Programmation objet en Java

Vincent Vidal

Maître de Conférences
Enseignements : IUT Lyon 1 - pôle AP - Licence ESSIR - bureau 2ème étage
Recherche : Laboratoire LIRIS - bât. Nautibus
E-mail : vincent.vidal@univ-lyon1.fr

Supports de cours et TPs : `http://spiralconnect.univ-lyon1.fr` module "S2 - M2103 - Java et POO"
48H prévues $\approx$ 39H de cours-TDs + 6H de TPs, 1H - interros, et 2H - examen final

Évaluation : Contrôle continu + examen final + Bonus/Malus TP

Plan

Java ne dispose pas de "vrais" tableaux nD...

- On peut "simuler" les tableaux 2D avec des tableaux 1D de références sur des tableaux. Par contre la taille de chaque "sous-tableau" ne sera pas obligatoirement la même...
  Et les lignes d'un tableau nD ne seront pas contiguës en mémoire.

Java ne dispose pas de "vrais" tableaux nD...

- On peut "simuler" les tableaux 2D avec des tableaux 1D de références sur des tableaux. Par contre la taille de chaque "sous-tableau" ne sera pas obligatoirement la même...
  Et les lignes d'un tableau nD ne seront pas contiguës en mémoire.

Java ne dispose pas de "vrais" tableaux nD...

- On peut "simuler" les tableaux 2D avec des tableaux 1D de références sur des tableaux. Par contre la taille de chaque "sous-tableau" ne sera pas obligatoirement la même...
  Et les lignes d'un tableau nD ne seront pas contiguës en mémoire.

Déclaration d'un tableau 2D

Les 3 déclarations suivantes sont équivalentes :

```
int t [] [] ;
int [] t [] ;
int [] [] t ;
```

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] } ;
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} } ;
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] };
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} };
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] };
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} };
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] } ;
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} } ;
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] };
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} };
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Initialiseurs d'un tableau 2D

```
int t1 [] [] = { new int [4], new int[3] } ;
int t2 [] [] = { {1,3,5}, {0,2,4,6,8,10}, {1,2,3,5,7} } ;
```

- t1.length vaut 2.
  t1[0].length vaut 4.
  t1[1].length vaut 3.
- t2.length vaut 3.
  t2[0].length vaut 3.
  t2[1].length vaut 6...

Sans les initialiseurs...

```
int t [] [] ;
t = new int [2][] ; // tableau de 2 tableaux de int
t[0] = new int [3] ;
t[1] = new int [4] ;

int t2[][] = new int [][2] ; // engendre une erreur de compilation

int t3[][] = new int [5][2] ;// Facilité de Java (matrice rectangle)
```

Exemple de parcours d'un tableau 2D

```
static void affiche(int t[][])
{
   int col ;
   for(int lig=0; lig<t.length; lig++)
   {
      for(col=0; col<t[lig].length; col++)
         System.out.print(t[lig][col]+ " ") ;

      System.out.println() ;
   }
}
```

Exemple de parcours d'un tableau 2D en JDK 5.0

```
static void afficheJDK5(int t[][])
{
   for(int [] ligne : t)
   {
      for(int valCase : ligne)
         System.out.print(valCase+ " ") ;

      System.out.println() ;
   }
}
```

Ce parcours JDK 5.0 n'est valide que pour un accès aux valeurs des cases, pas pour les modifier.

Depuis JDK 5.0 le dernier argument d'une méthode peut être variable en nombre

```
static int somme(int ... valeurs)
{
   int s = 0 ;
   for(int argi : valeurs) // le dernier argument est
   {                       // interprété comme un
      s += argi ;          // tableau 1D
   }
   return s ;
}
// exemples d'appel valide:
somme() ;
somme(5, 7, 9) ;
somme(tab1D) ; // tab1D étant de type int []
```

Ellipse et surdéfinition de méthodes

Règle : on cherche en premier une méthode sans ellipse qui correspond, en utilisant les conversions non-dégradantes si nécessaire. Si aucune méthode n'est trouvée, alors on effectue la recherche en faisant intervenir les méthodes à ellipse.

- void f(int ... valeurs) et void f(int [] t) ne peuvent pas coexister.
- void f(int i1, int i2) et void f(int ... valeurs) peuvent coexister.
- void f(double d1, double d2) et void f(int ... valeurs) peuvent coexister.

Ellipse et surdéfinition de méthodes

Règle : on cherche en premier une méthode sans ellipse qui correspond, en utilisant les conversions non-dégradantes si nécessaire. Si aucune méthode n'est trouvée, alors on effectue la recherche en faisant intervenir les méthodes à ellipse.

- `void f(int ... valeurs)` et `void f(int [] t)` ne peuvent pas coexister.
- `void f(int i1, int i2)` et `void f(int ... valeurs)` peuvent coexister.
- `void f(double d1, double d2)` et `void f(int ... valeurs)` peuvent coexister.

Ellipse et surdéfinition de méthodes

Règle : on cherche en premier une méthode sans ellipse qui correspond, en utilisant les conversions non-dégradantes si nécessaire. Si aucune méthode n'est trouvée, alors on effectue la recherche en faisant intervenir les méthodes à ellipse.

- void f(int ... valeurs) et void f(int [] t) ne peuvent pas coexister.
- void f(int i1, int i2) et void f(int ... valeurs) peuvent coexister.
- void f(double d1, double d2) et void f(int ... valeurs) peuvent coexister.

Ellipse et surdéfinition de méthodes

Règle : on cherche en premier une méthode sans ellipse qui correspond, en utilisant les conversions non-dégradantes si nécessaire. Si aucune méthode n'est trouvée, alors on effectue la recherche en faisant intervenir les méthodes à ellipse.

- `void f(int ... valeurs)` et `void f(int [] t)` ne peuvent pas coexister.
- `void f(int i1, int i2)` et `void f(int ... valeurs)` peuvent coexister.
- `void f(double d1, double d2)` et `void f(int ... valeurs)` peuvent coexister.

D'un type primitif vers une chaîne

```
int n = 47 ;
String chi = String.valueOf(n) ;

double x = 54.3 ;
String chd = String.valueOf(x) ;
```

La méthode statique valueOf **de la classe String est surdéfinie pour chaque type primitif.**

Remarque : 2 autres solutions : "" + n ou Integer.toString(n)

D'un type primitif vers une chaîne

```
int n = 47 ;
String chi = String.valueOf(n) ;

double x = 54.3 ;
String chd = String.valueOf(x) ;
```

La méthode statique valueOf **de la classe String est surdéfinie pour chaque type primitif.**

Remarque : 2 autres solutions : "" + n ou Integer.toString(n)

D'une chaîne vers un type primitif

Il faudra utiliser une méthode statique parseXXX de la classe enveloppe associée au type primitif (e.g. Integer pour int).

```
String ch = "1234";
int n = Integer.parseInt(ch) ;

String ch = "42.7";
double x = Double.parseDouble(ch) ;
```

Ce type de conversion impose des contraintes de formatage de la chaîne : par exemple une chaîne contenant un entier ne doit pas commencer par '+'. En cas d'échec de conversion une exception *NumberFormatException* est lancée.

Plan

Décrire votre code dans votre code lui-même plutôt que dans un document séparé vous aide à garder votre documentation à jour.

Page de référence pour Javadoc :
http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments

Les commentaires /** ... */

Objectif principal : Décrire les spécifications d'une API.

Plus de détails sur comment écrire les commentaires Javadoc :
http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Décrire votre code dans votre code lui-même plutôt que dans un document séparé vous aide à garder votre documentation à jour.

Page de référence pour Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments`

Les commentaires /** ... */

Objectif principal : Décrire les spécifications d'une API.

Plus de détails sur comment écrire les commentaires Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`

Décrire votre code dans votre code lui-même plutôt que dans un document séparé vous aide à garder votre documentation à jour.

Page de référence pour Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments`

Les commentaires /** ... */

Objectif principal : Décrire les spécifications d'une API.

Plus de détails sur comment écrire les commentaires Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`

Décrire votre code dans votre code lui-même plutôt que dans un document séparé vous aide à garder votre documentation à jour.

Page de référence pour Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments`

Les commentaires /** ... */

Objectif principal : **Décrire les spécifications d'une API.**

Plus de détails sur comment écrire les commentaires Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`

Décrire votre code dans votre code lui-même plutôt que dans un document séparé vous aide à garder votre documentation à jour.

Page de référence pour Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html#javadocdocuments`

Les commentaires /** ... */

Objectif principal : **Décrire les spécifications d'une API.**

Plus de détails sur comment écrire les commentaires Javadoc :
`http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`

Quelques règles pour Javadoc

- Un commentaire Javadoc n'est pas un tutoriel ou un guide d'utilisation, mais **une spécification**. Si cela est vraiment utile à la compréhension, alors on placera un lien (doc, site Web) vers une explication plus approfondie.
- On ne décrit pas les détails d'implémentations, mais le **comportement d'une méthode**.
- On utilise la 3ème personne dans une description et on évite l'impératif.

Quelques règles pour Javadoc

- Un commentaire Javadoc n'est pas un tutoriel ou un guide d'utilisation, mais **une spécification**. Si cela est vraiment utile à la compréhension, alors on placera un lien (doc, site Web) vers une explication plus approfondie.
- On ne décrit pas les détails d'implémentations, mais le **comportement d'une méthode**.
- On utilise la 3ème personne dans une description et on évite l'impératif.

Quelques règles pour Javadoc

- Un commentaire Javadoc n'est pas un tutoriel ou un guide d'utilisation, mais **une spécification**. Si cela est vraiment utile à la compréhension, alors on placera un lien (doc, site Web) vers une explication plus approfondie.
- On ne décrit pas les détails d'implémentations, mais le **comportement d'une méthode**.
- On utilise la 3ème personne dans une description et on évite l'impératif.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

Quelques règles pour Javadoc

- Conditions aux bornes, intervalles des paramètres, comportement dans les cas critiques (e.g. objet `null`, division par zéro).
- Les assertions (pré-conditions et post-conditions) décrites doivent être **indépendantes de l'implémentation**.
- En l'absence de commentaires à ce sujet, une méthode est supposée *thread-safe*.
- On peut éviter de réécrire entièrement les commentaires des méthodes si : elle *redéfinit* une méthode d'une classe parent (@Override), ou si elle *implémente une méthode d'une interface* (cf. cours sur l'héritage). Dans ces cas un lien vers la méthode redéfinie sera rajouté par Javadoc.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

L'ordre (à respecter) des tags Javadoc

1. @author *nom* (classes et interfaces) ;
2. @version %I%, %G% (classes et interfaces) ;
3. @param *nom description* (méthodes et constructeurs) ;
4. @return (méthodes qui retournent qqc différent de void) ;
5. @exception ou @throws (si des exceptions sont jetées) ;
6. @see *type* ;
7. @since *1.0* ;
8. @serial ;
9. @deprecated (depuis quand et quoi utiliser à la place).

Un même tag peut apparaître sur plusieurs lignes successives : @author, @param, @exception, et @see.

Exemple de documentation Javadoc

```java
/**
* Addition de deux entiers.
* @param a Le premier entier.
* @param b Le second entier.
* @return La somme de a et de b.
*/
public int addition (int a, int b) {
   return a + b ;
}
```

Exemple de documentation Javadoc

```
/**
* Division entière.
* @param a Le premier entier.
* @param b Le second entier.
* @return Le résultat de la division entière de a par b.
* @throws ArithmeticException lorsque b vaut 0.
*/
public int division (int a, int b) throws
    ArithmeticException {
   return a / b ;
}
```

Exemple de documentation Javadoc

```
/**
* Classe représentant un enseignant.
* @author Jean Dupont
* @version 4.2
* @see Etudiant
*/
public class Enseignant extends Personne {
    ...
}
```

Exemple de documentation Javadoc

http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#examples

Comment générer la documentation Javadoc ?

Grâce au programme javadoc.exe fourni dans le Java SDK, et customisé via des Javadoc doclets :

- **NetBeans** : *cliquez-droit sur le projet* et choisissez Generate Javadoc. Par défaut, le doclet génère les fichiers de documentation dans le répertoire javadoc de votre répertoire utilisateur. Shift+F1 pour rechercher dans la documentation générée.
- **eclipse** : *cliquer sur le menu Projet*. Puis sélectionnez l'option "Générer la Javadoc".
- **IntelliJ** : *cliquer sur le menu Tools*. Puis sélectionnez l'option "Generate Javadoc...".

Comment générer la documentation Javadoc ?

Grâce au programme javadoc.exe fourni dans le Java SDK, et customisé via des Javadoc doclets :

- **NetBeans** : *cliquez-droit sur le projet* et choisissez Generate Javadoc. Par défaut, le doclet génère les fichiers de documentation dans le répertoire javadoc de votre répertoire utilisateur. Shift+F1 pour rechercher dans la documentation générée.
- **eclipse** : *cliquer sur le menu Projet*. Puis sélectionnez l'option "Générer la Javadoc".
- **IntelliJ** : *cliquer sur le menu Tools*. Puis sélectionnez l'option "Generate Javadoc...".

Comment générer la documentation Javadoc ?

Grâce au programme javadoc.exe fourni dans le Java SDK, et customisé via des Javadoc doclets :

- **NetBeans** : *cliquez-droit sur le projet* et choisissez Generate Javadoc. Par défaut, le doclet génère les fichiers de documentation dans le répertoire javadoc de votre répertoire utilisateur. Shift+F1 pour rechercher dans la documentation générée.
- **eclipse** : *cliquer sur le menu Projet*. Puis sélectionnez l'option "Générer la Javadoc".
- **IntelliJ** : *cliquer sur le menu Tools*. Puis sélectionnez l'option "Generate Javadoc...".

Comment générer la documentation Javadoc ?

Grâce au programme javadoc.exe fourni dans le Java SDK, et customisé via des Javadoc doclets :

- **NetBeans** : *cliquez-droit sur le projet* et choisissez Generate Javadoc. Par défaut, le doclet génère les fichiers de documentation dans le répertoire javadoc de votre répertoire utilisateur. Shift+F1 pour rechercher dans la documentation générée.
- **eclipse** : *cliquer sur le menu Projet*. Puis sélectionnez l'option "Générer la Javadoc".
- **IntelliJ** : *cliquer sur le menu Tools*. Puis sélectionnez l'option "Generate Javadoc...".