TP 3 : Débogage, refactorisation

Débogage

Cette partie du TP consiste en plusieurs petits exercices destinés à vous apprendre à utiliser les fonctionnalités de débogage symbolique du JDT pour déboguer un code existant sans le modifier (et sans avoir recours à `toString()` et `System.out.println()`...).

Pour utiliser efficacement la perspective *Debug* du JDT, il est fortement conseillé de retenir les raccourcis clavier associés aux actions *Step Into*, *Step Over* et *Step Return* (cf. le cours sur la partie débogage du JDT).

Récupérer et importer dans l'espace de travail le projet situé à l'adresse suivante :

`http://www.di.ens.fr/~labatut/ED6/tp-3/Debug.zip`

Localiser la méthode `main()`, elle contient des appels à des méthodes mettant en évidence des bogues dans d'autres classes du projet.

Exercice 1 : La classe `Algorithms` implémente deux versions de la méthode `swap()` censée échanger les objets passés en arguments. Expliquer pourquoi la première version de cette méthode ne peut fonctionner. Vérifier avec le débogueur votre explication.

La deuxième méthode propose un moyen de contourner le problème. Vérifier son bon fonctionnement en comparant les variables du cadre de pile de `swap()` de la classe `Algorithms` et les variables du cadre de pile de `testSwap()` de la classe `Buggy`.

Exercice 2 : La méthode `binarySearch()` de la classe `Algorithms` est boguée (cf. l'affichage de la méthode `testBinarySearch()` de la classe `Buggy`).

Utiliser (efficacement) le débogueur pour comprendre le problème (modifier éventuellement le code pour que les appels à cette méthode soient isolés sur une ligne) et le corriger.

Exercice 3 : Corriger le ou les problèmes de la classe `GrowableArray` testée dans la méthode `testGrowableArray()` de la classe `Buggy` (utiliser le bon type de point d'arrêt...).

Exercice 4 : Identifier la cause des bogues de la classe `List` mis en évidence par la méthode `testList()` de la classe `Buggy` puis les corriger (utiliser en particulier la vue *Variables* pour parcourir les listes en mémoire).

Refactorisation

Récupérer et importer dans l'espace de travail le projet situé à l'adresse suivante :

`http://www.di.ens.fr/~labatut/ED6/tp-3/Refactor.zip`

Exercice 5 : La classe `Refactor` crée une nouvelle boîte de dialogue `About`. En utilisant (presque) *exclusivement* les outils de refactorisation d'Eclipse, le code de ces classes peut être rendu beaucoup plus lisible.

- Faire apparaître clairement les dépendances des classes `Refactor`, `Logo` et `About` à l'aide d'*Organize Imports*.
- Donner aux classes `Refactor`, `Logo` et `About` des noms plus explicites quant à leur rôle (et renommer également les identifiants des variables de ces types).
- Dans l'ancienne classe `About`, mettre en évidence des morceaux de code aux rôles bien distincts et les isoler dans des méthodes séparées.
- Deux des nouvelles méthodes font quasiment le même travail, modifier l'une des deux pour rendre l'autre inutile et finalement la supprimer.
- Dans chacune des nouvelles méthodes, le positionnement pourrait être indiqué en argument avec une variable de type `java.awt.Rectangle`. Rajouter un tel argument.
- Renommer les variables pour supprimer les indices superflus.
- Convertir les deux classes anonymes en classes imbriquées.

Dans la suite, les classes considérées ne sont pas testées à l'aide d'une méthode `main()` mais, pour chacune d'elles, une classe auxiliaire `NomDeLaClasseTest` contient des méthodes `testNomDuTest()` et constitue une « suite » de tests unitaires. Pour lancer une telle suite de tests unitaires, utiliser *[Mouse-R / Run As / JUnit Test]* : une nouvelle vue apparaît alors indiquant le succès ou l'échec des tests.

Exercice 6 : Identifier le ou les problèmes de *design* logiciel dans la classe *Matcher*. Utiliser des opérations de refactorisation pour les corriger sans casser les tests unitaires.

Exercice 7 : Même chose pour la classe *Configuration*.

Exercice 8 : Même chose pour la classe *Expression*.

Analyse statique

Installer le greffon FindBugs pour Eclipse en suivant les instructions indiquées à l'adresse suivante (le répertoire d'installation devra probablement être modifié) :

`http://findbugs.cs.umd.edu/eclipse/`

Récupérer et importer dans l'espace de travail le projet situé à l'adresse suivante :

`http://www.di.ens.fr/~labatut/ED6/tp-3/Bugs.zip`

Exercice 9 :

- Utiliser FindBugs (*[Mouse-R / FindBugs / FindBugs]*) pour mettre en évidence les problèmes potentiels dans la classe `Bugs` (comparer avec la liste d'avertissements initialement fournie par Eclipse).
- Comprendre chacun des problèmes suggérés en se référant à leur description complète :

  `http://findbugs.sourceforge.net/bugDescriptions.html`

- Certains problèmes subsistent encore dans le code fourni, mais FindBugs n'a pas pu les mettre en évidence. En parcourant la liste des *Bugs Patterns* précédente ainsi que les parties du code sans problème apparant, essayer de trouver les problèmes restant.