TP de PPO en Java

Polytech Lille GIS4 2018-2019

Objectifs : Eclipse - Tests unitaires en JUnit.

Ce TP s'appuie sur l'application de gestion de comptes bancaires développée lors de TPs précédents et reprise comme exemple dans le cours sur les tests unitaires. Tenir à disposition les classes correspondantes que vous avez programmées : `Compte` (version primitive et avec historisation des opérations de crédit/débit), `CompteEpargne`, `Banque`, les classes d'exception `CompteInexistant` et `OperationNonValide`, la classe principale `Guichet`.

1 Eclipse

1.1 Prise en main

1. A la racine de votre compte créer un répertoire `eclipse/workspace`
   ```
   bash> cd
   bash> mkdir eclipse
   bash> cd eclipse
   bash> mkdir workspace
   ```
2. Lancer `eclipse` dans sa version `neon`
   ```
   bash> /usr/local/eclipseNeon/eclipse &
   ```
3. Sélectionner le workspace `eclipse/workspace` que vous venez de créer
4. Ouvrir la "perspective Java" : onglet "Window/Perspective/Open Perspective/Java"
5. Créer un projet Java de nom `banque0` : 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.
6. Importer les sources de vos classes `Compte` (dans sa version primitive), `CompteEpargne`, `Banque`, `CompteInexistant`, `OperationNonValide`, `Guichet` 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 `Guichet` cliquer sur `Scanner` (si la Javadoc n'apparait pas, faire : onglet "Window/Show View/Javadoc").
7. Compilation et exécution :
   - Compilation : automatique (si l'option : onglet "Project/Build Automatically" 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/banque0`

— sous eclipse : onglet “Window/Show View/Navigator”.
— Exécution : dans la vue ‘Package Explorer” faites bouton droit sur `Guichet.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 l’exécution par l’onglet “Run” ou le bouton correspondant de la barre de menu principale.

1.2 Packager

Il s’agit ici de créer une version packagée du projet précédent.

1. Créer un nouveau projet nommé `banquePack`
2. Créer un package `banque` : bouton droit sur “src”, “New/Package” de nom `banque`. Le package doit apparaitre dans “src” (noter au passage la possibilité de créer des nouvelles classes ou interfaces sous ce même menu “New”)
3. Copier les sources des classes du projet `banque0` sauf `Guichet` dans ce package `banque` du projet `banquePack`
   — les sélectionner dans `banque0/src/(default package)`
   — faire “bouton droit/Copy”
   — et sur `banquePack/src/banque` faire “bouton droit/Paste”
   — vérifier qu’Eclipse les a correctement packagées : la clause `‘‘package banque;’’` a été automatiquement insérée en tête de leur code.
4. Ranger la classe `Guichet` dans un autre package nommé `‘‘applications’’`
   — des erreurs apparaissent dues au fait que `Guichet` et les classes de base ne sont plus dans le même package, ce qui pose des problèmes de visibilité : `Guichet` (du package `‘‘applications’’` doit maintenant importer les classes du package `banque`.
   — résoudre ces erreurs en exploitant les préconisations faites par Eclipse (en marge gauche des lignes de code de la classe `Guichet`).
   — protéger (au moins par `protected`) les variables d’instance des classes de base (du package `banque`).
   — vérifier l’exécution correcte du programme `Guichet`.

2 Junit sous Eclipse

2.1 Scénario de base

Il s’agit ici d’expérimenter ce qui a été vu en cours sur les tests unitaires avec Junit (suivre le poly). Travailler dans le projet `banquePack`.

1. dans la classe `banque.Compte`, programmer la méthode `reinit()` dans sa version V1 (p. 5). Remarquer au passage des capacités d’expansion automatique de code d’Eclipse (proposition des méthodes applicables à une variable ou `this`).
2. Créer un package `test` (p. 21)
3. Créer la classe de test `CompteTest`
   — sur le package `test` faire “bouton droit/New/JUnit Test Case”, entrer son nom `CompteTest`
   — 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 de la p. 22
   — exécuter ces tests : sur `CompteTest`, “bouton droit/Run As/JUnit Test”
   — le compte-rendu de JUnit apparait dans une nouvelle vue (comme p. 24 du poly)

4. Faire la correction de la méthode `reinit()` de la classe `banque.Compte` (version V2, p. 11) et rejouer les tests (p. 25 du poly).
5. Tester aussi l'état "débiteur" (p. 26), corriger et rejouer les tests (p. 27).
6. Factoriser les situations (cf. p. 29 et 30 du poly).
7. Test d'exception (p.31 à 33 du poly) : créer une nouvelle classe de test `BanqueTest` pour tester des exceptions
   — `CompteInexistant` (p. 32)
   — faire de même pour tester les cas d'`OperationNonValide`
8. Test et héritage (p. 34 à 37 du poly)
   — créer la classe de test `CompteEpargneTest` (p. 35)
   — et essayer les 2 solutions par redéfinition ou par `@Ignore`.
9. Suite de tests (p. 39 et 40)
   — dans le package `test` créer une nouvelle classe de test `GestionBanqueTest` regroupant les classes de test précédentes conformément au code p.40 du poly
   — l'éxécuter.
10. exécuter des tests en mode ligne (p. 23).
    — dans un terminal, placer vous dans le répertoire `eclipse/workspace/banquePack/bin`
    — vérifier qu'il contient bien les `.class` compilés automatiquement par Eclipse
    — pour exécuter une classe de tests (par exemple `test.CompteTest`) faire :
    `java -cp .:/usr/local/junit/junit.jar org.junit.runner.JUnitCore test.CompteTest`
    Pour rappel l'option `-cp` permet de compléter le `CLASSPATH` pour l'exécution d'une commande, ici avec la bibliothèque de JUnit.

2.2 Points d'orgue

Changer de choix d'implémentation et profiter des tests rédigés précédemment (poly p. 41).

2.2.1 Comptes historisés

1. créer un nouveau projet `banqueHistorisee`
2. copier les classes précédentes et remplacer la classe `Compte` par sa version avec historisation des opérations de débits/crédits (voir. sujet de TP 2)
3. y ajouter la fonctionnalité `reinit()`.
4. rejouer les tests (qui ne doivent pas être modifiés).

2.2.2 Gestion des comptes

Travailler dans un nouveau projet `banqueUtil` en copiant le projet précédant. Changer la gestion interne des comptes dans la banque (sans changer son protocole) grâce aux collections de `java.util` en remplaçant le tableau de `Compte` par une `List<Compte>` ou une `Map<Integer,Compte>`. Exploiter les tests précédents (qui ne doivent pas être modifiés).