CSC4102 : Qualité du code JAVA et introduction aux idiomes JAVA

Denis Conan

Janvier 2022
1. Motivations et objectifs
2. Outils d’analyse statique de code = Pour aider !
3. Définition idiome/patron d’implémentation
4. Idiomes JAVA, premiers exemples, méthodes de la classe Object communes à tous les objets
5. Mise en pratique en TP (2h) + HP (3h)
1 Motivations et objectifs

1.1 Qui demande à améliorer la qualité du code ?
1.2 Pourquoi les idiomes ?
1.3 Quels sont les objectifs de la séance ?
1.4 Après la séance — À quoi ça sert ? où ? quand ?
1.1 Qui demande à améliorer la qualité du code ?

« *The Google FindBugs Fixit* » [Ayewah and Pugh, 2010]

In May 2009, Google conducted a company wide *FindBugs* “fixit”\(^1\). Hundreds of engineers reviewed thousands of *FindBugs* warnings.

- **Static analysis tools [such as *FindBugs*] scan software looking for issues that might cause defective behavior.**

- **Most interesting software quality problems were eventually found and fixed without *FindBugs*, but *FindBugs* could have found these problems early, when they are cheap to remediate.**

  - These observations [are attributed] to the success of Google’s testing and monitoring practices.

- **Engineers viewed *FindBugs* warnings as worth fixing.**

  - Over 77% of the reviews identified the warnings as real defects and recommended fixing the underlying problems.

---

1. C’est ce que l’on appelle une revue de code, ici avec *FindBugs*, qui a été repris dans *SpotBugs* [Ayewah et al., 2008]
1.2 Pourquoi les idiomes ?

G.L. Steele, ACM Grace Murray Hopper Award, 1988
Foreword of « Effective JAVA » [Bloch, 2008]

■ If you ever studied a second language yourself and then tried to use it outside the classroom, you know that there are three things you must master :

1. How the language is structured (grammar)
2. How to manage things you want to talk about (vocabulary)
3. The customary and effective ways to say everyday things (usage)

■ It is much the same with a programming language.

1. You need to understand the core language [UML, JAVA = object-orientation]
2. You need to know the vocabulary [e.g. JAVA standard library for collections]
3. You need to be familiar with the customary and effective ways to structure your code

— C’est ce qui s’appelle les patrons d’implémentation ou idiomes
1.3 Quels sont les objectifs de la séance ?

- Étudier la qualité du code produit dans le Sprint 1
  - On paie de la dette technique
  - Avec les outils SpotBugs et CheckStyle
    - On atténue les efforts nécessaires à l’aide d’outil d’analyse statique du code

- Étudier l’application d’idiomes JAVA
  - On améliore la qualité du code en améliorant nos connaissances sur le langage
  - À titre d’exemple, sur les méthodes communes à toutes les classes : equals, hashCode, toString, etc.

- Continuer le développement de l’application de l’étude de cas
  - On annonce dès maintenant les nouveaux cas d’utilisation pour l’ensemble du Sprint 2
1.4 Après la séance — À quoi ça sert ? où ? quand ?

- Périodiquement, utiliser les outils d’analyse statique de code dans Eclipse
- Maintenir la qualité (passage des tests) avec l’intégration continue

[perfect.] Pour ceux qui veulent aller plus loin, possibilité de générer un site Web Maven, qui présente les rapports de qualité

[perfect.] Pour les « professionnels », possibilité de faire échouer la construction du logiciel en cas de problèmes détectés avec les outils CheckStyle et SpotBugs

- Dans ce module, nous n’irons pas jusque là
2 Outils d’analyse statique de code = Pour aider !

2.1 Analyse statique de code
2.2 SpotBugs
2.3 CheckStyle
2.1 Analyse statique de code

- Texte
  - Lexicale
  - Tokens
  - Syntaxe
  - AST
  - Sémantique

- Édition de liens
  - Optimisation, Allocation mémoire/trgistre

- MIPS
  - Génération de code

- Représentation Intermédiaire
  - intermediate.IR

- Intermédiaire

- Arbre Sémantique = AST + Attributs + Table Symbole
  - java_cup.runtime.Symbol
  - syntax.ast.ASTNode

1. Image extraite du cours CSC4536 : http://www-inf.int-evry.fr/COURS/CSC4536/web/
2.2 SpotBugs

Catégories (https://spotbugs.github.io/)

- **Correctness bug** : Probable bug, an apparent coding mistake resulting in code that was probably not what the developer intended
  - We strive for a low false positive rate

- **Bad Practice** : Violations of recommended and essential coding practice
  - Examples include hash code and equals problems, cloneable idiom, dropped exceptions, serializable problems, and misuse of finalize
  - We strive to make this analysis accurate, although some groups may not care about some of the bad practices

- **Dodgy** (douteux, risqué) : Code that is confusing, anomalous, or written in a way that leads itself to errors
  - Examples include dead local stores, switch fall through, unconfirmed casts, and redundant null check of value known to be null
  - More false positives accepted

1. **SpotBugs** doesn't try to identify all defects in a particular category
   [Ayewah et al., 2008]
### 2.2.1 Exemples, a/e

1. \( X \) \( x_2 = \text{new } X_2(...) \);  
2. \( h_2.put(x_2, \text{"c\'est un objet"}; \)

- « X2 doesn’t define a hashCode() method but is used in a hashed data structure in main(String[]) »

- « A class defines an equals(Object) method but not a hashCode() method, and thus doesn’t fulfill the requirement that equal objects have equal hashCodes. An instance of this class is used in a hash data structure, making the need to fix this problem of highest importance. »

---

Le problème vient de ce que la classe possède une méthode equals mais pas de méthode hashCode

- Rappelons-nous qu’un dictionnaire (ajout avec méthode put utilise hashCode puis equals pour chercher un objet
2.2.2 Exemples, b/e

```java
pi = null;
for(it= vPersonnes.iterator();it.hasNext();pi=it.next()) {
    if(pi.equals(new Personne("Dupont"+5, "Jules","19102271271"+5))) {
        it.remove();
    }
}
```

■ « Déréférencement d'un pointeur null dans la méthode main(String[]) »

■ « Un pointeur à null est déréférencé ici. Ceci va mener à une NullPointerException quand le code sera exécuté. »

Le problème est à la ligne 2 : l’affectation « pi = it.next() » n’a lieu qu’en fin de boucle; donc, lors de l’entrée dans la boucle for, pi vaut sa valeur avant la boucle, c’est-à-dire ici null
2.2.3 Exemples, c/e

```java
if (p.getPrenom() == p1.getPrenom()) {
// ...
```

- « Comparaison d’objets String utilisant == ou !=. »
- « Ce code compare des objets String au moyen de l’égalité par référence des opérateurs == ou !=. À moins que les deux chaînes ne soient des constantes dans le fichier source ou aient été internalisées au moyen de la méthode `String.intern()`, deux chaînes identiques peuvent être représentées par deux objets String différents. Envisagez d’utiliser la méthode `equals(Object)` à la place. »

- **SpotBugs** signale l’utilisation de l’opérateur « == » à la place de la méthode « equals »
- C’est sans aucun doute une erreur pour des variables qui sont des références sur des objets
2.2.4 Exemples, d/e

1 Calendar o2 = o1;
2 System.out.println("o1 == o2: " + (o1 == o2) + ", o1.equals(o2): " + (o1.equals(o2)));

- This method compares a local variable with itself, and may indicate a typo or a logic error. Make sure that you are comparing the right things.
- L’alerte vient de la combinaison de l’affectation de la ligne 1 avec l’appel de la méthode equals à la ligne 2
2.2.5 Exemples, e/e

```java
public Audio(final String code, final Localisation localisation,
              final String titre, final String auteur, final String annee,
              final Genre genre, final String classif) {
    super(code, localisation, titre, auteur, annee, genre);
    if (classif == null) {
        throw new IllegalArgumentException("Ctr\Audio\classification\=\" +
                                             classification");
    }
    this.classification = classif;
}
```

« Lecture du champ Audio.classification non initialisé dans new Audio(...). »

« Ce constructeur lit un champ qui n’a pas encore été initialisé. Une des causes les plus fréquentes est l’utilisation accidentelle par le développeur du champ au lieu d’un des paramètres du constructeur. »

Le problème est à la ligne 6 : on voulait sans aucun doute utiliser l’argument classif au lieu de l’attribut classification

- Mais, dans ce cas, on sait que c’est null qui sera affiché
2.3 CheckStyle

- Comme son nom le suggère, adhésion à un standard de codage
  - Deux fournis par défaut dans Eclipse
    - Sun Checks et Google checks
- Pour nous, l'objectif « numéro 1 » est la documentation Javadoc
  - Configuration particulière de CheckStyle fournie en TP
    - Feuille de style construite à partir de Sun Checks
      - Nous autorisons les tabulations, les caractères « _ » dans les noms de paquetages, les lignes à plus de 80 colonnes, etc.
      - Nous ne développons pas une bibliothèque
        - Uniquement pour quelques classes
  - Génération de la documentation dans Eclipse (menu Project/Generate Javadoc)
    - Possibilité pour ceux qui le souhaitent de générer avec MAVEN (mvn site)

2.3.1 Documentation Javadoc

Classe eu.telecomsudparis.csc4102.util.Datutil

```java
/**
 * compare la date avec aujourd’hui : vrai si aujourd’hui.
 * @param date la date à comparer avec le jour courant.
 * @return vrai si aujourd’hui
 * @throws IllegalArgumentException si @code date est @code null
 */
public static boolean dateEstAujourdhui(final LocalDate date) {
    if (date == null) {
        throw new IllegalArgumentException("date == null");
    }
    return date.equals(aujourdhui());
}
```

http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[Javadoc, JSE8]

Notez que les commentaires Javadoc commencent par la chaîne de caractères « /** »
Vérification de la présence et correction « du Javadoc » avec CheckStyle

```java
/**
 * Bla bla.
 * @param e explication sur e.
 */
public void ajouter(final int a, final int b) { }
```

- « Balise javadoc @param inutilisé pour ‘e’. »
- « Balise javadoc @param manquante pour ‘a’. »
- « Balise javadoc @param manquante pour ‘b’. »
2.3.2 Autres exemples I

- Aussi présenté dans [Bloch, 2008], Item 8

```java
1  private int Valeur;
2  public void MaMethode() { //...
```

- « Le nom ‘Valeur’ n’est pas conforme à l’expression ‘^[a-z][a-zA-Z0-9]*$’. »
- « Le nom ‘MaMethode’ n’est pas conforme à l’expression ‘^[a-z][a-zA-Z0-9]*$’. »

- **Standard d’écriture lexicale du code JAVA**

<table>
<thead>
<tr>
<th>Paquetage</th>
<th>seance5.mediathequesimplifie, eu.telecomsudparis.csc4102</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classe, interface</td>
<td>Timer, HashMap, HttpServlet</td>
</tr>
<tr>
<td>Méthode, attribut</td>
<td>remove, ensureCapacity, getVote</td>
</tr>
<tr>
<td>Constantes</td>
<td>MIN_VALUE, NEGATIVE_INFINITY</td>
</tr>
<tr>
<td>Variable locale</td>
<td>i, somme, sommeVote</td>
</tr>
<tr>
<td>Paramètre de type</td>
<td>T, V, K, T1</td>
</tr>
</tbody>
</table>
2.3.2 Autres exemples II

1 public int b;

- « La variable ‘b’ devrait être privée et avoir des accesseurs. »

1 public String toString() { return "A\[Valeur=" + valeur + "]"; }

- « overrides java.lang.Object.toString »
  - Ajouter @Override, sinon on risque une erreur difficile à trouver
    - P. ex. si on écrit
      « public String toString(boolean versionCourte) {...} »

- Autre exemple plus difficile à trouver pour le corriger :
  « public boolean equals(Document obj) {...} »
2.3.2 Autres exemples III

```java
public int ajouter(int a, int b) {
    return a + b;
}
```

- « Le paramètre a devrait être final. »
- « Le paramètre b devrait être final. »
- Pour se prémunir du changement de la valeur des arguments en oubliant que ces changements sont oubliés en fin de méthode
  - On écrira « public int ajouter(final int a, final int b)… »
3 Définition idiome/patron d’implémentation

■ Une des premières occurrences du terme

■ « Idioms — Because collections are so powerful, there are a small set of standard tricks that experienced Smalltalkers know to play with them. If you are reading code that uses one of these idioms, you may be puzzled at first. This section introduces the problems you can solve using collections in unusual ways. » [Beck, 1995]

■ « By idioms, we mean some (often unusual) combination of abstractions specific to some programming language, which are used for implementing some other abstraction that is not part of this language. » [Czarnecki and Eisenecker, 2000]

■ Notre position dans ce module : les idiomes permettent d’approfondir des concepts de base importants


■ Dans CSC4102, à titre d’exemple, parcours de quelques idiomes

— Méthodes de la classe Object communes à toutes les classes
— Plus dans le document « pour aller plus loin » de cette séance
4 Idiomes JAVA, premiers exemples, méthodes de la classe `Object` communes à tous les objets

4.1 Méthode `equals`
4.2 Méthode `hashCode`
4.3 Méthode `toString`
4.4 Les autres méthodes de la classe `Object`
4.1 Méthode equals

- Repris de l’Item 8 de [Bloch, 2018]

- Contrat² [Class Object, JSE8]
  - Reflexive : for any non-null reference value $x$, $x.equals(x)$ should return true
  - Symmetric : for any non-null reference values $x$ and $y$, $x.equals(y)$ should return true if and only if $y.equals(x)$ returns true
  - Transitive : for any non-null reference values $x$, $y$, and $z$, if $x.equals(y)$ returns true and $y.equals(z)$ returns true, then $x.equals(z)$ should return true
  - Consistent : for any non-null reference values $x$ and $y$, multiple invocations of $x.equals(y)$ consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified
  - For any non-null reference value $x$, $x.equals(null)$ should return false

- Cf. « pour aller plus loin » avec des exemples de non-respect des propriétés

- Conclusion : Utilisez l’assistant Eclipse pour générer les méthodes equals + Choisir les attributs définissant de manière unique un objet de la classe

2. Prend en argument un Object : utiliser @Override pour éviter une erreur
4.2 Méthode `hashCode`

- Repris de l’Item 9 de [Bloch, 2018]

- Redéfinir `equals` impose de redéfinir aussi `hashCode` :
  \[ \forall o_1, o_2 \in Classe : o_1.equals(o_2) \implies o_1.hashCode() = o_2.hashCode() \]

```java
class CleErronee {
    private String nom, prenom;
    public CleErronee(final String n, final String p) {nom=n;prenom=p;}
    @Override public boolean equals(final Object obj){//...avec nom,prenom
        // pas de méthode `hashCode`
    }
}
```

```java
Map<CleErronee,Client> d=new HashMap<CleErronee,Client>();d.put(c1, c);
```

- Résultat de l’exécution :
  
  ```
  366712642!=1829164700, null
  ```

- As much as is reasonably practical, the `hashCode` method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer.) [Class Object, JSE8]
4.3 Méthode `toString`

- Repris de l'`Item 10` de [Bloch, 2018]
- Toujours redéfinir `toString`
  - Par défaut, nom de la classe + @ + représentation hexadécimale non signée de la valeur donnée par `hashCode()`

Classe `seance7.methodescommunesatouslesobjets.EnfantAvecToString`

```java
1 public class EnfantAvecToString extends ParentAvecToString {
2     private int b;
3     public EnfantAvecToString(final int a, final int b) {super(a);this.b = b;}
4     @Override public String toString() {
5         return "EnfantAvecToString[b=\"+b+\", toString()=\"+super.toString()\"]\";}
}
```

Classe `seance7.methodescommunesatouslesobjets.ExemplesToString`

```java
1     EnfantSansToString e = new EnfantSansToString(1,2);
2     System.out.println(e + ",\" + Long.toHexString(e.hashCode()) +\"\";
3     EnfantAvecToString e2 = new EnfantAvecToString(1,2);
4     System.out.println(e2);
```

- Résultat de l'exécution :

```
EnfantSansToString@15db9742, 15db9742
EnfantAvecToString [b=2, toString()=ParentAvecToString [a=1]]
```
4.3.1 Assoc. bidirectionnelles et toString

Classe seance7.methodescommunesatouslesobjets.associationbidirectionnelle.Ensemble

```java
private List<Element> elems;

public Ensemble(final int v) { elems = new ArrayList<Element>(); }

public void ajouterUnA(final int v) { elems.add(new Element(this, v)); }

@Override // redefinition : contenu toString() de elems
public String toString() { return "Ensemble\n[elems=\n\n]"; }
```

Classe seance7.methodescommunesatouslesobjets.associationbidirectionnelle.Element

```java
private int v;
private Ensemble ens;

public Element(final Ensemble ens, final int v) { this.v=v; this.ens=ens; }

@Override // redefinition : contenu de v et toString() de ens
public String toString() { return "Element\n[v=\n, \nens=\n]"; }
```

Classe seance7.methodescommunesatouslesobjets.associationbidirectionnelle.ExempleBoucleInfinie

```java
Ensemble b = new Ensemble(1); b.ajouterUnA(2); System.out.println(b);
```

- Résultat de l'exécution :

```
Exception in thread "main" java.lang.StackOverflowError
```

- Il faut éviter la formation du cycle d'appels

- Ensemble::toString doit utiliser des getters de Element au lieu d'utiliser Element::toString
### 4.3.2 Énumérations avec `toString` *

**Classe `seance7.methodescommunesatouslesobjets.EnumerationSansRedefinitionDeToString`**

```java
public enum EnumerationSansRedefinitionDeToString {
    Premier, Deuxieme;
}
```

**Classe `seance7.methodescommunesatouslesobjets.EnumerationAvecRedefinitionDeToString`**

```java
public enum EnumerationAvecRedefinitionDeToString {
    Premier("Mon premier"), Deuxieme("Mon second");
    private String libelle;
    // notez que le constructeur est privé
    private EnumerationAvecRedefinitionDeToString(final String libelle) {
        this.libelle = libelle; }
    @Override // redefinition : le libelle, plutôt que le nom de l'objet
    public String toString() { return libelle; } }
```

**Classe `seance7.methodescommunesatouslesobjets.ExemplesToStringEnumeration`**

```java
System.out.println(EnumerationSansRedefinitionDeToString.Deuxieme);
System.out.println(EnumerationAvecRedefinitionDeToString.Deuxieme);
```

---

**Résultat de l’exécution :**

```
Deuxieme  
Mon second
```

---

2. Le constructeur de `EnumerationAvecRedefinitionDeToString` est `private` car il n’existe aucune raison de le mettre `public` puisqu’aucun nouvel objet n’est instanciable depuis l’extérieur.
4.4 Les autres méthodes de la classe Object

[Class Object, JSE8]

- **getClass**: _Returns the runtime class of this object_

- **Méthodes pour la concurrence** :
  - S’utilise avec la directive Synchronized (hors programme CSC4102)
  - **notify**: _wakes up a single thread that is waiting on this object’s monitor_
    - _If any threads are waiting on this object, one of them is chosen to be awakened_
    - _The choice is arbitrary and occurs at the discretion of the implementation_
  - **notifyAll**: _wakes up all threads that are waiting on this object’s monitor_
  - **wait**: _causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed_
5 Mise en pratique en TP (2h) + HP (3h)

- Qualité du code avec **SpotBugs** puis **CheckStyle**
- Idiomes : programmation des méthodes communes à tous les objets
- Continuation du développement de l’application de l’étude de cas
- Rendu de la séance en HP (3h) : PDF + JAVA dans « sprint2 »
- Compléments « **Pour aller plus loin** » sur les idiomes JAVA
The Google FindBugs Fixit.

Using Static Analysis to Find Bugs.

*Smalltalk Best Practice Patterns Volume 1 : Coding*.
Prentice Hall.

*Effective Java, 2nd Edition*.
Addison-Wesley.

*Effective Java, 3rd Edition*.
Addison-Wesley.
Références II

Class Object (JSE8).
Javadoc of the class Object of JAVA SE 8.

Generative Programming : Methods, Tools, and Applications.
Addison-Wesley.

Javadoc (JSE8).
Javadoc Tool.