

## Activité 3 C306 - Ingénierie du logiciel

### 1. Première implémentation

#### 1. Donner une copie d'écran des rapports fournis par Jenkins (checkstyle, spotbugs, PMD, couverture de code) sur ce qui a été produit à l'activité 2.

Étant donné que j'ai réalisé le solveur de Sudoku avant de mettre en place Jenkins, les rapports que j'ai sont uniquement ceux de l'activité 3.

Les rapports spotbugs et checkstyle de l'activité 2 ne dévoilaient aucune erreur, comme on peut le voir dans mon rapport sur cette activité. Concernant PMD, il restait une erreur de niveau 2 dû au fait de l'utilisation d'un `System.out.println` :

#### PMD Results

The following document contains the results of PMD 6.38.0.

#### Violations By Priority

##### Priority 2

org/unc/nc/App.java

Rule	Violation	Line
SystemPrintln	System.out.println is used	13

J'ai donc modifié mon pom.xml pour ne plus accepter que l'affichage des erreurs de priorité 1. Et comme il n'y en a aucune, le rapport PMD ne relevait donc plus aucune erreur.

```
<targetJdk>11</targetJdk>
<includeTests>true</includeTests>
<verbose>true</verbose>
<showPmdLog>true</showPmdLog>
<printFailingErrors>true</printFailingErrors>
<failOnViolation>false</failOnViolation>
<skip>false</skip>
<minimumPriority>1</minimumPriority>
<skipEmptyReport>false</skipEmptyReport>
```

## 2. Ecrire l'interface pour un 'solveur' de grille de sudoku défini dans l'activité précédente.

J'ai écrit une interface 'Resolveur' que j'ai ensuite implémenté dans ma classe 'GrilleImpl' :

```
/**
 * Interface de résolution du sudoku.
 */
public interface Resolveur {

    /**
     * Méthode de résolution du sudoku.
     *
     * @return true si le sudoku est complété
     * @throws HorsBornesException erreur bornes
     * @throws ValeurImpossibleException erreur valeur
     * @throws CaractereInterditException erreur caractère
     */
    boolean solve() throws HorsBornesException,
        ValeurImpossibleException, CaractereInterditException;
}
```

## 3. Ecrire les tests correspondant aux méthodes définies dans votre interface.

J'ai écrit des tests avec les grilles de dimension 9 et 16 fournies dans le devoir. Je compare une grille complétée à partir d'un fichier txt d'une grille de sudoku déjà complété avec une grille qui elle n'est pas complétée à partir d'un des fichier txt incomplet avant de lancer la fonction de résolution, et si les deux grilles sont identiques à la fin, le test est considéré comme validé :

```
@Test
private void grille9solve() throws CaractereInterditException, IOException,
HorsBornesException, ValeurImpossibleException {
    GrilleImpl grille9 = new GrilleImpl(9);
    String path = "src/test/resources/sudoku-9x9.txt";
    File file = new File(path);
    grille9 = GrilleParserUtils.parse(file, grille9);

    GrilleImpl grille9solved = new GrilleImpl(9);
    String path2 = "src/test/resources/sudoku-9x9-resolu.txt";
    File file2 = new File(path2);
    grille9solved = GrilleParserUtils.parse(file2, grille9solved);

    grille9.solve();

    assertEquals(Arrays.deepToString(grille9.grille),
Arrays.deepToString(grille9solved.grille));
}
```

#### 4. Ecrire une implémentation simple de 'résolveur' (une version récursive utilisant un parcours en profondeur s'écrit en quelques lignes).

Voici l'implémentation de l'interface 'Résolveur' dans 'GrilleImpl', j'utilise une solution de backtracking, où on se sert du principe de récurrence où la méthode se rappelle elle-même jusqu'à ce que les conditions de résolution du sudoku soient satisfaites :

```
/**
 * Méthode de résolution de la grille de Sudoku par backtracking.
 *
 * @return boolean indiquant que la grille est résolue
 * @throws HorsBornesException      erreur de bornes
 * @throws ValeurImpossibleException erreur de valeur
 * @throws CaractereInterditException erreur de caractère
 */
public boolean solve() throws
    HorsBornesException, ValeurImpossibleException,
    CaractereInterditException {
    for (int row = 0; row < this.getDimension(); row++) {
        for (int column = 0; column < this.getDimension(); column++) {
            if (this.grille[row][column] == Grille.EMPTY) {
                for (char caracteresPossible : this.caracteresPossibles) {
                    if (checkAll(row, column, caracteresPossible)) {
                        this.setValue(row, column, caracteresPossible);
                        if (solve()) {
                            return true;
                        } else {
                            this.setValue(row, column, Grille.EMPTY);
                        }
                    }
                }
            }
        }
        return false;
    }
}
return true; // sudoku résolu
}
```

5. Donner une copie d'écran des rapports fournis par Jenkins (checkstyle, SpotBugs, PMD, tests et couverture), il ne doit pas rester d'erreurs et la couverture doit être complète.

PMD : Il remonte des erreurs quant à l'utilisation de FileInputStream, qui est utilisé dans le Parser fourni dans le devoir. Donc je me vois mal de le corriger.

## PMD Results

The following document contains the results of [PMD](#) 6.38.0.

## Violations By Priority

### Priority 1

[org/unc/nc/GrilleParserUtils.java](#)

Rule	Violation	Line
<a href="#">AvoidFileStream</a>	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter	69

SpotBugs : il sort un avertissement quant à la méthode 'possible' dans l'interface 'Grille'. Comme ce fichier était lui aussi fourni, je n'ai pas cherché à régler ce 'bug'.

## SpotBugs Bug Detector Report

The following document contains the results of [SpotBugs](#).

SpotBugs Version is 4.2.2

Threshold is *medium*

Effort is *default*

## Summary

Classes	Bugs	Errors	Missing Classes
9	1	0	0

## Files


Class	Bugs
<a href="#">org.unc.nc.Grille</a>	1

[org.unc.nc.Grille](#)

Bug	Category	Details	Line	Priority
org.unc.nc.Grille.POSSIBLE devrait être sorti de l'interface et mis en package protected	MALICIOUS_CODE	<a href="#">MS_OOI_PKGPROTECT</a>	23	Medium

Checkstyle : Aucun problème

## Checkstyle Results


The following document contains the results of [Checkstyle](#) 8.29 with google\_checks.xml ruleset. 

## Summary

Files	Info	Warnings	Errors
9	0	0	0

Concernant la couverture du code, j'ai sciemment mis les tests utilisant le Parser en private (donc non testés durant le build) dû à l'interprétation que fait Jenkins avec les fichiers txt. Les fichiers fournis sont en LF (Line Feed) qui indique un saut de ligne propre à ce que l'on rencontre dans les fichiers UNIX / Linux. De fait, le Parser est conçu en s'attendant à ce type de fichier, et il fonctionne correctement sur ma machine et les fichiers sont bien enregistrés comme tel une fois poussés sur GitHub. Mais pour une raison que j'ignore, Jenkins va les convertir en CR+LF (Carriage Return Line Feed) qui est la méthode de saut de ligne tel qu'on la retrouve sur Windows. Et comme mon Jenkins est installé sur une machine tournant sous Windows, je pense que c'est cela qui provoque la conversion à la volée lors du build. Si mon Jenkins était sur une machine Linux je ne pense pas qu'il y aurait ce problème. Mais comme on peut le constater, sans ces deux tests utilisant le Parser, le build passe :

## Construction #6 (4 déc. 2021 à 12:57:...

 [Ajouter une description](#)



Changes

1. ([details](#) / [githubweb](#))



[Started by GitHub push by EWEN14](#)



**Revision:** d78c46e1049f98238f19a166e4dd994c70998a5a

**Repository:** <https://github.com/EWEN14/Activites-Genie-Logiciel>

- [refs/remotes/origin/master](#)



[Résultats des tests](#) (aucune erreur)

## Construction du module

 [active-deux](#) 2 mn 11 s

Sinon, en termes de couverture, les tests couvrent l'ensemble de la classe 'GrilleImpl' :

62% classes, 77% lines covered in package 'org.unc.nc'

Element	Class, %	Method, %	Line, %	Branch, %
exceptions	100% (3/3)	100% (3/3)	100% (3/3)	100% (0/0)
App	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
CalculUtils	0% (0/1)	0% (0/3)	0% (0/8)	0% (0/2)
Grille	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
GrilleImpl	100% (1/1)	100% (13/13)	100% (80/80)	100% (46/46)
GrilleParserUtils	0% (0/1)	0% (0/2)	0% (0/15)	0% (0/4)

Ainsi que presque l'intégralité du Parser, sauf l'Exception de lecture qui n'est jamais levée et qui normalement ne peut pas l'être du fait que le reader est limité à la dimension donnée.

GrilleParserUtils.java × GrilleImpl.java × Resolveur.java × Grille.java × Coverage: GrilleImplTest ×

```

30 @ public static GrilleImpl parse(InputStream in,
31     CaractereInterditException, HorsBornesException,
32
33     Reader reader = new InputStreamReader(in, StandardCharsets.UTF_8);
34     int dimension = grille.getDimension();
35     char[] buffer = new char[dimension];
36
37     for (int line = 0; line < dimension; line++) {
38         int lus = reader.read(buffer);
39         if (lus != dimension) {
40             throw new EOFException("format incorrect");
41         }

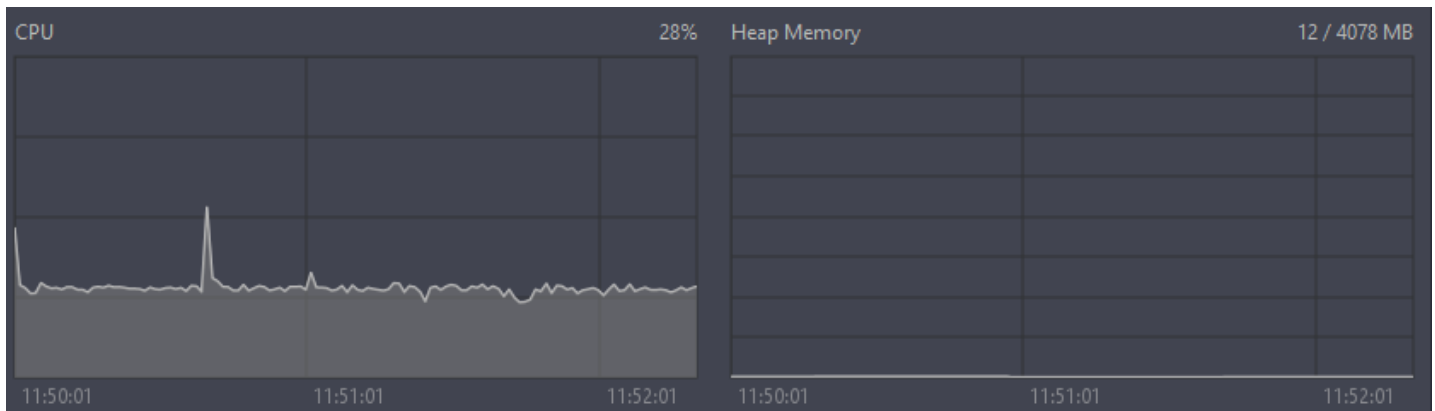
```

75% classes, 89% lines covered in package 'org.unc.nc'

Element	Class, %	Method, %	Line, %	Branch, %
exceptions	100% (3/3)	100% (3/3)	100% (3/3)	100% (0/0)
App	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
CalculUtils	0% (0/1)	0% (0/3)	0% (0/8)	0% (0/2)
Grille	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
GrilleImpl	100% (1/1)	100% (13/13)	100% (80/80)	100% (46/46)
GrilleParserUtils	100% (1/1)	100% (2/2)	86% (13/15)	100% (0/0)

Globalement, le programme est assez lent, lors des tests je lance une résolution de grille en dimension 9 et 16. Si celle en dimension 9 se résout en moins d'une seconde, celle de 16 peut durer entre 2 et 3min30s. Celle de dimension 25 est beaucoup trop longue (environ 30 minutes sur mon ordinateur) et donc je ne l'ai pas inclus dans les tests.

Voici les usages en CPU et mémoire lors de l'exécution de ces tests :



Minimum utilisation CPU : 23%

Minimum utilisation mémoire : 8 Mb

Maximum utilisation CPU : 53%

Maximum utilisation mémoire : 17Mb

Temps sur ce test : 3 minutes et 22 secondes

On peut voir sur cette capture que c'est l'appel de la fonction de résolution qui consomme le plus de ressources :

Method	Samples
97.0% [truncated]	4 535
> 35.2% org.unc.nc.GrilleImpl.solve()	1 644
> 8.3% org.unc.nc.GrilleParserUtilsTest.grille16solve()	389
> 8.3% invoke0	389
> 8.0% jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Object, Object[])	375
> 7.6% jdk.internal.reflect.NativeMethodAccessorImpl.invoke(Object, Object[])	357
> 6.1% java.lang.reflect.Method.invoke(Object, Object[]) → jdk.internal.reflect.NativeMethodAccessorImpl.invoke(Object, Object[])	286
> 3.4% org.junit.platform.commons.util.ReflectionUtils.invokeMethod(Method, Object, Object[]) → jdk.internal.reflect.NativeMethodAccessorImpl.invoke(Object, C	161
> 1.8% org.junit.jupiter.engine.extension.TimeoutExtension.interceptTestableMethod(InvocationInterceptor\$Invocation, ReflectiveInvocationContext, ExtensionCont	86
> 1.7% org.junit.jupiter.engine.execution.MethodInvocation.proceed() → jdk.internal.reflect.NativeMethodAccessorImpl.invoke(Object, Object[])	81

**2. Refactoriser pour trouver des améliorations sans changer votre algorithme (et donner un nouveau rapport de performances)**

**3. Enrichissez votre algorithme pour écrire un solveur plus 'intelligent' et évaluez ses performances.**

Je n'ai pas pris le temps de l'appliquer, mais il aurait été possible de mettre en place un algorithme basée sur le principe de « liens dansants » (Dancing Links) et de l'utilisation de l'algorithme X. Dans les faits, il est question d'utiliser des nœuds et des matrices afin d'optimiser la récursivité en comparant plusieurs nœuds au cours d'une même récursion plutôt que de le faire nœud par nœud comme c'est le cas dans mon code. Par conséquent, le code sera bien plus optimisé et rapide. Dans les exemples trouvés, un des auteurs a pu remarquer que sa solution pour une grille de dimension 25 était cinq fois plus rapide avec cette méthode de liens dansants qu'avec une solution de backtracking (récursion simple).

J'ai pu trouver cette méthode de liens dansants sur plusieurs sources, dont voici deux liens :

<https://medium.com/javarevisited/building-a-sudoku-solver-in-java-with-dancing-links-180274b0b6c1>

<https://www.baeldung.com/java-sudoku>

Étant donné la quantité de code supplémentaire assez conséquente qu'implique ces solutions et que leur manière de fonctionner reste encore assez complexe pour moi, j'ai préféré ne pas les mettre en place (ça et le manque de temps).

