EF-1
Algorithmique - Programmation

Séance 1 (introduction : Java & Raisonnement)

Hervé Blanchon & Francis Brunet-Manquat
Université Pierre Mendès France
IUT 2 – Département Informatique

Objectif du module

Renforcer vos connaissances, compétences en algorithmique et en programmation OO

- ✓ Algorithmique (pour réfléchir à une implantation)
  - sur les vecteurs,
  - les listes chainées,
  - les arbres
- ✓ Programmation Orientée Objet (pour modéliser les données)
  - classe (attribut, méthode), objet
  - composition
  - héritage
  - classe abstraite/interface
- ✓ Java (pour l'implantation effective)
  - syntaxe du langage
  - compilation/exécution
  - IDE (Integrated Development Environment) : NetBeans
  - collections
  - exceptions

Contenu de la séance 1

- ✓ Introduction à Java // NetBeans
- ✓ Notion de vecteur ou de tableau
- ✓ Algorithmique sur les vecteurs

INTRODUCTION À JAVA

Premier programme

✓ Fichier **PremierProgramme.java**

```
package IntroJava;
public class PremierProgramme {
   public static void main(String[] args) {
      System.out.println("Bonjour !");
   }
}
```

- Compilation
  - **javac PremierProgramme.java**
- Exécution
  - **java PremierProgramme**

Déclaration d'une variable en Java

Syntaxe Java

✓ nomType nomVariable;

✓ nomType nomVariable = valeur;

Exemples

✓ int x;

✓ int x = 0;

Type primitif	Signification	Place occupée en mémoire
byte	Entier très court [-128 à +127]	1 octet / 8 bits
short	Entier court [-32768 à +32767]	2 octets / 16 bits
int	Entier [-2 147 483 648 (-2^{31}) à +2 147 483 647 (2^{31-1})]	4 octets / 32 bits
long	Entier long [-2^{63} à $+2^{63}$ -1]	8 octets /64 bits
float	Nombre réel [$-1.4 * 10^{-45}$ à $+3.4 * 10^{38}$]	4 octets / 32 bits
double	Nombre réel double précision [$4.9 * 10^{-324}$ à $+1.7 * 10^{308}$]	8 octets / 64 bits
char	Caractère unicode [65536 caractères possibles]	2 octets / 16 bits
boolean	variable booléenne [valeurs : true ou false]	1 octets / 8 bits

Opérateurs arithmétiques en Java

Opérateur	Dénomination	Effet	Exemple	Résultat (x vaut 7)
+	addition	Ajoute deux valeurs	x + 3	10
-	soustraction	Soustrait deux valeurs	x - 3	4
*	multiplication	Multiplie deux valeurs	x * 3	21
/	division	Divise deux valeurs	x / 3	2.3333333
%	modulo	Résultat de la division entière	x % 2	1 ((3 * 2) + **1** vaut 7)

Opérateurs d'affectation en Java

✓ Affecte, assigne, une valeur à une variable

- ici x

Opérateur	Effet *x prend pour valeur …*	Exemple	Résultat
x = y	*y*	x = 3	x prend pour valeur 3
x += y	*x + y*	x += 3	x prend pour valeur 6 (3+3)
x -= y	*x - y*	x -= 2	x prend pour valeur 4 (6-2)
x *= y	*x * y*	x *= 10	x prend pour valeur 40 (4*10)
x /= y	*x / y*	x /= 2	x prend pour valeur 20 (40/2)
x %= y	*x % y*	x %= 7	6 ((2 * 7) + **6** vaut 20)

Opérateurs de comparaison en Java

✓ Comparer deux valeurs comparables

Opérateur	Dénomination	Exemple	Résultat
==	égalité	x == 3	***true*** si x est égal à 3 ***false*** sinon
<	infériorité stricte	x < 3	***true*** si x est strictement inférieur à 3 ***false*** sinon
<=	infériorité large	x <= 3	***true*** si x est inférieur ou égal à 3 ***false*** sinon
>	supériorité stricte	x > 3	***true*** si x est strictement supérieur à 3 ***false*** sinon
>=	supériorité large	x >= 3	***true*** si x est supérieur ou égal à 3 ***false*** sinon
!=	inégalité	x != 2	***true*** si x est différent de 3 ***false*** sinon

✓ **Attention** :
- ne pas confonde == (égalité) avec = (affectation)

Entrées/sorties

✓ Lire et afficher un entier

```
package IntroJava;

import java.util.Scanner;  // bibliothèque définissant un Scanner

public class entreesortie {
    public static void main(String[] args) {
    // Creation d'un lecteur d'entrées à la console
    Scanner entree = new Scanner(System.in);
    // Demander à l'utilisateur de saisir un entier
    System.out.print("Saisir un entier : ");
    // Déclaration d'une variable de type entier
    int monInt;
    // Initialisation du contenu de la variable monInt
    monInt = entree.nextInt();
    // Affichage de l'entier saisi
    System.out.println("L'entier saisi est : " + monInt);
    }
}
```


Instruction conditionnelle

✓ Résumé

en Java

```
if (condition1) {
   seq_instructions1;
{} else if (condition2..n) {
   seq_instructions2..n;}
[] else {
   seq_instructionsn+1;]
}
```

✓ Notes
- {} indique que la section **else if** peut se répéter 0, 1 ou plusieurs fois
- [] indique de la partie **else** est optionnelle
- *condition* est une expression booléenne

Instruction conditionnelle

✓ Formes simples

en Java

```
if (condition1) {
   seq_instructions1;
} else {
   seq_instructions2;
}
```

```
if (condition1) {
   seq_instructions1;
}
```

⚠ si la condition est fausse, on ne fait rien !

Instruction d'itération

✓ Procédure avec itération **`while`** :

```
private static void affichetablede7vWhile() {
    // déclaration d'une variable d'indice i
    int i;
    // initialisation de i
    i = 1;
    // tant que i <=10 faire
    while (i <= 10) {
        // afficher la ligne correspondant à i
        System.out.println(i + " x 7 = " + i * 7);
        // incrémenter i en vu de l'affichage de la ligne
        // suivante si nécessaire
        i = i + 1;
    }
}
```

✓ **`while (condition) {séquence d'instructions}`**

- tant que condition est vraie exécuter séq. inst.

Instruction d'itération

✓ Procédure avec itération **`do while`** :

```
private static void affichetablede7vDoWhile() {
    // déclaration d'une variable d'indice i
    int i;
    // initialisation de i
    i = 1;
    // faire ... tant que i <=10 faire
    do {
        // afficher la ligne correspondant à i
        System.out.println(i + " x 7 = " + i * 7);
        // incrémenter i en vu de l'affichage de la ligne
        // suivante si nécessaire
        i = i + 1;
    } while (i < 10); // attention nouvelle condition
}
```

✓ **`do {séquence d'instructions} while (condition)`**

- exécuter séq. inst. tant que condition est vraie (**au moins une fois**)

Instruction d'itération

✓ Procédure avec itération **`for`** :

```
private static void affichetablede7vFor() {
    for(int i = 1; i <= 10; i++) {
        // afficher la ligne correspondant à i
        System.out.println(i + " x 7 = " + i * 7);
    }
}
```

✓ **`for( décl. & init. variable itération;`**
`condition;`
`modification variable itération)`
`{séquence d'instructions}`

1. initialiser variable itération
2. tant que condition est vrai
   - exécuter séquence d'instructions
   - exécuter modification variable itération

Exercices

✓ Triangle isocèle de base et hauteur **`n`**

✓ Triangle de hauteur **`n`** et base **`2n-1`**

- Implanter deux procédures qui affichent les triangles avec **`n`** en argument

`afficheTriangle1(3)`

produit

```
*
**
***
```

`afficheTriangle2(3)`

produit

```
*
**
***
**
*
```

Penser à utiliser le **débogueur** si nécessaire !

Expressions booléennes

- ✓ Les expressions booléennes ne font pas seulement intervenir des opérateurs de comparaison …
- ✓ … mais aussi des opérateurs logiques
  - cond1 || cond2 ; cond1 && cond2 ; !condition

Opérateur	Dénomination	Effet	Exemple	Résultat (x vaut 7, y vaut 12)
||	OU logique	Vérifie qu'au moins une des conditions (cond1, cond2) est vraie	(x == 7) || (y == 15) (x == 12) || (y == 7)	true false
&&	ET logique	Vérifie que les deux conditions (cond1, cond2) sont vraies	(x == 7) || (y == 12) (x == 7) || (y == 15)	true false
!	NÉGATION logique	Inverse la valeur de la condition	(y == 12) (y == 15)	true false

Négation (notée ! en Java)

A	!A
vrai	faux
faux	vrai

- ✓ (A == faux) *équivalent à* (non A)
- ✓ (A == vrai) *équivalent à* A
- ✓ (!(!A)) = (!!A) *équivalent à* A

ET (&&) & OU (||)

- ✓ Comme la négation les opérateurs **ET** & **OU** sont des opérateurs booléens

A	B	A ET B
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

A	B	A OU B
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

⚠ Attention

- le **ou logique** est vrai si au moins un parmi A et B est vrai
- ce n'est pas le « fromage **ou** dessert » de la carte du restaurant qui est un **ou exclusif**
  - i.e. soit fromage, soit dessert, mais pas les deux !

ET (&&) & OU (||)

- ✓ Observation sur l'évaluation du **et** & du **ou**

A	B	A && B
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

- ✓ et
  - si A est vrai alors
    - A et B = B
  - si A est faux alors
    - A et B = A
    - **pas besoin d'examiner B**

A	B	A || B
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

- ✓ ou
  - si A est vrai alors
    - A ou B = A
    - **pas besoin d'examiner B**
  - si A est faux alors
    - A ou B = B

Éviter les évaluations inutiles

- ✓ En Java, les opérateurs booléens …
  - **&&**
  - **||**
- ✓ … implantent une évaluation « court-circuit »
  (“short-circuit evaluation” en anglais)
  - qui vise à en faire le moins possible !

Évaluation court-circuit de `A && B`

- ✓ Vue sous forme de table

A	B	A && B
vrai	vrai	vrai
vrai	faux	faux
faux	non examinée	faux

- ✓ Note
  - on n’a que 3 situations car lorsque **A** est fausse, **B** n’est pas examinée

Évaluation court-circuit de `A || B`

- ✓ Vue sous forme de table

A	B	A || B
vrai	vrai	vrai
vrai	faux	faux
faux	non examinée	faux

- ✓ Note
  - on n’a que 3 situations car lorsque **A** est fausse, **B** n’est pas examinée

Négation d’expression booléenne

- ✓ Loi de Morgan
  - NON(A ET B) = NON(A) OU NON(B)
  - NON(A OU B) = NON(A) ET NON(B)
- ✓ … en Java
  - `!(A && B) = !A || !B`
  - `!(A || B) = !A && !B`

NOTION DE VECTEUR

Notion de vecteur

- ✓ Une collection de données (éléments)
  - … de même type
    - simple *(ex. un vecteur d'entiers)*
    - objet *(ex. un vecteurs d'étudiants)*
    - vecteur *(ex. un vecteur de vecteurs d'entiers)*
  - … indicées (numérotées)
- ✓ Exemple
  - Vecteur **v** d'**entiers** *indicé* de *0* à *5*

Notion de vecteur

- ✓ Vecteur **v** d'**entiers** *indicé* de *0* à *5*

	0	*1*	*2*	*3*	*4*	*5*
v	**5**	**12**	**7**	**10**	**6**	**8**

- ✓ Notations
  - **v**[*0..5*] = [**5**, **12**, **7**, **10**, **6**, **8**]
  - [*0..5*] est l'intervalle des indices de **v**
    - *0* est la borne inférieure *(plus petit des indices)*
    - *5* est la borne supérieure *(plus grand des indices)*
  - **v**[*i*] est l'élément d'indice *i* du vecteur **v**
    - **v**[*0*] = **5** ; **v**[*3*] = **10** ; **v**[*5*] = **8**
    - **v**[*i*] n'est défini que si *i* est dans l'intervalle des indices
    - **v**[*-1*] et **v**[*6*] **ne sont pas définis** !!

Variable de type vecteur en Java

- ✓ Déclaration d'un vecteur d'entier

```
int[] vectEnt;
// vecEnt est un vecteur d'entier
// pour l'instant il est vide
```

- ✓ Création du vecteur vectEnt de taille 12

```
vectEnt = new int[12];
// réserver pour vectEnt une zone contigüe
// en mémoire pour stocker 12 entier
```

- ✓ Déclaration et création de vectEnt de 12 entiers

```
int[] vectEnt = new int[12];
```

Variable de type vecteur en Java

- ✓ Soit **v** un vecteur déclaré comme variable & créé
  - **v.length** est la taille, la longueur, de **v**
    - exemple : **vectEnt.length == 6**
  - le premier élément de **v** est à l'indice **0**
  - le dernier élément de **v** est à l'indice **v.length – 1**
    - exemple : indice du dernier élément de **vectEnt == 5**

Initialisation

- ✓ Soit **v** un vecteur déclaré comme variable & créé
  - on connaît les indices des première et dernière valeurs de **v**(**[0 .. v.length-1]**)
- ✓ Initialiser le vecteur **v** ⟺ donner une valeur à tous les **v[i]** pour **i ∈[0..v.length-1]**

Initialisation (v1 – **main()**)

- ✓ Donner une valeur à tous les **v[i]** pour **i ∈[0..v.length-1]**
  - une instruction pour chaque i

```
public static void main(String[] args) {
   // déclaration d'une variable de type vecteur d'entier
   int[] vectEnt;
   // création d'un vecteur de 6 entiers de nom vectEnt
   vectEnt = new int[6];
   // initialisation "manuelle" de chacun des entiers vectEnt
   vectEnt[0] = 5;
   vectEnt[1] = 12;
   vectEnt[2] = 7;
   vectEnt[3] = 10;
   vectEnt[4] = 6;
   vectEnt[5] = 8;
}
```


Initialisation (v2 – **main()**)

- ✓ Donner une valeur à tous les **v[i]** pour **i ∈[0..v.length-1]**
  - une itération qui parcours chaque i (+ hasard)

```
public static void main(String[] args) {
   // déclaration d'une variable de type vecteur d'entiers
   int[] vectEnt;
   // création d'un vecteur de 6 entiers de nom vectEnt
   vectEnt = new int[6];
   // initialisation avec des entiers [0..100]
   // tirés selon une distribution uniforme (hasard)
   for (int i = 0; i < vectEnt.length; i++) {
      vectEnt[i] = (int) (Math.random() * 100);
   }
}
```

Initialisation (v3 – `main()`)

- ✓ Donner une valeur à tous les `v[i]` pour `i` ∈ `[0..v.length-1]`
  - une itération qui parcours chaque i (+ lecture)

```
public static void main(String[] args) {
   // déclaration d'une variable de type vecteur d'entiers
   int[] vectEnt;
   // création d'un vecteur de 6 entiers de nom vectEnt
   vectEnt = new int[6];
   // création d'un scanner pour la lecture
   Scanner lecteur = new Scanner(System.in);
   System.out.print("Saisir " + vectEnt.length + " valeurs : ");
   // lire vectEnt.length valeurs
   for (int i = 0; i < vectEnt.length; i++) {
      vectEnt[i] = lecteur.nextInt();
   }
}
```


Déclaration & initialisation combinées

- ✓ On peut combiner la déclaration & l'initialisation d'un vecteur au moyen d'un agrégat
  - `typeElement[] vectElement = {valeur`$_0$`, valeur`$_1$`, … , valeur`$_{vectElement.length-1}$`}`

- ✓ Exemple :

```
public static void main(String[] args) {
    // déclaration d'une variable de type vecteur d'entiers
    // et initialisation au moyen d'un agrégat
    int[] vectEnt = {5, 12, 7, 10, 6, 8};
}
```


Affichage avec une procédure

- ✓ On veut écrire une procédure de prototype :

```
private static void afficheVect(int[] unVectEnt)
```

qui affiche les valeurs contenues dans un vecteur d'entiers

- ✓ Il faut parcourir complètement le vecteur jusqu'au bout
  - avec un `for`

```
for (int i = 0; i < unVectEnt.length; i++) {
   System.out.print(unVectEnt[i] + " ");
}
```

  - avec un `while`

```
int i = 0; // initialisation indice consulté
while (i < unVectEnt.length) {
   System.out.print(unVectEnt[i] + " ");
   i = i + 1; // passer au suivant
} // {i == unVectEnt.length}
```


Initialisation (v4 – `initRandomVectEnt()`)

- ✓ Donner une valeur à tous les `v[i]` pour `i` ∈ `[0..v.length-1]`
  - une itération qui parcours chaque i (+ hasard)

- ✓ On veut écrire une procédure de prototype :

```
private static void initRandomVectEnt(int[] leVecteur)
```

qui range des valeurs tirées au hasard dans chacun des éléments du vecteur leVecteur

Initialisation (v4 – initRandomVectEnt())

```
public class VecteurInitAvecProc {
  public static void main(String[] args) {
    // déclaration d'une variable de type vecteur d'entiers
    int[] vectEnt;
    // création d'un vecteur de 6 entiers de nom vectEnt
    vectEnt = new int[6];
    initRandomVectEnt(vectEnt);
    afficheVect(vectEnt);
  }
  private static void initRandomVectEnt(int[] leVecteur) {
    for (int i = 0; i < leVecteur.length; i++) {
      leVecteur[i] = (int) (Math.random() * 100);
    }
  }
  private static void afficheVect(int[] unVectEnt) {
    for (int i = 0; i < unVectEnt.length; i++) {
      System.out.print(unVectEnt[i] + " ");
    }
  }
}
```


ALGORITHMIQUE SUR LES VECTEURS

Somme des éléments d'un vecteur

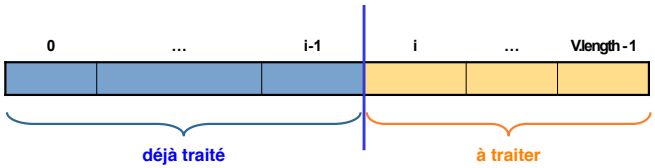
- ✓ On veut écrire une fonction qui retourne la somme des éléments d'un vecteur d'entiers contenant des entiers [0..100]

```
private static int sommeVectEnt(int[] vectEnt)
```

- ✓ Cette fonction retourne un entier

Somme des éléments d'un vecteur

- ✓ Dessin (ébauche)



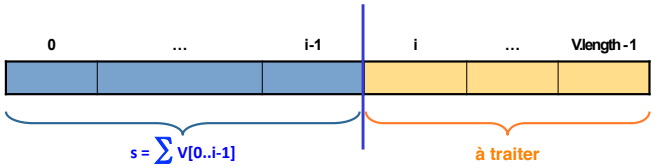
- Qu'a-t-on fait sur la zone traitée ?
  - rappel : on doit faire une somme

Somme des éléments d'un vecteur

- ✓ Pour aller vers la fonction :
  - formaliser l'algorithme (faire un raisonnement par récurrence)
  - commencer par produire une hypothèse !
    - Faire un dessin
  - en français
    - À un instant donné, l'algorithme qui fait la somme des éléments du vecteur à fait la somme des éléments d'un sous-vecteur V[0..i-1]
  - formalisation
    - s = ∑ V[0..i-1]

Somme des éléments d'un vecteur

- ✓ Dessin complet



Somme des éléments d'un vecteur

- ✓ *raisonnement par récurrence*

Hypothèse s = ∑ V[0..i-1]

Q : Quelles sont les situations intéressantes ?
R : Il y en a deux !

- ➤ i = V.length ⇨ ✳ `{return s;}` *{s = ∑ V[0..V.length-1]}*
- ➤ i < V.length ⇨ `s = s + V[i]; i = i + 1;` ➡ ***H***

Q : Dans quelle situation retourne-t-on à l'hypothèse ?
R : Lorsque i ≤ V.length-1 !

Itération `while (i < V.length) {…}`

Q : Connaît-on un vecteur pour lequel on peut rendre le résultat sans calcul ?
R : Oui, le vecteur vide !

Initialisation `i = 0; s = 0;` ➡ ***H***

Somme des éléments d'un vecteur

- ✓ *raisonnement par récurrence*

Hypothèse s = ∑ V[0..i-1]

- ➤ i = V.length ⇨ ✳ `{return s;}` *{s = ∑ V[0..V.length-1]}*
- ➤ i < V.length ⇨ `s = s + V[i]; i = i + 1;` ➡ ***H***

Itération `while (i <= V.length) {…}`

Initialisation `i = 0; s = 0;` ➡ ***H***

Somme des éléments d'un vecteur

```
private static int sommeVectEnt(int[] vectEnt) {
   // déclaration des 2 variables entières utiles
   int s, i;

   s = 0;      // initialisation de la somme
   i = 0;     // initialisation de l'indice parcours

   while (i < vectEnt.length) {    // itération
      s = s + vectEnt[i];    // accumuler
      i = i + 1;             // avancer
   }

   // {i == vectEnt.length, s == Σ vectEnt[0..vectEnt.length-1]}
   return s; // retourner le résultat
}
```


Exercice : nombre d'entiers < 50

- ✓ On veut écrire une fonction qui retourne le nombre des éléments strictement inférieurs à 50 dans un vecteur d'entiers contenant des entiers [0..100]

```
private static int nbInf50VectEnt(int[] vectEnt)
```

- ✓ Cette fonction retourne un entier

Exercice : Recherche entier présent ?

- ✓ On veut écrire une fonction booléenne (prédicat) qui retourne vrai si une valeur **`val`** est présente dans un d'un vecteur d'entiers contenant des entiers [0..100]

```
private static boolean estPresenteVectVal(
                              int[] vectEnt,
                              int val)
```

- ✓ Cette fonction retourne un booléen

Exercices

- ✓ Donner l'algorithme et l'implantation en Java des fonctions suivantes
  - **`private static int valMinVectEnt(int[] vectEnt)`**
    - qui retourne la plus petite valeur de vectEnt
  - **`private static float moyenneVectEnt(int[] vectEnt)`**
    - qui retourne la moyenne des éléments de vectEnt
  - **`private static bool estTrieVectEnt(int[] vectEnt)`**
    - qui retourne vrai si vectEnt est trié dans l'ordre croissant ; faux sinon
  - **`private static bool estPalindrome(char[] vectCar)`**
    - qui retourne vrai si vectCar est un palindrome ; faux sinon
      - *« désigne un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche »* source wikipédia