

# TME 1 : Programmation, compilation et exécution en Java

**Objectifs pédagogiques : savoir organiser une arborescence, savoir compiler et exécuter en ligne de commande, savoir documenter un code, savoir utiliser des classes de l'API.**

## 1.1 Préambule général sur l'ensemble des TME.

Au cours de la série des TME associés à l'UE LI-314, nous allons concevoir et programmer progressivement en Java une application simple mettant en oeuvre un algorithme évolutionniste. Les algorithmes évolutionnistes constituent un modèle informatique très simplifié du processus d'évolution génétique qui opère au sein d'une population d'êtres vivants. Ces algorithmes se dotent d'une population d'individus qui représentent un ensemble de solutions possibles à un problème donné.

Le principe de fonctionnement d'un algorithme génétique est illustré sur la figure 1, tirée du site <http://khayyam.developpez.com/articles/algo/genetic/>

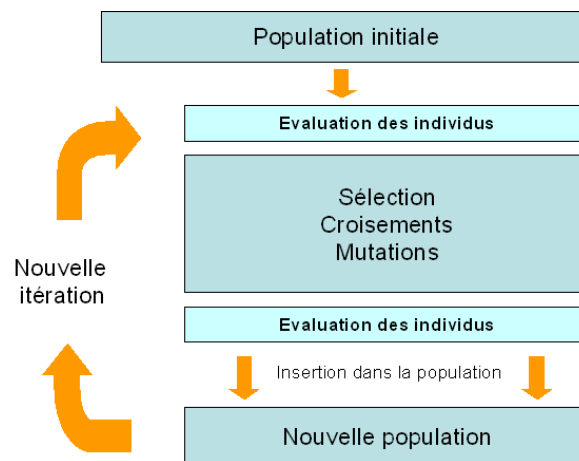


Figure 1: Illustration du principe de fonctionnement d'un algorithme génétique

Le problème est représenté par un environnement. Les individus disposent d'une *fitness* qui mesure l'adéquation de leur comportement au problème qui leur est posé par l'environnement. La population d'individus évolue de génération en génération. Lors du passage d'une génération à la suivante, on élimine un certain nombre d'agents dont la fitness est faible et on les remplace par des nouveaux individus qui sont les descendants d'individus généralement plus performants. Au fil des générations, la fitness moyenne des individus de la population augmente globalement, si bien que les individus résolvent de mieux en mieux le problème. Vous découvrirez tous ces points au fil des TME successifs.

**Le caractère incrémental des TME impose une contrainte : en général, il sera nécessaire que vous ayez fini le TME précédent pour pouvoir attaquer le TME suivant.**

## 1.2 Mise en place du TME 1.

Dans ce premier TME, nous allons nous familiariser avec les outils de base de l'environnement de développement Java JDK. Nous aurons principalement besoin d'un terminal et d'éditeur de texte comme *Emacs* ou *gedit/nedit*. **Dans le cadre de ce TME, il est interdit d'utiliser eclipse.**

Pour la mise en place du TME, créez un répertoire `P0BJ/` (ou `LI314/` ou `Pobj/` ou ce qui vous semble le plus adapté) et un sous-répertoire `tme1/` dans lequel vous vous placerez. Créez ensuite deux sous-répertoire `src/` et `bin/` contenant respectivement les sources et les binaires (code compilé) du TME 2. Vous ajouterez un dernier répertoire `doc/` pour la documentation auto-générée du projet.

Pour être sûr de ne pas entrer en conflit avec d'autres applications du même type, vous utiliserez le nom de paquetage `pobj.algogen` pour votre implémentation. Vous construirez l'application dans le paquetage `pobj.algogen`, les fichiers sources (`.java`) d'un paquetage doivent se trouver dans des répertoires synonymes. En dessous de `src/`, vous créez donc un répertoire `pobj/` et un répertoire `alogogen/` sous `pobj/`.

L'arborescence que vous devez obtenir est la suivante:

```
TME1/
- src/
  - pobj/
    - alogogen/
  - bin/
  - doc/
```

L'objectif du premier TME est de poser quelques classes de base pour faire fonctionner un algorithme génétique. Votre application sera composée d'une classe `Population`, une classe `Individu`, une classe `Environnement` et d'une classe principale de lancement, `PopulationMain`. Dans un premier temps dans ce TME, la classe `Population` sera représentée par une classe `PopulationArray` dont l'implémentation repose sur un tableau.

Vous développerez au cours du TME la méthode `main()` qui doit permettre de tester au fur et à mesure votre développement.

### 1.3 Classes de base

Dans cet exercice, nous souhaitons définir les classes `Individu` et `PopulationArray`, dans le paquetage `pobj.algogen`, représentant une population d'individus, conformément au diagramme de la Figure 2.

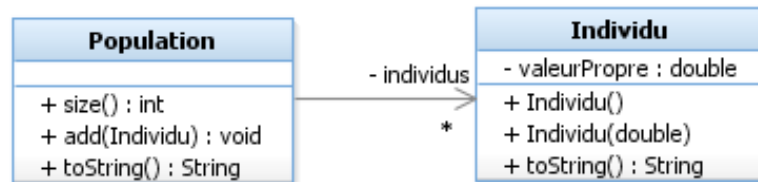


Figure 2: Diagramme UML des classes `Population` et `Individu`.

#### 1.3.1 Classe `Individu`

Un individu possède une valeur propre qui est représentée par un nombre réel (type `double` en Java). La valeur propre de l'individu est la propriété qui le caractérise. Dans ce TME, elle est fixe pour un individu. Le constructeur sans paramètre tire une valeur aléatoire pour cette valeur propre, celui avec paramètre initialise la valeur propre à la valeur passée en paramètre.

⇒ **Donnez l'implémentation de la classe `Individu`** en Java, en prenant soin de fournir toutes les méthodes usuelles.

**NOMBRES ALÉATOIRES EN JAVA.** Deux options sont possibles :

1. `Math.random()` fournit un double entre  $[0, 1[$ .
2. En utilisant plutôt la classe `Random` du package `java.util` on peut aussi faire : `Random r = new Random();` puis invoquer `r.nextDouble()` qui fournit un double entre  $[0, 1[$ , ou `r.nextInt(max)` qui fournit un entier dans  $[0, max[$ .

**Préférez cette seconde solution.**

### 1.3.2 Evaluation d'un individu

A présent, vous allez munir chaque individu d'une mesure de sa qualité: la *fitness*. À l'initialisation, la fitness est fixée à 0. Lors de l'évaluation d'un individu, sa fitness est fixée en invoquant une méthode `setFitness` qui passe la nouvelle fitness en paramètre. Pour afficher un individu, vous afficherez à présent sa valeur propre et la valeur de sa fitness.

⇒ Mettez à jour votre implémentation de la classe `Individu` et le diagramme de classes de votre application.

**COMPILATION EN LIGNE DE COMMANDE.** En Java, chaque classe (publique) doit être définie dans un fichier propre et portant le même nom que la classe, avec l'extension `.java`. Ce fichier doit se trouver dans un répertoire correspondant au paquetage courant. Ainsi, pour une classe `pak1.pak2.pak3.MaClasse`, le fichier source sera `pak1/pak2/pak3/MaClasse.java` à partir de la racine des sources du projet. Pour la classe `pobj.algogen.PopulationArray`, il faudra ainsi créer le fichier `pobj/algogen/PopulationArray.java`.

⇒ L'étape suivante concerne la **compilation du code source**. Pour cela, utilisez la commande :  
`javac «nom du fichier à compiler».java`

Des options souvent utilisées sont les suivantes :

- `-classpath «chemin jar»` : pour indiquer des bibliothèques tierces nécessaires pour compiler,
- `-d «répertoire»` : pour indiquer où placer les fichiers compilés.

Pour compiler votre fichier `Individu.java`, vous utiliserez donc la ligne de commande suivante :

```
javac -d ./bin src/pobj/algogen/Individu.java
```

Avant compilation, le répertoire `bin/` était vide. Jetez maintenant un coup d'œil à la structure du répertoire `bin/` après compilation. Les fichiers compilés portent le suffixe `.class`, ils contiennent du code compilé portable (bytecode).

**Remarque** : si vous n'indiquez pas d'option `-d`, les fichiers compilés seront placés au même endroit que les fichiers sources, **ce qui n'est pas une bonne idée**.

**Remarque 2** : quand on écrit `import pobj.banque.Individu` le compilateur cherche un fichier `pobj/banque/Individu.java` dans le `sourcepath` ou `pobj/banque/Individu.class` dans les chemins du `classpath`.

**Remarque 3** : vous pouvez aussi indiquer plusieurs fichiers simultanément en utilisant des jokers `javac *.java`.

### 1.3.3 Construction d'une population aléatoire

Vous trouverez ci-dessous une implémentation de la classe `PopulationArray` qui utilise un tableau d'individus:

PopulationArray.java

```

package pobj.algogen;
import java.util.Arrays;
public class PopulationArray {
    private static final int POP_SIZE = 20;
    private Individu[] individus;
    private int size = 0;

    public PopulationArray() {
        individus = new Individu [POP_SIZE];
    }

    public int size() {
        return size;
    }

```

```

18 public void add(Individu individu) {
19     if (size < individus.length) {
20         individus[size] = individu;
21         size++;
22     } else {
23         throw new ArrayIndexOutOfBoundsException("Plus de place !");
24     }
25 }
26
27 @Override
28 public String toString() {
29     return Arrays.toString(individus);
30 }
31 }
32

```

⇒ Importez ce code dans un fichier **PopulationArray.java**. Compilez ce fichier.

Que se passe-t-il ? Pour résoudre le problème, il faut soit positionner le `classpath` dans votre environnement, soit utiliser l'option `-classpath` «votre classpath» à la compilation.

Pour compiler votre fichier **PopulationArray.java**, la bonne ligne de commande est donc :

```
javac -d ./bin -classpath ./bin src/pobj/algogen/PopulationArray.java
```

Vous noterez que le chemin indiqué pour ranger les fichiers `.class` désigne la racine de l'application `./bin` et pas les répertoires correspondants aux packages accessibles depuis cette racine.

On souhaite se donner la capacité de construire des populations aléatoires. Pour créer une population aléatoire, vous allez créer une nouvelle classe **PopulationFactory** munie d'opérations permettant de créer et d'initialiser des populations. Déclarez ses opérations en `static`, de façon à pouvoir y accéder sans instancier la classe.

⇒ Donnez une implémentation de cette classe et de sa méthode **createRandomPopulation**, dont la signature est déclarée ci-dessus.

```

1 /**
2  * Opération permettant d'obtenir une nouvelle population générée aléatoirement.
3  * @param size la taille de la population à créer.
4  * @return une population composée de "size" individus générés aléatoirement.
5  */
6 public static PopulationArray createRandomPopulation(int size)

```

## 1.4 Programme principal

Un programme principal Java est décrit dans une classe définissant une *procédure* (ou méthode statique) `main()`.

⇒ Dans une nouvelle classe **pobj.algogen.PopulationMain**, définissez un programme principal **main()** prenant la taille de la population en ligne de commande.

Ce programme construit une population constituée d'individus aléatoires et affiche le contenu de la population.

**PARAMÈTRES D'UN PROGRAMME JAVA.** La méthode `public static void main (String [] args)` sert de point d'entrée au programme. La variable `args` va porter les paramètres passés au programme, sous la forme d'un tableau de `String`. On peut utiliser `args.length` pour contrôler le nombre d'arguments.

La conversion des `String` en entiers `int` se fait à l'aide de la méthode `static int Integer.parseInt(String s)`. Attention cette méthode peut lever une `NumberFormatException` si la `String` argument `s` n'est pas un entier.

⇒ **Compilez votre programme** principal en prenant soin d'utiliser le répertoire de destination `bin/` pour les fichiers compilés.

Pour lancer un programme Java, démarrez une machine virtuelle java et passez en argument le nom de la classe de programme principal à lancer (suivi des arguments du programme principal).

Pour **lancer votre programme** `PopulationMain` (classe `pobj.algogen.PopulationMain`) avec par exemple une taille de 10 individus, tapez donc la commande suivante :

```
java pObj.algogen.PopulationMain 10
```

**Remarque** : cette commande peut être lancée dans le répertoire principal des fichiers compilés (`bin/`). Sinon, précisez le chemin avec l'option `-classpath`.

### 1.4.1 Classe `Population` avec des `ArrayList`

⇒ Renommez la classe `PopulationArray` en `Population` et modifiez son implémentation pour utiliser des tableaux dynamiques (interface `List` et classe `ArrayList` de `java.util`, cf. annexe).

⇒ Quel est l'intérêt d'utiliser une `java.util.List` plutôt que des tableaux ? Y a-t-il des inconvénients ? La vue client de la classe `Population` a-t-elle changé ?

La classe `ArrayList` du package `java.util` permet de manipuler des tableaux de taille dynamique. La fin du TD1 porte un extrait de programme illustrant les principales méthodes disponibles. Consultez la documentation en ligne (documentation de l'API) pour obtenir plus d'informations.

## 1.5 Documentation du code

Dans tout projet de taille conséquente, on n'écrit pas du code pour soi mais pour les autres programmeurs participant au projet, ou pour les futurs utilisateurs de l'application. La documentation du code occupe donc une place fondamentale. L'environnement de développement Java propose l'outil `javadoc` permettant de générer des documentations hypertexte à partir de commentaires spéciaux placés dans le code source des classes.

Les commentaires spéciaux commencent par `/**` et se terminent par `*/`. Ils se placent juste au-dessus de ce que l'on veut documenter, en priorité les classes elles-mêmes et les constructeurs et méthodes publiques (mais une bonne pratique consiste à tout documenter). Des balises spéciales comme `@param` ou `@return` sont également proposées pour standardiser la mise en page. Voici par exemple le code source d'une classe `Point` totalement documentée :

`pobj.axiom5.noyau.Point.java`

<pre> package pObj.axiom5.noyau;  /**  * Classe de représentation de Point dans un repère cartésien  */ public class Point {     /** abscisse du point */     private double x;     /** ordonnée du point */     private double y;      /**      * Construit un point de coordonnées initiales spécifiées      * @param x l'abscisse initiale du point      * @param y l'ordonnée initiale du point      */     public Point(double x, double y) {         this.x = x;         this.y = y;     } </pre>	<pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 </pre>
---	--

```

/**
 * Accède a l'abscisse de ce point
 * @return l'abscisse x du point
 */
public double getX() {
    return x;
}
/**
 * Accède à l'ordonnée de ce point
 * @return l'ordonnée y du point
 */
public double getY() {
    return y;
}
/**
 * Effectue une translation du vecteur de translation spécifié
 * @param dx translation sur l'axe des abscisses
 * @param dy translation sur l'axe des ordonnées
 */
public void translater(int dx, int dy) {
    x+=dx;
    y+=dy;
}
}

```

En suivant ce modèle, documentez vos classes et votre programme principal.

Pour générer la documentation, utilisez la commande `javadoc`:

`javadoc «paquetages ou fichiers sources à documenter»`

Pour générer la documentation privée (point de vue fournisseur), tapez :

`javadoc -private -charset utf-8 -sourcepath src/ -d doc-private/ pobj.algogen`

## 1.6 Création d'une archive et remise du TME (Impératif)

Le TME que vous venez de réaliser va resservir par la suite. Il est donc impératif de le terminer.

Chaque semaine, il est obligatoire de rendre une version par email à votre chargé de TME avant la fin de la séance. Si vous le souhaitez, vous pouvez aussi rendre **une seconde version améliorée avant le début du TME suivant**.

Pour rendre votre TME mais aussi pour le réutiliser dans des TME ultérieurs, il faut créer une archive jar. Pour créer l'archive en ligne de commande, il faut se mettre à la racine de votre code source (le répertoire `src/`) et taper la commande

`jar cvf pobj.algogen-Nom1-Nom2.jar .`

**Remarque** : il suffit de fournir les fichiers sources java, sans les binaires ni le `javadoc` éventuellement créé. Ces derniers éléments peuvent être générés à l'ouverture de votre projet.

Vous enverrez votre archive à votre encadrant de TME à l'adresse email qu'il vous fournira lors du premier TME.

Vous prendrez soin de mettre **[POBJ TME]** dans le sujet de votre mail, ainsi que vos noms. **Mettez vous et votre binôme en copie du mail** de façon à pouvoir vérifier le contenu du mail, et pour servir de sauvegarde entre les séances (et aussi éviter les problèmes si un membre du binôme est absent etc...).

**Outre une archive, votre mail contiendra une trace d'exécution dans le corps du message.**

Par ailleurs, chaque TME comporte une question à laquelle il vous est demandé de répondre par écrit dans le mail que vous envoyez à votre encadrant. Ces consignes sont valables pour tous les TME.

**Question :** quelle est la méthode la plus longue (en nombre de lignes de code) de votre application ? Copiez-collez cette méthode dans le corps du mail que vous enverrez à votre encadrant, commentaires compris. Ajoutez une trace d'exécution.

## 1.7 API de Java

L'ensemble des bibliothèques standard de Java (Java API) sont commentées en **Javadoc**. Vous devez vous familiariser avec l'utilisation de cette documentation. Commencez par regarder les méthodes des classes `java.util.Random` et `java.util.ArrayList`.

**Conseil :** Votre environnement de travail graphique, sous Linux, permet la gestion de bureaux virtuels. Une bonne pratique de programmation est d'utiliser un bureau virtuel séparé avec une fenêtre de navigation Web maximisée et pointant sur la documentation de l'API Java<sup>1</sup>. On peut naviguer entre les bureaux virtuels à la souris (cliquer dans les mini-bureaux dans la barre des tâches) et au clavier (en général `ctrl+alt`+flèches gauche ou droite). Sous eclipse (à partir du TME2), si la configuration est bien positionnée, survolez un nom de classe ou opération pour voir sa documentation, ou ouvrez une fenêtre **javadoc** avec F1.

---

<sup>1</sup>La recherche d'information au sein de la documentation de l'API Java sur le site d'Oracle n'est pas toujours évidente. Un moteur de recherche dédié est disponible à l'adresse <http://javasearch.developpez.com/index.php?ver=3>