ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H-301
Programmation orientée objet

TP1
Introduction au langage JAVA
Classes et objets

?

- Langage de programmation **orienté objet** créé par Sun Microsystems.
- Points forts :
  - langage **portable** entre différents OS.
  - **librairies standards** (API Java) : String, Math, File, Zip, Network, Mail, …

- Le code Java est compilé en un langage intermédiaire (*bytecode*) sur une **machine virtuelle** (JVM) multi plates-formes.
- **Tout est objet** (mis à part les types primitifs).

Terminologie Java

- **JVM** : *Java Virtual Machine*
- **JRE** : *Java Runtime Environment*
  - Ensemble d'outils pour **exécuter** des programmes Java sur toutes les plates-formes supportées.
  - Est constitué d'une *JVM* et d'une bibliothèque standard à toutes les plates-formes.
- **JDK** : *Java Development Kit*
  - *JRE* + outils de developpement comme le **compilateur** (*javac*) qui produit du *bytecode* qui sera interprété par la *JVM*.
- **IDE** : *Integrated Development Environment*
  - Ensemble d'outils pour le **développement** logiciel (Ex : Eclipse, NetBeans)

Un premier programme

- Fichier `HelloWorld.java` :

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

- Compilation (va créer un fichier `HelloWorld.class`):

```
$> javac HelloWorld.java
```

- Exécution :

```
$> java HelloWorld
```

Syntaxe Java

- La syntaxe de Java est très proche de celle de C++

- Référence :
  - http://java.sun.com/docs/books/tutorial/java/

Types primitifs, variables et portée

- Les **types primitifs** les plus utilisés sont :
  - `boolean` : valeur de vérité (`true` / `false`)
  - `char` : caractère sur 16 bits
  - `int` : entier sur 32 bits
  - `double` : réel *IEEE754* sur 64 bits

- La déclaration et l'assignation des **variables** fonctionnent comme en C++ :
  - `char uneLettre = 'r';`
  - `double somme = 25.63;`

- La notion de **portée** est la même qu'en C++ (blocs `{}`).

Tableaux

- Comme en C++, les **tableaux** Java ont une taille *N* fixe définie par le développeur.
  Les éléments du tableaux sont indicés de *0* à *N-1*.

- Déclarer un tableau :
  - `int[] unTableau = new int[10];`
  - `char[] unAutreTableau = {'J' , 'a' , 'v' , 'a'};`
- Accéder à une case d'un tableau :
  - `unTableau[7] = 700;`
- Taille d'un tableau :
  - `int length = unTableau.length;`

Opérateurs et expressions

- Les **opérateurs** ressemblent à ceux de C++. Dans l'ordre de priorités :
  - multiplication : * , / (attention à la division entière) , %
  - addition : + , -
  - relation : < , <= , > , >=
  - egalité : == , !=
  - *et* logique : &&
  - *ou* logique : | |

- **Expressions** : Le contrôle des priorités est comme en algèbre, avec des parenthèses.

Instructions de contrôle de flux

- `if(condition) {instructions}`
- `if(condition) {instructions} else {instructions}`
- `while(condition) {instructions}`
- `do {instructions} while(condition);`
- `for(initialisation;condition;increment){…}`

For-Each : Autre forme du `for`

- Syntaxe facilitée pour **parcourir des tableaux** *(ou des Collections : voir plus loin)*

```
int sum = 0;
int[] numbers = {1,2,3,4};

for (int item : numbers){
  sum += item;
}
// sum == 10
```

Classe String

- En Java, les **chaînes de caractères** sont des objets
- Création : `String greeting = "Hello world!";`
- Longueur : `int length = greeting.length();`
- Accès à un caractère : `char c = greeting.charAt(2);`
- Concaténation :
  - `string1.concat(string2);`
  - `String maChaine = “Hello, ” +” world” + “!”;`
- Affichage en console :
  - `System.out.println(“I say : “ + greeting);`
- http://java.sun.com/docs/books/tutorial/java/data/strings.html

Conversion de types

- Convertir des **String en nombres** :
  - `double a = Double.valueOf("10.5");`
  - `int b = Integer.valueOf("20");`
  - `double c = a + b;`

- Convertir des **nombres en String** :
  - `int i = 123;`
  - `String nombre = String.valueOf(i);`
  - **ou** : `nombre = Integer.parseInt(i);`

- La classe `Double` est un "*Wrapper*" pour le type primitif `double`, `Integer` un "*Wrapper*" pour `int`

Arguments de la ligne de commande

```
public class program
{
   public static void main(String[] args)  {
      if (args.length == 2) {
         double a = Double.valueOf(args[0]);
         double b = Double.valueOf(args[1]);
         System.out.println(a+b);
      }
      else {
         System.out.println("erreur");
      }
   }
}
```

```
$> javac program.class
$> java program 2.4 7
9.4
$> _
```

- Environnement de développement (IDE)
- Multi plates-formes (Linux, Windows, OSX, …)
- Logiciel libre

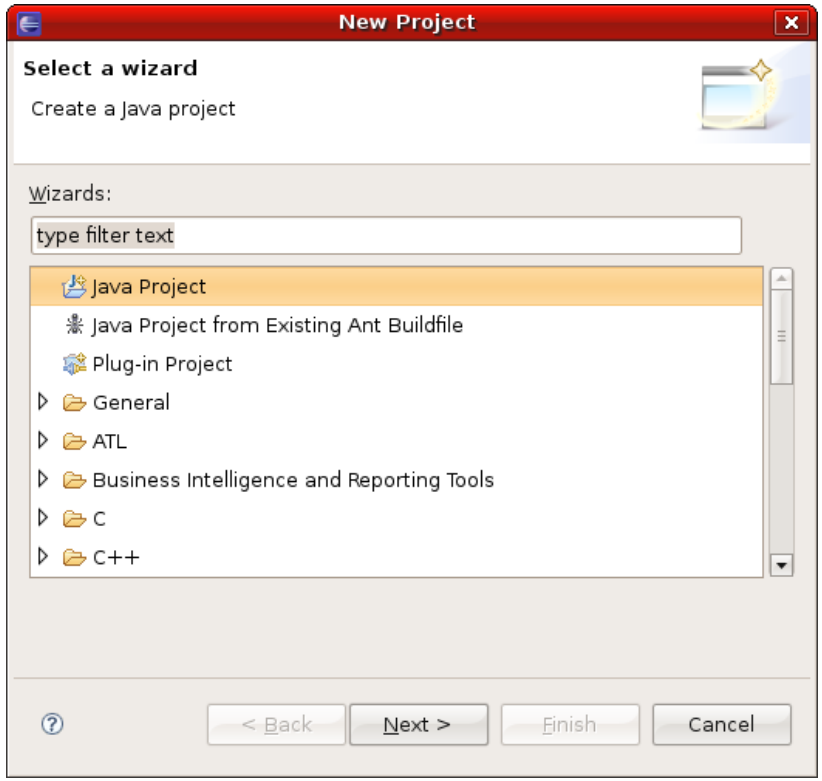
- Pour travailler chez vous :
  - Télécharger et installer la *JDK SUN 7*
    - http://java.sun.com/javase/downloads/index.jsp#jdk
  - Télécharger et installer *Eclipse IDE for Java Developpers*
    - http://www.eclipse.org/downloads/

Lancement d'Eclipse dans les salles

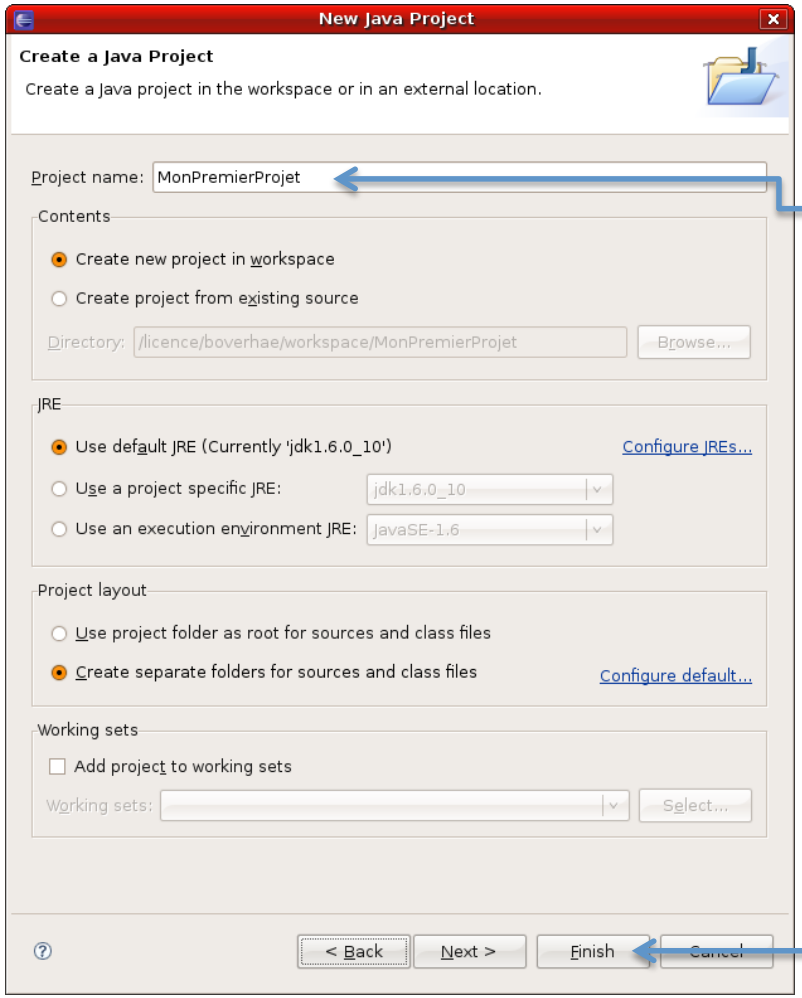
Cliquer sur Workbench

Créer un nouveau projet

- File -> New… -> Project
- Choisir « Java Project » et cliquer sur Next



Créer un nouveau projet



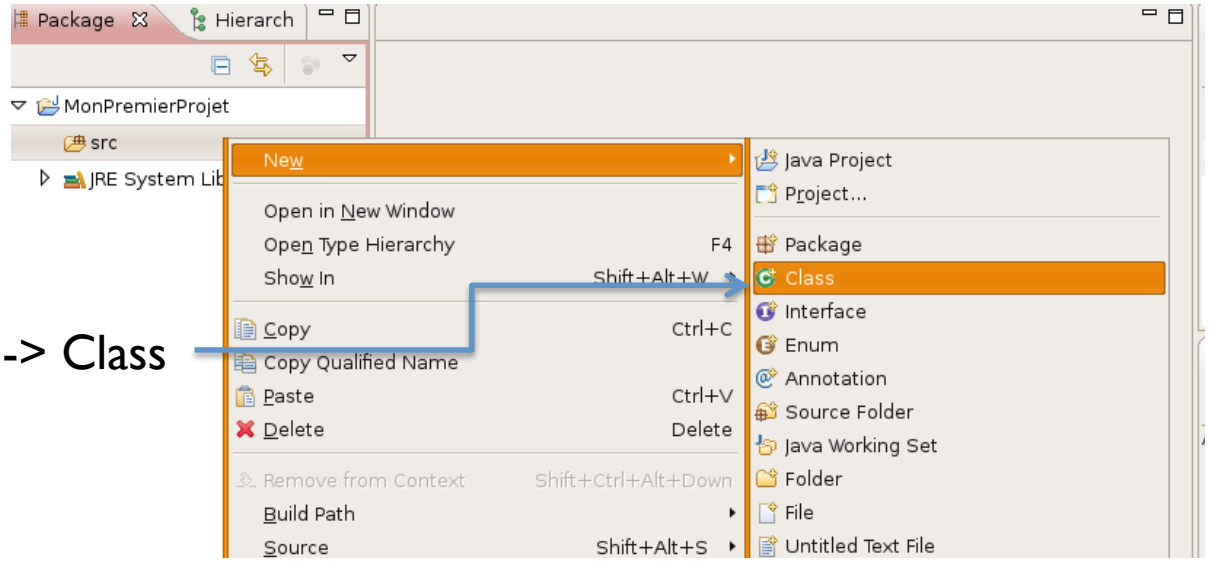
Donner un nom au projet

Cliquer sur Finish

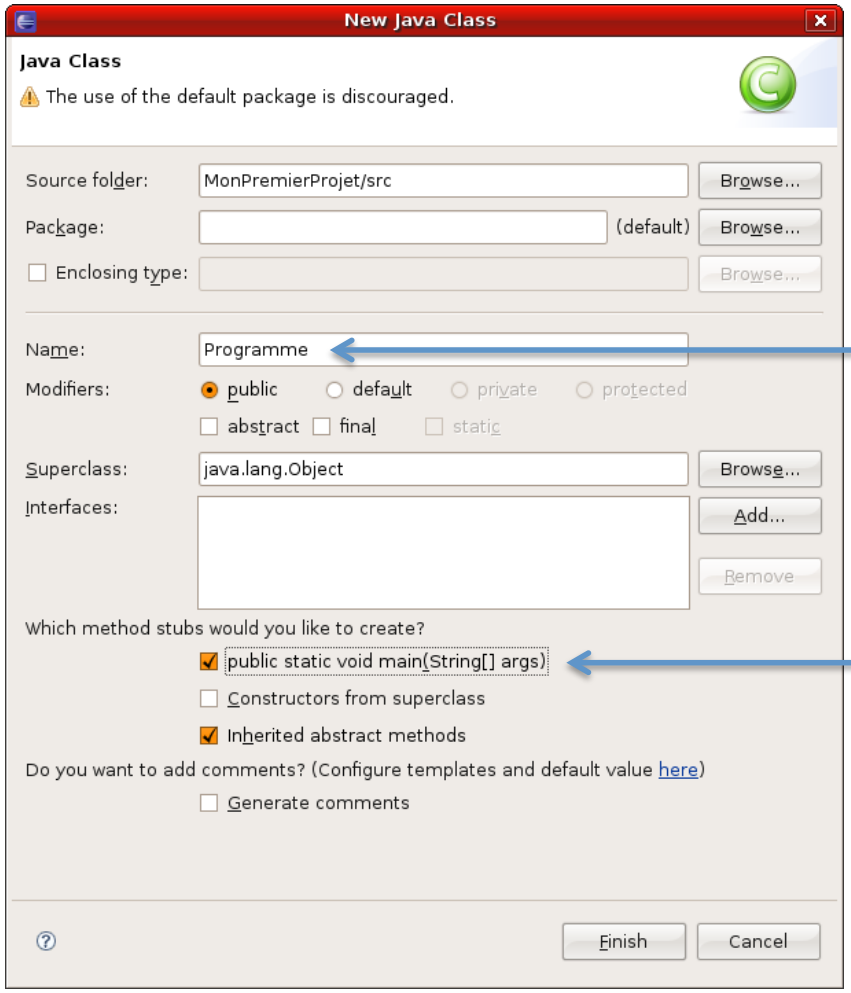
Créer une classe Programme

- Classe principale qui contiendra la méthode main (point d'entrée).

Bouton droit sur src -> New -> Class



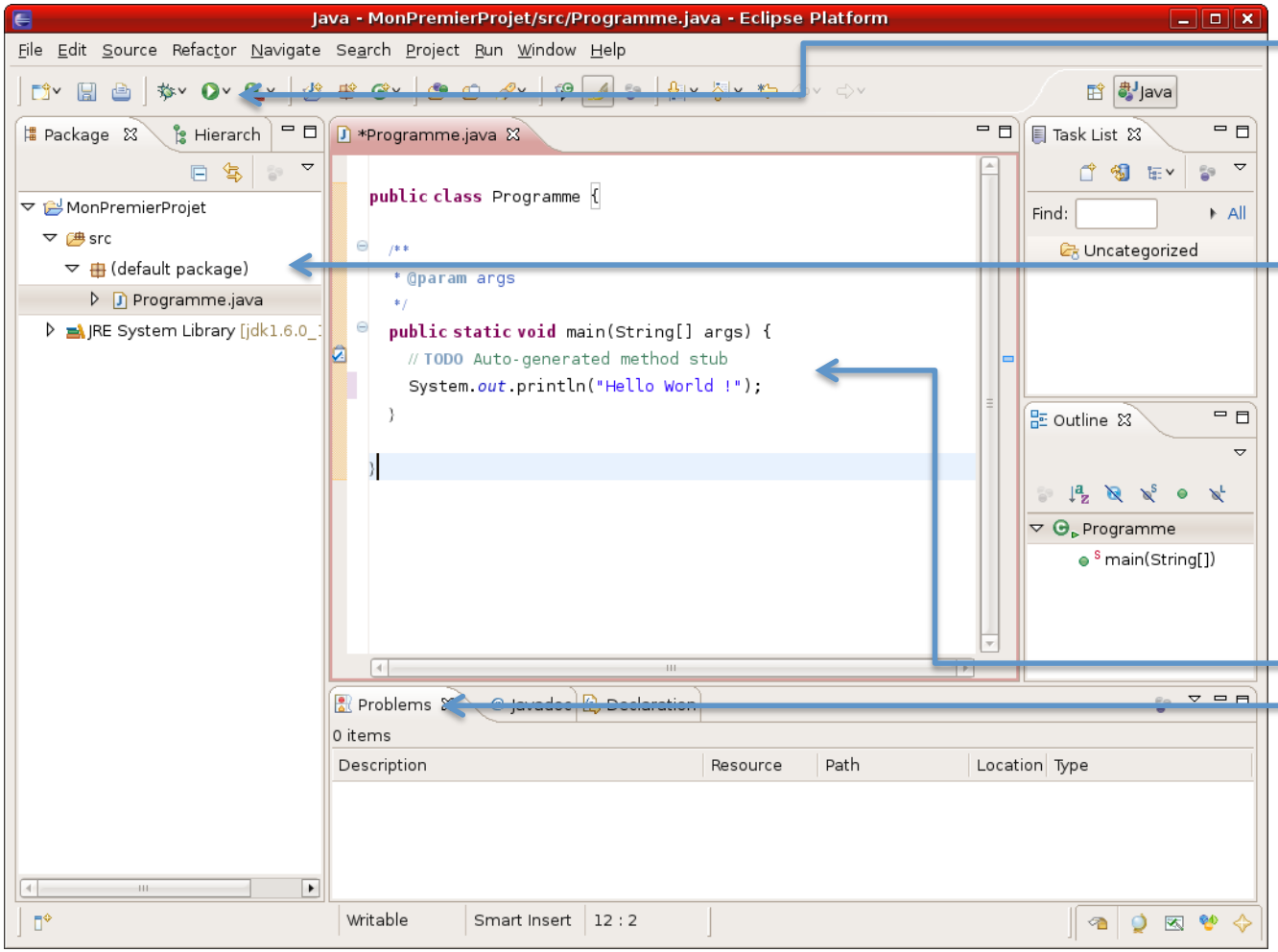
Créer une classe Programme



Nommer la classe

Créer une méthode main

Vue de développement Java



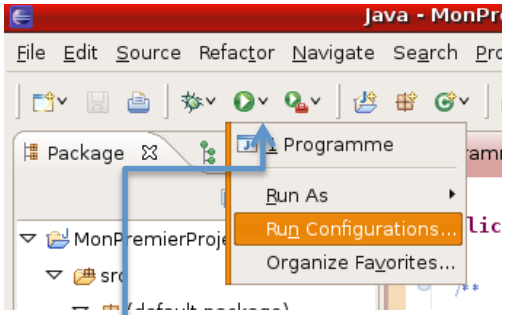
Exécution

Liste des fichiers

Code source

Erreurs et avertissements

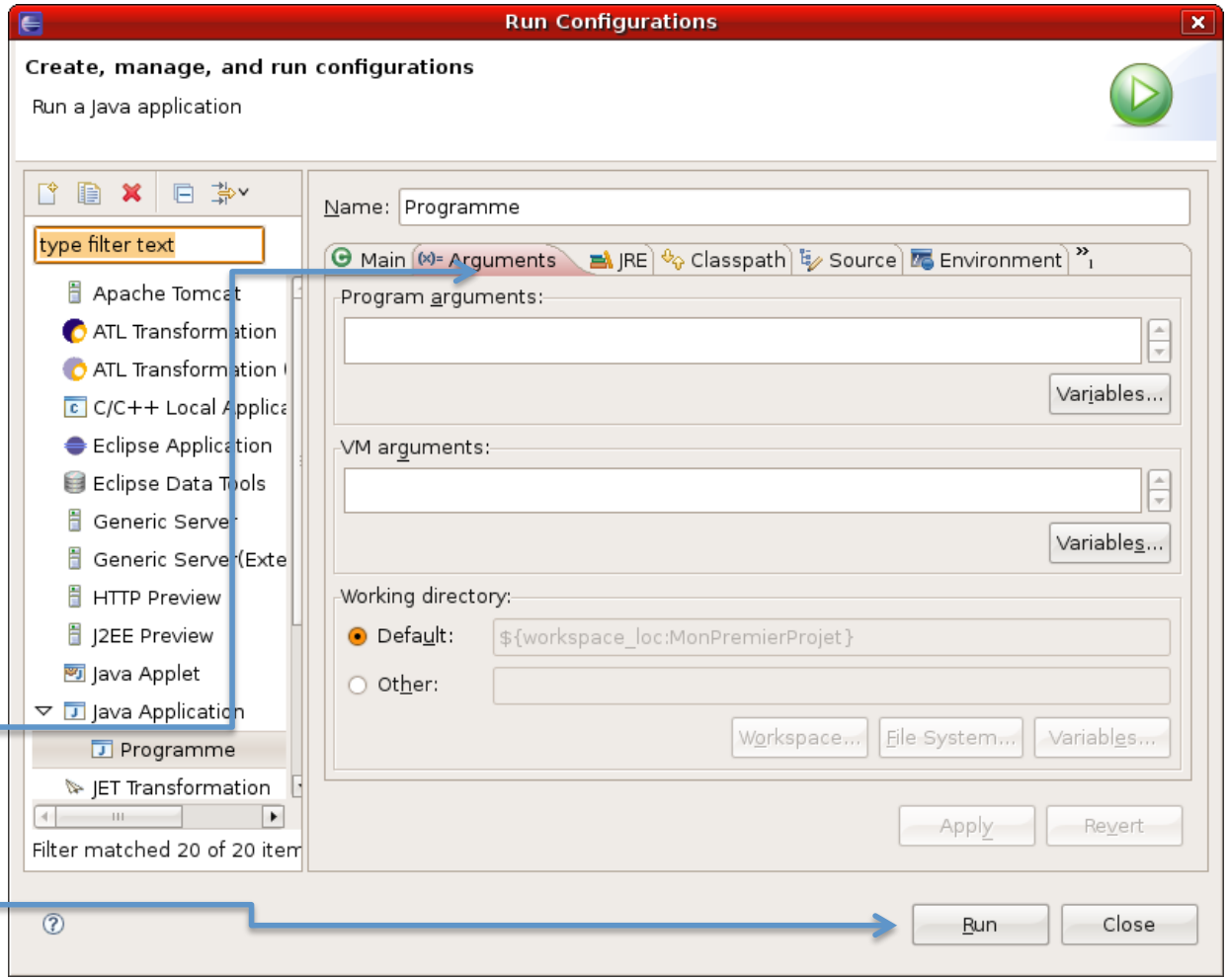
Passer des arguments au programme



Bouton Play,
Run Configurations…

Onglet « Arguments »,
Program arguments

Run



Classes et Objets

- Une **classe** peut être vue comme un nouveau type de donnée, un modèle, qui possède :
  - ses propres **variables** (attributs)
  - son propre **comportement** (méthodes ou fonctions de classe).
- Par exemple : une personne, un vélo, …

- Un **objet** est une **instance** d'une classe qui a une existence propre.
- Par exemple : la personne Jean Dupont, le vélo vert de Frédéric, …

Définition de classes et attributs

- En Java, on va créer un fichier par classe (`MaClasse.java`)
- **Attributs** : variables des instances de la classe
  - variables de type simple, des tableaux ou des objets.
- Les attributs ont un **niveau de protection** :
  - `public` pour les attributs visibles à l'extérieur de la classe.
  - `private` pour les attributs visibles uniquement à l'intérieur.

- **Définition d'une classe avec des attributs :**

```
public class Person{
    public String firstName;
    public String lastName;
    public double salary= 1000; //valeur par défaut
}
```

Définition de méthodes

- Les **méthodes** définissent le **comportement** des objets de la classe.
- Elles peuvent prendre ou nom des paramètres et renvoyer ou non une valeur de retour.
- Elles ont aussi un niveau de protection (public/private).

- Définition d'une classe avec une méthode :
  -
```
public class Person{
    public String firstName;
    public String lastName;

    public String toString(){
      return firstName + " " + lastName;
    }
}
```

Définition de constructeurs

- Un **constructeur** est une méthode spéciale appellée lors de la création d'un objet.
- Cette méthode n'a pas de type et ne renvoie rien.
- Un constructeur doit avoir le **même nom** que la classe.

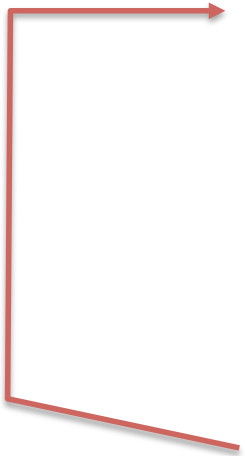
- Définition d'une classe avec un constructeur :

```
public class Person{
  public String firstName;
  public String lastName;

  public Person(String fn, String ln){
    firstName = fn;
    lastName = ln;
  }
}
```

Définition de constructeurs

- Il peut y avoir **plusieurs constructeurs** par classe, avec différents paramètres.
- `this()` permet de réutiliser le code d'un constructeur dans un autre (factorisation).
- Exemple :



```
public class Person{
  //…
  public Person(String ln)
  {
    lastName = ln;
    firstName = "";
  }

  public Person(String ln, String fn)
  {
    this(ln);
    firstName= fn;
  }
}
```

Mot clé `this`

- `this` permet de se référer aux attributs ou méthodes de la classe courante quand il y a **ambiguité**.

- Exemple :

```
public class Person{
    public String lastName;
    public String firstName;

    public Person(String lastName, String firstName){
      this.lastName = lastName;
      this.firstName = firstName;
    }
}
```

Accesseurs

- Un **accesseur** est une méthode pour accéder, en lecture ou en écriture aux attributs d'une classe.
- Un accesseur en lecture renverra l'attribut.
- Un accesseur en écriture est généralement `void`, prend un paramètre et modifie l'attribut, après avoir vérifié que cette modification ne portait pas atteinte à l'intégrité de l'objet.
- Exemple :

```
public class Person {
  private double salary=1000;

  public double getSalary(){
    return salary;
  }

  public void setSalary(double salary){
    if(salary >= 0)
      this.salary = salary;
  }
}
```

Créer et utiliser des objets

- Pour **créer un objet**, il faut déclarer une variable du type de l'objet et instancier l'objet à l'aide du mot clé new et d'un de ses constructeurs.

  - `Person jean = new Person("dupond", "jean");`

- Une fois l'objet créé, on peut **accéder** à ses attributs ou méthodes publiques à l'aide d'un point.

  - `jean.lastName = "durant";`
  - `System.out.println(jean.toString());`

Conventions de nommage Java

- Nommer les attributs, variables, objets et méthodes en *lowerCamelCase*.
- Nommer les classes et les librairies en *UpperCamelCase*.
- Préférer l'anglais et ne pas mélanger les langues.
- Commencer le nom des accesseurs en lecture par `get` et les accesseurs en écriture par `set`.