TP de PPO en JAVA

Polytech Lille GIS2A3

Objectifs : Tests unitaires en JUnit - Eclipse

1 Prise en main

1.1 Eclipse

1. Lancer eclipse dans sa version neon
   ```
   bash> /usr/local/eclipseNeon/eclipse &
   ```
2. Sélectionner le workspace `eclipse/workspace` par défaut
3. Ouvrir la "perspective Java" : onglet "Window/Perspective/Open Perspective/Java
4. Créer un projet Java de nom Progression : onglet "File/New/Java Project", entrer le nom du projet puis cliquer sur "Finish" (options par défaut). Il apparait dans la vue "Package Explorer" avec un répertoire "src" pour y programmer vos sources.
5. Importer les sources de vos classes ProgressionArithmetique et ProgressionGeometrique dans leurs versions initiales dans "src" : bouton droit sur "src" puis "Import.../General/File System", cliquer sur "Next", sélectionner votre répertoire de TP où vous les aviez programmées puis les classes précédentes.
   — Vérifier qu'elles apparaissent bien sous "src" dans "(default package)" = pas de package.
   — Remarquer leur présentation arborescente : zoomer sur une classe, ses constituants apparaissent (variables d'instances, méthodes, constructeurs, ...). Cliquer sur un constituant pour se positionner dans le code.
   — Voir la Javadoc du langage : par exemple dans le code de la classe ProgressionAritmetique cliquer sur `String` (si la Javadoc n'apparait pas, faire : onglet "Window/Show View/Javadoc").
6. Compilation et exécution :
   — Compilation : automatique(si l'option : onglet "Project/BuildAutomatically" est cochée, ce qui est le cas par défaut). Par défaut Eclipse range les `.class` dans un répertoire `bin` du projet. Vous pouvez vous en rendre compte :
     — sur le système : répertoire `eclipse/workspace/`
     — sous eclipse : onglet "Window/Show View/Navigator".
   — Exécution : dans la vue "Package Explorer" faites bouton droit sur `TestArithmetique.java` puis "Run As/Java Application". Son exécution apparait dans la "Console" en bas (sinon faire : "Window/Show View/Console"). Par la suite il sera possible de relancer lÕexécution par l'onglet "Run" ou le bouton correspondant de la barre de menu principale.

1.2 Tests

1. Créer la classe de test `ProgressionGeometriqueTest`
   — sur le package faire "bouton droit/New/JUnitTestCase", entrer son nom `ProgressionGeometriqueTest`
   — puis OK sur "Add JUnit 4 library to the build path"

— un schéma de code de méthode de test apparait (annoté par `@Test`).
— programmer selon ce shéma les méthodes de test
— exécuter ces tests : sur `ProgressionGeometriqueTest`, "bouton droit/Run As/JUnit Test"
— le compte-rendu de JUnit apparait dans une nouvelle vue

2. Copier le répertoire `~aetien/public/PPO/tp6/source`. Il contient le code source de la classe `ProgressionGeometriqueTest` qui vous a permis de tester votre code à la fin du TP4.
3. Dans un éditeur de texte tel que `Kate` ouvrir la classe de test `ProgressionGeometriqueTest` et comparer avec ce que vous avez fait. Normalement, aux `try catch` près, vous devriez avoir le même code. Les `try catch` étaient là pour vous mettre des messages d'erreur compréhensibles à la place de l'affichage de la pile. Bien évidemment, ils sont désormais inutiles.

2 Tests pour la bibliothèque

Créer un nouveau projet `bibliotheque` dans Eclipse. Importez y votre dernière version de la bliothèque du TP5, de la même façon que précédemment.

2.1 Partie guidée

1. Créer un package `test` qui contiendra tous vos tests.
2. Créer la classe de test `OuvrageTest`
3. Écrire la méthode de test `testEmprunterValeurEmprunte()` qui verifie qu'à la création d'un ouvrage la valeur de sa variable d'instance `emprunte` vaut `false` puis `true` après avoir appelé la méthode `emprunter()` sur cet ouvrage.

   Attention, il est important que si vous lancez deux fois le même test, sans avoir rien modifié dans le code source, le résultat du test soit le même. De même, l'ordre dans lequel vous lancez les tests ne doit pas avoir d'impact sur leur résultat. Donc l'état de votre système avant le test et après doit être le même.
4. De la même façon écrire la méthode de test `testEmprunterValeurCompteur()` qui verifie que l'appel de la méthode `emprunter()` sur un ouvrage incrémente bien la valeur de sa variable d'instance `compteur`.
5. Écrire la méthode de test `testEmprunterExceptionLancee()` qui lance une exception quand l'ouvrage que l'on veut emprunter n'est pas disponible.

   Remarque pour la seule méthode `emprunte()` on a créé 3 méthodes de tests afin de tester le cas normal et le cas où l'ouvrage n'est pas disponible. Et votre méthode ne fait pas grand chose. *Attention donc à l'avenir à la façon dont vous écrivez vos méthodes pour qu'elles soient testables. Normalement, il faut tester toutes les branches. Il faut donc mieux découper vos méthodes pour pouvoir écrire au moins un test pour chacune d'elles.*
6. Écrire la méthode de test pour tester le comportement de la méthode `retourner()`.

2.2 A vous de jouer

1. Créer la classe de test `RevueTest` et les différentes méthodes de tests. Bien évidemment, seules les méthodes surchargées (c'est-à-dire redéfinies) dans la classe `Revue` doivent être testées.
2. Créer la classe de test `BibliothequeTest` et les différentes méthodes de tests. Personnellement, j'en ai écrit 13 dans la classe pour la dernière version du TP5.