DÉVELOPPEMENT D'APPLICATIONS WEB

FRONTEND | **BACKEND** | BASES DE DONNÉES

JAVA

Introduction

POLYTECH DIJON

charles.meunier@u-bourgogne.fr

I JAVA

Mais encore ?

Plusieurs paradigmes

JAVA est un langage de programmation inspiré du C++ qui prend en charge plusieurs paradigmes de programmation :

- La programmation **Impérative**
- La programmation **Orientée Objets** (classes, interfaces, héritage, polymorphisme).
- La programmation **générique** (équivalent des templates en C++)
- La programmation **fonctionnelle** (lambdas et streams)
- La programmation **parallèle** (multithreading)
- La programmation **déclarative** (via la programmation fonctionnelle)

Impératif vs déclaratif

"Soit un premier tableau d'entiers, créez un second tableau à partir des entiers impairs contenus dans le premier tableau et dont la valeur sera multipliée par deux"

Programmation impérative

décrit le traitement à réaliser étape par étape

```
int[] entiers = {1, 2, 3, 4, 5, 6, 7};

ArrayList<Integer> doubleImpairs = new ArrayList<>();

for(int i = 0; i < entiers.length; ++i)
{
  int entier = entiers[i];

  if(entier % 2 == 1)
    doubleImpairs.add(entier * 2);
}
```

Programmation déclarative

décrit le résultat attendu

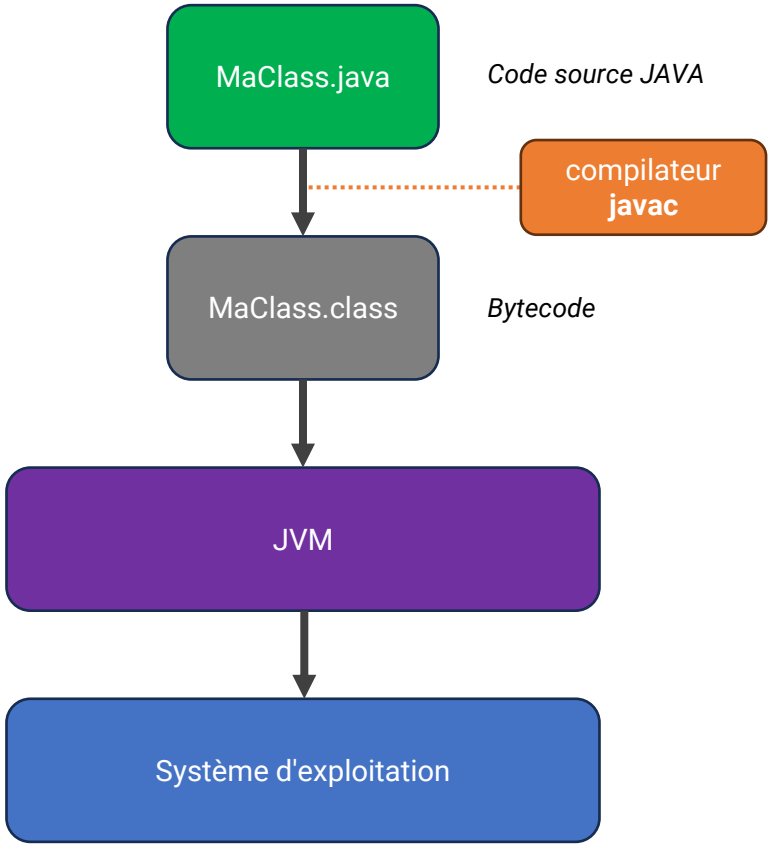
```
int[] entiers = {1, 2, 3, 4, 5, 6, 7};

int[] doubleImpairs = Arrays.stream(entiers)
                            .filter(n -> n % 2 == 1)
                            .map(n -> n * 2)
                            .toArray();
```

Un langage hybride

JAVA est un langage **semi-compilé**.

1. Le code source est compilé pour donner un **byte-code**.
2. La **Machine Virtuelle Java** (JVM) interprètera ensuite ce byte-code en fonction de la plateforme d'exécution.



La JVM permet aux applications JAVA d'être "**portables**".

- La JVM doit exister sur la plateforme cible
- L'application ne doit pas utiliser des API propres à un système d'exploitation

JDK

Le **Java Development Kit** (JDK) est un élément essentiel pour la création et l'exécution d'applications JAVA.

Il fournit :

- Un **compilateur JAVA** (javac)
- Le **Java Runtime Environment** (JRE) qui inclut
  - la JVM
  - les bibliothèques standard JAVA
- Des **outils de développement**
  - jdb (debogueur JAVA),
  - java (pour exécuter les applications JAVA compilées)
  - javadoc (pour générer automatiquement la documentation à partir des codes sources)
  - jar (pour la création et la manipulation de fichiers JAR)

JVM

La Java Virtual Machine (JVM) fournit un environnement d'exécution pour les applications JAVA.

Interprétation du bytecode
afin de produire un code machine compatible avec la plateforme cible

Gestion de la mémoire
et libération automatique des objets qui ne sont plus utilisés (garbage collector ou ramasse miettes)

Gestion des exceptions
qui se produisent durant l'exécution de l'application.

Gestion des threads
dans le cadre des applications multithreads

II PREMIÈRE APPLICATION

Point d'entrée de l'application

Une application JAVA doit contenir une classe proposant une fonction static main qui servira de point d'entrée au programme à son exécution :

MyApplication.java (1)

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

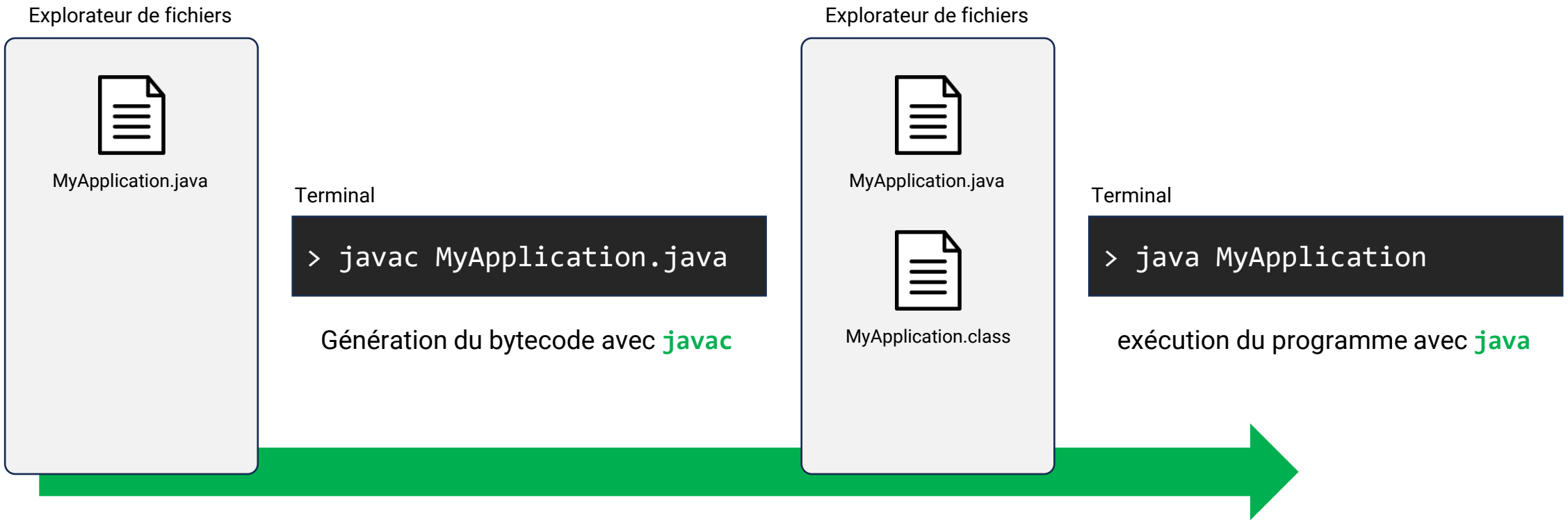
(2) (3) (4)

1. Le nom de la classe doit être le même que le nom du fichier (casse comprise)
2. La fonction main est une méthode publique et statique.

   Une seule fonction main par application.
3. Le paramètre args contient les paramètres transmis au programme lors de son exécution.
4. La fonction main ne retourne rien (void). Par défaut, un programme java retourne toujours le code 0. Pour retourner un code d'erreur, on utilise :

```
System.exit(-23);
```

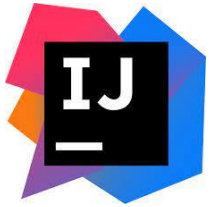
Compiler un programme JAVA



Environnements de développement intégrés

ECLIPSE

NETBEANS



INTELLIJ IDEA

VS CODE

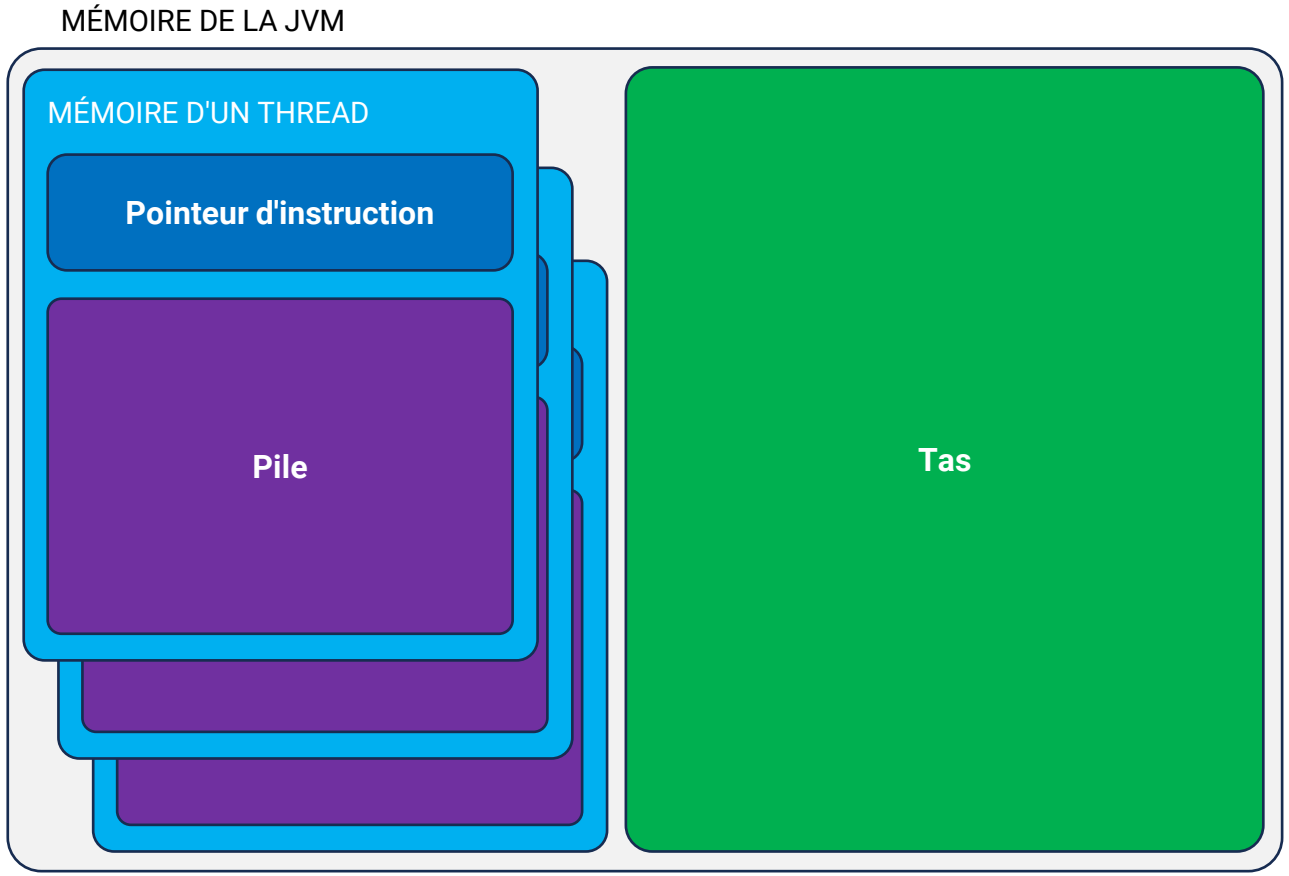
XCODE

II

GESTION DE LA MÉMOIRE

Types et garbage collector

Pile et Tas



Pointeur d'instruction
Contient l'adresse de la prochaine instruction à exécuter pour un thread donné.

Pile
Stocke les variables locales, les paramètres passés aux fonctions, les valeurs de retour des fonctions, dans le contexte d'un thread donné.

Tas
Commun à tous les threads, stocke tous les objets complexes du programme (tableaux, instances de classes).

TYPES PRIMITIFS

JAVA fournit un ensemble de types dits primitifs :

Entiers

- byte *8 bits signé*
- short *16 bits signé*
- int *32 bits signé*
- long *64 bits signé*

Décimaux

- float *32 bits*
- double *64 bits*

Caractères

- char *16 bits unicode*

Booléens

- boolean ***true*** *ou* ***false***

Les types primitifs ne sont pas des objets. Ils ne nécessitent pas d'instanciation à leur création et ne disposent pas de méthode.

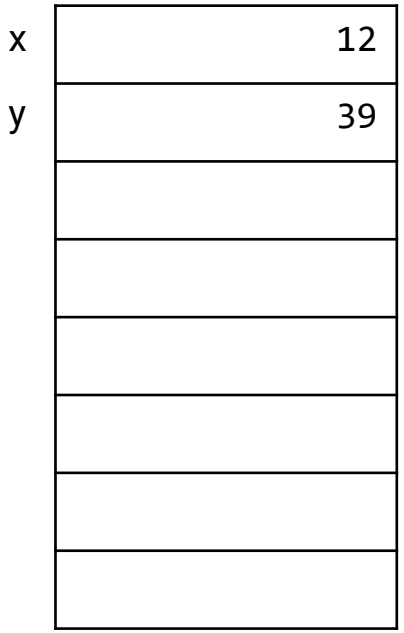
TYPES PRIMITIFS VS RÉFÉRENCES

Les variables locales de **type primitif** sont **stockées sur la pile** et **contiennent directement la valeur** de la donnée :

Code Java

```java
int x = 12;
int y = 39;
```

Pile



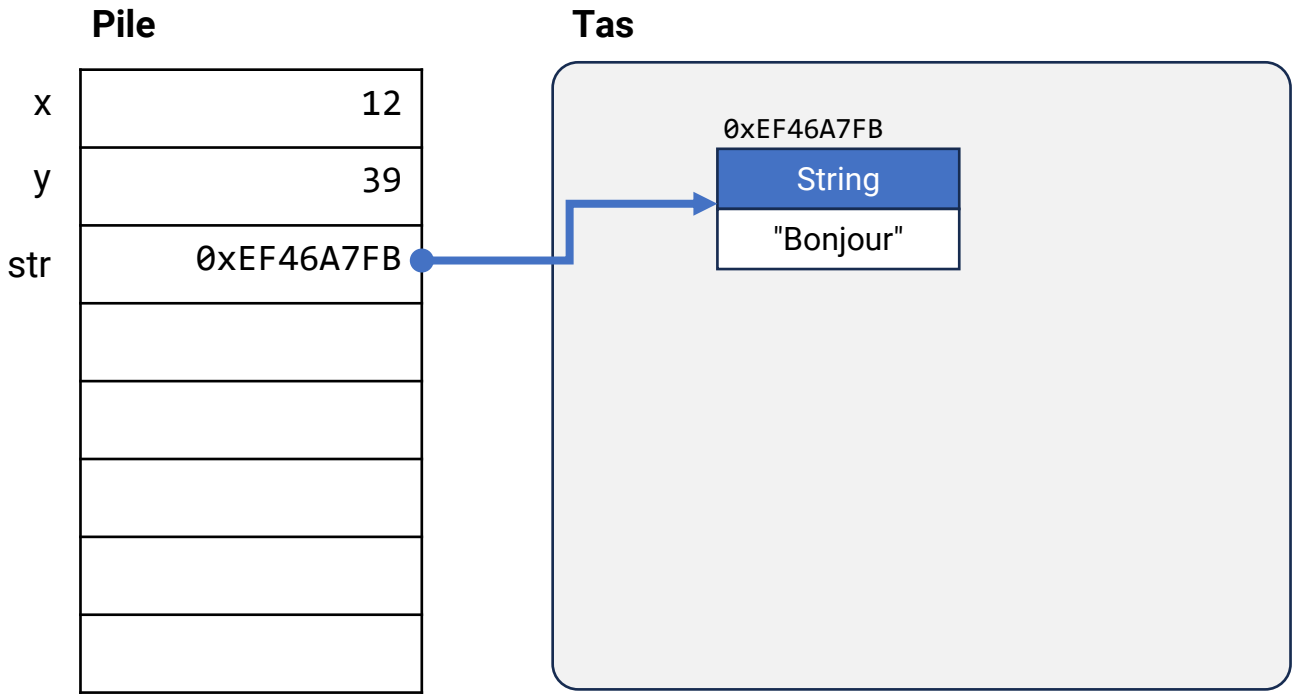
Tas

TYPES PRIMITIFS VS RÉFÉRENCES

Les variables locales de **type complexe** sont **stockées sur la pile** et contiennent **une référence vers la donnée** qui est **stockée dans le tas** :

Code Java

```
int x = 12;
int y = 39;
String str = "Bonjour"
```



AFFECTATION ET COMPARAISON

Les opérateurs d'affectation (=) et de comparaison (==) de JAVA travaillent avec les valeurs contenues dans les variables manipulées.

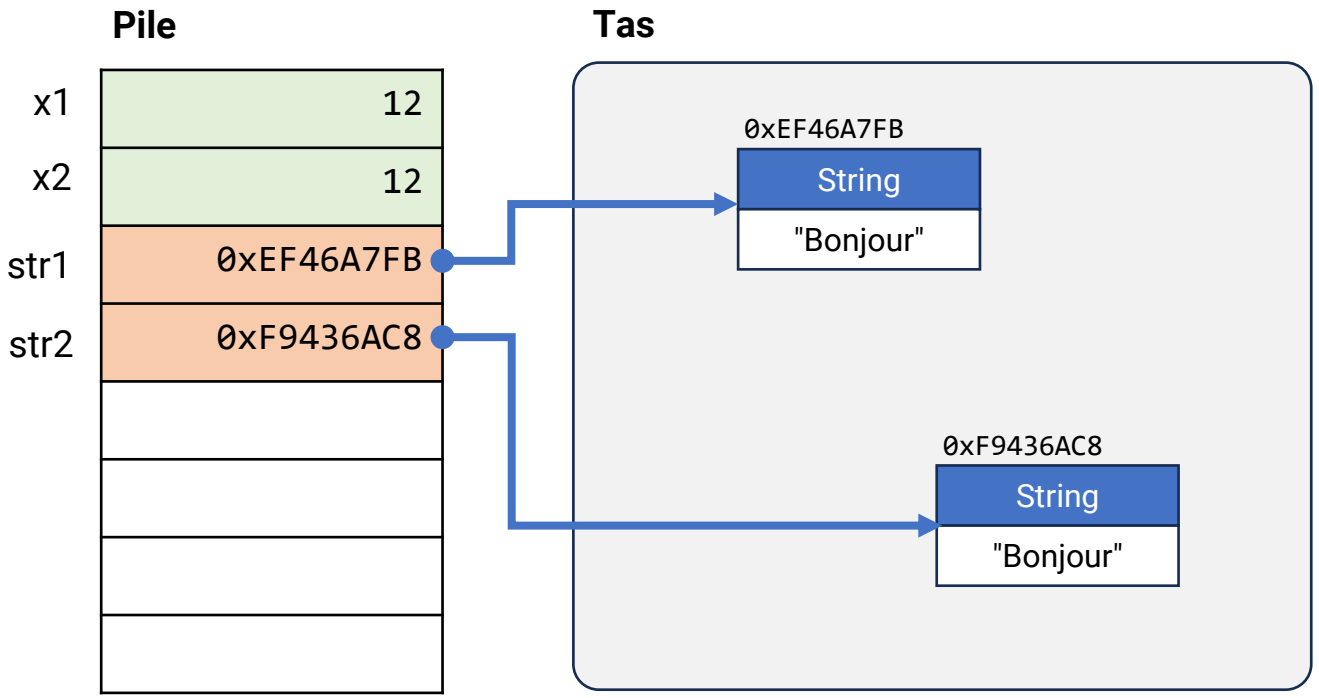
Pour une variable de type primitif, il s'agit de la valeur effectivement stockée dans la variable.

Pour une variable de type complexe, il s'agit de l'adresse de la donnée stockée dans le tas.

Code Java

```java
int x1 = 12;
int x2 = 12;
x1 == x2        // -> true

String str1 = "Bonjour"
String str2 = "Bonjour"
str1 == str2    // -> false
```



GARBAGE COLLECTOR

Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

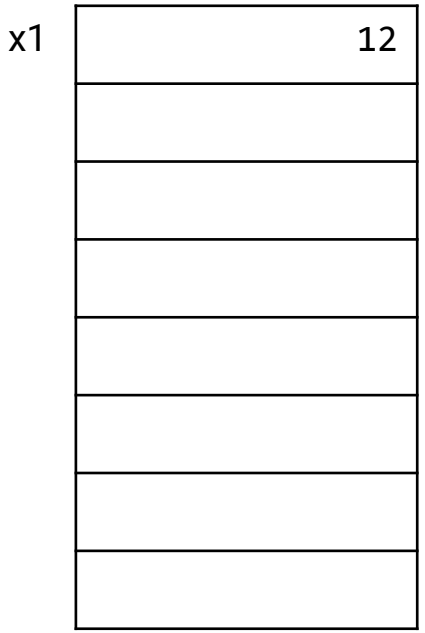
Code Java

Pointeur d'instruction

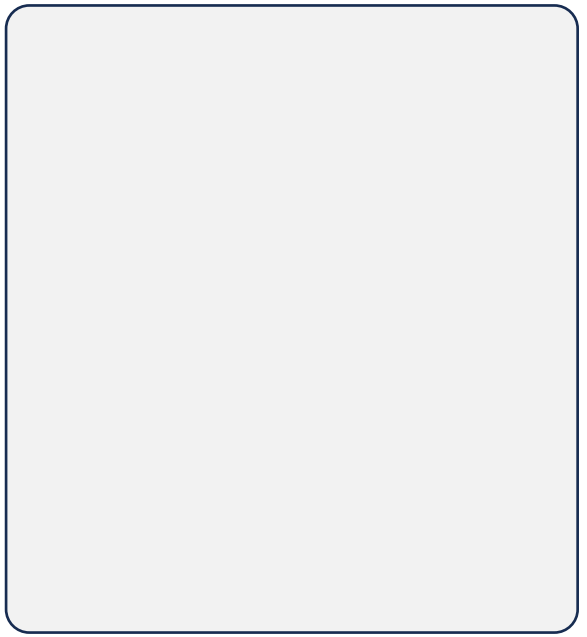
```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pile



Tas



GARBAGE COLLECTOR

Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

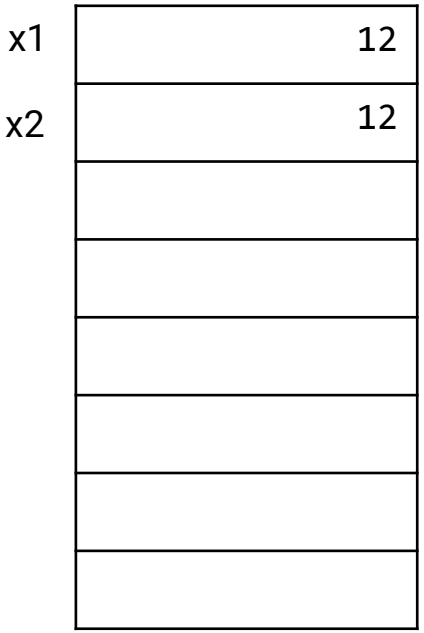
Code Java

Pointeur d'instruction

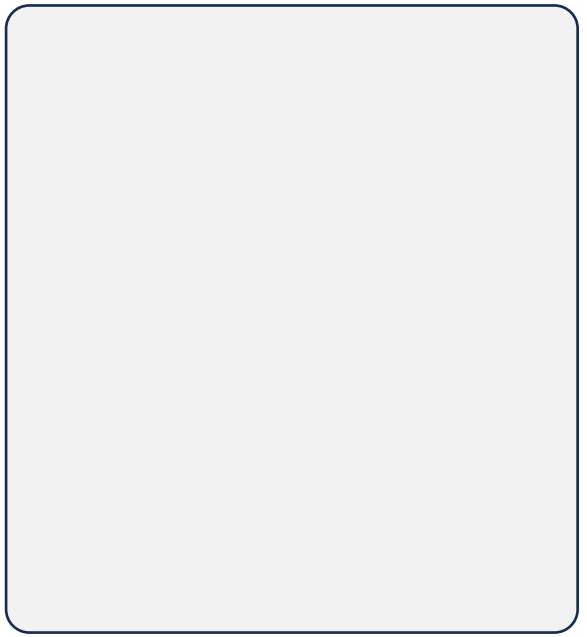
```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pile



Tas



GARBAGE COLLECTOR

Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

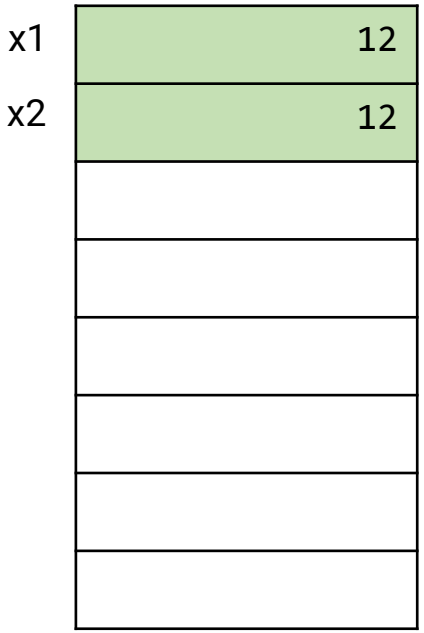
Code Java

Pointeur d'instruction

```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pile



Tas

GARBAGE COLLECTOR

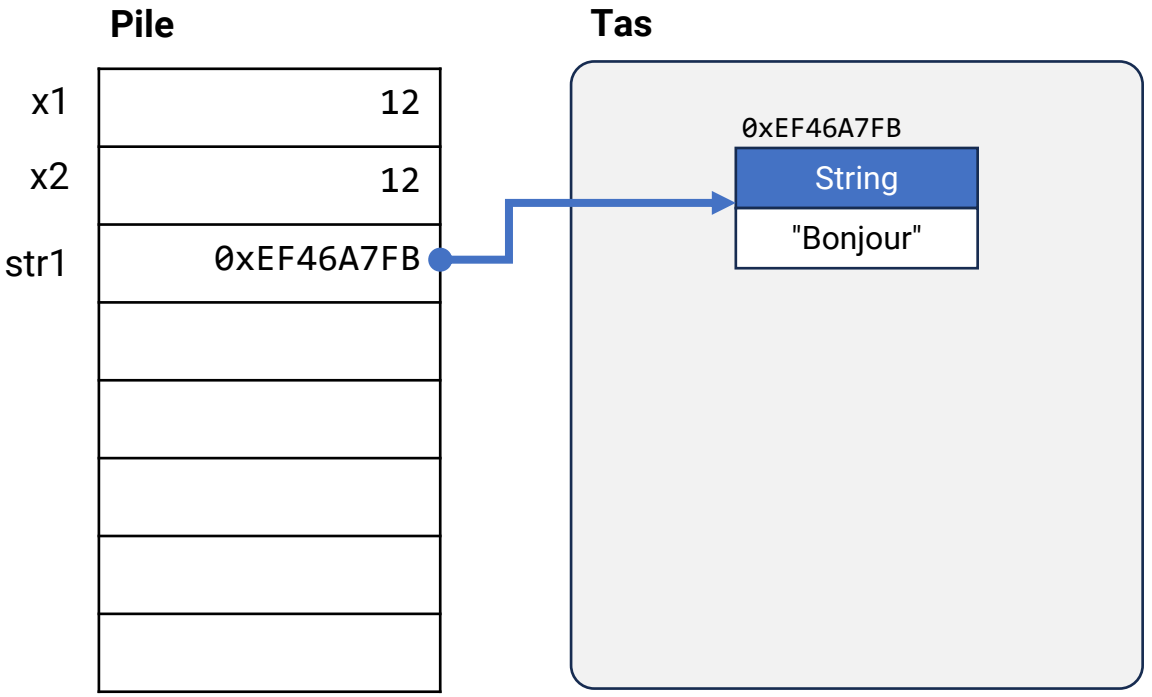
Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

Code Java

```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pointeur d'instruction



GARBAGE COLLECTOR

Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

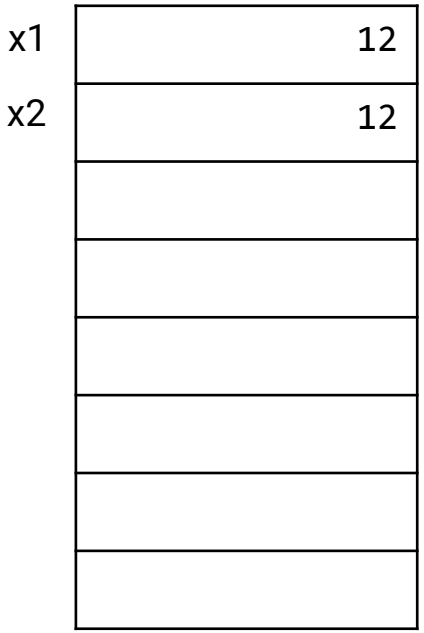
Code Java

```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pointeur d'instruction

Pile



Tas

GARBAGE COLLECTOR

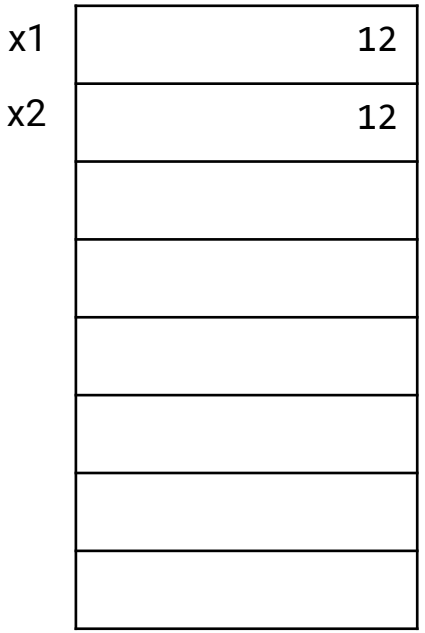
Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

Code Java

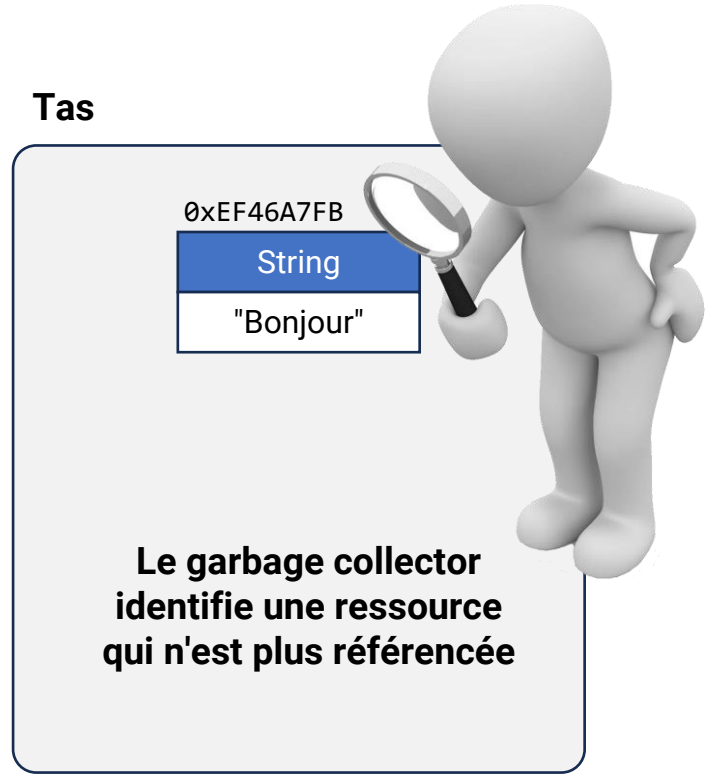
```
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pile



Tas



GARBAGE COLLECTOR

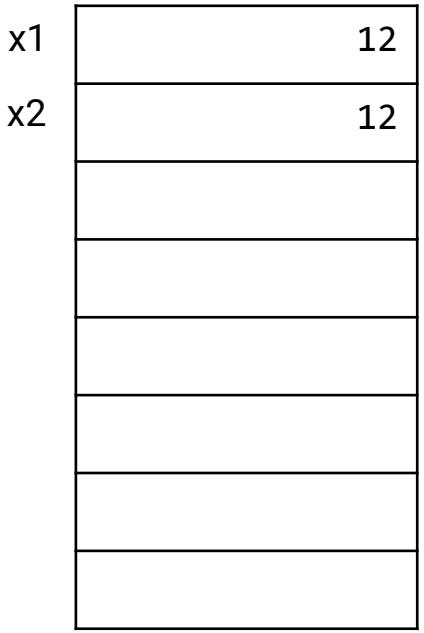
Le garbage collector identifie les ressources qui ne sont plus référencées et libère automatiquement la mémoire qui leur était allouée.

Code Java

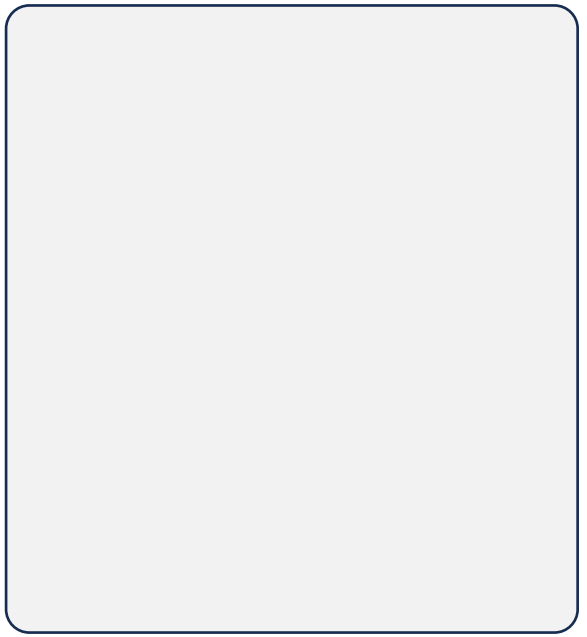
```java
int x1 = 12;
int x2 = 12;

if(x1 == x2)
{
  String str1 = "Bonjour"
}
```

Pile



Tas



IV SYNTAXE GÉNÉRALE

Déclaration de variables

```
//Types primitifs
int entier = 12;                                    //!\ Une variable doit toujours être initialisée /!\
float decimal = 27.8;
boolean bool = true;

//Types complexes
Float objetDecimal = 12.8F;
System.out.println(objetDecimal.isInfinite());      // -> false

int[] tableauEntiers = {1, 2, 3, 4};
System.out.println(tableauEntiers.length);          // -> 4

ArrayList<entier> listeEntiers = new ArrayList<>();
listeEntiers.add(12);
System.out.println(listeEntiers.size());             // -> 1
```

Alternatives

if / else if / else

```
if(a == b) {
  ...
}
else if(a == c) {
 ...
}
else {
 ...
}
```

switch

```
switch(a) {
  case 0:
    ...
    break;
  case 1:
    ...
    break;
  default:
    ...
}
```

Boucles

for

```
int[] tab = { ... };

for(int i = 0; i < tab.length; ++i) {
  ...
}
```

ou encore

```
int[] tab = { ... };

for(int i: tab) {
  ...
}
```

while

```
while(condition) {
 ...
}
```

do/while

```
do {
 ...
} while (condition)
```

Fonctions

Les fonctions n'existent pas en Java

Classes

MaClasse.java (1)

```
public class MaClasse {
  //Attributs de la classe (2)
  private int entier;
  private final entierConstant; (3)

  //Constructeur de la classe
  MaClasse() { (4)
    ...
  }

  MaClasse(int entier) { (5)
    ...
  }
}
```

1. Rappel : le nom de la classe doit être le même que le nom du fichier (casse comprise)
2. Bonne pratique : lister tous les attributs d'une classe au début de cette dernière.
3. final permet de déclarer un attribut en lecture seule. L'attribut est initialisé dans le constructeur et ça valeur ne peut ensuite plus être modifiée
4. Le constructeur porte le même nom que la classe.
5. Une classe peut avoir plusieurs constructeur.

Méthodes

```
public class MaClasse {
  //Attributs de la classe
  private int entier;

  int getEntier() {
    return entier;
  }
}
```

En JAVA, une fonction est nécessairement un membre d'une classe. **Toutes les fonctions sont donc des méthodes**.

Ceci est valable pour la fonction `main` également.

Enumérations

```
public enum Status {
  ADMIN,
  TEACHER,
  STUDENT
}
```

Les énumérations permettent de définir un ensemble de valeurs possibles pour un type donnée.

```
//Déclarer une variable de type Status
Status myStatus = Status.STUDENT;
```

En JAVA les **enum** sont considérées comme des classes qui peuvent avoir des attributs et un constructeur.

```
for(Status status: Status.values()) {
  System.out.println(status);
}
```

Il est possible d'obtenir l'ensemble des valeurs d'une énumération sous la forme d'un tableau.

Héritage

```java
public class Human {
  public void think() {
    System.out.prinln("Science, literature, ecology, ...");
  }
}
```

```java
public class Student extends Human {
  public void think() {
    System.out.prinln("Alcool, ..., more alcool?");
  }
}
```

Le mot clé extends permet de spécialiser une autre classe dans une relation d'héritage.

L'héritage multiple n'existe pas en JAVA.

Le polymorphisme permet de redéfinir le comportement des fonctions de la classe mère.

Abstraction

```
public abstract class Shape {
  protected String color = "Blue";

  public abstract void draw();
}
```

```
public class Square extends Shape {
  public void draw() {
    System.out.prinln("I know how to draw a " + this.color + " square");
  }
}
```

Si une classe contient une méthode abstraite, est doit être définie comme abstract (la classe et les méthodes abstraites).

Toute classe qui hérite d'une classe abstraite est elle-même abstraite à moins qu'elle ne définissent le code de toutes les méthodes abstraites dont elle hérite.

Les classes abstraites ne peuvent pas être instanciées.

Interface

```java
public interface Shape {
  public void draw();
}
```

```java
public class Square implements Shape {
  public void draw() {
    System.out.prinln("I know how to draw a square :)");
  }
}
```

Les interfaces permettent de définir un contrat garantissant que les classes qui implémentent une interface proposent bien toutes les méthodes décrites par l'interface.

Une classe peut implémenter plusieurs interfaces.