvez les modifier. Chaque fichier s'ouvre dans un nouvel onglet, n'oubliez pas de faire de temps à autres le tri !

3. Configurations et exécution : IntelliJ peut sauver plusieurs configurations afin d'exécuter votre projet. On peut également choisir le mode d'exécution (debug par exemple).

4. Console : La sortie « console » de votre programme dans Eclipse. C'est également ce qui apparaîtra sur la console de votre OS lors de l'exécution



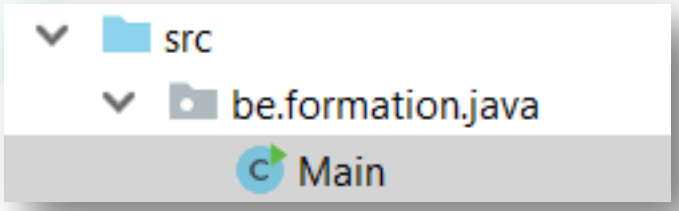
Les Packages

CHAPITRE 4

Les Packages

- C'est l'architecture d'un projet java. Ils regroupent les fichiers du programme et d'autres packages
- Les noms de package sont séparés par des '.' qui marquent une hiérarchie
- Attention : la structure hiérarchique est aussi bien logique que physique (écrite sur disque)

C:\Users\(…)\HelloWorld\src\be\formation\java



Les Packages

- L'utilisation de fichiers du programmes contenus dans un autre package est possible en faisant un *import*.

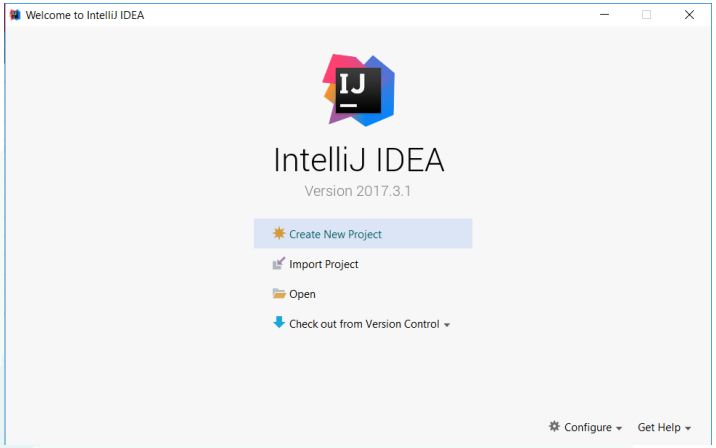
```java
package be.formation.java;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

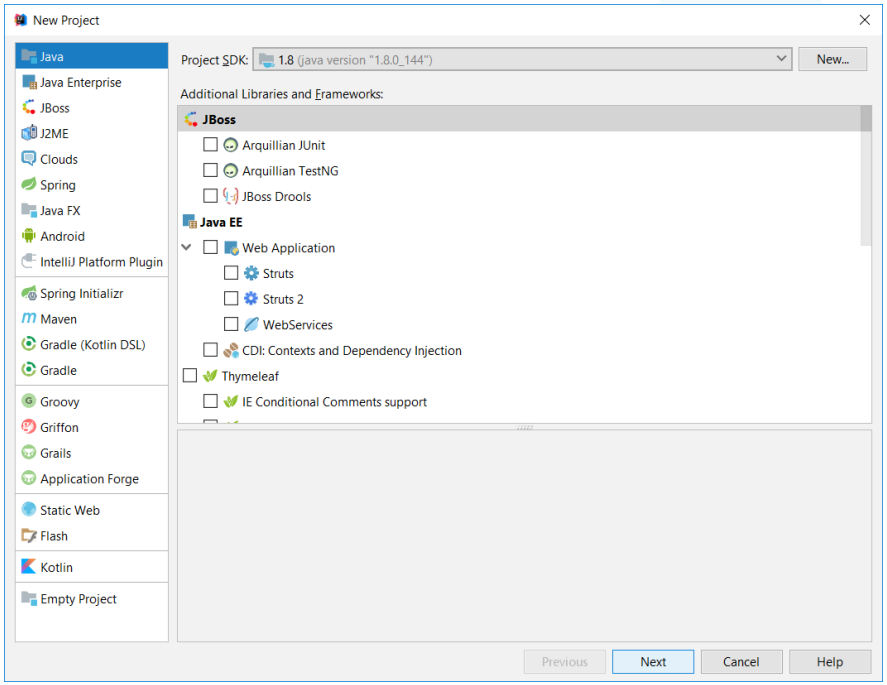
Un Premier programme

CHAPITRE 5

Un premier programme

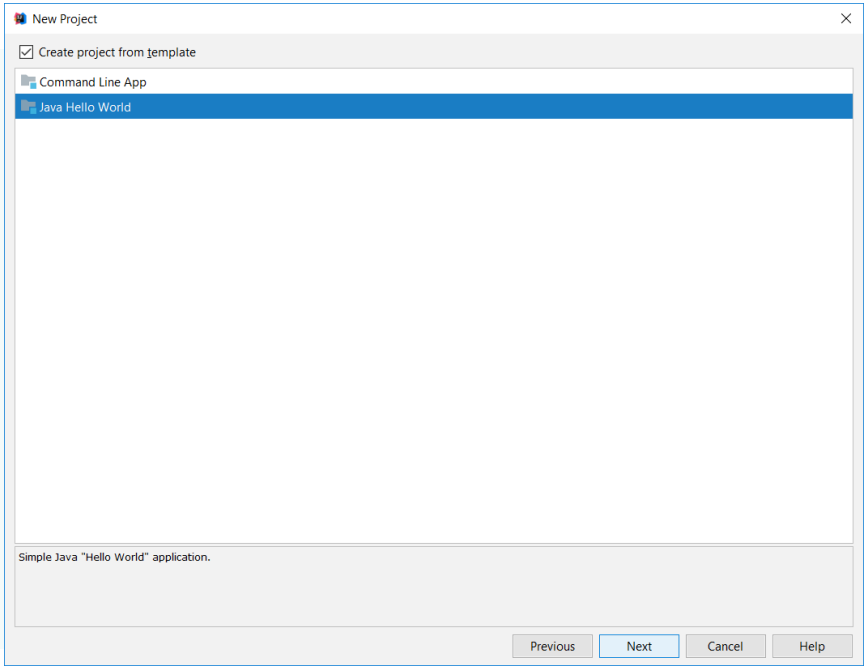
- Premier programme : un « Hello World » en Java
  - Créez un nouveau projet Java





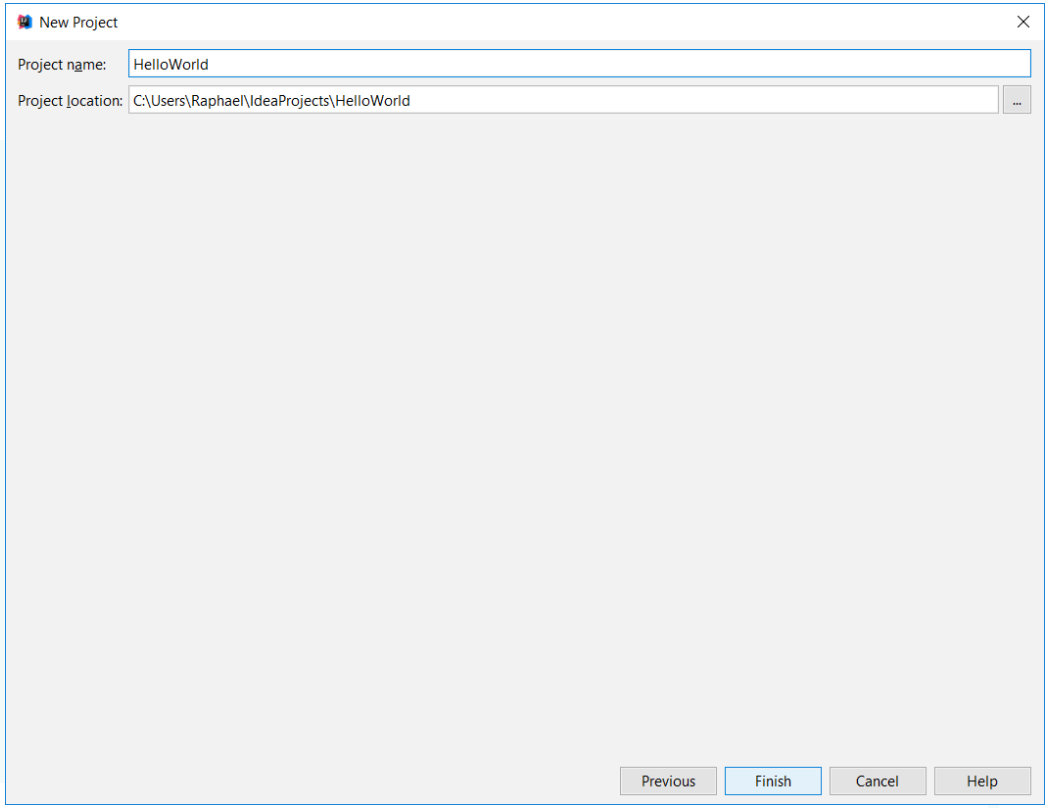
Un premier programme

- Vous pouvez choisir un template de base pour le projet
- Dans notre cas, sélectionnez « Java Hello World »



Un premier programme

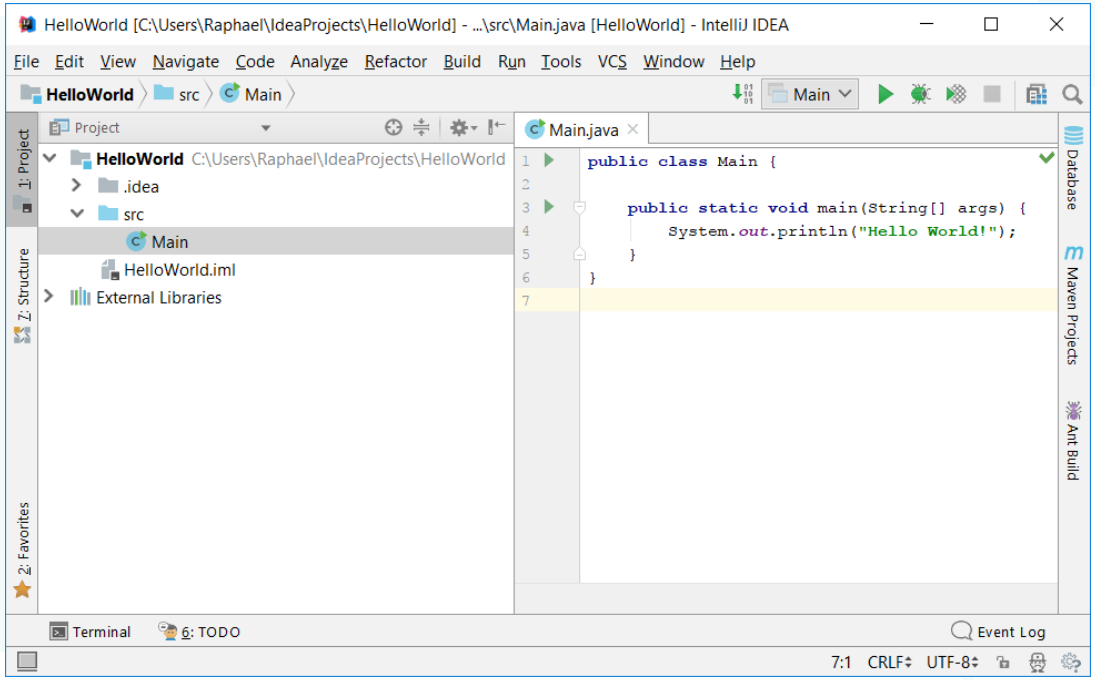
- Nommez votre projet, vous pouvez également modifier son emplacement au besoin.



Un premier programme

- Votre projet a été créé !

- Exécutez le dans IntelliJ grâce au bouton « Run » ou Shift + F10



Un premier programme

- En réalité, lorsqu'on exécute le projet, notre IDE utilise les commandes suivantes :

  - javac (pour java compiler)

  Ouvrez la console de commande Windows (cmd.exe)

  Naviguer jusqu'au dossier source (qui contient nos packages)

  Utiliser la commande :

  javac vos\packages\VotreClasse.java

  - java

  Dans ce même dossier, utiliser la commande :

  java vos.packages.VotreClasse

Fondamental

CogniTIC

Building Together Better IT Solutions

Plan du cours

```
int a, b, c;
a = 2;
b = 1;
c = a + b;
```

Les variables

CHAPITRE 1

Les variables

- Qu'est-ce qu'une variable ?
  - Symbole qui associe un nom à une valeur en mémoire.
  - Le nom doit être un identifiant unique et différent des mots-réservés.
  - Une variable peut changer de valeur au cours du temps.

Les variables

- Convention de nommage : camelCase
  - Pas d'accents
  - Pas d'espaces
  - Commencent par une minuscule
  - Majuscules internes si le nom est composé de plusieurs mots

Les variables : types

- Les variables que vous créez en Java doivent être typées.
  - Cela permet au compilateur de vérifier les valeurs que vous donnez aux variables. Impossible par exemple de définir une variable de type numérique et de lui affecter une chaîne de caractères.
  - C'est également utilisé pour allouer l'espace mémoire nécessaire aux données qui seront stockées.

Les variables : types

- Petit rappel…
  - Les ordinateurs travaillent en binaire
  - 1 bit = 0 ou 1

Les variables : types

- Les variables numériques

byte (8-bit),

Entiers entre -128 et 127

- short (16-bit),

Entiers entre -32 768 et 32 767

- int (32-bit) défaut pour les entiers,

Entiers entre -2 147 483 648 et 2 147 483 647

Les variables : types

- long (64-bit),

Entiers entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807

- float (32-bit),

Nombres à virgule à précision simple

- double (64-bit) défaut pour les nombres à virgule,

Nombres à virgule à précision double

Les variables : types

- Les booléens

boolean (1 bit théorique),

true ou false

Les variables : types

- Les caractères et chaînes de caractères

  char (16-bit),

  Un caractère

  - String,

  Une chaîne de caractère

Les variables : types

- On note que tous les types de variables s'écrivent avec des minuscules, sauf String. En réalité, ce sont tous des types primitifs, alors que String est une classe.
- Bien que le type String n'est pas un type primitif, il en a souvent le comportement.
- Nous en reparlerons quand nous verrons la programmation orientée objet !

Les variables : déclaration

- En Java, on est obligé de déclarer ses variables.

```
byte myByte;
short myShort;
int myInt;
long myLong;

float myFloat;
double myDouble;

boolean myBoolean;

char myChar;
String myString;
```

Les variables : initialisation

- En plus d'être déclarée, une variable doit également être initialisée avant de pouvoir être utilisée.

```
myByte = 123;
myShort = 12345;
myInt = 1234567890;
myLong = 9876543210L;

myFloat = 123.456F;
myDouble = 321.654;

myBoolean = true;

myChar = 'a';
myString = "abc";
```

Les variables

- On peut également déclarer et initialiser ses variables en une ligne :

```
byte myByte = 123;
short myShort = 12345;
int myInt = 1234567890;
long myLong = 9876543210L;

float myFloat = 123.456F;
double myDouble = 321.654;

boolean myBoolean = true;

char myChar = 'a';
String myString = "abc";
```

Les variables

- Ou déclarer plusieurs variables de même type :

```
int a, b, c;
byte d = 4, e;
long f = 321, g = 123;
```

Les variables

- Constantes :
  - Une constante stocke une valeur qui ne changera pas au cours de l'exécution du programme. Pour cela, on utilise le mot-clé final.
  - Par convention, on écrit les constantes en majuscules

```
final int SPEED_OF_SOUND = 340;
```

```
int a = 3;
int b = 2;
float c = (float) a / b;
```

Les Conversions

CHAPITRE 2

Les conversions

- Conversions implicites
  - Le compilateur convertit implicitement les valeurs lorsqu'il peut valider qu'il n'y aura aucune perte d'information :

```
byte b = 127;
int i = b;

float f = 132.465F;
double d = f;

char c = 321;
```

Les conversions

- Conversions explicites
  - Lorsqu'il y a un risque de perte de données, c'est au développeur de prendre la responsabilité de convertir explicitement les données.
  - Selon le type de données source et destination, on doit utiliser des mécanismes différents.

Les conversions

- Entre types différents mais compatibles : le cast

```
int i = 123;
byte b = (byte) i;
```

- De type numérique en String : String.valueOf()

```
int i = 123;
String s = String.valueOf(i);
```

Les conversions

- De chaîne de caractères en type numérique : le parse

```
String input = "123";
byte b = Byte.parseByte(input);
short s = Short.parseShort(input);
int i = Integer.parseInt(input);
long l = Long.parseLong(input);
float f = Float.parseFloat(input);
double d = Double.parseDouble(input);
```

```
c = a + b;
d = c * 2;
```

Les opérateurs

CHAPITRE 3

L'opérateur d'affectation

Opérateur	Signification
=	Opérateur d'affectation usuel

Les opérateurs arithmétique

Opérateur	Signification
+	Addition et concaténation de chaînes
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division entière)

Les opérateurs arithmétique

- Exemples :
  - L'opérateur « + »

```
c = a + b;
d = c + 2;
```

  - L'opérateur « - »

```
b = b - 1;
c = b - a;
```

Les opérateurs arithmétique

- L'opérateur « * »

```
a = b * c;
```

- L'opérateur « / »

```
e = a / c;
```

- L'opérateur « % »

```
b = a % c;
```

Les opérateurs relationnels

Opérateur	Signification
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à

Les opérateurs relationnels

Opérateur	Signification
==	Égal à
!=	Différent de

Les opérateurs logiques

Opérateur	Signification
&&	AND logique
||	OR logique inclusif
!	Négation
^	OR exclusif (XOR)

Autres opérateurs utiles

Opérateur	Signification
++	Incrémentation (i = i + 1)
--	Décrémentation (i = i - 1)

Autres opérateurs utiles

Opérateur	Signification
+=	Affectation élargie Exemple : i = i – 5 devient i -= 5
-=	
*=	
/=	

Les priorités des opérateurs

Opérateurs
(), ++, --
*, /, %
+, -
<, <=, >, >=
==, !=
^
&&
||
=, +=,…

Les Conditions

CHAPITRE 4

Les conditions

Les conditions de type if … else if … else

if

Le code compris dans ce bloc ne s'exécute que si la condition est remplie (= renvoie true)

else if (optionnel)

- Le code compris dans ce bloc ne s'exécute que si la condition du premier if n'est pas remplie, mais que la condition du else if l'est

else (optionnel)

- Le code compris dans ce bloc ne s'exécute que si la condition du if, ainsi qu'aucune condition des else if n'est remplie (= renvoie false)

Les conditions

Exemple de if … else :

```
int a = 5;

if (a > 2) {
    System.out.println("a est vaut plus que 2");
} else {
    System.out.println("a vaut 2, ou moins");
}
```

Les conditions

Exemple if … else if … else:

```
int a = 5;

if (a > 2) {
    System.out.println("a est vaut plus que 2");
} else if (a == 2) {
    System.out.println("a vaut 2");
} else {
    System.out.println("a vaut moins de 2");
}
```

Les conditions

Le switch

Il permet d'évaluer l'égalité d'une variable à plusieurs valeurs possibles et d'exécuter le bloc correspondant :

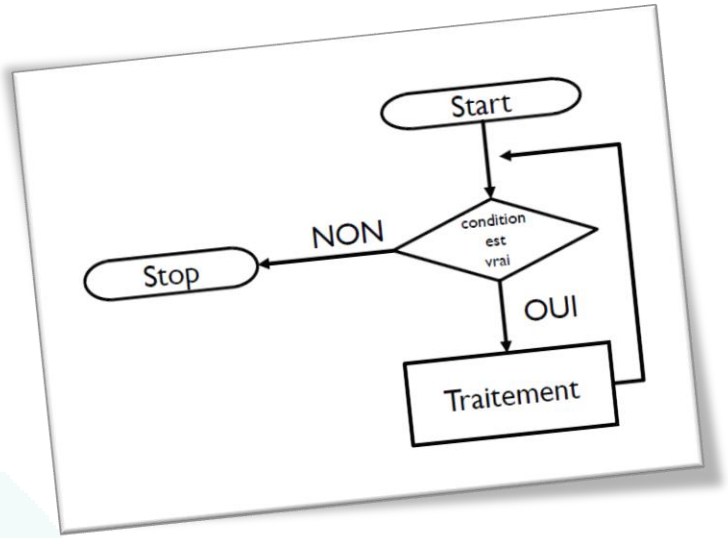
```
int a = 5;

switch (a) {
    case 5:
        System.out.println("a vaut 5");
        break;
    case 4:
        System.out.println("a vaut 4");
        break;
    case 3:
        System.out.println("a vaut 3");
        break;
    default:
        System.out.println("a ne vaut ni 3, ni 4, ni 5");
        break;
}
```

Les conditions

Jusqu'à Java 6, il n'était possible de faire un switch que sur des int et des char.

Depuis Java 7, le switch peut aussi s'appliquer à des String.

Attention ! Le mot-clé break n'est pas obligatoire en Java, mais dans le cas où il n'est pas indiqué, les cas suivant sont exécutés en cascade.

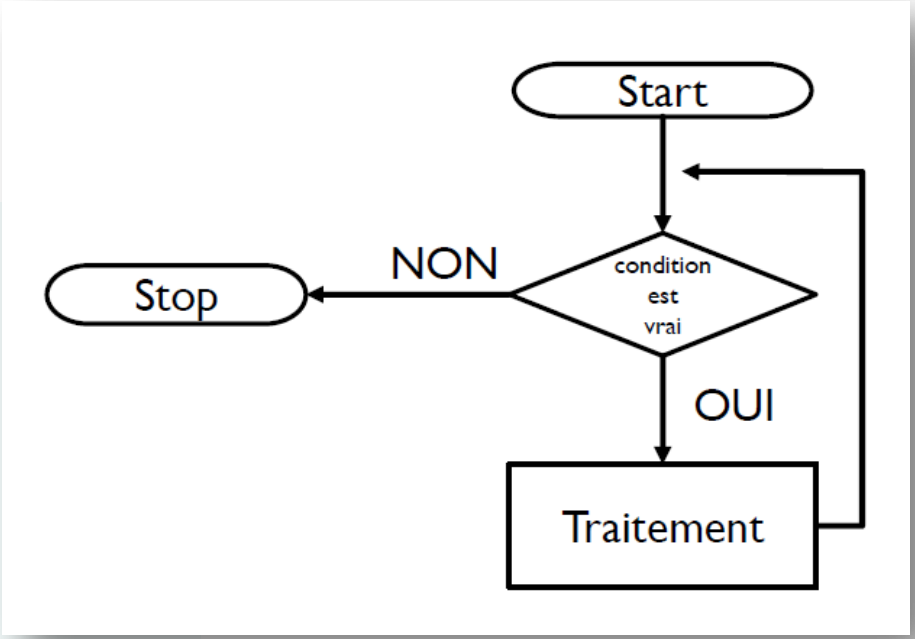


Les boucles

CHAPITRE 5

Les boucles

Les boucles while



Les boucles

Les boucles while

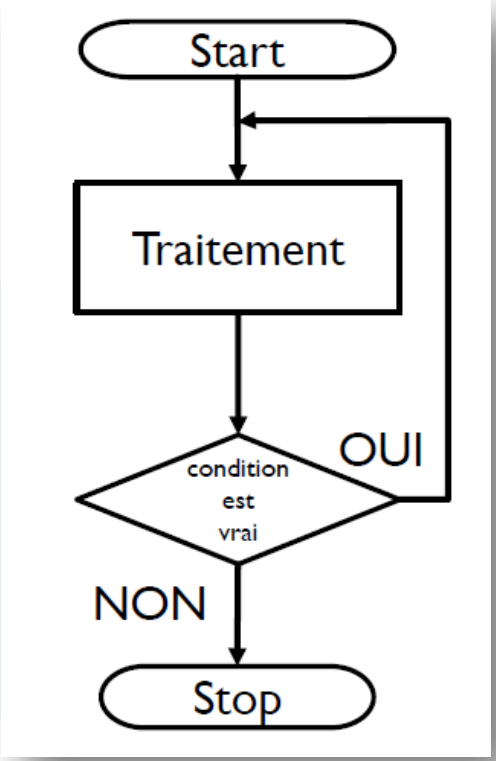
Une boucle while s'exécute tant que la condition indiquée renvoie true. Si celle-ci renvoie false, elle n'itère plus.

Syntaxe :

```
int i = 0;

while (i < 10) {
    System.out.println(i);
    i++;
}
```

Les boucles

Les boucles do … while



Les boucles

Les boucles do … while

Les boucles do… while ressemblent beaucoup aux boucles while, à la seule différence près qu'elles s'exécuteront toujours une première fois, même si la condition indiquée n'est pas remplie.

Syntaxe :

```
int i = 0;

do {
    System.out.println(i);
    i++;
} while (i < 10);
```

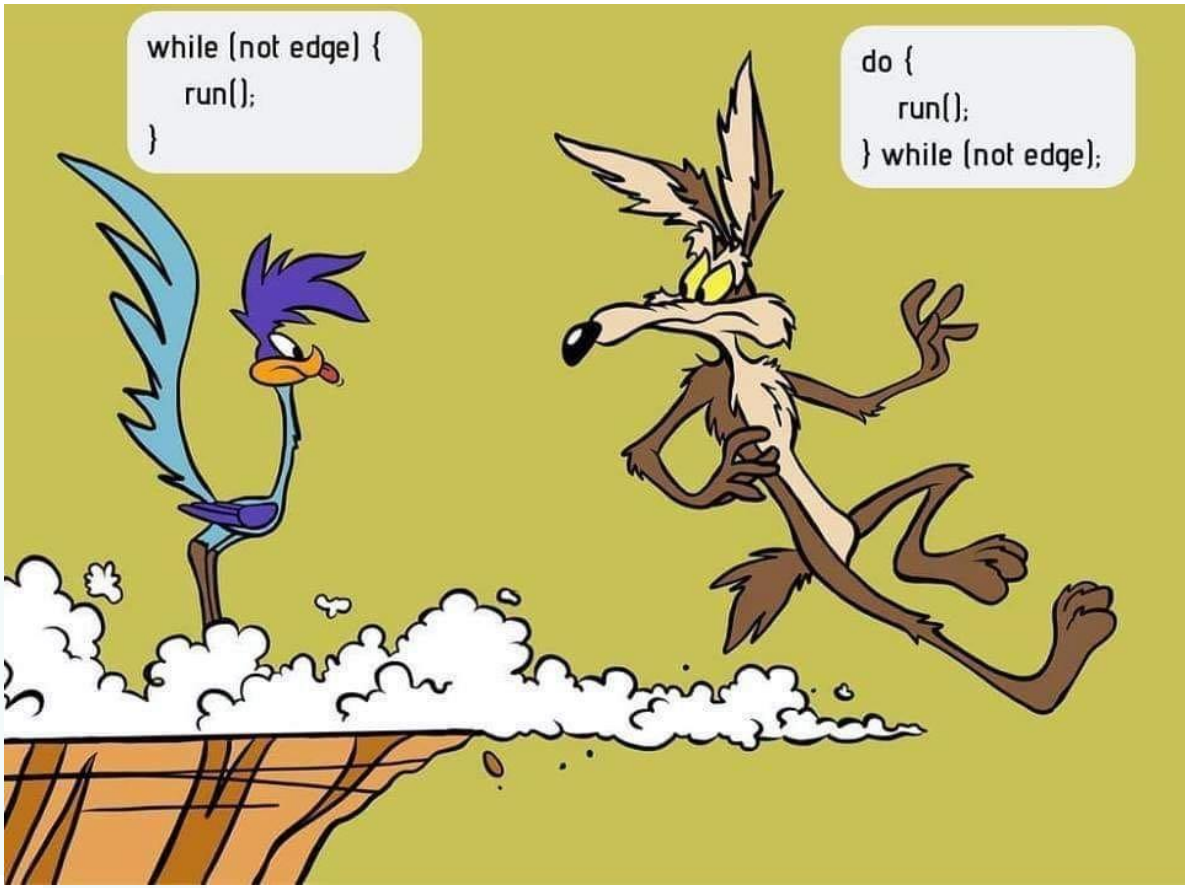
Les boucles

while(cond) {…}

Vs

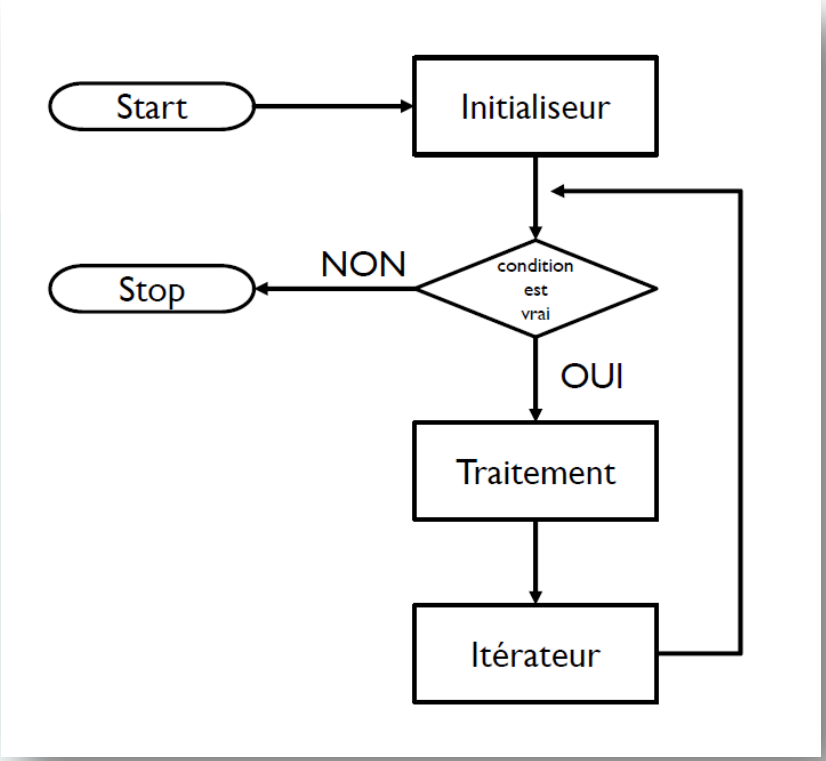
do {…} while(cond);

Les boucles



Les boucles

Les boucles for



Les boucles

Les boucles for

Une boucle for permet de définir le nombre d'itération que l'on souhaite, c'est-à-dire le nombre de fois que le bloc de code dans la boucle va être exécuté.

Syntaxe :

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Les boucles

Les boucles foreach

La boucle foreach est une variante de la boucle for utilisée afin d'itérer sur des tableaux ou des collections.

Syntaxe :

```
for (int item : myArray) {
    System.out.println(item);
}
```

Les boucles

Informations complémentaires sur les boucles

On peut arrêter l'itération en cours et passer directement à l'itération suivante grâce au mot-clé continue

On peut arrêter toute itération dans la boucle grâce au mot-clé break

```
List<int[]> crazyList;
```

Tableaux et Collections

CHAPITRE 6

Les tableaux

- Les tableaux
  - C'est simplement un ensemble d'éléments de même type. La taille du tableau est fixée à la création et ne peut pas être modifiée.
  - On peut retrouver les éléments grâce à leur indice, qui représente en quelque sorte leur place dans le tableau.
  - L'index d'un tableau commence à 0, ce qui signifie qu'un tableau de 5 élément contiendra des éléments aux index 0, 1, 2, 3 et 4.

Les tableaux

- Déclaration et initialisation

```
int[] myArray = new int[5];

int[] myOtherArray = new int[]{9, 3, 4, 2, 6};
```

La taille du tableau doit être fixée explicitement ou implicitement dès l'initialisation !

Les tableaux

- Accès et modification

  Les accès et les modifications se vont via un index :

```
int a = myOtherArray[0];

myOtherArray[5] = 123;
```

La tentative d'accéder ou de modifier un emplacement inexistant du tableau causera une ArrayIndexOutOfBoundsException.

Les tableaux

- Les tableaux à plusieurs dimensions
  - Certaines données sont représentées par un tableau à plusieurs dimensions. En réalité, on manipule simplement un tableau qui contient lui-même des tableaux.
  - Il n'y a pas de limite au nombre de dimensions, mais au-delà de trois dimensions, le tableau devient complexe à manipuler.

Les collections

Les Collections

Les collections ressemblent beaucoup aux tableaux, mais chaque type de collection a un comportement bien précis.

Nous ne verrons pas tous les types de collections qui existent en Java, car certaines sont très peu utilisées, ou que dans certains cas particuliers.

Les collections

« List » :

- Contiennent une collection ordonnée d'éléments.
- L'utilisateur contrôle l'endroit de la liste où chaque élément est inséré.
- L'accès aux éléments se fait via leur *index* (leur position dans la liste), ou via une recherche dans cette liste.
- Une liste accepte des doublons.

Les collections

Les listes en Java :

- ArrayList

Les listes

Les ArrayList

Comme les autres listes, les ArrayList ressemblent beaucoup aux tableaux, si ce n'est que leur taille n'est pas fixe, elles s'adaptent au nombre d'éléments qu'on y insère.

Il existe des méthodes spécifiques permettant l'ajout ou le renvoi d'éléments à l'ArrayList.

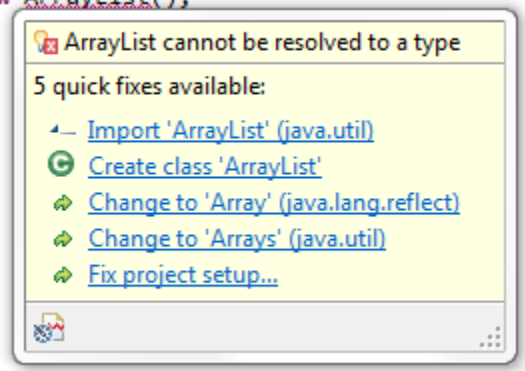
Plus aisées à manipuler, elles demandent de fait plus de ressources.

Les listes

Déclaration et instanciation :

Pour utiliser une ArrayList, il faut importer « ArrayList » !

```
ArrayList arrayList = new ArrayList();
```



Déclaration d'une ArrayList de taille indéfinie, sans définir le type de contenu (déconseillé) :

```
ArrayList arrayList = new ArrayList();
arrayList.add(1);
```

Les listes

Bonne pratique :

« Typer » l'ArrayList, nous verrons pourquoi plus tard

Donner une taille initiale à son ArrayList

Exemple :

```
ArrayList<Integer> arrayList = new ArrayList<Integer>(5);
arrayList.add(1);
```

⇒ arrayList est une ArrayList prévue pour contenir 5 éléments de type Integer

⇒ Si on ne déclare pas de taille, l'ArrayList prendra une taille par défaut

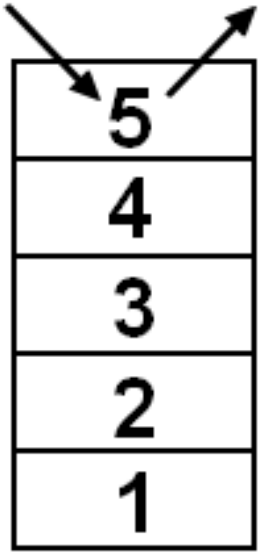
Les listes

On utilise entre autres les méthodes *add*, *get* ou *remove* pour ajouter, recevoir ou retirer des éléments.

Les collections

« Stack » :

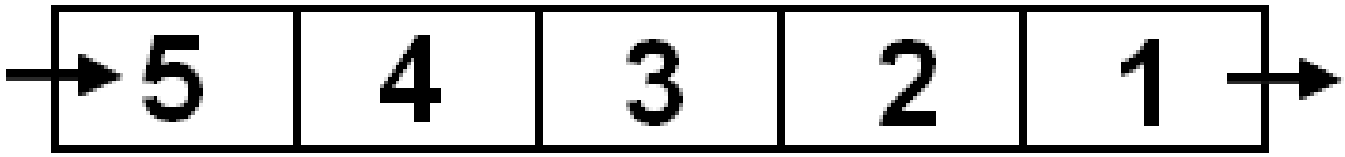
- Se comporte comme une pile d'éléments
- Suit le principe « LIFO » (Last In First Out)
- Exemple : une pile d'assiette



Les collections

« Queue » :

- Se comporte comme une file d'éléments
- Suit le principe « FIFO » (First In First Out)
- Exemple : une file d'attente

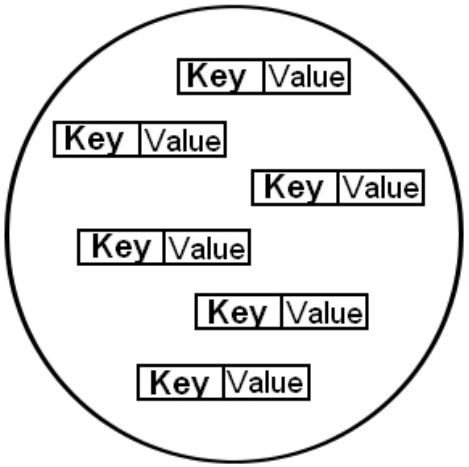


Les stacks et les queues

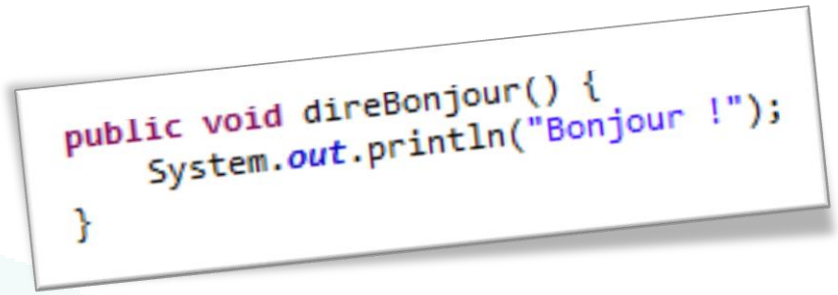
- Les ArrayDeque (Deque est prononcé « Deck »)
  - En Java, il existe des collections qui peut se comporter à la demande comme une pile ou comme une file. C'est le cas par exemple de l'ArrayDeque.
  - On la déclare et l'initialise comme une ArrayList.
  - On utilise entre autres les méthodes *addFirst, addLast, getFirst, getLast, removeFirst* ou *removeLast* selon le besoin.

Les Map

Une « Map » est ce qu'on appelle communément un dictionnaire, c'est-à-dire qu'elle contient des couples clé - valeur.



- Le type de Map le plus courant est la HashMap



Les méthodes

CHAPITRE 7

Les méthodes

- Utilité
  - L'utilisation de méthodes permet d'éviter la répétition de morceaux de codes. On appelle cela « factoriser » le code.
  - En Programmation Orienté Objet, elles permettent de définir le comportement de l'objet à l'appel de cette méthode.
- Convention
  - En Java, les noms des méthodes suivent la même convention que le nom des variables.

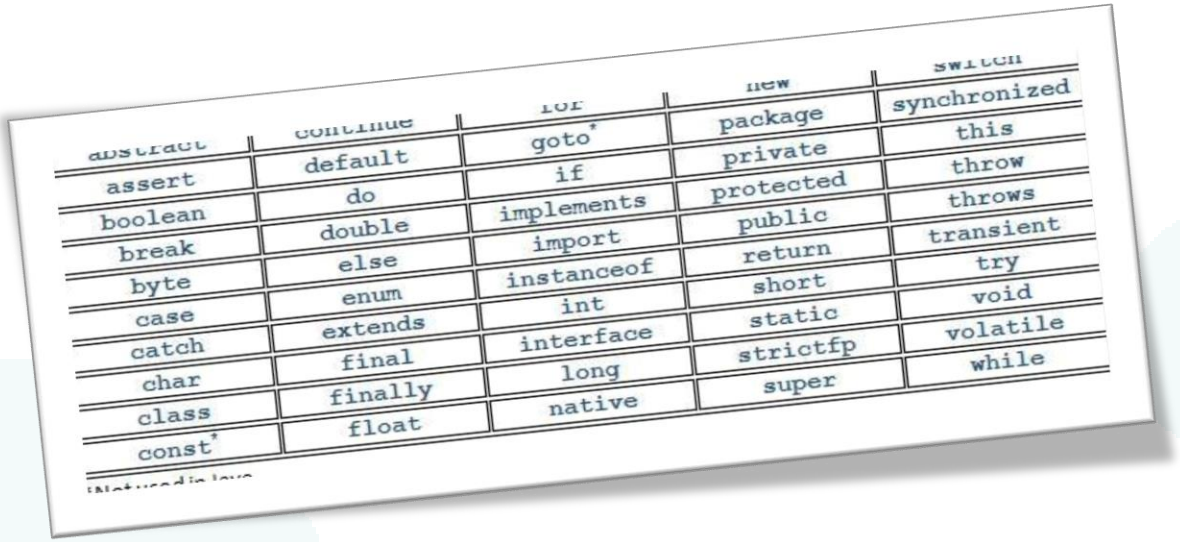
Les méthodes

Exemple :

```
public void direBonjour() {
    System.out.println("Bonjour !");
}
```

Cette méthode affiche simplement le texte « Bonjour ! » en console dès qu'elle est appelée.

Pour l'instant, ne vous préoccupez pas des mots public et void devant la méthode, ils seront expliqués lorsque nous aborderons l'Orienté Objet en Java.



abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const*	float	native	super	while

Autres informations utiles

Informations Utiles

Commentaires

Les commentaires peuvent être uni- ou multi-lignes.

- Commentaire uni-ligne

```
// Ceci est un commentaire uni-ligne
```

- Commentaire multi-ligne

```
/*
 * Ceci est
 * un commentaire
 * multi-ligne
 */
```

Informations Utiles

Lire une entrée clavier

Dans un programme console, il est possible de lire une entrée de l'utilisateur.

Le code pour obtenir cette entrée est :

```
Scanner sc = new Scanner(System.in);
String input = sc.nextLine();
```

```
int inputInt = sc.nextInt();
sc.nextLine();
```

Informations Utiles

Les mots réservés en Java

abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const*	float	native	super	while

*Not used in Java