Introduction à la programmation en Java

UFR Sciences de Nice

Licence Math-Info 2006-2007

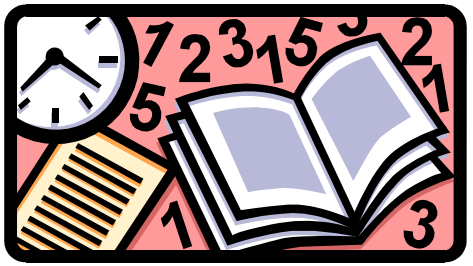
Module L1I1

Frédéric MALLET

Jean-Paul ROY

COURS 7

Rappels



Où en sommes-nous ?

- Manipuler des variables
  - Déclaration & typage
  - Initialisation & affectation
- Manipuler des références
  - Déclaration de classes, champs, constructeurs et méthodes
  - Instanciation d'objets
  - Notation pointée
- Portée des variables
  - Champs, variable locale, paramètre
- Méthodes d'instances ou méthodes statiques
- Manipuler des tableaux
  - Tableaux de valeurs ou de références
- Manipuler les structures de contrôle
  - Alternative et répétition

Type primitif, variable et affectation

- 8 types primitifs en Java
  - **Codage** (nombre de bits), **domaine**, **opérations**
  - **Entiers** : approximation des entiers relatifs Z
    - `int` (32 bits), `long` (64 bits), `short` (16 bits), `byte` (8 bits)
  - **Nombres approchés** : approximation des nombres réels R
    - `float` (32 bits), `double` (64 bits)
  - **Caractères** : lettres, ponctuation, chiffres, …
    - `char` (16 bits) = code unicode UTF-16
  - **Booléen** : vrai ou faux
    - `boolean`

Type primitif, variable et affectation

- Variable= case mémoire, **type et valeur** « variable »
- Représente une valeur inconnue à la compilation
  - Valeur donnée par l'utilisateur ou un autre programme ;
  - Résultat d'un calcul intermédiaire.
- **Doit être déclarée**, i.e. associée à un type
  - `int a;`
  - `boolean b1, b2 ;`
- **Doit avoir une valeur** avant d'être lue
  - `char cc = 'a';`

Type primitif, variable et affectation

- **L'opérateur d'affectation**
  - Permet de donner une valeur à une variable déclarée
  - SYNOPSIS : `<variable> = <expression> ;`
  - Exemples :
    - `int a, b = 5;`
    - `a = b*10 + 5 ;`
    - `b = b*10 + a ;`
- Des **opérateurs d'affectation combinés** existent
  - `+=`, `-=`, `*=`, `/=`, `%=`, …
  - SYNOPSIS : `<variable> <operateur> <expression> ;`
  - Exemple : `a += b * 10;`

Classe, référence et objet

- Les programmes manipulent des données variées
  - **I**mage, **T**exte, **T**éléphone, **F**iguresGéométriques, …
- Une **classe** représente un ensemble de données
  - Domaine de valeurs, opérations standards
  - Ce sont également des types => codage
- Composition de types primitifs
  - **C**ouleur = (RVB) **r**ouge (int), **v**ert (int), **b**leu (int)
  - ou = (CMJN) **c**yan, **m**agenta, **j**aune, **n**oir
  - **P**ersonne = nom (?), prénom (?), âge (?)

Classe, référence et objet

```
class CouleurRVB {
  int rouge;
  int vert;
  int bleu;
}
```

```
class CouleurCMJN {
  int cyan, magenta;
  int jaune, noir;
}
```

```
class Téléphone {
  int indicatif;
  byte v1, v2, v3, v4;
}
```

```
class Personne {
  String nom, prénom;
  int âge;
  Téléphone numéro;
}
```

Les champs sont des variables

Classe, référence et objet

- **Référence** ≡ variable de type non primitif

```
CouleurRVB c1;
Personne p1, p2;
```

  - `c1`, `p1` et `p2` sont des références.
- Valeurs possibles pour la référence `c1`
  - `null` : l'objet néant (quel que soit le type)
  - un **objet** de type `CouleurRVB`
- 3 opérateurs pour les références
  - **opérateur point** : pour accéder aux champs
    - Impossible si la référence vaut `null`
    - `c1.rouge` = champ `rouge` de la référence `c1`
    - `p1.nom` = champ `nom` de la référence `p1`
  - `==` et `!=` pour comparer les valeurs (même objet)

Classe, référence et objet

- Pour créer un nouvel objet, on utilise l'opérateur `new`
  - Appel (invoque) le constructeur de la classe associée
  - Exemple : `new CouleurRVB(…);`
- Comment créer la couleur RVB bleu :
  - `CouleurRVB bleu = new CouleurRVB();`
  - `bleu.rouge = 0;`
  - `bleu.vert = 0;`
  - `bleu.bleu = 255;`
- Comment créer le magenta ?

Constructeur : `CouleurRVB`

- Pour initialiser les champs à des valeurs utilisateurs

```
class CouleurRVB {
  int rouge, vert, bleu;
  CouleurRVB(int r, int v, int b) {
    this.rouge = r;
    this.vert = v;
    this.bleu = b;
  }
}
```

- Lors de la création d'un objet, l'opérateur `new` invoque un des constructeurs de la classe
  - `CouleurRVB bleu = new CouleurRVB(0, 0, 255);`
  - `CouleurRVB cyan = new CouleurRVB(0, 255, 255);`

Notes

Constructeur : **Personne**

- Pour initialiser les champs à des valeurs utilisateurs

```
class Personne {
  String nom, prénom;
  int âge = -1;
  Personne(String nom, String prénom) {
    this.nom = nom;
    this.prénom = prénom;
  }
}
```

- Pour créer une **Personne**, il faut au moins, le nom et le prénom
  - `Personne _007 = new Personne("James", "Bond");`
  - `_007.âge = 44;`

Méthodes d'instances et statiques

- Les opérations associées aux classes doivent être définies dans la classe
  - Elles sont appelées **méthodes**
- **Personne** : accesseurs/modificateurs

```
class Personne {
  String getNom() {
    return this.nom;
  }
  int getÂge() {
    return this.âge;
  }
  void setÂge(int nouvelÂge) {
    this.âge = nouvelÂge ;
  }
}
```


Méthodes d'instances et statiques

- Leur comportement dépend des valeurs des champs
- Il faut une référence pour les utiliser
  - `_007.getNom();` est de type `String` et vaut `"Bond"`
- Attention à l'utilisation des modificateurs
  - `Personne _007 = new Personne("James", "Bond");`
  - `Personne p1 = _007;`
  - `p1.setÂge(25);`
  - Que vaut `_007.getÂge()` ?

Champ ou paramètre

- La méthode **setÂge** a besoin d'une information temporaire : le paramètre **nouvelÂge**
- Une méthode **assombrir** dans la classe **CouleurRVB**

```
class CouleurRVB {
  int rouge, vert, bleu;
  void assombrir() {
    this.rouge = this.rouge * 8 / 10 ;
    this.vert = this.vert * 8 / 10;
    this.bleu = this.bleu * 8 / 10;
  }
}
```

Méthodes d'instances et statiques

- Leur comportement ne dépend pas d'un objet en particulier, il est associé à une classe !
- **`String`** vers **`CouleurRVB`**

```
CouleurRVB string2Couleur(String val) {
  si val est "blanc" alors 255,255,255
  si val est "bleu" alors 0,0,255
  si val est "rouge" alors 255,0,0
  sinon 0,0,0
}
```

  - Dépend d'un objet ? Duquel ?
  - Dans quelle classe ?

Constantes - final

- On aimerait avoir des constantes pour représenter les couleurs classiques et ne pas construire de nouveaux objets à chaque fois

```
class Palette {
  final CouleurRVB bleu = new CouleurRVB(0,0,255);
  final CouleurRVB rouge = new CouleurRVB(255,0,0);
  final CouleurRVB cyan = new CouleurRVB(0,255,255);
}
```

  - Comment les utiliser ?
  - Peut-on faire plus pratique ?
  - Ré-écrire la méthode **`string2Couleur`**
  - Y a-t-il un problème avec la méthode **`assombrir`** ?

Variables locales

```
class CouleurRVB {
  int rouge, vert, bleu;

  CouleurRVB assombrir() {
    int r = this.rouge * 8 / 10 ;
    int v = this.vert * 8 / 10;
    int b = this.bleu * 8 / 10;
    return new CouleurRVB(r,v,b);
  }
}
```


Notes

Champ ou paramètre

```
class CouleurRVB {
  int rouge, vert, bleu;

  CouleurRVB assombrir(int percent) {
    assert(percent>=0 && percent<100);
    int r = this.rouge * percent / 100 ;
    int v = this.vert * percent / 100;
    int b = this.bleu * percent / 100;
    return new CouleurRVB(r,v,b);
  }
}
```


Structures de contrôle : alternative

- Les instructions sont exécutées en séquence
  - 1 après l'autre
- Si il y a un choix : **`if`** ou **`if/else`**
  - La condition de choix est un booléen (**`true`** ou **`false`**)

```
if(montant<prix) {
  System.out.println("montant insuffisant");
}
```

```
if(montant<prix) {
   System.out.println("montant insuffisant");
} else {
   System.out.println("vente conclue");
}
```


Structures de contrôle : répétition `for`

- La répétition **`for`** doit être utilisée pour énumérer des valeurs dans un intervalle connu :
  - Pour tout i entier ∈ [0,n[
    `for(int i=0; i<n; i++) { … }`
  - Pour tout i entier pair ∈ [4,n]
    `for(int i=4; i<=n; i+=2) { … }`
  - Pour tout x nombre approché ∈ [a,b] multiple de 0.1
    `for(double x=a; x<=b; x += 0.1) { … }`
  - Pour tout c caractère de l'alphabet minuscule
    `for(char c='a'; c<='z'; c++) { … }`

Structures de contrôle : répétition `for`

- Calculer la somme des n premiers entiers
  - Générer les entiers entre [1,n]
  - Les accumuler (dans une variable intermédiaire)

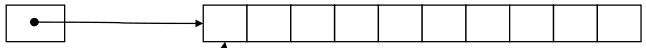
```
int somme(int n) {
  int s=0;
  for(int i=1; i<=n; i++) {
    s = s + i;
  }
  return s;
}
```

// somme(4) équivaut à
int s = 0;
s = s + 1;
s = s + 2;
s = s + 3;
s = s + 4;
s = (((0 + 1) + 2) + 3) + 4

Calculer le produit des n premiers entiers pairs ? Factorielle ?

Rappels sur les tableaux

- Déclaration de tableaux
  - `int[] t1,t2;` *// t1,t2 deux références sur tableau d'entiers*
  - `double[] t3;` *// Soit t3 un tableau de nombres approchés*
  - `String[] t4;` *// Soit t4 un tableau de chaînes de caractères*
- Construction d'un tableau : **`new`**
  - Les tableaux sont des objets
  - `t1 = new int[10];` // t1 référence un tableau de 10 entiers

    t1 
  - `t2 = t1;` // t2 référence le même tableau que t1

    t2 
  - `t4 = new String[3];` // t4 est un tableau de chaînes de caractères

    // aucune chaîne n'est construite

Rappels sur les tableaux

- Accès à la longueur d'un tableau (champ **`length`**)
  - `t1.length` → 10, `t2.length` → 10, `t4.length` → 3
  - `t3.length` → ? `NullPointerException`
- Accès aux éléments d'un tableau
  - Par leur indice, entier compris entre 0 et length-1
  - `t1[0]` est une variable entière, le premier élément du tableau `t1`
  - `t1[t1.length-1]` est le dernier élément du tableau `t1`
  - `t2[1]` est la même variable que `t1[1]`
  - `t4[2]` est le 3ème élément de `t4`, de type `String`
  - `t3[5]` ? `NullPointerException`

Sur l'initialisation d'un tableau

```
int[] tab1 = {4, 6, 3, 7, 9, 2, 3};
tab1.length ——→ 7
```

Quel est l'indice de 9 dans **`tab1`** ?

```
String[] tab2 = {"chou", "pou", "hibou"};
tab2.length ——→ 3
tab2[0] ——→ "chou"
```

Quel est l'indice de "hibou" dans **`tab2`** ?

```
int[][] tab3 = {{1,2,3}, {4,5,6,7}};
tab3.length     ——→ 2
tab3[0].length  ——→ 3
tab3[1].length  ——→ 4
```

Quel est le premier élément de **`tab3`** ?

Notes

Tableau et boucle `for`

- Calculer la somme des éléments d'un tableau t
  - Parcourir tous les éléments
  - On accède aux éléments par leur indice `[0, t.length[`

```
int somme(int[] t) {
  int s = 0;
  for(int i=0; i<t.length; i++){
    s = s + t[i];
  }
  return s;                  // renvoie le résultat
}
void test(){
  int[] t1 = {5,6,9,-2};
  int res = somme(t1);       // passe t1 en paramètre
  System.out.println("somme="+res);
}
```

```
int s = 0;
s = s + t[0];
s = s + t[1];
…
```


Structures de contrôle : répétition `while`

- La répétition **`while`** doit être utilisée lorsqu'on répète une action plusieurs fois.
- Exemple : Trouver l'indice de la première occurrence de 3 dans un tableau t
  - Connaît-on *a priori* le nombre de répétition ?

```
int indiceDe(int[] t, int x) {
  int i = 0;                    // pour parcourir les éléments de t
  while(i<t.length && t[i] != x) {
    i++;
  }
  // i >= t.length || t[i] == x
  if(i>=t.length) return -1;    // on n'a pas trouvé x
  else // i<t.length, donc t[i] == x
    return i;                   // on a trouvé x dans t
}
```


Répétition `while` ou `for`

- Parfois, utiliser **`for`** est plus lisible
- Exemple : première occurrence de x dans t

```
/** @return l'indice de x dans t, -1 si pas de x dans t */
int indiceDe(int[] t, int x) {
  // On suppose qu'on ne trouvera pas x,
  // il faudra alors parcourir tout le tableau
  for(int i=0; i<t.length; i++){
    if(t[i] == x) return i;
  }
  // Si on avait trouvé x, on aurait fait return et quitté
  // Donc on n'a pas trouvé x et on a atteint la fin du tableau
  return -1;
}
```


Répétition `while` ou `for`

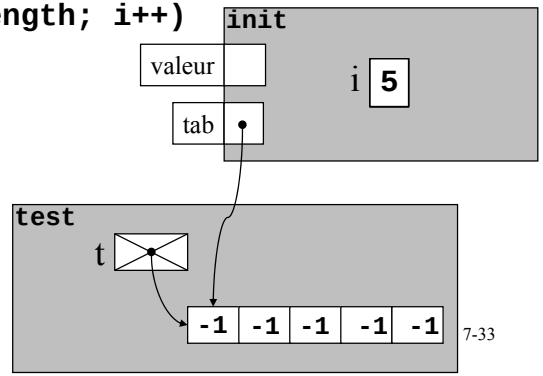
- **Parfois**, utiliser **`for`** est plus lisible
- Exemple : première occurrence de x dans t

```
/** @return l'indice de x dans t, -1 si pas de x dans t */
int indiceDe(String[] t, String x) {
  // On suppose qu'on ne trouvera pas x,
  // Il faudra alors parcourir tout le tableau
  for(int i=0; i<t.length; i++){
    if(x.equals(t[i])) return i;
  }
  // Si on avait trouvé x, on aurait fait return et quitté
  // Donc on n'a pas trouvé x et on a atteint la fin du tableau
  return -1;
}
```

Passer un tableau en paramètre

- Les tableaux peuvent être passés en paramètres
  - Attention, les tableaux sont des objets, ils sont manipulés par l'intermédiaire de références !
  - Exemple : Ecrire une méthode qui initialise un tableau à une valeur constante

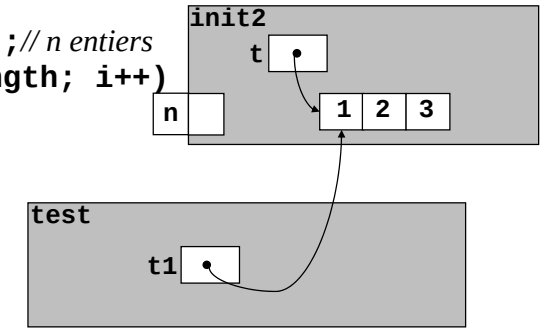
```
void init(int[] tab, int valeur){
  for(int i=0; i<tab.length; i++)
    tab[i] = valeur;
}
void test() {
  int[] t;
  t = new int[5];
  init(t, -1);
}
```



Renvoyer des tableaux

- Les méthodes peuvent créer des tableaux et les renvoyer comme résultat
  - Exemple : Une méthode qui créer un tableau contenant les entiers de 1 à n.

```
/** @return un tableau d'entiers {1, 2, …, n } */
int[] init2(int n) {
  int[] t = new int[n];// n entiers
  for(int i=0; i<t.length; i++)
    t[i] = i+1;
  return t;
}
void test() {
  int[] t1;
  t1 = init2(3);
} // la référence temporaire t a disparu, mais pas le tableau !
```



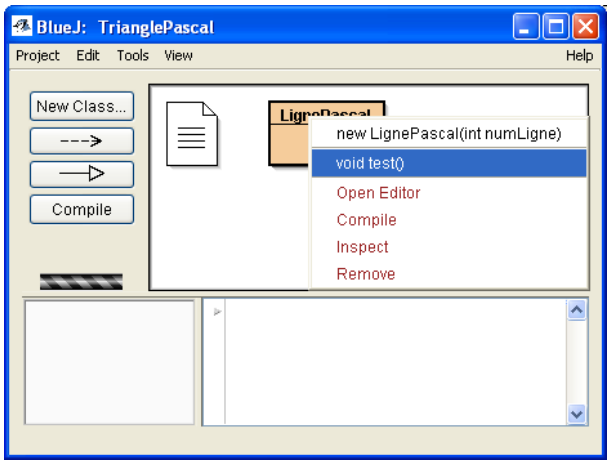
Une méthode statique pour tester

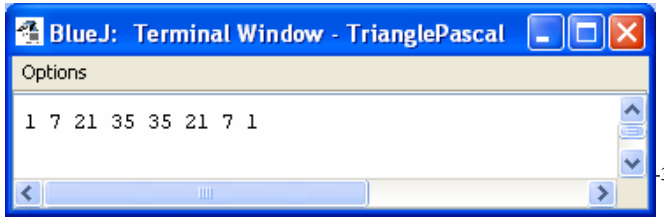
```
public class LignePascal {        // cf. cours 6 page 29 …

   private int[] ligne;

   public LignePascal(int numLigne) {
      La ième ligne de Pascal a i+1 nombres
      Le premier élément de chaque ligne est 1
      On part de la deuxième ligne, jusqu'à numLigne
        Pour chaque élément ligne[n] sauf le premier (qui vaut 1)
          ligne[n] vaut ligne[n] (ligne precedente) + ligne[n-1]
        }
      }
   }

   static void test() {
       LignePascal pas = new LignePascal(7);
       for(int i=0; i<pas.ligne.length; i+=1)
         System.out.print(pas.ligne[i] + "\t");
       System.out.println();
   }
}
```

Méthode de classe

Notes

♦On se sert de la méthode **test(…)** comme d'un lanceur pour exécuter un programme. C'est avec elle que l'on créera les objets et non plus avec un clic droit de souris !…





Fonctions à valeur tableau

♦Une méthode peut très bien avoir un résultat de type tableau.

$$n \longmapsto \{1, 2, 3, 4, \ldots, n\}$$

$$\{t_0, t_1, \ldots, t_{n-1}\} \longmapsto \{f(t_0), f(t_1), \ldots, f(t_{n-1})\}$$

$$\{t_0, t_1, \ldots, t_{n-1}\} \longmapsto \{t_0, t_{n-1}\}$$

♦Dans la classe précédente `LignePascal`, transformons le constructeur en fonction (méthode ayant un résultat) !


```
public class LignePascal1 {                  // sans champ !

    public int[] calculeLigne(int numLigne) {
      int[] ligne = new int[numLigne+1];
      ligne[0] = 1;
      for(int p=1; p<=numLigne; p+=1)
        for(int n=p; n>=1; n-=1)
          ligne[n] = ligne[n-1] + ligne[n];
      return ligne;
    }

    static void test(int n, int p) {
      LignePascal1 pas = new LignePascal1();
      int[] tab = pas.calculeLigne(n);
      System.out.print("binomial(" + n + "," + p + ")=");
      System.out.println(tab[p]);
    }

}
```

Méthode d'instance

Méthode de classe (static)

Notes