Cours1
Java pour l'algorithmique

Année 2016-2017 J-F. Kamp - IUT de Vannes

PLAN

- Java simplifié
- Stratégie de développement
- Structure générale d’un programme en Java
- Types primitifs et la classe String
- Opérateurs
- Alternative
- Itération
- Entrées / Sorties
- Tableaux
- Sous-programmes

Java simplifié

- Pas d’objets

  => pas de new (sauf …)

- Pas de vrai lanceur (main)

  => un principal comme TestAlgo

- Des méthodes = sous-programmes

  => pas de distinction entre fonction et procédure

- 1 fichier java ⇔ 1 fichier TestAlgo

  MonAlgo.java (avec un principal)

  java Start MonAlgo

Stratégie de développement

- Découpage en méthodes

  => void afficherTab (int[] tab, int nb)

  => rôle bien identifié

- Chaque méthode est testée individuellement (test unitaire)

```
void testAfficherTab() {
    int[] monTab = new int[10];
    // test cas normal
    afficherTab ( monTab, 10 );
    // autres tests
    …
}
```

- Le principal appelle les cas de test

```
void principal() {
    testAfficherTab();
    …
}
```

Structure générale

```
public class MonAlgo {
   // Déclaration des CONSTANTES
   final int TAILLE = 50 ;    // exemple CSTE

   void principal() {
      testMaMeth1();
      …
   }

   void testMaMeth1() {
      // déroulement du test de maMeth1(…)
      …
   }

   int maMeth1 ( int[] tab, double d ) {
      // déclaration des variables locales
      int ret;
      …
      …
      return ret;
   }

}
```

8 types de base en Java
types primitifs

- boolean : type booleen (2 bits, true, false)
- char : type caractère ('1', '?', '(', ...)
- byte : type entier sur 8 bits
- short : type entier sur 16 bits
- int : type entier sur 32 bits
- long : type entier sur 64 bits
- float : type réel sur 32 bits
- double : type réel sur 64 bits

Classe String

- String = chaîne de caractères en Java

  Exemples acceptés :

  ```
  int var1 = 52;
  String st1 = "Résultat = ";
  String st2 = st1 + var1; // idem TestAlgo
  ```

- !! String est une classe : piège !!

  En réalité il faudrait faire

  ```
  String st1 = new String ("Résultat = ");
  ```

  « st1 » ne contient PAS "Résultat = " (le contenu de la case mémoire) MAIS l'adresse de la case mémoire

Classe String

Exemple de piège : comparaison du contenu de 2 chaînes de caractères

```
String var1 = new String ( "Toto" ) ;
String var2 = new String ( "Toto" ) ;
```

"Toto" = "Toto" MAIS var1 ≠ var2

Ce test ne renvoie pas le résultat espéré :

var1 = = var2 renvoie false

// on compare des adresses !!

Ce qu'il faut faire : var1.equals (var2)

Opérateurs

- affectation : =
- opérateurs mathématiques : *, /, -, +

  Uniquement les types entiers et réels.

  + aussi utilisé pour concaténer chaînes de caract.
- modulo : % (types entiers et réels)
- incrémentation de 1 : ++ (types entiers et réels)
- décrémentation de 1 : -- (types entiers et réels)
- égalité : = =
- inégalité : !=
- supérieur, inférieur :

  >, >=, <, <= (types entiers et réels)
- et, ou, non :

  &&, ||, ! (types booléens uniquement)

Alternative simple

```
if ( <condition> ) {
        instructions ;
        ...
}
[ else {
        instructions ;
        ...
} ]
```

Alternatives multiples

```
if (<condition1>) {

        instructions ;
}

else if (<condition2>) {

        instructions ;
}

else {

        instructions ;
}
```

Alternatives multiples : switch

```
switch (uneVariable) { // uniquement byte, short, int, char

        case <valeur1> :
                // exécuté si (uneVariable == valeur1)
                <instructions>;
                // on quitte le switch
                break;

        case <valeur2> :
                // exécuté si (uneVariable == valeur2)
                <instructions>;
                // on quitte le switch
                break;

        default :
                // facultatif
                // exécuté si tout le reste a échoué
                <instructions par défaut>;
                // on quitte le switch
                break;
}
```

Itération

Equivalent « tantque » TestAlgo

```
while ( <condition de continuation> ) {

        instructions ;
        ...
}
```

Equivalent « repeter » TestAlgo sauf…

```
do {
        ...
        instructions ;
        ...
} while ( <condition de continuation> ) ;
// !! Condition de continuation, PAS
// condition d'arrêt comme TestAlgo !!
```

Itération

```
for ( <init>; <condition de continuation>; <incr> ) {
        ...
        <instructions> ;
}
```

La boucle « for » est strictement équivalente à la boucle « while » suivante :

```
<init>;

while ( <condition de continuation> ) {

        ...
        <instructions> ;

        <incr> ;
}
```

Itération

« FOR » Java N’EST PAS « POUR » TestAlgo

Le « pour » TestAlgo est un simple compteur.
Si il ne peut pas compter, il ne s’exécute pas.

```
entier borne := -1;
pour  i := 0 à borne pas 1
        ...
finpour
```

pas d’exécution : boucle infinie

Le « for » Java est un véritable « while ». Il peut boucler à l’infini ou ne pas exécuter le corps.

```
int borne = -1;
for ( i = 0; i<=borne; i++ ) {
        // corps de la boucle
}
```

le corps de la boucle n’est jamais exécuté

Entrées / Sorties

Affichage à l'écran

System.out.println (String str) ;

où *str* = la concaténation (+) de n'importe quelle chaîne de caractères constante ("toto") avec n'importe quel type primitif et n'importe quel autre type *String* (idem TestAlgo)

Exemple :

```
String maChaine = "Le résultat";
int res = 306;
System.out.println ( maChaine + " est : " + res ) ;
```

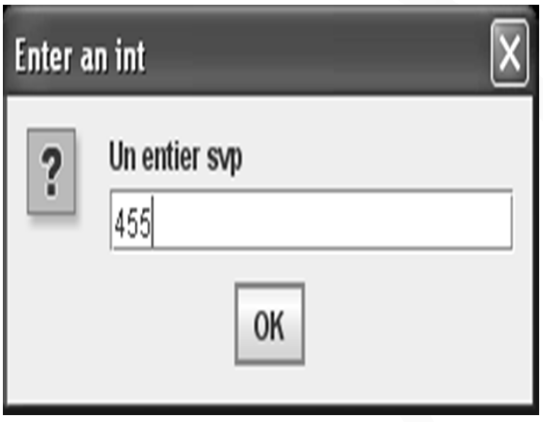
Entrées / Sorties

Saisie par boite de dialogue.

Une classe spécifique SimpleInput permet de faire de la saisie de types primitifs et String par boite de dialogue graphique.

Exemple, saisie d’un nombre entier au clavier :

```
int unNbre ;
unNbre = SimpleInput.getInt ("Un entier svp");
```



Tableaux simples

Points communs avec TestAlgo :

- chaque case du tableau contient le même < type > de données (int ou double ou String ou …)

- si N est le nombre de cases, chaque case est numérotée de 0 à (N - 1)

- un tableau est toujours passé à un sous-programme par adresse

Tableaux simples

Différences avec TestAlgo :

- la taille N du tableau ne doit pas être spécifiée au moment de sa déclaration (allocation dynamique)
- une fois alloué, un tableau « monTab » connaît sa taille N : monTab.length renvoie la valeur N
- une fois alloué, chaque case du tableau a un contenu par défaut (zéro)
- « monTab » est une variable qui contient l’adresse de la première case du tableau (null par défaut)

Tableaux simples

```
//  Exemple, déclaration d'un tableau d'entiers

int [ ] monTab ;      // A ce stade, monTab contient
                      // la valeur  « null » et est donc
                      // inutilisable !

//  Par définition, « monTab » contient l'adresse
//  de la première case du tableau


//  Allocation dynamique du tableau

int taille = 25;

monTab = new int [ taille ] ;


//  Affectation d'une valeur Y à la case n°X
//  ( 0 <= X < N )

monTab [ X ] = Y ;


//  Récupération de la taille du tableau dans la
//  variable « laTaille »

int laTaille = monTab.length ; // laTaille contient 25
```

Sous-programmes

Points communs avec TestAlgo :

- Un sous-programme peut être défini n'importe où à l'intérieur d'une classe Java
- Un sous-programme reçoit des données initiales par l'intermédiaire de ses paramètres
- Un tableau est toujours passé à un sous-programme par adresse (référence)
- Un sous-programme est appelé par « principal() » ou n'importe quel autre sous-programme

Sous-programmes

Différences avec TestAlgo :

- En Java, un sous-programme s’appelle méthode et il n’y a pas de différence entre fonction et procédure

- Une méthode Java peut renvoyer zéro ou un résultat par l’intermédiaire de l’instruction return (= retourne TestAlgo), y compris un tableau

Sous-programmes

Différences avec TestAlgo :

- Le choix de passage d'une variable par valeur ou par adresse n'est PAS possible (l'équivalent de @ n'existe pas) : le type primitif (int, double, …) est toujours passé par valeur

- Lorsqu'une méthode ne renvoie pas de résultat par l'intermédiaire de « return » le type de retour déclaré doit être void

Sous-programmes

Un exemple concret de méthode Java

```
/**
   *  Cette méthode calcule la somme
   *  des éléments contenu dans le
   *  tableau passé en paramètre
   *  @param leTab le tableau des valeurs
   *  @param nbElem le nbre de valeurs
   *  @return la somme des valeurs
   */
int somTab ( int [ ] leTab, int nbElem ) {
   int som, i ;    // variables locales
   som = 0 ;
   for ( i = 0; i < nbElem; i++ ) {
      som = som + leTab[i] ;
   }
   return som ;
}
```