Le Langage Java

1re année

J. Beleho (bej) C. Leruste (clr) M. Codutti (mcd)
P. Bettens (pbt) F. Servais (srv) C. Leignel (clg)
D.P. Bishop (bis) S. Drobisz (sdr)

Haute École de Bruxelles-Brabant — École Supérieure d'Informatique

Année académique 2016 / 2017

Séance 4

Lisibilité et javadoc

- Écrire du code lisible
- Écrire du code illisible
- Refactorisation
- La documentation
- javadoc
- Les tags
- Étude de cas

« Any fool can write code that a computer can understand. Good programmers write code that humans can understand. »
Martin Fowler

« It's not a bug - it's an undocumented feature. »
Author Unknown

```
public class h{public static void
main(String[]
args){System.out.println("Hi");}}
```

Lisibilité du code

Un code est **souvent lu** ;

- lorsqu'il est écrit / mis au point ;
- correction de bug ;
- évolution du code ;

$\Longrightarrow$ **La lisibilité est essentielle**

Lisibilité du code : indentation

1

Indenter correctement son code

Lisibilité du code : choix des noms

2

Bien choisir le **nom des variables**

```
int u=clavier.nextInt(),n=clavier.nextInt(),
t=clavier.nextInt();
double p=u*n*(1+t/100.0);
System.out.println(p);
```

```java
package esi.java.cours;

import java.util.Scanner ;

public class CalculPrixVente {
    public static void main ( String[] args ) {
        Scanner clavier = new Scanner ( System.in ) ;
        double àPayer;
        int prixUnitaire = clavier.nextInt();
        int nombreArticles = clavier.nextInt();
        int tauxTVA = clavier.nextInt();

        àPayer = prixUnitaire * nombreArticles * (1 + tauxTVA/100.0);

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

Crédit photo

Lisibilité du code : décomposition

3
Décomposer les expressions trop longues

```
àPayer = prixUnitaireHTVA * (1 + tauxTVA/100.0) * nombreArticles;

prixUnitaireTTC = prixUnitaireHTVA * (1 + tauxTVA/100.0);
àPayer = prixUnitaireTTC * nombreArticles;
```

Lisibilité du code : constantes

4

Utiliser des **constantes**

```
final double TAUX_TVA = 0.21;
```

Illisibilité du code

-1

Surcharger de **commentaires**

Un commentaire explique ce que le code fait mais pas comment il le fait

D'autres conventions ?

Crédit photo

Conventions d'écriture

Conventions d'écriture Java

- http://www.oracle.com/...codeconv...

La refactorisation

Refactorisation

Improving the design of existing code

`JUnit` VCS (git/svn)

La documentation

Crédit photo

Motivation

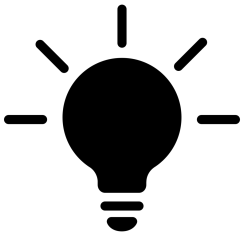
Pour qui ?
Qu'écrire ?

Motivation

Où mettre la documentation ?

- Avec le code
  - Plus facile pour le maintenir
  - Plus de chance de garder la synchronisation avec le code
- Mais le programmeur-utilisateur n'a pas à voir le code pour l'utiliser

Motivation



litterate programming

- la documentation accompagne le code
- un outil extrait cette documentation pour en faire un document facile à lire
- toute la documentation suit la même structure, le même style
  $\rightarrow$ plus facile à lire


code
javadoc
doc

Javadoc

javadoc

- Commentaire `javadoc` identifié par `/** ... */`

```
/**
     Calcule et retourne le maximum de 2 nombres.
*/
```

- La documentation est produite au format `HTML`
- On commente essentiellement
  - la classe : rôle et fonctionnement
  - les méthodes publiques : ce que ça fait, paramètres et résultats
- Se met **juste au dessus** de ce qui est commenté

Les tags

Utilisation de **tags** pour identifier certains éléments

Les plus courants :

- **@param** : décrit les paramètres
- **@return** : décrit ce qui est retourné
- **@throws** : spécifie les exceptions lancées
- **@author** : note sur l'auteur

Les tags

Exemple

```
/**
 * Donne la racine carrée d'un nombre.
 * @param nb le nombre dont on veut la racine carrée.
 * @return la racine carrée du nombre.
 * @throws IllegalArgumentException si le nombre est négatif.
*/
public static double sqrt( double nb ) {...}
```

- Les types sont déduits de la signature et ajoutés à la documentation
- La première phrase (terminée par un .) sert de résumé

Les tags

Résumé

Modifier and Type	Method and Description
`static double`	`sqrt(double nb)` Donne la racine carrée d'un nombre.

Détail

sqrt

```
public static double sqrt(double nb)
```

Donne la racine carrée d'un nombre.

Parameters:

`nb` - le nombre dont on veut la racine carrée.

Returns:

la racine carrée du nombre.

Throws:

`java.lang.IllegalArgumentException` - si le nombre est négatif.

Le code HTML

Peut contenir des balises HTML

Exemple :

```
/**
* Indique si l'année est bissextile. Pour rappel :
* <ul>
*   <li>Une année qui n'est pas divisible par 4 n'est pas bissextile
*       (ex: 2009)</li>
*   <li>Une année qui est divisible par 4</li>
*     <ul>
*     <li>est en général bissextile (ex: 2008)</li>
*     <li>sauf si c'est un multiple de 100 mais pas de 400 (ex: 1900, 2100)</li>
*     <li>les multiples de 400 sont donc bien bissextiles (ex: 2000, 2400)</li>
*     </ul>
*   </ul>
* Plus formellement, <code>a</code> est bissextile si et seulement si <br/>
* <code>a MOD 400 = 0 OU (a MOD 4 = 0 ET a MOD 100 != 0)</code>
* @param année l'année dont on se demande si elle est bissextile
* @return vrai si l'année est bissextile
*/
```

Le code HTML

Ce qui donne

estBissextile

```
public static boolean estBissextile(int année)
```

Indique si l'année est bissextile. Pour rappel :

- Une année qui n'est pas divisible par 4 n'est pas bissextile (ex: 2009)
- Une année qui est divisible par 4
  - est en général bissextile (ex: 2008)
  - sauf si c'est un multiple de 100 mais pas de 400 (ex: 1900, 2100)
  - les multiples de 400 sont donc bien bissextiles (ex: 2000, 2400)

Plus formellement, `a` est bissextile si et seulement si

```
a MOD 400 = 0 OU (a MOD 4 = 0 ET a MOD 100 != 0)
```

Parameters:

`année` - l'année dont on se demande si elle est bissextile

Returns:

vrai si l'année est bissextile

Production de la documentation

Commande javadoc

- javadoc Temps.java
- javadoc *.java
- javadoc -d doc *.java
- javadoc -charset utf-8 *.java
- ...et beaucoup d'autres options (cf. documentation de javadoc)

Une bonne documentation

Une bonne *javadoc* décrit le **quoi** mais jamais le **comment**

- $\longrightarrow$ Ne jamais parler de ce qui est privé
- Mauvais exemples :
  - *On utilise un for pour parcourir le tableau.*
  - *Pour aller plus vite, on stocke le prix hors tva dans une variable temporaire.*

Une bonne documentation

Ne pas écrire ce que `javadoc` écrit lui-même :

- Mauvais exemples :
  - *nb - un entier qui ...*
  - *La méthode sqrt ...*
  - *Cette méthode ne retourne rien.*
- Pour en savoir plus :

`http://www.oracle.com/technetwork/articles/java/index-137868.html`

Étude de cas

Crédits

Ces slides sont le support pour la présentation orale de l'unité d'enseignement **DEV1-JAV** à HE2B-ÉSI

Crédits

Les distributions Ubuntu et/ou debian
du système d'exploitation GNU Linux.
LaTeX/Beamer comme système d'édition.
Git et GitHub pour la gestion des versions et le suivi.
GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project