Java : Introduction

Mamadou Kaba Traoré
Maître de Conférences en Informatique
Université Blaise Pascal, Clermont-Ferrand 2. LIMOS CNRS UMR 6158
e-mail : traore@isima.fr, cel. 06.20.20.25.35

Plan

1. Structure d'un programme

```
class nom_de_la_classe {
	// Exemple de commentaire (fin à la fin de la ligne)

	/* Autre exemple de commentaire (fin à la rencontre
	du symbole de fin de commentaire */

	// Méthode principale : nécessaire pour toute classe exécutable
	public static void main(String[] arg) {
		/* Ici, la déclaration des données */
		/* Ici, les instructions du programme (déclarations de données possibles ici également) */

		// Attention : majuscules et minuscules sont différenciées par le compilateur

		// Conseils à suivre :
		// - le nom d'une classe commence par une majuscule (ex : Serie, Simulateur, Voiture)
		// - le nom de tout autre entité commence par une minuscule (ex : liste, machine, auto)
		// - si un nom est composé, tous les mots qui suivent le premier commencent
		//   par une majuscule (ex : maSerie, SerieHarmonique, maSerieHarmonique)
	}
}
```

Exemple :

```
class Bienvenue {
        public static void main(String[] param) {
                String message = "Bonjour ! Ce programme a été créé le 27/09/03";
                System.out.println(message);        // Affichage du message textuel
                return;                             // Fin du programme (facultatif)
        }
}
```

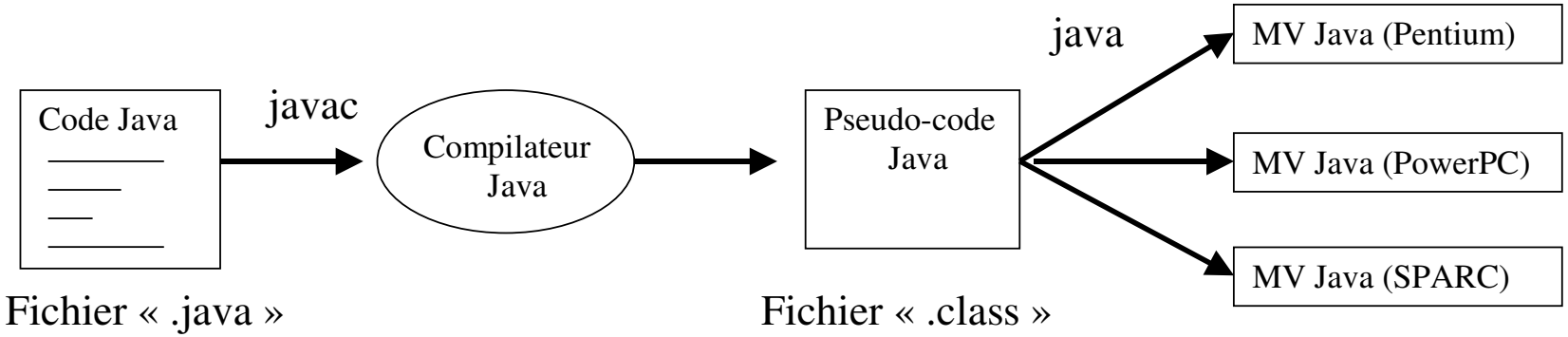
2. Compilation et exécution

1. Saisie du programme dans un éditeur de texte quelconque (Bloc note, Xemacs, Edit, …)
2. Enregistrement du fichier avec l'extension « .java » (ex : bienvenue.java)
3. Compilation avec : javac bienvenue.java ⇒ Il y a création du fichier Bienvenue.class
4. Exécution avec : java Bienvenue
5. Si l'argument param est utilisé dans le code de la méthode main(), une exécution paramétrée est possible avec : java Bienvenue liste_de_mots
   param désigne dans ce cas la liste de mots (ou tableau de mots), et chacun des mots est considéré dans le programme comme représentant respectivement param[0], param[1], ...

3. Environnement de programmation

Il est nécessaire que le JDK (Java Development Kit, gratuit sur le site de Sun : **http://www.sun.com**) soit installé (version minimale : JDK1.2), et les éventuelles configurations effectuées, ou que vous travaillez sous un environnement de développement intégré (Borland Jbuilder, Symantec Visual Café, …).

Un kit de développement Java est constitué d'un compilateur et d'une Machine Virtuelle (ou interpréteur), qui fonctionnent selon le schéma suivant :



Si le fichier « .java » contient plusieurs classes, alors le compilateur créé un fichier « .class » pour chacune de ces classes. Une seule au plus de ces classes peut avoir une méthode main(), et peut donc être exécutable (les classes non exécutables ont vocation à être utilisées par d'autres classes).

4. Structures syntaxiques élémentaires

- Types de base : 6 types numériques, un type booléen, un type alphabétique, et une classe chaîne
  - byte (entier compris entre –128 et 127)
  - short (entier -32768 et 32767)
  - int (entier entre –2.147.483.648 et 2.147.483.647)
  - long (entier entre -9.223.372.036.854.775.808 et 9.223.372.036.854.775.807)
  - float (réel entre $-3{,}4\times10^{-38}$ à $3{,}4\times10^{38}$)
  - double (réel double précision entre $-1{,}7\times10^{-308}$ et $1{,}7\times10^{308}$)
  - boolean (booléen valant true ou false)
  - char (caractère alphabétique dont la valeur est mise entre apostrophes)
  - String (chaîne de caractères, à valeur entre guillemets) seul type non primitif

- Variables :
  - Syntaxe : type_de_donnée nom_de_la variable;
  - Exemples :

```
byte b;
short s; long l; float f;          // Plusieurs variables de types différents
double d, e, f;                    // Plusieurs variables de type identique
int i = 10;double g = 1.2e3;       // Déclarations et initialisations
char c = '\n', d ='\u0082';        // Quelques caractères d'échappement
String titre = "Java";             // Objet chaîne de caractères
```

- Affectation :
  - Syntaxe : nom_de_la_donnée = valeur;
  - Exemples : x = 2 ; a = true ; c = "algorithmique" ;

- Opérateurs élémentaires : arithmétiques, logiques, et sur chaîne
  - Opérateurs arithmétiques : +, -, *, /
  - Opérateurs de comparaison : <, >, ==, !=, >=, <=, && (Et logique), || (Ou logique)
  - Opérateurs sur chaînes : +, .length, [ ]

- Expressions : combinaisons de valeurs, de variables, et d'opérateursQuelques exemples : x + 2, i < j, (x +3) * (y – 5) / (x + y)Règles de priorité standards sur les opérateurs, utilisation conseillée des parenthèses
  - Chaînage possible dans certains cas : x = y = z = 3 ;
  - Expressions abrégées : x +=2 ; // x = x + 2
    i /= (j+3) ; // Attention à l'expression incomplète: i /= j + 3
  - Expressions condensées : // Supposons que x = 2 ;
    x++ ; // x vaut 3
    ++x ; // x vaut 4
    y = x++ ; // y vaut 4, puis x vaut 5
    y = ++x ; // x vaut 6, puis y vaut 6
    // Mêmes constats pour l'opérateur de décrémentation --

- Bloc : définit l'espace de visibilité pour un ensemble de données
  - Syntaxe : { }
  - Imbricable, et utilisable partout où une instruction est utilisable ; les variables déclarées dans un bloc cessent d'exister en dehors du bloc
  - Exemple :

```
int i = 4 ; int j = 2 ;
{
      int j = 3 ;                 // nouvelle variable j
      int y = i + j ;             // y vaut 7 (= 4 + 3)
      {
            int i = 2, y = 0 ;    // Nouvelles variables i et y
            int j = y – i ;       // j vaut –2 (= 0 – 2)
      }
      j = j + i – y ;             // j vaut 0 (= 3 + 4 – 7)
}
int x = i + j ;                   // x vaut 6 (= 4 + 2)
                                  // Ici, y n'existe pas
```

5. Alternatives

- Test unaire
  - Syntaxe: if (condition) instruction ou bloc
  - Exemples:
```
if (y > x) max = y ;
if ((n > 0) && (m >0)) { n *= m ; m-- ;}
```

- Test binaire
  - Syntaxe:
```
if (condition) instruction ou bloc
else instruction ou bloc
```
  - Exemple:
```
if (y > x) max = y;
else max = x ;
```
  - Imbrications :
```
if (x > y) {
        if (x > z) max = x ;
        else max = z;
}
else {
        if (y > z) max = y;  else max = z;
}
```
  - Condensé : (condition) ? valeur_1 : valeur_2 ;
  - Exemples :
```
z = (x > y) ? x :y ;
s = (((u > v) ? u :v) > w) ? ((u > v) ? u :v):w;
```

- Test n-aire
  - Syntaxe:
```
switch (variable_scalaire) {
case valeur_1 : suite d'instructions ou/et de blocs
case valeur_2 : suite d'instructions ou/et de blocs
…
case valeur_n : suite d'instructions ou/et de blocs
default : suite d'instructions ou/et de blocs (cette dernière clause est optionnelle)
}
```
  - Exemple:
```
switch (operation) {
case '+':   za = xa+ya; zb = xb+yb; break ;
case '-':   za = xa-ya; zb = xb-yb; break ;
case '*':   za = (xa*ya)-(xb*yb); zb = (xb*ya)+(xa*yb); break
case '/':   {     double m = ((ya*ya)+(yb*yb)) ;
                  za = ((xa*ya)+(xb*yb))/m; zb = ((xb*ya)+(xa*yb))/m;
            }
            break ;
default :   za =zb = 0 ;
}
```
  - Remarque : sans l'instruction break entre deux « case » consécutifs, l'exécution du premier d'entre eux, se poursuit par l'exécution du second.

6. Répétitions

- Nombre indéterminé d'itérations, avec test initial
  - Syntaxe: while (condition) instruction ou bloc
  - Exemple: fact =1;
    while (n > 0) fact*=n--;

- Variante avec test final
  - Syntaxe: do instruction ou bloc while (condition) ;
  - Exemple: fact =1;
    do fact*=n-- while (n > 0); // si au départ n < 0 ⇒ on a fact = n

- Nombre déterminé d'itérations
  - Syntaxe: for (initialisation ;test_d_arrêt ;incrément) instruction ou bloc
  - Exemple: fact =1;
    for (int i = 1; i < n; i++) fact*=i;
  - Remarque: il est possible d'avoir plusieurs initialisations, et/ou plusieurs incréments
  - Exemple :
```
for (i = 0, j = n ; i < j; i++, j--) {
        /* Des instructions ici */
}
```

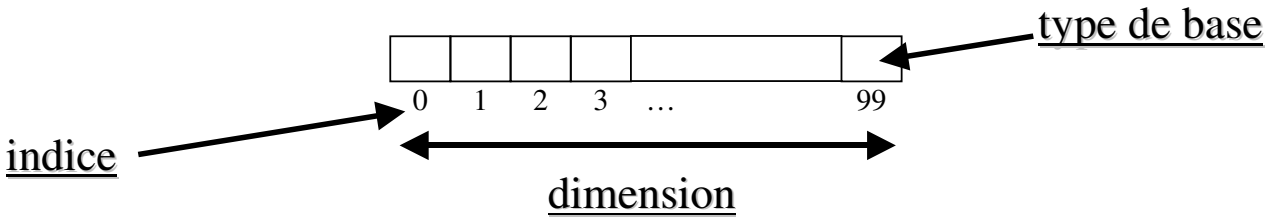
- Boucles infinies:
```
for (i = 0 ;;i++) {
        // Boucle infinie par absence de test (sortie par break)
}
for ( ;;) {
        // Boucle infinie par excellence
}
for (i = 0 ; i < n;) {
        // Boucle infinie par absence d'incrément
}
while (true) {
        // Boucle infinie couramment utilisée
}
```

- Etiquette et rupture :
```
infinie : while (true) {
                do {
                        if (n < 0) break ;              // Sortie du do…while
                        if (m > 0) break infinie;  // Sortie du while
                } while (true)
        }
```

7. Tableaux

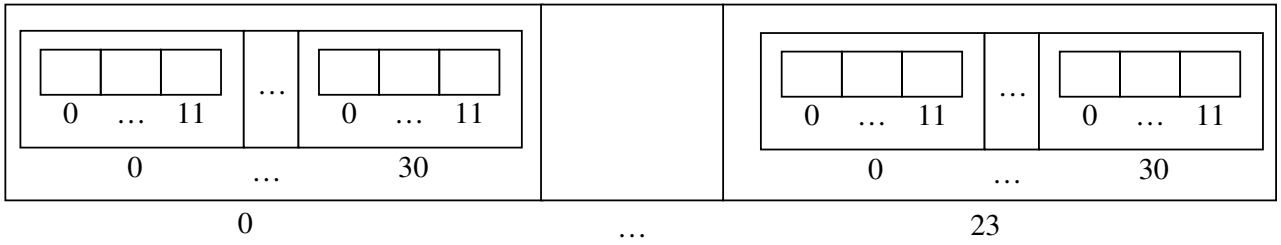
- Tableau unidimensionnel
  - Syntaxe: nom_du_type_de_base[] nom_de_la_variable_tableau ;
  - Exemple: int[] liste;
  - Autre syntaxe : nom_du_type_de_base nom_de_la_variable_tableau[] ;
  - Exemple: int liste[];
  - Création : nom_de_la_variable_tableau = new nom_du_type_de_base[dimension]
  - Exemple: liste = new int[100]; // Les indices commencent toujours à 0
  - Accès : nom_de_la_variable_tableau[indice]
  - Exemple:

```
(for i = 0 ; i < 100 ; i++) {
        s+= liste[i];
        liste[i] = 0;
}
```



- Tableau multidimensionnel
  - Syntaxe: nom_du_type_de_base[][]…[] nom_de_la_variable_tableau ;
  - Autre syntaxe : nom_du_type_de_base nom_de_la_variable_tableau[][]…[] ;
  - Exemple: int[][] matrice = new int[10][10];

```
String emploiDuTemps[][][] = new String[24][31][12];
(for i = 0 ; i < 24 ; i++) {
      (for j = 0 ; j < 31 ; j++) {
            (for k = 0 ; k < 12 ; k++) {
                  emploiDuTemps[i][j][k] = "libre";
            }
      }
}
```



8. Fonctions

- Définition
  - Format:

```
type_du_résultat nom_de_la_fonction(déclaration_d'argument) {
        // Déclaration de variables,
        // et instructions de traitement.
        // Puis renvoi de résultat par : Return résultat;
}
```

  - Type du résultat : type primitif, ou classe, ou void (si pas de retour).
  - Déclaration d'arguments : type_de_l'argument nom_de_l'argument
  - Exemples :

```
int maximum (int x, int y) {                    // Cas classique
        int max = (y > x) ?x : y; return max;
}
int maximum2 (int x, int y) {                   // Plusieurs return
        if (y > x) return y; else return x;
}
double pi() {return 3.14;}                      // Pas d'argument
void saluer () {                                // Pas de retour
        System.out.println("Bonjour !"); return;
}
```

- Appel
  - Fonction main() : ne s'appelle pas de manière explicite ;
    est appelée par la Machine Virtuelle, lors de l'exécution de la classe.
  - Autre fonction : nom_de_la_fonction(liste_des_arguments_effectifs)
  - Exemples : n = maximum(5, 2) ;
    m = maximum2(n, p) ;
    surface = pi()*r*r ;
    saluer() ;

- Passage par valeur, passage par référence
  - Principe : tout argument de type élémentaire est passé par valeur, et tout argument instance de classe est passé par référence (String par exemple)
  - Exemple :

```
void echanger(int x, int y) {          // Echange sans effet sur les arguments
        int temp = x; x = y; y = temp; return ;
}
void echanger(int[] x, int[] y) {      // Echange qui affecte les arguments
        /* les tableaux sont traités par Java comme des classes */
        for (int i = 0 ; i < x.length ; i++) {
                int temp = x[i]; x[i] = y[i;] y[i] = temp;
        }
}
```

- Pile d'exécution : les entités créées lors de l'exécution d'une fonction, sont détruites à la fin
  - Exemple :
```
void initialiser(int[] liste) {    // création et initialisation inutiles
    liste = new int[10] ;
    for (int i = 0 ; i < 10 ; i++) liste[i] = 0 ;
    return ;
}
int[] u ; initialiser(u) ;    // Appel sans intérêt pour l'argument u
/* au retour de l'appel, la place réservée à la liste est perdue */
```
  - Remarque : chaque type primitif possède un homologue défini sous forme de classe (Integer pour int, Float pour float, Double pour double, et Boolean pour boolean).

- Surcharge
  - Principe : désigner par le même nom des fonctions de signatures différentes
  - Exemples :
```
int maximum(int u, int v) {return (u >v) ?u :v}
float maximum(float u, float v) {return (u >v) ?u :v}
int maximum(int u, int v, int w) {return maximum(u, maximum(v, w)) ;}
```
  - Remarque : on ne peut pas surcharger deux fonctions dont les signatures ne diffèrent que par le type de retour.
  - Exemple :
```
int maximum(int u, int v) {return (u >v) ?u :v}
boolean maximum(int u, int v) {return (u >v) ?true :false}
```

9. Exceptions

- Notion
  - Définition : erreur générée par une méthode lors de son exécution (certaines exceptions sont prédéfinies dans Java, d'autres peuvent être définies par l'utilisateur)
  - Exemple : java.io.IOException est une exception d'Entrées-Sorties

- Gestion d'exceptions standards : déclaration de zone critique, et interception en cas d'erreur
  - Déclaration : try { /* Instructions de la zone critique */}
  - Interception : catch() { /* Traitement de l'exception levée */}
  - Exemple :

```
public void saisie(byte[] b) {
        /* Quelques déclarations de variables ici */
        try {                                       // Zone critique
                System.in.read(b);                  // Lecture critique au clavier
                /* Autres traitements */
        } catch(java.io.IOException e) {            // Si une erreur survient
            System.out.println("Erreur en lecture");
        }
}
```

- Délégation : non gestion locale de l'exception,
  et transfert de la responsabilité de cette gestion aux appelants potentiels.
  - Principe : pour ne pas devoir utiliser de bloc try…catch, il faut signaler la possibilité de levée d'exception dans la signature de la fonction contenant une zone sensible.
  - Exemple :
```
public void saisie(byte[] b) throws java.io.IOException {
    /* Quelques déclarations de variables */
    System.in.read(b);          // Plus besoin de try…catch
    /*  Une exception levée par read() sera relayée par saisie(), à l'appelant */
}
```

10. Packages et importation

- Création de package
  - Utilité : si une classe doit être réutilisée par d'autres classes, il est préférable de la stocker dans une librairie (package en Java)
  - Principe : mettre la déclaration de package comme en-tête à la définition de la classe
  - Exemple : package maLibrairie ;
    class MaClasse {/* Définition de la classe */}
  - Inclusions : package maLibrairie.utilitaires ;
    class Calculateur {/* Définition de la classe */}

- Désignation d'une classe, propriété d'un package, par une entité extérieure au package :
  - Syntaxe : nom_du_package.nom_de_la_classe
  - Exemple : java.io.IOException

- Désignation directe de la classe : clause d'importation, à mettre avant
  - Condition : import nom_du_package.nom_de_la_classe
  - Exemples : import java.io.IOException ; // La classe peut être désignée par son seul nom
    import java.io.* ; // Toutes les classes de java.io sont concernées
  - Remarque : en règle générale, une fonction est la propriété d'une classe, et la notation pointée est alors nécessaire pour appeler cette fonction (ex : IOException.writeln() ;)

11. Entrées – Sorties standards

- Affichage à l'écran : System.out.print (println pour créer une nouvelle ligne à la suite)
  - Argument : toute entité chaîne (sinon conversion automatique en chaîne)
  - Exemple :

```
public static void main(String[] arg) {
        int jj = 27, aa = 2003; String mm = "Septembre";
        System.out.println("Bonjour !");
        System.out.print("Ce programme a été créé le " + jj + mm + aa);
}
```

- Saisie au clavier : System.in.read
  - Argument : tableau d'octets
  - Exemple :

```
public static void main(String[] arg) {
        byte b = new byte[5];                          // Tableau à saisir
        try {                                          // Zone critique
                System.in.read(b);                     // Lecture de la donnée
                String s = new String(b,0,5);          // Conversion en chaîne
                System.out.println("\n Lu :" + s);     // Affichage
        } catch(java.io.IOException e) {               // Gestion d'erreur
              System.out.println("Erreur en lecture");
        }
}
```

- Flux de données
  - Principe : gestion des entrées-sorties sous forme de flux (classes prédéfinies)
  - Exemple :

```
public static void main(String[] arg) {
        int n ; float x;                          // Données à lire
        DataInputStream cl = new DataInputStream(System.in);
        // Cette classe gère des flux d'entrée autres que byte
        DataOutputStream e = new DataOutputStream(System.out);
        // Cette classe gère des flux  de sortie autres que String
        try {
            System.out.print("Entier : ");
            n = cl.readInt();                     // Plus besoin de conversion
            System.out.print("Réel : ");
            x = cl.readFloat();
            System.out.print("Valeurs lues :" + '\t');
            e.writeInt(n);
            System.out.print('\t');
            e.writeFloat(x);
        } catch(java.io.IOException e) {
            System.out.println("Erreur en lecture ou en écriture");
        }
}
```

- Entrées-Sorties à tampon
  - Principe : stockage des flux dans un buffer, avant déchargement dans la variable
  - Exemple :

```
public static void main(String[] arg) {
        int n ; float x;                // Données à lire
        InputStreamReader isr = new InputStreamReader(System.in);
        // Lecteur de flux d'entrée
        BufferedReader br = new BufferedReader(isr);
        // Lecteur de flux d'entrée à tampon (buffer associée)
        String s;                       // Variable pour la lecture
        System.out.print("Entier :");
        s = br.readLine();              // Ligne lue, puis buffer vidé
        n = Integer.parseInt(s);        // Conversion en entier
        System.out.print("Réel :");
        y = Integer.parseInt(br.readLine());
        int max = (x > y) ?x :y;
        System.out.print("Maximum : " + '\t' + max);
}
```

12. Travaux pratiques

- Librairie d'Entrées-Sorties

Afin de disposer d'une librairie standard d'Entrées, utilisable pour le reste des travaux pratiques, réaliser, dans un package nommé utilitaires, une classe dont la structure est la suivante :

```
class Entree {          // les fonctions doivent être publiques pour être accessibles
    BufferedReader br ;                     // Déclaration de l'outil de lecture
    public Entree() {                   // Création de l'outil de lecture}
    public int lireEntier() {           // Saisie d'un entier}
    public float lireReel() {           // Saisie d'un réel}
    public double lireDouble(){         // Saisie d'un réel double précision}
    public boolean lireBooleen(){       // Saisie d'une valeur booléenne}
    public String lireChaine(){         // Saisie d'une chaîne de caractères}
    public char lireCaractere(){        // Saisie d'un caractère}
}
```

Tester cette classe avec une classe exécutable, qui doit nécessairement utiliser, les deux clauses suivantes :

```
import utilitaire.Entree ;      // Importation de la classe de gestion des entrées
Entree e = new Entree() ;       // Création d'une entité de saisies
```

- Utilisation de la ligne de commande

Réaliser une classe Optimum capable de déterminer et d'afficher le minimum et le maximum d'une liste d'entiers de taille quelconque. Cette liste est fournie en argument de la ligne de commande.

Exemple d'exécution : java Optimum 12 14 8 24 33 9
Réponse : Le minimum de {12, 14, 8, 24, 33, 9} est 8 et le maximum est 33

Autre exemple d'exécution : java Optimum
Réponse : Utilisation : java Optimum liste-de-valeurs

- Test de comportement des Data Stream

Réaliser une classe Bissextile qui affiche la liste de toutes les années bissextiles comprises entre deux dates lues au clavier. Pour la lecture et l'affichage, utiliser les classes DataInputStream et DataOutputStream.

L'algorithme de détermination du caractère bissextile ou non d'une année est le suivant :

une année est bissextile si son millésime est multiple de 4, sauf les années de début de siècle (donc divisibles par 100) qui ne sont bissextiles que si leur millésime est divisible par 400

- Trieuse

Réaliser une classe qui propose le menu suivant :

1. Saisir une liste d'entiers
2. Quitter

Si l'utilisateur choisit l'option 2, le programme s'arrête. Sinon, le sous menu suivant est proposé :

1. Tri par insertion
2. Tri bulle
3. Quitter

En cas de choix de l'option 3 du sous-menu, le programme revient au menu principal. Sinon, la liste triée est affichée, ainsi que le nombre d'opérations d'affectation effectuées, aussi bien que le nombre d'opérations de test. Le sous-menu est alors proposé de nouveau.

La liste est saisie dans un tableau. Le nombre d'entiers de la liste est variable. Un deuxième tableau doit être utilisé pour garder une copie du tableau, qui servira à restaurer les valeurs initiales, une fois que le tableau aura été trié.

Enrichir la collection de méthodes de tri, avec d'autres méthodes. On s'intéressera à comparer les performances des méthodes de tri, sur plusieurs jeux de données (qui pourront être générés aléatoirement).