A3P/IPO 2022/2023

Cours 3

Sommaire

`private`

- Le fait de déclarer dans une classe (A) les attributs `private` a pour but de les protéger des accès d'une méthode d'une autre classe (B).
- Mais cela n'empêche pas une méthode de la même classe (A) d'accéder aux attributs de **tout objet** de cette classe (A).
- *Cette propriété est indispensable pour écrire des méthodes de test d'égalité.*

Tests

- ```
  if ( condition )
     uneInstruction;
  else
     uneInstruction;
  ```
  (expression booléenne)

  _ *facultatif*
  /

- ***uneInstruction;***

  peut toujours être remplacée par un bloc d'instructions

  ```
  {
     une ou;
     plusieurs;
     instructions;
  }
  ```
  ```

Expression booléenne

- ```
  if ( condition )
    return true;
  else
    return false;
  ```

  est équivalent à

- ```
  return condition;
  ```

```
if ( condition )
  return false;
else
  return true;
```

est équivalent à :

```
return !condition;
```

- `x != y` est préférable à `!( x == y )`

`if` non indépendants

- ```
  if ( a < 0 )  instruction1;
  if ( a > 0 )  instruction2;
  if ( a == 0 ) instruction3;
  ```

- ```
  if ( a < 0 ) instruction1;
  else if ( a > 0 ) instruction2;
  else if ( a == 0 ) instruction3;
  ```

- ```
  if ( a < 0 ) instruction1;
  else if ( a > 0 ) instruction2;
  else instruction3;
  ```
  ```

if indépendants ou non

- ```
  if ( d == 1 )
    instruction1;
  if ( d == 2 )
    instruction2;
  if ( d == 3 )
    instruction3;
  ```

- ```
  if ( a == 0 )
    instruction1;
  if ( b == 0 )
    instruction2;
  if ( c == 0 )
    instruction3;
  ```

- **else if !**

« Égalités »

- `x=y;` instruction d'affectation à la variable `x` de la valeur de l'expression `y`

- `x==y` expression booléenne de comparaison binaire des expressions `x` et `y` (comparaison des références si `x` et `y` sont des objets)
  Attention aux String !

- `x.equals(y)` expression booléenne de comparaison du contenu des objets `x` et `y`

Exemple

- `String vS1 = "mot";`
  `String vS2 = "mot";`

- Comparaison « binaire » :
  `( vS1 == vS2 )` ? `true` ou `false` ?

- Comparaison d'objets :
  `vS1.equals( vS2 )` ?
  `true` ou `false` ?

Tableaux

- **Déclaration** : `typeElements[] nomTableau`
  (`typeElements` peut être **n'importe quel type** existant, y compris un type tableau !) *C'est une référence !*
- **Création** : `nomTableau = new typeElements[xEntierPositif];`
- **Taille** : `int vNombreDeCases = nomTableau.length;`
  et indices toujours de *0* à *vNombreDeCases-1*
- **Accès** à la case d'indice `i` : `nomTableau[i]`
  (en lecture et en écriture)
- **Tout** en un : `typeElements[] nomTableau = { valeur0, ..., valeurN-1 };`
  (la taille est calculée automatiquement)
- `System.out.println( nomTableau );` **?**

Exemples

- Attribut :

```
private String[] aMots;
```

- Paramètre :

```
...( final double[] pNotes )
```

- Variable locale :

```
int[] vNombres = new int[10];
int[] vNombres = { 10, 25, -15 };
```

Boucles

- **for** (*déclaration_initialisation*;
  condition_de_continuation;
  instruction_de_progression) {
  instructions_à_répéter;
  } // **for**

À utiliser à chaque fois qu'on peut déterminer avant le début de la boucle, le nombre de fois qu'elle va tourner.

est équivalente à :

- *déclaration_initialisation*;
  while (*condition_de_continuation*) {
  instructions_à_répéter;
  instruction_de_progression;
  } // **while**

System.out.println

- ```
  System.out.println(
    "x=" + this.aX + "!" );
  ```

est équivalent à :

- ```
  System.out.print(
    "x=" + this.aX + "!" + "\n" );
  ```

et à :

- ```
  System.out.print( "x=" );
  System.out.print( this.aX );
  System.out.println( "!" );
  ```
  ```

`int` et `double`

- **`int`** <- abréviation d' Integer *(entier)*

  sur n=8/16/**32**/64 bits => 2^n valeurs => limité à -2^{n-1} … $+2^{n-1}-1$

- **`double`** <- précision double *(réel sur 32/**64** bits)*

  - limité à **environ** -10^{+300} … -10^{-300}, 0.0, $+10^{-300}$ … $+10^{+300}$
  - tous ces nombres de sont pas représentables :
    par exemple, 0.1 (= 0,0625 + 0,03125 + …)
  - erreurs de calcul, propagation/aggravation
  - jamais `==` `!=` `<=` `>=` **|x-y| < ε** remplace `x==y`

- Conversion de type *(cast)* : `(NouveauType)expression`

  - avec ou sans modification de la valeur
  - Exemple de la partie entière : `int vE = (int)vRéel;`

`vRéel` est inchangée ! ➔ nouvelle expression !

Exemples d' **expressions**

- Entières : `12, -354, -this.aX, (int)pR, 1+vI/3, pC.getDiametre()`
- Réelles : `1.5, 6E-23, 1+vI/3.0, (double)vI, Math.sqrt( 2.0 )`
- Booléennes : `true, false, vOK, vI<0 && x==y, this.aEstVisible()`
- String : `"mot", "", "res="+vI, aCurrentRoom.getDescription()`
- Références : `this, null, new Cercle(), this.getSoleil()`

Exemples d' **instructions**

- **Affectation** : `variable=`*`expression`*`;`
- **Affichage** : `System.out.println(` *`expression`* `);`
- **Retour** : `return` *`expression`*`;`
- **Appel de procédure** : `objet.procedure(` *`expr1`*`,` *`expr2`* `);`
- **Test** : `if (` *`exprBooléenne`* `)` …
- **Tant que** : `while (` *`exprBooléenne`* `)` …
- **Pour** : `for ( int var=`*`exprEntière`*`;` *`exprBooléenne`*`;` **`var++`** `)` …

A ne pas confondre :

- Classe / objet
- Variables / méthodes
- Attributs / paramètres / variables locales
- Fonctions / procédures / constructeurs
- Définition… / appel… …de méthode
- Types primitifs / objets
- Expression / instruction
- Retour de valeur / affichage
- Affectation / comparaison

Forum !

A NE PAS OUBLIER

- Avez-vous pensé à regarder les liens de la colonne de droite ?

- Syntaxe interactive (TP 2.1)
- Guide de style

- Atelier de débogage

- Résa 3.3, dont CRAPEL