



06

Chapitre

Code propre

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« Codez toujours comme si le type qui sera chargé de maintenir votre code est un violent psychopathe qui sait où vous habitez. »

John Woods

Plan du chapitre

1

Pourquoi faire
propre ?

Question

- Qu'est ce qu'un code propre ?
 - Que vous a t-on enseigné pour faire du code propre ?

Code propre

- Exemple
 - Calcul des 100 premiers nombres premiers
 - Que pensez-vous de ce bout de code ?
 - Que pose t-il comme problème ?

Plan du chapitre

1

Pourquoi faire
propre ?

2

Pourquoi
la pratique
classique
est fausse ?

**Les commentaires sont néfastes au
code**

Les commentaires mentent

- Le code évolue toujours plus vite que les commentaires

```
// returns null if register doesn't exist
public void registerItem( Item item ) throws NoRegistryException {
    /* ... */
}
```

Les commentaires n'apportent aucune information supplémentaire

- Commentaires qui parodient le code : inutiles et lourds

```
// Default constructor
protected AnnualDateRule() {
}

/** The day of the month */
private int dayOfMonth;

/** @return the day of the month */
public int getDayOfMonth() {
    return dayOfMonth;
}
```


Les commentaires conduisent à une incompréhension

- Commentaires bâclés : conduisent à une incompréhension

```
/*  
 * Returns whether the light is on.  
 */  
bool getLightStatus( Light light) {  
    if (!light.isOn()) {  
        resetLight(light);  
    }  
    return light.isOn();  
}
```

**Les commentaires rendent le code
illisible**

Les commentaires bruite le code

- Commentaires de fin de bloc mélangent information et repère

```
public static int main( String args[] ) {
    try {
        while ((line = in.readLine()) != null) {
            lineCount++;
            charCount += line.length();
            String words[] = line.split("\\W");
            wordCount += words.length;
        } //while
        System.out.println("wordCount = " + wordCount);
        System.out.println("lineCount = " + lineCount);
        System.out.println("charCount = " + charCount);
    } // try
    catch (IOException e) {
        System.err.println("Error:" + e.getMessage());
    } //catch
} //main
```


**Les commentaires décrédibilisent
le développeur**

Les fautes d'orthographe

```
int i; /* Conter variable for "for" loop. */
int t; /* Total of additions for calculaton */
int d; /* Individual number for calclatuion */
/* "for" loop */
for (i=0; i<100; i++) { /* increment i by one until hunderd */
    d = f(); /* get the calue for d */
    t = t + d; /* ad it to t */
}
```

Les copier / coller malheureux

- Les copier/coller malheureux.

```
/* The version. */  
private String version;  
/* The licenceName. */  
private String licenceName;  
/* The version. */  
private String info;
```

Bilan

Bilan

■ Constat

- Les commentaires introduisent de la confusion ou pire des mensonges
- Les commentaires créent du bruit dans le code
- Les commentaires nuisent à la lisibilité du code

■ Conclusion

- Il faut supprimer les commentaires et la documentation autant que faire se peut
- Il faut que le code soit sa propre documentation

Attention : Cas particulier des API

- Dans ce cas, la documentaire (type Javadoc) est **obligatoire**

```
/**
 * Computes the matching between the reference regions
 * and the segmentation output regions.
 * @param segmentation the output region map.
 * @param reference the reference region map.
 * @result the region map with the best matching.
 */
RegionMap *matching( RegionMap *segmentation, RegionMap *reference ) {
    ...
}
```

- Mais, il faut consacrer du temps
- **Remarque** : Ces API sont plutôt stables par essence. Cette documentation ne ment pas

Plan du chapitre

1

Pourquoi faire
propre ?

2

Pourquoi
la pratique
classique
est fausse ?

3

Comment faire
propre ?

Nouvelle pratique du codage : code propre

20

- Le code est la seule chose qui soit maintenue
 - Il faut supprimer les commentaires autant que faire se peut
- Mais, supprimer les commentaires et la documentation a des contreparties :
 - Il faut repenser l'écriture du code
 - Il faut dépenser du temps et de l'énergie au moment de l'écriture du code
 - « *N'importe quel programmeur peut écrire du code que l'ordinateur comprend. Les bons programmeurs écrivent du code que les humains peuvent comprendre.* » Martin Fowler.
- Présentation d'une pratique (méthode eXtreme Programming)

Nouvelle pratique du codage : code propre

21

- Supprimer les commentaires et la documentation a des contreparties :
 - Il faut repenser l'écriture du code
 - Il faut dépenser du temps et de l'énergie au moment de l'écriture du code
 - « *N'importe quel programmeur peut écrire du code que l'ordinateur comprend. Les bons programmeurs écrivent du code que les humains peuvent comprendre.* » Martin Fowler.
- Présentation d'une pratique (méthode eXtreme Programming)

**Tous les commentaires ne
sont pas à supprimer**

Règle n°1

**→ Utiliser judicieusement les
commentaires**

Commentaires indispensables

1) Compléments d'information sur des instructions non auto-documentables

```
// format matched hh:mm:ss GMT, MMM dd, yyyy  
Pattern timeMatcher = Pattern.compile(  
    "\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

Commentaires acceptables

2/ Messages entre développeurs

```
// Warning: this method is kept for the sake of compatibility
@deprecated
void getMaximum( ArrayList<Cell> cells ) {
    ...
}
```

```
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 39
//
```


Commentaires acceptables

- TODO / FIXME
 - Commentaire provisoire

```
// TODO Change the sort algorithm to heap sort algorithm
public void sort( Ordonable list ) {
    ...
}
```

Règle n°2

→ Utiliser avantageusement les noms d'identificateurs

Choisir des noms explicites

- Les noms sont partout
 - Identificateurs, projet, fichiers, dossiers...
 - Il faut les rendre explicites
 - Ce sont les premiers commentaires d'un programme

```
int d; // elapsed time in days
```



```
int elapsedTimeInDays;
```

Nommer selon l'intention

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<>();
    for (int[] x : _theList) {
        if (x[0] == 4) {
            list1.add(x);
        }
    }
    return list1;
}
```



```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<>();
    for (Cell cell : _gameBoard) {
        if (cell.isFlagged()) {
            flaggedCells.add(cell);
        }
    }
    return flaggedCells;
}
```

Remplacer un commentaire par une variable

```
// Does the module from the global list <module> depend on  
// the subsystem we are part of?  
if (smodule.getDependSubsystems().contains(module.getSubSystem()))
```



```
List<Module> moduleDependees = smodule.getDependSubsystems();  
String ourSubSystem = module.getSubSystem();  
if (moduleDependees.contains(ourSubSystem))
```

Remplacer un commentaire par une méthode

```
// Check to see if the employee is eligible for full benefits
if ((_employee.flags & HOURLY_FLAG) && _employee.age > 65) {
    /* ... */
}
```



```
private final boolean isEligibleForFullBenefits() {
    return (_employee.flags & HOURLY_FLAG) && _employee.age > 65;
}

if (isEligibleForFullBenefits()) {
    /* ... */
}
```

Ne pas en faire trop

```
for (int mapIndex = 1; mapIndex < INDEX_SIZE; ++mapIndex) {  
    setMapIndex(mapIndex);  
}
```



```
for (int i = 1; i < INDEX_SIZE; ++i) {  
    setMapIndex(i);  
}
```