A3P/IPO 2024/2025

Cours 2

© Denis BUREAU (ESIEE Paris)

Sommaire

1. Commentaires
2. L'ordre des choses …
3. **Expressions / instructions**
4. Définition de nouvelles méthodes + **`return`**
5. Accesseurs et modificateurs + les types
6. Constructeurs (3 sortes)
   + instanciation + **`this()`**
7. **`null`**
8. **Récursivité**
9. Thème du projet

0. Rappels

- ```
  public void vaBas() {
     this.deplacementVertical(20);
  }
  ```

- ```
  cercle1.vaBas();
  => cercle1.deplacementVertical(20);
  ```

- ```
  cercle2.vaBas();
  => cercle2.deplacementVertical(20);
  ```

- ```
  cercle3.vaBas();
  => cercle3.deplacementVertical(20);
  ```

1.1. Commentaires (déjà vus)

- `// jusqu'à la fin de la ligne`
  - commenter les **}** fermantes
  - commenter des blocs de code (F8/F7)
- **Pour une classe ou une méthode**,
  { et **}** seront chacune seule sur leur ligne
-
```
/**
 * plusieurs lignes
 * pour la javadoc
 * @param pNom explication
 */
```

1.2 Commentaires : 3 sortes

Non vus par le compilateur, mais utiles !

- Fin de ligne : `} // commentaire`
  Débuts de ligne : `// instruction;`
- Intra-ligne ou multi-lignes :

```
vX = vX + /* vN + */ 1;
```

```
/* instruction1;
instruction2; */
```

- Javadoc *(avant !)* :

```
/** explication,
 *  plus de détails
 */
```

2. Ordres dans une classe

- 1) attributs
  ordre sans importance
- 2) constructeurs
  ordre sans importance
- 3) autres méthodes *(même si elles s'appellent)*
  ordre sans importance

(1 2 3 : ordre habituel => « imposé »)

- Instructions entre { et }
  ordre très important !

Paramètres
ordre très important !

3. Expressions ≠ Instructions

- Une **expression** a toujours une valeur
  (vaut quelque chose une fois calculée)
- Une **instruction** contient souvent une expression :
  - `variable = expression;`
    différent de
    `( variable == expression )`
    *(**expression** booléenne)*
  - `return expression;`
    différent de
    `System.out.println( expression );`

(instruction d'affectation)

(instruction de terminaison)

(instruction d'affichage)

4.1. Définition de méthode

- = signature + corps
- **Signature** =

```
public typeRetour^α nom( paramètres )
```

- **Corps** =

```
{
    instructions^β;
} // nom()
```

- $^{\alpha}$ procédure => `void`
- $^{\alpha}$ constructeur => ne rien mettre !
- $^{\beta}$ fonction => `return expression;`

4.2. Définition de méthode ?

- = signature + corps
- Signature =

```
public
typeRetour
nom
(
paramètres
)
```

- Corps = `{ instructions; } // nom()`

4.3. `return`

- `return` *`expression`*`;`
  obligatoire pour terminer une fonction
  (retourne une et une seule valeur)

- `return;`
  facultatif pour terminer prématurément une procédure
  (ne retourne rien)

- Ce n'est pas un appel de méthode !
  Donc évitez `return(` *`expression`* `);`

5. Accesseurs & modificateurs

- Accesseur = **fonction** qui retourne la valeur d'un attribut :
-
```
public typeAttribut getNomAttribut()
{ return this.aNomAttribut;
} // getNomAttribut()
```

- Modificateur = **procédure** qui modifie la valeur d'un attribut :
-
```
public void setNomAttribut(
    final typeAttribut pNomAttribut )
{ this.aNomAttribut = pNomAttribut;
} // setNomAttribut(.)
```

6.0 Les types

- Rappel : 2 sortes
  primitifs (int, …) et **références** (Classe, …)

- Utilisables pour :
  - attributs
  - paramètres
  - variables locales
  - valeurs de retour de fonction

- Type X utilisable à l'intérieur de la classe X !

6.1. Constructeurs

Plusieurs constructeurs dans une classe :

- Par défaut = sans paramètre

```
public NomClasse()
{ this.aNomAttribut1 = valeur_fixe1;
  this.aNomAttribut2 = valeur_fixe2;
} // NomClasse()
```

- **Naturel** = paramètres correspondant exactement aux attributs

```
public NomClasse(
        final typeAttribut1 pNomAttribut1,
        final typeAttribut2 pNomAttribut2 )
{ this.aNomAttribut1 = pNomAttribut1;
  this.aNomAttribut2 = pNomAttribut2;
} // NomClasse(..)
```

- quelconques …

6.2. Nouvel objet = instance

- Déclaration d'(une référence vers) un objet :
  `Cercle vC1;` ***aucun cercle n'est créé !***

  instancier, instanciation :
- Création d'un objet =
  `vC1 = new Cercle();`
  => appel du constructeur sans paramètre
  `vC1 = new Cercle( 30, "red" );`
  => appel du constructeur à 2 paramètres si ∃
- **`new Cercle()`** : expression/instruction ?

6.3. Duplication de code

- **TOUT FAIRE POUR L'ÉVITER !**

- Cercle() :
  `this.aX=1; ♥.aY=2; ♥.aD=3; ♥.aC="red";`
- Cercle(…pX, …pY, …pD, …pC) :
  `this.aX=pX; ♥.aY=pY; ♥.aD=pD; ♥.aC=pC;`

- Dans ce cas, le 1er constructeur s'écrit :
  `this( 1, 2, 3, "red" );`
  ! première instruction d'un constructeur !

7.1 `null`

- `null`, identique quelle que soit la classe, puisque signifie « pas d'objet » ;
  si `variable` peut valoir `null`, alors
  ni `variable.attribut`,
  ni `variable.méthode()` sinon :

- **`NullPointerException`**

- `Cercle vC1;` ==> *référence* ≡ adresse
  `null` ==> 0 , `new Cercle()` ==> ≠0

7.2 **NullPointerException**

- `this.aSoleil = null;`
  `…=this.getPositionSoleil(this.aSoleil);`

```
… getPositionSoleil(final Cercle pC) {
  int vPos = pC.getPosition();
  if (…)  return null;  else  …;
}
```

- Recherche de la cause :
  `a.b()``.c()  a ?  a.b() ?`
  aMaison.getPositionSoleil(aSoleil3)*.length()*

8.1 Récursivité

- - méthode qui s'appelle elle-même
  - => un paramètre qui change
  - => un test d'arrêt !

- ```
  public void maMethode( final int pN )
  {
    if ( pN < 1 )    return;
    faireUnPas();
    this.maMethode( pN - 1 );
  } // maMethode(.)
  ```
  ```

8.2 StackOverflowError

- - méthode qui s'appelle elle-même
  - => un paramètre qui change
  - => un test d'arrêt !
  - récursion « infinie » !?

-
```
public void maMethode( final int pN )
{
   if ( pN > 1 )    return;
   faireUnPas();
   this.maMethode( pN - 1 );
} // maMethode(.)
```

A propos du projet

- Vous devez avoir lu les ressources indiquées dans la séance **Résa** *(+ Charte de l'unité !)*
- **Vous devez avoir réfléchi au thème de votre jeu :** inventez une histoire (un scénario) et soyez capable de la résumer en une phrase selon le modèle indiqué dans les archives.
- Date limite de saisie : ***prochaine séance Résa*** 3.1

Apprendre par cœur
tous les encadrés rouges

Premier Contrôle …

Lien cliquable sur la page web

Besoin de post-Assistance
pour mieux préparer ce contrôle ?

Lundi entre 18h et 19h

(apportez un micro-casque)

Apprendre par cœur
tous les encadrés rouges

+ terminer/refaire (seul) tous les TP & TD