

## 1. Rappels généraux

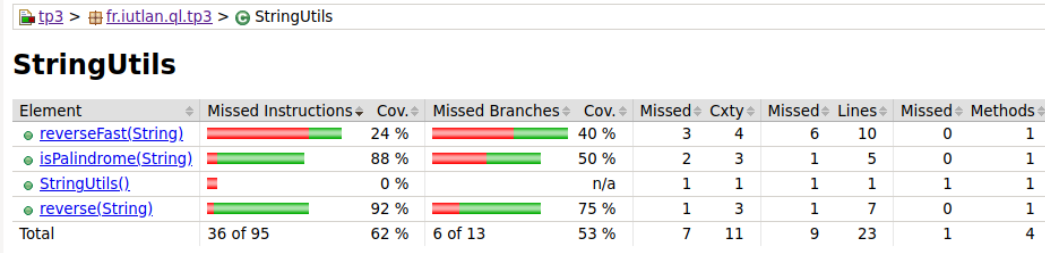
Tous les TD et TP se dérouleront sur **Linux**.

N'oubliez pas de déposer votre travail sur Moodle à *chaque* séance. Vous pouvez le continuer chez vous après et le re-déposer, mais il est interdit de tricher en échangeant des morceaux de code source entre vous. C'est à vous de faire le travail intégralement.

## 2. Présentation générale

Jacoco est un outil qui permet de calculer la *couverture de code*. C'est à dire la proportion d'instructions du logiciel à tester qui sont effectivement vérifiées par des tests unitaires. Est-ce qu'un projet peut être considéré comme fiable si des cas particuliers ne sont pas vérifiés par des tests unitaires ? Par exemple si une méthode doit pouvoir recevoir un paramètre `null` sans planter, mais qu'on ne le vérifie pas et qu'en réalité, ça la fait planter. . .

Jacoco s'utilise après avoir programmé des tests. Il affiche des statistiques comme celles-ci :



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
reverseFast(String)		24 %		40 %	3	4	6	10	0	1
isPalindrome(String)		88 %		50 %	2	3	1	5	0	1
StringUtils()		0 %	n/a	n/a	1	1	1	1	1	1
reverse(String)		92 %		75 %	1	3	1	7	0	1
Total	36 of 95	62 %	6 of 13	53 %	7	11	9	23	1	4

Figure 1: Rapport de Jacoco

Jacoco considère chaque instruction du programme, et chaque alternative. Normalement, les tests doivent aller partout. C'est à dire qu'en fournissant des paramètres soigneusement réfléchis, les tests font exécuter chaque instruction des méthodes à tester. Jacoco colore en rouge les instructions n'ont jamais été sollicitées lors des tests (*missed branches* et *missed instructions*). C'est donc la priorité des testeurs de rajouter ce qu'il faut, et ça va être votre travail dans ce TP.

## 3. Mise en place du TP

1. Créez un nouveau projet dans le *workspace* Eclipse :
  - GroupID : `fr.iutlan.ql`
  - ArtifactID : `tp4`
2. Remplacez le fichier `pom.xml` par [ce fichier pom.xml](#) (enregistrer la cible sous... et changez le nom).
3. Mettez à jour le projet Maven (menu **Project**, item **Update Maven Project**).
4. Dans le dossier `src/main/java`, créez le *package* `fr.iutlan.ql.tp4` et une classe `StringUtils`. Ensuite, remplacez cette classe par [StringUtils.java](#) (enregistrer le lien sous...).
5. Téléchargez le fichier compressé [TestsStringUtils.zip](#).
6. Décompressez ce fichier compressé et déplacez ses sources dans le dossier `src/test/java/fr/iutlan/ql/tp4`.

7. Vérifiez bien la structure finale des sources, `src/main` et `src/test` et les packages, utilisez le remaniement (*refactoring*) pour corriger des défauts.
8. Faites tourner les tests fournis. Comme ils sont dans plusieurs classes, on les appelle une *suite de tests*. Ils doivent tous passer.
9. Téléchargez le fichier [morseEncode.py](#) (enregistrer la cible sous...), où vous voulez pour pouvoir le lancer facilement. Ajoutez le droit `x` dessus : `chmod +x morseEncode.py`.

## 4. Couverture de code

Le but du TP est de compléter ces tests afin de vérifier toutes les fonctionnalités de la classe `StringUtils`. Actuellement, il y a six tests répartis dans autant de classes, mais ils ne vérifient pas tous les cas. Dans les méthodes à tester, il y a des alternatives et des boucles, or ces tests ne les sollicitent pas toutes. Il pourrait y avoir des bugs dans les instructions non testées.

### 4.1. Analyse de la couverture dans Eclipse

Il y a un menu dans Eclipse, clic droit sur le projet, puis **Coverage As**, puis **JUnit Test**, figure 2.

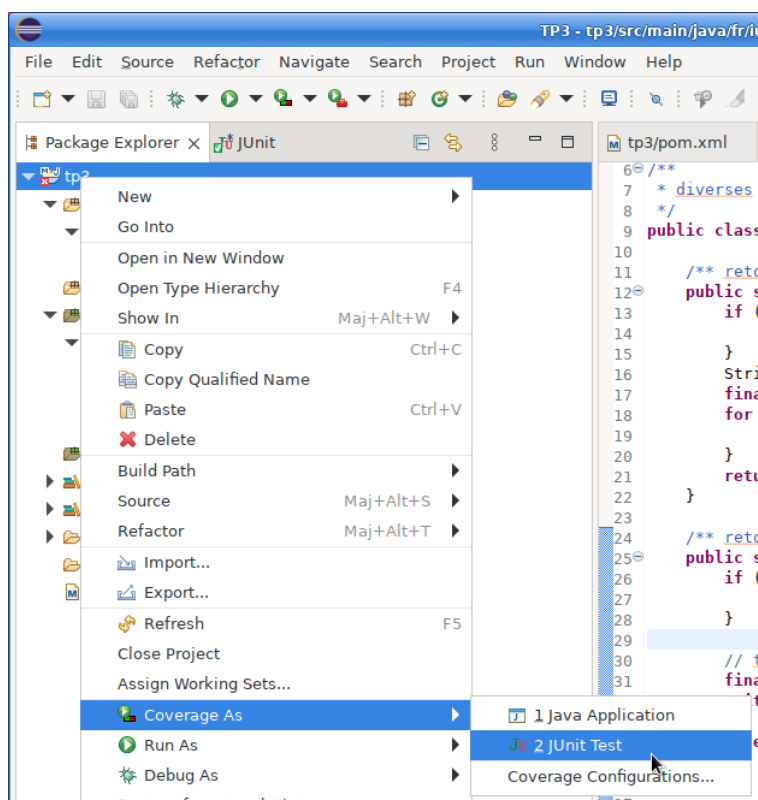


Figure 2: Menu Couverture des tests

Ça va lancer Jacoco qui colore les sources, en vert si l'exécution est passée là, en jaune si tous les cas n'ont pas été vus et en rouge si l'exécution n'est jamais passée par là.

NB: Jacoco est aussi appliqué aux tests eux-mêmes, pour être sûr que tous les tests et leurs méthodes annexes servent à quelque chose, mais il ne faut tenir compte que de ce qu'il y a dans `src/main/java`. Le taux de couverture y est beaucoup plus faible.

Pour enlever les couleurs, il suffit de cliquer sur le bouton en forme de double croix dans l'onglet Coverage, figure 3.

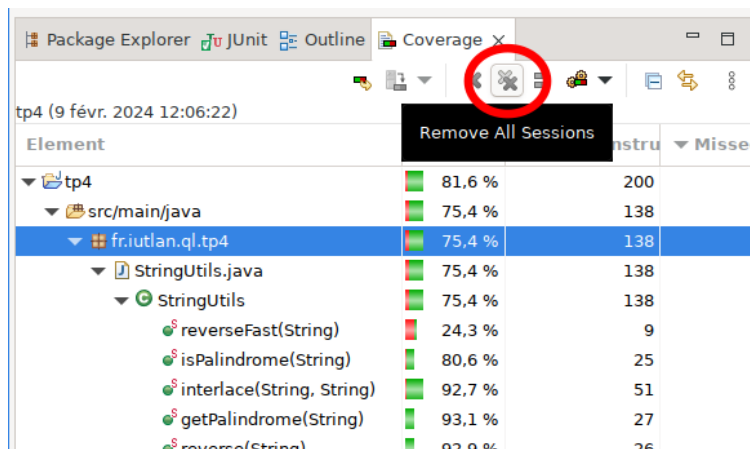


Figure 3: Effacer les couleurs

## 4.2. Analyse par Jacoco

La commande `mvn clean verify` fait le même travail que dans Eclipse. Ce qui est intéressant, c'est qu'elle produit un rapport qui peut être ajouté au dossier d'un projet.

1. Tapez `mvn clean verify` dans le dossier du projet (là où est le `pom.xml`)
2. Allez ensuite dans `target/site/jacoco`. Vous y voyez un `index.html`, ouvrez-le et parcourez-le en cliquant sur les différents liens, voir figure 1.

## 4.3. Vérification de la couverture par maven

Regardez le fichier `pom.xml`. Vers la fin, il y a ce groupe de lignes, pour configurer Jacoco en tant que `test global`. C'est à dire que le résultat de Jacoco est lui-même testé, et le projet est rejeté si le taux de couverture du code n'est pas suffisant :

```
<limit>
  <counter>LINE</counter>
  <value>COVEREDRATIO</value>
  <minimum>0.25</minimum>
</limit>
```

La balise `<minimum>0.25</minimum>` permet d'indiquer un seuil de couverture minimal que doit valider le projet. Ici 25%, c'est parce que les tests qui vous sont fournis valident à peine mieux que ça. L'objectif est de couvrir 100%.

- 👉 Modifiez le seuil de couverture exigé à 100%.
- 👉 Ajoutez les tests pour arriver à ce taux de couverture.

Vous allez avoir un peu de mal avec le décodage Morse. Lisez les explications qui sont dans le script [morseEncode.py](#). Le décodage fait appel à un parcours d'arbre binaire, mais ça a été codé sous la forme d'une gigantesque arborescence de tests. C'est efficace, mais c'est très peu lisible.

Il est recommandé d'utiliser des tests paramétrés. Vous devrez relire le TP3. Concernant les tests, comme il faut passer partout, vous pourrez utiliser le script `morseEncode.py` pour générer des séquences à tester. Il y a un grand nombre d'exceptions, il faudra comprendre comment les provoquer volontairement. Vous pourrez placer ces séquences dans un test paramétré, du type « tests qui doivent tous échouer ».

## 5. Rendu du TP

Vous devrez remettre votre travail à la fin de chaque séance. C'est ainsi que c'est noté, en contrôle continu.

Ouvrez un terminal bash dans le dossier de votre TP, là où il y a le fichier `pom.xml`, puis tapez ceci :



```
mvn clean
cd ..
tar cfvz tp4.tgz tp4
```

Puis déposez le fichier `tp4.tgz` dans la zone de dépôt du TP4 sur [Moodle R4.02 Qualité de développement](#).

**ATTENTION** votre travail est personnel. Si vous copiez pour quelque raison que ce soit le travail d'un autre, vous serez tous les deux pénalisés par un zéro.

En cas de souci quelconque, envoyez un mail à [pierre.nerzic@univ-rennes1.fr](mailto:pierre.nerzic@univ-rennes1.fr) pour expliquer le problème.