Master 1 Informatique :
Spécialité Compétences Complémentaires dans les services du Numérique (CCN)

Réalisé par : Neda YOUSEFIAN

Mini-projet de Meth

mini-éditeur de texte

Professeur : Noël Plouzeau

Avril 2020

Table des matières

1 Introduction

L'objectif de ce mini-projet est de réaliser un programme en java pour un éditeur de texte. On définit différentes actions pour l'utilisateur : saisir un caractère (insert), supprimer (delete), couper (cut), copier (copy), coller (paste), déplacer le curseur (move cursor), définir un marqueur (set marker). Pour cela, on utilise les concepts utilisateurs suivants :

- Buffer [1] (mémoire tampon)
- Clipboard (presse-papiers)
- Sélection et curseur

On utilise un objet de java de type **StringBuffer**. Ce dernier nous permet de représenter des séquences de caractères extensibles et inscriptibles. Les caractères peuvent être insérés ou supprimés en utilisant respectivement les méthodes insert() et delete().

2 Choix architecturaux

Afin d'organiser le projet, on crée quatre dossiers : receiver, commende, invoker et Client. Nous allons les décrire dans les sous-sections suivantes.

2.1 Diagramme de classe

On représente un *design pattern* dans le diagramme de classe. Dans ce *pattern*, il y a plusieurs classes. Un client qui a une fonction principale (main). Les classes de *concreteCommand* sont pour exécuter une opération à partir de *receiver*. La *concreteCommand* étend l'interface *Command* et y implémente la méthode *execute* en invoquant l'opération correspondante sur le *receiver*. La *concreteCommand* définit le lien entre le *receiver* et l'action à exécuter. La *command* déclare l'interface pour exécuter l'opération. L'invoker lui, demande à la Commande de transmettre la requête. On représente dans la figure suivante une diagramme de classe :

1. Un buffer est une partie de la mémoire qui est utilisée pour stocker un flux de données provenant de périphériques.

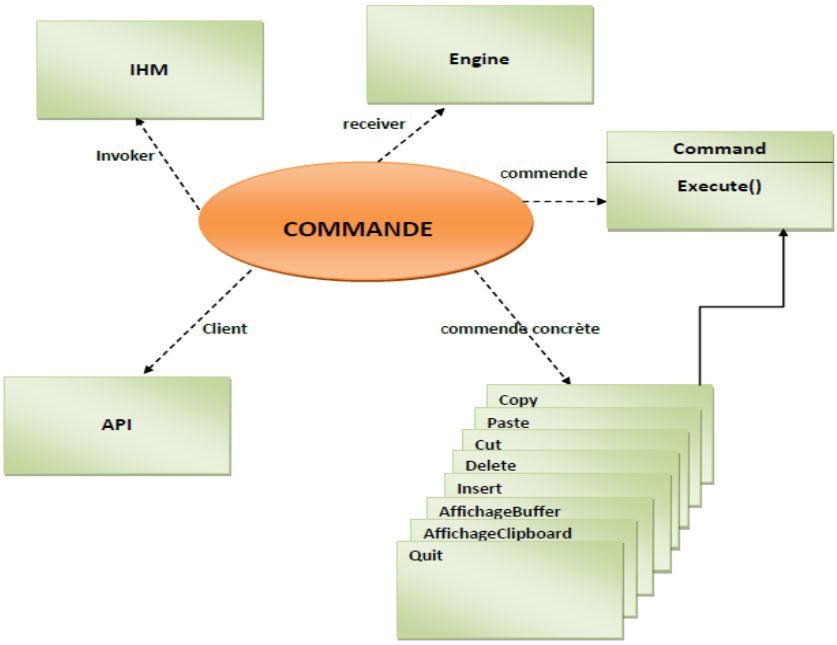


FIGURE 1 – Représentation du diagramme de classe

2.2 Diagramme de séquence

Dans cette sous-section, on présente un diagramme de séquence. Le client va demander à une commande de s'exécuter. L'*invoker* va prendre la commande, l'encapsuler et la placer dans une file d'attente dans le cas où il y a autre chose à faire avant cette commande. La *concreteCommand* qui est en charge de la commande demandée, envoie le résultat au *receiver*.

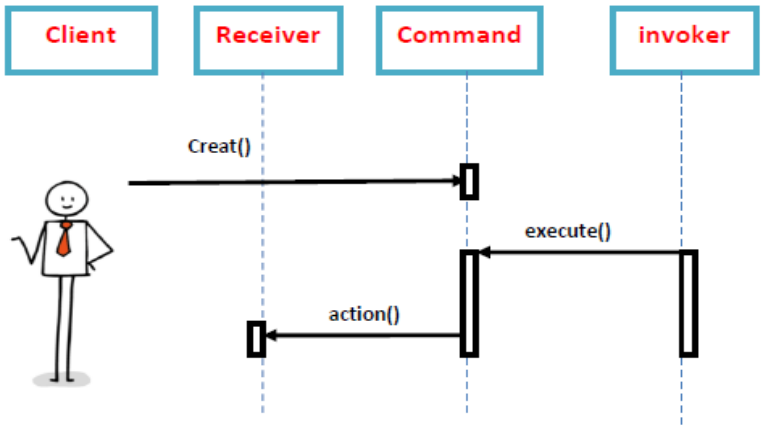
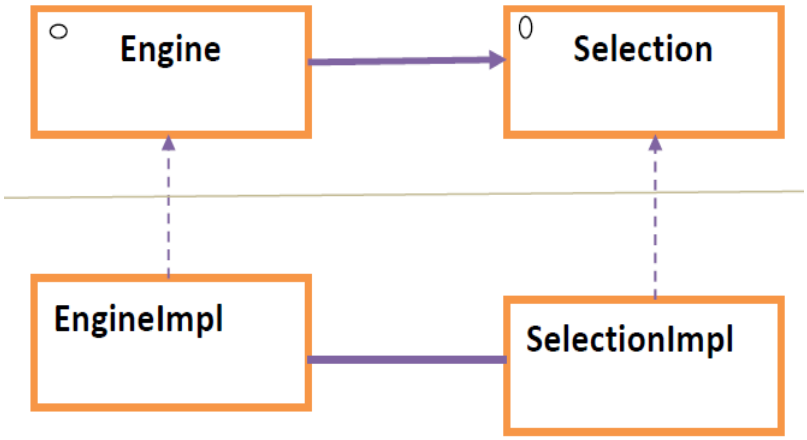


FIGURE 2 – Représentation du diagramme de séquence

2.3 UML

Dans deux interfaces *Engine* et *Selection*, on regroupe un ensemble d'opérations qu'on va implémenter respectivement dans *EngienImpl* et *SelectionImpl*. Le chronogramme suivant présente la relation entre les interfaces utilisés dans *receiver*



2.4 receiver

Dans le dossier receiver, on crée deux interfaces **Engine.java** et **Selection.java** et on les implémente respectivement dans **EngineImpl.java** et **SelectionImpl.java**. On fait les tests unitaires dans **EngineTest.java** .
La figure (3) représente deux tableaux de *Unified Modeling Language* (UML) de la classe Engine.java et Selection.java :

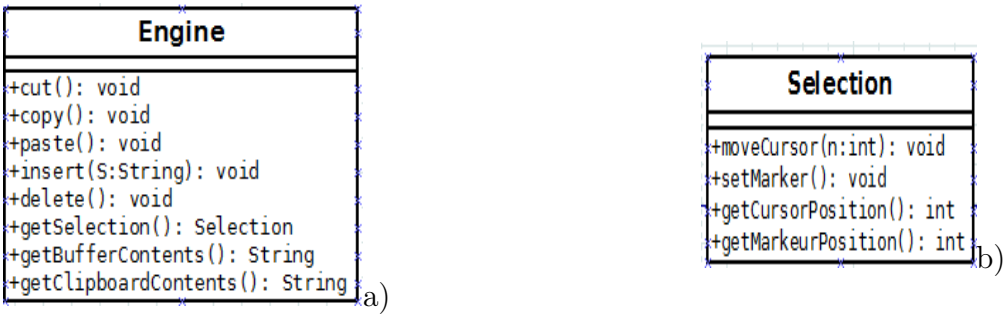


FIGURE 3 – Représentation d'un UML pour la classe a) Engine.java b) Selection.java

Nous allons maintenant décrire les classes de *receiver*.

2.4.1 Description de la classe EngineImpl.java

Cette classe nous permet d'implémenter les actions couper, copier, coller, supprimer et saisir un caractère. On crée le constructeur de *EnginImpl* pour instancier nos objets.

```
public  EnginImpl(){
    buffer = new StringBuffer();
    cboard = "";
    selection = new SelectionImpl(buffer);
}
```

- **Méthode public void cut()** : cela permet de couper un texte dans le presse-papiers. On prend la chaîne de caractères grâce à la méthode substring(int start, int end). Cette méthode renvoie un nouvel objet String contenant la chaîne donnée où start est inclusif et end exclusif. start est la position de marqueur **getMarkeurPosition()** et end la position de curseur **getCursorPosition()**. On supprime cette chaîne de caractères dans le texte avec la méthode delete(int start, int end). On la stocke dans la mémoire tampon(buffer). Ces méthodes sont implantées dans la classe *Selection*. On pose une condition :

```
if (start > end ){
        buffer.substring(end, start);
        buffer.delete((end, start);
    }
     else{
         buffer.substring((start, end);
         buffer.delete((start, end);
     }
```

- **Méthode public void copy()** : cela permet de copier un texte dans le presse-papiers. On a la chaîne de caractères avec la méthode de copie grâce à *substring(start , end)*. Cette méthode renvoie un nouvel objet String contenant la chaîne donnée de start à end. Concernant start et end, les informations sont données dans la présentation de la méthode de coupe.
  On pose une condition :

```
if (start > end ){
        buffer.substring(end, start);
    }
     else{
         buffer.substring(start, end);
     }
```

- **Méthode public void paste()** : cela permet de coller le texte depuis le presse-papiers. On pose une condition dans cette méthode : si on souhaite

coller un texte à la place d'un autre texte, tout d'abord, on supprime la chaîne de caractères en appliquant la méthode *StringBuffer.delete(start, end)* sur cette dernière. La chaîne commence au début spécifié et s'étend jusqu'au caractère défini comme fin de la chaîne. On l'insère ensuite par la méthode *StringBuffer.insert()*. Si on veut coller le texte à la position du curseur, on utilise juste la méthode *StringBuffer.insert()*.

```
if (start > end ){
    buffer.delete(end, start );
    buffer.insert(end , getClipboardContents());
    selection.moveCursor(getClipboardContents().length());
}
else{
    buffer.insert(end , getClipboardContents());
    selection.moveCursor(getClipboardContents().length());
}
```

- **Méthode public void delete()** : cela permet de supprimer un caractère dans le texte. Pour cela, on applique la méthode Java StringBuffer.deleteCharAt(int index) afin de le supprimer où *index* est la position de curseur *selection.getCursorPosition()*. Lorsqu'on supprime un caractère, le déplacement de curseur est noté -1 (selection.moveCursor(-1)).

- **Méthode public insert(String S)** : cela permet d'insérer une chaîne de caractères. Cette méthode a un paramètre de type *String*. On applique la méthode *insert(index , S)* où ***index*** est la position du curseur (selection.getCursorPosition()) et ***S*** un paramètre dans la méthode insert. On utilise la méthode *getCursorPosition()* de la classe *SelectionImpl.java*. Lorsqu'on ajoute un caractère le déplacement est noté (selection.moveCursor(S.length()).

- **Méthode public Selection getSelection()** : cela permet de retourner la sélection.
- **Méthode public String getBufferContents()** : cela permet de renvoyer le texte dans le buffer (mémoire tampon) et de retourner le contenu de StringBuffer.
- **Méthode public String getClipboardContents()** : cela permet de renvoyer le texte dans le presse-papiers et de retourner le contenu du presse-papiers.

2.4.2 Description de classe SelectionImpl.java

C'est une classe qui nous permet de sélectionner du texte. Pour cela, on définit et on modifie la position du curseur, qu'on déplace. On instancie l'objet *Selection* à partir de cette classe grâce à un constructeur défini. Ce constructeur

a un paramètre de de type StringBuffer. On initialise à chaque instance de classe sa propre variable.

```
public SelectionImpl(StringBuffer variable) {
    str = new String("");
    this.variable = variable;
    this.cursor = 0;
    this.markeur = 0;
}
```

- **Méthode public void moveCursor(int n)** : cette méthode est utilisée pour déplacer le curseur à un nombre donné n. Lorsque le curseur est entre le ime et le $i+1ime$ caractère du buffer, on dit que la position du curseur est i. Lorsque la position du curseur est 0, alors le curseur est avant le premier caractère (début du buffer).

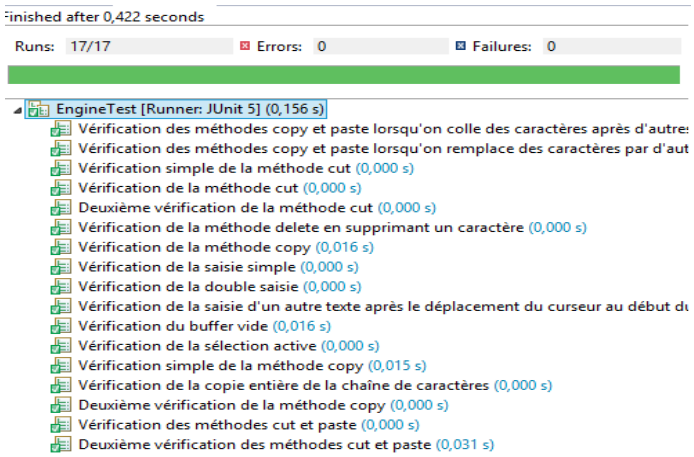
- **Méthode public void setMarker()** : cette méthode modifie la position du marqueur

- **Méthode public int getCursorPosition()** : cette méthode retourne le contenu du presse-papiers sous la forme d'une chaîne de caractères

- **Méthode public int getMarkeurPosition()** : cette méthode retourne la position du marqueur

2.4.3 Description de classe EngineTest.java

On réalise des tests grâce à JUnit 5 qui est un framework de test unitaire de Java. Ce framework nous permet de réaliser des assertions qui testent les résultats. On fait dix-sept tests dans cette classe avec des couvertures vertes. Ces couvertures vertes nous indiquent que toutes les méthodes ont bien été vérifiées.



Nous faisons une synthèse sur des tests appliqués dans *EngineTest*. On réalise cinq premiers tests simples :

- **test1 :** on saisit le texte *ABCD* dans un buffer vide
- **test2 :** on saisit le double texte *ABCDE* et *EFG*. Le contenu du buffer est *ABCDEFG*
- **test3 :** on saisit le texte *ABC*, on déplace le curseur au début du texte, on saisit le deuxième texte *DEF*. Le contenu du buffer est *DEFABC*
- **test4 :** on fait une vérification de la mémoire tampon, qui est vide.
- **test5 :** on vérifie si la sélection de texte est active. On saisit le texte *ABC*, le curseur se déplace deux fois à gauche *moveCursor(-2)*. On insère le deuxième texte *DEF*. Le contenu du buffer est alors *ADEFBC*

On fait quatre tests pour la méthode *Copy()* :

- **test6 :** on insère le texte *ABC*. On déplace le curseur trois fois à gauche *(moveCursor(-3))*. On applique la méthode cut mais il ne se passe rien. Le curseur ne se déplace pas *(moveCursor(0))*. Les contenus du presse-papiers et du buffer sont respectivement une chaîne vide et *ABC*
- **test7 :** on sélectionne tout le texte *ABC*. La position du curseur est à la fin du texte. En appliquant la méthode *copy()*, les contenus du presse-papiers et de la mémoire tampon sont identiques, à savoir *ABC*
- **test8 :** dans ce test, on va utiliser une partie du texte. La position du curseur est à la fin du texte, on déplace le curseur à gauche*(moveCursor(-2))* et on sélectionne *BC*. Les contenus du presse-papiers et du buffer sont respectivement *BC* et *ABC*
- **test9 :** on insère une chaîne de caractères *ABC*. Pour ce test, on déplace deux fois le curseur à gauche *(moveCursor(-2))* et on sélectionne le caractère *A* en déplaçant le curseur une fois à gauche *(moveCursor(-1))*. Les contenus du presse-papiers et du buffer sont respectivement *A* et *ABC*

On réalise trois tests pour la méthode *Cut()* :

- **test10 :** on insère le texte *ABCD*. On déplace le curseur quatre fois à gauche *(moveCursor(-4))*. On applique la méthode cut mais il ne se passe rien, le curseur ne se déplace pas *(moveCursor(0))*. Les contenus du presse-papiers et du buffer sont respectivement une chaîne vide et *ABCD*

- **test11 :** on insère le texte *ABCD*. On déplace le curseur trois fois à gauche *(moveCursor(-3))*. On sélectionne la chaîne de caractères *BC* en déplaçant le curseur deux fois à droite *(moveCursor(2))*. On applique la méthode *cut()*. Les contenus du presse-papiers et du buffer sont respectivement *BC* et *AD*

- **test12 :** on insère le texte *ABCD*. On déplace le curseur deux fois à gauche *(moveCursor(-2))*. On sélectionne le caractère *B* en déplaçant le curseur une fois à droite *(moveCursor(1))*. On applique la méthode *cut()*. Les contenus du presse-papiers et du buffer sont respectivement *B* et *ACD*

On réalise deux tests pour les méthodes copy et paste :

- **test13 :** on insère le texte *ABCD*. On déplace le curseur deux fois à gauche *(moveCursor(-2))*. On sélectionne la chaîne de caractères *CD* en déplaçant le curseur deux fois à droite *(moveCursor(2))*. On applique la méthode *copy()* afin de copier la chaîne de caractères *CD* dans le contenu du presse-papiers. On applique ensuite la méthode *paste()* afin de coller la chaîne de caractères *CD* après la position du curseur. Les contenus du presse-papiers et du buffer sont respectivement *CD* et *ABCDCD*

- **test14 :** on refait les étapes insertion de texte et sélection de texte du test13. On applique la méthode *copy()*. On applique la méthode *paste()*, on colle donc un texte sur un autre texte. Les contenus du presse-papiers et du buffer sont respectivement *AB* et *ABCD*

On réalise deux tests pour les méthodes cut et paste :

- **test15 :** on insère le texte *ABCDEF*. On déplace le curseur trois fois à gauche *(moveCursor(-3)*. On sélectionne la chaîne de caractères *DE* et on applique la méthode *cut()*. Les contenus du presse-papiers et du buffer sont respectivement *DE* et *ABCF*. Dans un second temps, on déplace le curseur une fois à gauche *(moveCursor(-1))*, le curseur est avant le caractère *F*. On applique la méthode *paste()*. Les contenus du presse-papiers et du buffer sont respectivement *DE* et *ABCFDE*

- **test16 :** on insère le texte *ABCDEF*. On déplace le curseur trois fois à gauche *(moveCursor(-3)*. On sélectionne la chaîne de caractères *ABC* en déplaçant le curseur encore trois fois à gauche. On applique la méthode *cut()*, les contenus du presse-papiers et du buffer sont respectivement *ABC* et *DEF*. La position du curseur est à 0 *(moveCursor(0))*. Dans un second temps, on déplace le curseur une fois à droite *(moveCursor(1))*, le curseur est sur le caractère *D*. On applique la méthode *paste()*. Les contenus du presse-papiers et du buffer sont respectivement *ABC* et *DABCEF*

On teste enfin la méthode *delet()* en supprimant un caractère :

- **test17 :** on insère le texte *ABCD*. On déplace le curseur deux fois à gauche *(moveCursor(-2))*. On sélectionne le caractère *C* en déplaçant le curseur une fois à droite *(moveCursor(1))*. En appliquant la méthode *delete()*, on supprime ce caractère. Le contenu du buffer est *ABD*

2.5 invoker

Dans cette partie, on réalise un exemple simple d'*invoker* pour le modèle de *design pattern*. Un exemple de modèle *IHM*[2] existe dans gitLab (hyperlien). Ce dossier est constitué d'une interface *IHM.java* et de son implantation *IHMImpl.java*. On ajoute au modèle existant sur gitLab les deux méthodes *getNumber()* et *getText()*. La méthode *getNumber()* insère un numéro dans la console et le retourne. La méthode *getText()* insère une chaîne de caractères dans la console et la retourne.

2.6 Client

Dans ce dossier, on crée un fichier *MainEditeur.java* afin de définir la fonction principale (*main*). On définit une fonction *configureCommands()* afin de configurer les commandes. Par exemple, pour **ihm.addCommand("C", new Copy(engine))**, "copy" est associé à la classe *copy* dans le dossier *commande* sur l'objet receiver. La liste suivante représente une commande de l'action sur l'API :

Commandes	Descriptions
I	Insère une chaîne de caractères
C	Copie le texte sélectionné dans le presse-papiers
V	Colle le texte dans le presse-papiers
D	Supprime un caractère
S	Sélectionne une partie du texte
X	Coupe le texte sélectionné
ABuf	Affiche le contenu de la mémoire
APP	Affiche le contenu du presse-papiers
Q	Permet de quitter l'éditeur de texte

Lorsqu'on exécute le fichier *MainEditeur.java*, ceci s'affiche sur la console :

2. *IHM* pour **I**nterfaces **H**omme **M**achine ou de *GUI* pour **G**raphical **U**ser **I**nterfaces

```
***************Bienvenue sur l'éditeur de texte*****************

*                     Choisissez les commandes                *

*****************************************************************
I       Insère un texte
C       Copie le texte sélectionné dans le presse-papiers
V       Colle le texte sélectionné dans le presse-papiers
D       Supprime un caractère
S       Sélectionne une partie du texte
X       Coupe le texte sélectionné et le met dans le presse-papiers
ABuf    Affiche le contenu de la mémoire-tampon
APP     Affiche le contenu du presse-papiers
Q       Permet de quitter l'éditeur de texte
*****************************************************************
*                                                               *
*                 Saisissez un caractère , S.V.P !              *
*                                                               *
*****************************************************************
```

On saisit la commande souhaitée à l'aide du tableau ci-dessus.

A)

```
I
M1 CCN
avr. 07, 2020 3:41:29 PM fr.istic.nyousefian.ACO.commande.Insert execute
INFOS: M1 CCN
I
N
avr. 07, 2020 3:41:36 PM fr.istic.nyousefian.ACO.commande.Insert execute
INFOS: M1 CCNN
D
avr. 07, 2020 3:41:40 PM fr.istic.nyousefian.ACO.commande.Delete execute
INFOS: M1 CCN
I
```

B)

```
C
avr. 07, 2020 3:42:06 PM fr.istic.nyousefian.ACO.commande.Copy execute
INFOS: M1 CCN
S
0
1
avr. 07, 2020 3:42:12 PM fr.istic.nyousefian.ACO.commande.Selection execute
INFOS: 0
avr. 07, 2020 3:42:12 PM fr.istic.nyousefian.ACO.commande.Selection execute
INFOS: 1
V
7
0
avr. 07, 2020 3:42:14 PM fr.istic.nyousefian.ACO.commande.Paste execute
INFOS: M1 CCN
avr. 07, 2020 3:42:14 PM fr.istic.nyousefian.ACO.commande.Paste execute
INFOS: M1 CCN M1 CCN
X
avr. 07, 2020 3:42:18 PM fr.istic.nyousefian.ACO.commande.Cut execute
INFOS: M1 CCN M1 CCN M1 CCN
avr. 07, 2020 3:42:18 PM fr.istic.nyousefian.ACO.commande.Cut execute
INFOS: M1 CCN M1 CCN M1 CCN
APP
M1 CCN M1 CCN M1 CCN
ABuf

Q
```

FIGURE 4 – Représentation :
A) insertion de la chaîne de caractères "M1 CCNN" avec la commande ***I*** et suppression du caractère 'N' avec la commande ***D***
B) copie, collage, sélection, coupe, affichage du contenu du presse-papiers, affichage du contenu de la mémoire et arrêt grâce aux commandes **C**, **V**, **S**, **X**, **APP**, **ABuf** et **Q**

2.7 commande

Ce dossier contient l'ensemble des fichiers action de l'utilisateur et le fichier *command.java*. *commande* est une interface qui nous permet d'exécuter le code. Nous allons maintenant décrire une de ces classes. Prenons comme exemple *copy*. La classe *copy* implémente l'interface *Command.java*. Elle possède un constructeur et une méthode *execute()*.
Voici la composition de la méthode *execute()* :
- engine.copy() : appelle la méthode correspondante *engine*.
- Logger.getGlobal().info(engine.getClipboardContents()) : *Logger* est utilisé pour voir le contenu du presse-papiers.

3 Conclusion

Ce mini projet d'éditeur du texte m'a permis d'approfondir mes connaissances du langage Java à partir d'un projet spécifique. J'ai également acquis des bases sur le contrôle d'une machine via IHM. J'ai pu mieux cerner ce qu'est une mémoire-tampon (buffer) et la classe StringBuffer. Le projet m'a enfin permis d'avoir des notions à propos du *design pattern*.
Le programme ne contient qu'un test donc le code *EngineTest.java* a été parcouru à 100%. L'ensemble du programme n'a cependant été parcouru qu'à 61.5%. On pourrait poursuivre le travail en faisant un test unitaire.