IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Tests – Tests avec JUnit(5)

Ingénierie du développement logiciel – IDL

1 Objectifs

Ce TP a pour but de vous faire découvrir les tests unitaires ainsi que de vous familiariser avec les technologies JUnit (5) et Eclemma.

- Se familiariser avec la notion de test unitaire
- Apprendre à écrire des tests avec JUnit (5)
- Exécuter des tests et mesurer leur couverture avec EclEmma

La base de code pour expérimenter JUnit et EclEmma en première partie de TP sera l'implémentation `bad4debug.zip` que vous avez préalablement corrigée. La deuxième partie du TP s'applique à votre projet de fil rouge de MAPD.

2 Important : évaluation

Votre production de TP est évaluée, vous devez donc la rendre pour la date fixée avec le responsable. Le dépôt se fait sur Moodle, dans l'espace dédié. Le livrable attendu est une archive au format `.tar.gz`[1] d'un répertoire à votre nom contenant les éléments suivants :

- les tests de mise en jambe de la première partie ;
- le plan de test de la deuxième partie (sous la forme des deux grilles complétées exportées au format PDF) ;
- les tests pour votre projet de la deuxième partie.

Pour vous faire gagner du temps dans le rendu, nous vous proposons deux grilles question 2 (section 7.3 du plan de test). Ces grilles sont disponibles sur Moodle, au format ODT (LibreOffice). Vous n'avez pas besoin de nous rendre autre chose concernant le plan de test. Néanmoins, si vous souhaitez un plan de test complet en MAPD, il vous faudra en rédiger un.

Note : la deuxième partie du TP est fortement liée à l'UE MAPD et le livrable est aussi très utile pour le fil rouge. Dans l'UE MAPD, il vous sera départ. Vous êtes fortement invités à joindre le plan de test au livrable de MAPD afin de faciliter la compréhension des évaluateurs de l'UE.

1. Et pas `.zip`, ni `.tar`, ni `.bz2`, ni `.rar`, ni `.7z`, ... Se référer à la documentation de `tar` pour la commande précise.

Afin d'éviter une double évaluation du même contenu entre les deux UEs, il vous est demandé de différencier les tests écrits pour IDL de ceux que vous ajouterez par la suite en MAPD (vous pouvez par exemple ajouter un commentaire court avant chaque test).

3 Outils

3.1 JUnit

JUnit est déjà installé dans votre environnement Eclipse mais vous devez configurer votre projet pour pouvoir l'utiliser. Pour cela il faut ajouter la bonne bibliothèque dans le *buildpath*.

3.2 Emma/EclEmma

EclEmma est un outil libre pour la couverture des tests Java sous Eclipse. Il est censé être installé sur les machines de l'école et est généralement installé par défaut avecv Eclipse pour Java, néanmoins. Il se peut néanmoins que vous deviez/souhaitiez l'installer sur votre machine personnelle. Le site de l'outil est : `http://www.eclemma.org/` il existe sur le *market place*. L'installation se passe normalement sans problème. Pour le lancement il faut utiliser le lanceur, bouton qui apparaît près du debuggeur, à sa gauche.

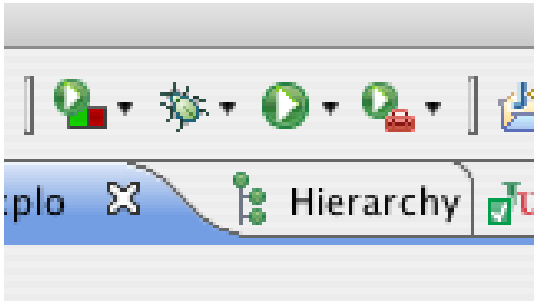

FIGURE 1 – EclEmma : le lanceur.

Mais un menu est aussi disponible dans le menu contextuel, utiliser le menu `Coverage As`.

Dans la vue EclEmma vous verrez alors apparaître le calcul de couverture de vos tests. Mais la coloration apparaît également dans le code source. Ces couleurs peuvent d'ailleurs être changées dans les préférences. Le taux de couverture est fait relativement à une unité qui peut-être changée dans le menu des préférences. Cette information apparaît dans l'onglet *Coverage* des propriétés du projet. Il s'agit : du nombre d'instructions, de ligne de code, de blocs de méthodes ou de types. L'information sur le taux de couverture (et la présentation) peut être changée dans la vue, il faut cliquer sur le petit triangle à droite. Ce point est important car dans un premier temps vous pouvez faire un fichier de test par classe, avec toutes les méthodes à tester. Ensuite choisir le taux par méthodes puis le taux par instructions qui est beaucoup plus fin.

Il est possible d'ajuster la configuration de couverture de façon à ne couvrir que le code source, pour cela aller dans *coverage configuration* et sélectionner les dossiers à analyser. Pour supprimer la coloration du code il faut aller dans la vue des couvertures et fermer toutes les sessions en cliquant sur la double croix noire.

Mais attention à bien configurer la couverture.

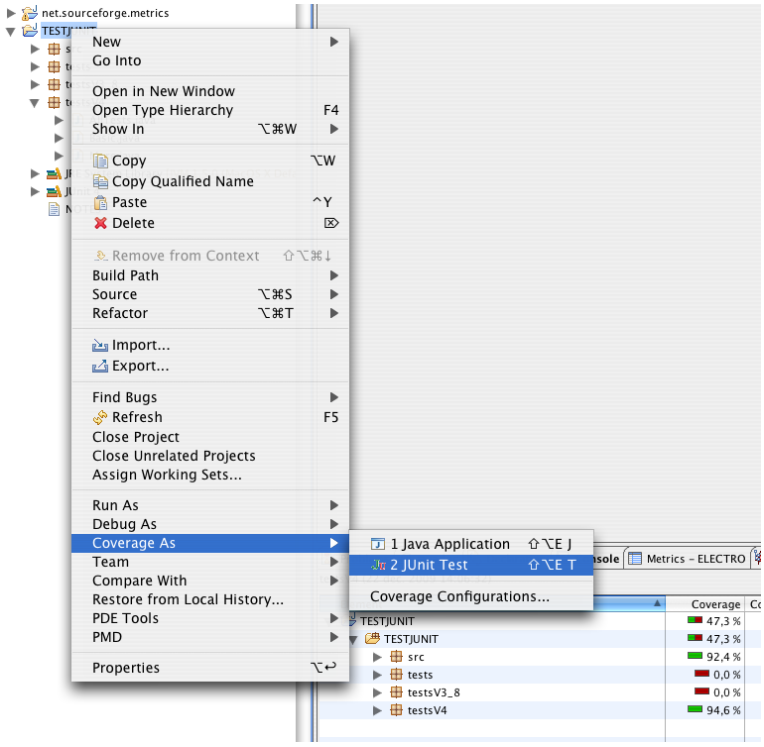

FIGURE 2 – EclEmma : le menu contextuel.

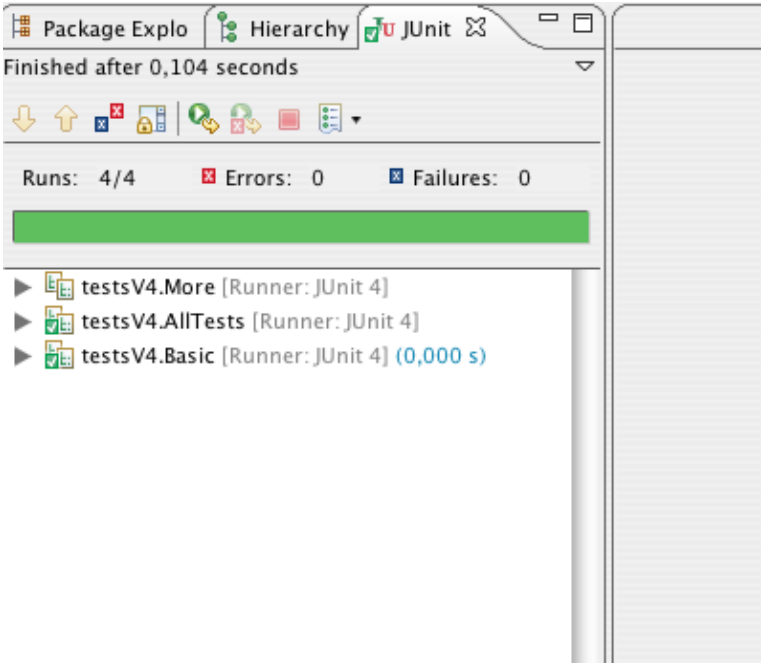

FIGURE 3 – JUNIT : une exécution de tests.

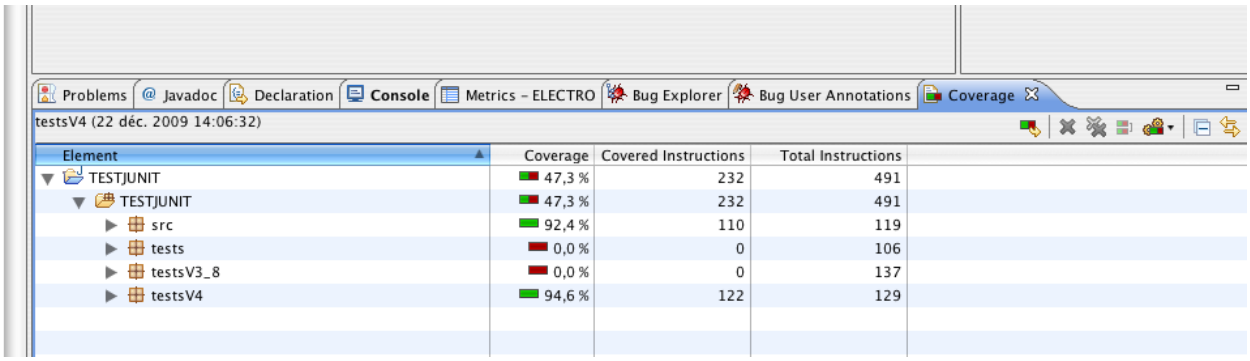

FIGURE 4 – EclEmma : la vue de couverture.

4 Travail à faire #1 : mise en jambes

L'objectif de cette section est de se familiariser avec Emma dans Eclipse, avec la notion de couverture ainsi qu'avec le système d'annotations de JUnit pour exprimer des tests. Si vous êtes déjà familier avec ces outils et que vous les comprenez bien, vous pouvez passer à la section suivante.

Notes de lecture et d'exécution de cette section :

- Les trois premières étapes ne sont que l'initialisation du travail (et sont donc des étapes obligatoires lorsque l'on écrit des tests) ;
- les étapes 4 et 5 sont la conception et l'écriture des tests. Si les principes sont compris, elles peuvent être raccourcies pour vous concentrer sur la section 5 (vous devrez de toute façon vous poser ces questions si vous souhaitez tester votre implémentation du réseau de Petri) ;
- l'étape 6 peut être exécutée avant et après chaque écriture de test (sans pour autant viser 100% de couverture) ;

Travail à faire :

1. Reprendre le code du modèle *bad4debug* qui est dans `bad4tests.zip` ;
2. Ajouter un répertoire de code pour contenir vos tests ;
3. Ajouter JUnit (5) dans les bibliothèques de votre projet ;
4. Créer une classe de tests pour la classe `Place` avec le test d'une partie de ses méthodes :
   - `toString()` ou `getName()` ;
   - `getTokens()` ;
   - `removeTokens()` ;
   - les constructeurs de `Place` ;

   Il est généralement préférable d'ordonner ses méthodes (annotation `@Order`) de la plus simple à la plus complexe en commençant par les constructeurs, puis les accesseurs et les autres en fonction de leur graphe d'appel. Dans le cas de la classe `Place` les méthodes d'ajout des arcs sont probablement les plus complexes et devraient être testées en dernier (mais ce n'est pas demandé dans ce travail préliminaire).
5. Exécuter vos tests JUnit. Trouver les cas limites ou significatifs à tester, éventuellement ajouter des méthodes auxiliaires qu'il faudra aussi tester. Voici des questions à se poser et à résoudre.
   - Nombre de tokens = 0 et création d'un vecteur ou null value ?
   - Nombre de tokens négatif ? erreur ou valeur absolue ?
   - Nombre de tokens positif et taille du vecteur 0 ou autre chose ?
   - Nombre de tokens dans l'opération `removeTokens` ?
   - Tester l'existence d'un arc avant l'ajout, cela peut demander des méthodes additionnelles

qu'il faut également tester.

6. Tester votre niveau de couverture, exclure les tests de la configuration de couverture et tester à nouveau la couverture. Il est intéressant de faire des allers-retours entre l'écriture de tests, la mesure de la couverture et l'observation de la coloration puis d'ajouter et de corriger les tests.
7. Créer un *setup*.

Utile : Fixer un ordre d'exécution des tests, en particulier s'il y a des effets de bords, peut être utile. Il y a des moyens prédéfinis et vous pouvez définir le vôtre, voir le bon article `https://www.baeldung.com/junit-5-test-order`. Il est aussi préférable de tester les plus simples d'abord.

En situation réelle, il faudrait évidemment aussi tester les autres classes.

5 Travail à faire #2 : conception et écriture de tests

Dans cette partie, vous devrez mettre en œuvre le processus de test, de l'analyse à l'écriture. **Vous devrez déposer ce travail sur le dépôt Moodle dédié afin d'être évalué.**

5.1 Analyse, plan de tests

Livrable : Le document de plan de test complété.

Un exemple de plan de test vous est fourni sur Moodle.

▷ **Compléter ce plan de test de sorte qu'il soit cohérent avec vos choix de conception de votre réseau de Petri (RdP) en MAPD [2] :**

1. **Section 7.2 du plan de test : compléter la spécification des tests afin de prendre en compte**
   - **la suppression de jetons d'une place ;**
   - **la création d'arcs ;**
   - **la gestion des arcs doublés.**
2. **Section 7.3 du plan de test : compléter la spécification des tests d'activation des transitions.**
   - **Concernant les transitions simples, certaines spécifications sont complètes, d'autres sont parcellaires (avec une ou plusieurs cases vides), d'autres sont inexistantes (et donc à écrire complètement) ;**
   - **Écrire la spécification pour le cas des transitions à entrées multiples.**

5.2 Conception et écriture de tests

Livrable : Les tests écrits en Java ainsi qu'un lien vers votre projet pour pouvoir les exécuter (une archive comprenant le tout est la bienvenue).

▷ **Travail préliminaire :**

2. De ce fait, vos plans de tests ainsi que vos tests peuvent être différents les uns des autres

1. **écrire le code permettant un affichage des éléments du RdP. Cet affichage se doit d'être simple et lisible. Vous pourrez vous inspirer de l'affichage proposé dans la section 7.1 du plan de tests.**
2. **écrire les tests de cet affichage (la section 7.1 du plan de test vous donne un bon point de départ).**

▷ **Écrire les tests en suivant votre plan de test.** [3] **Vous devrez fournir au moins quelques tests dans chacune des catégories suivantes :**

1. création d'éléments du RdP (état initial d'un élément créé, navigabilité des relations, etc.) ;
2. activation du RdP (dont au moins une activation de transition à entrées multiples) ;
3. destruction d'éléments du RdP (dont au moins une place ou une transition reliée à un arc).

3. Il n'est pas attendu que vous fournissiez la totalité des tests spécifiés par le plan de test qui ne sera pas forcément exhaustif dans le cadre de IDL. Néanmoins nous vous encourageons à compléter vos tests au fur et à mesure de l'avancée de l'UE MAPD en suivant ce plan de test.