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

DevEnv – Le déverminer pour Java sous Eclipse

Ingénierie du développement logiciel – IDL

Objectifs

- Apprendre à utiliser le déverminer (ou débogueur, *debugger* en anglais) sous Eclipse
- Déverminer un petit programme

Note : ce TP vise Eclipse, néanmoins, le déverminer est présent dans la plupart des IDE. Si vous utilisez un autre IDE et que vous vous sentez suffisamment à l'aise avec, vous pouvez toujours tenter d'adapter le TP à votre IDE.

1 Comment ça marche ?

Le déverminer est une application spécifique qui peut-être lancée à la main ou automatiquement lors d'une erreur. Dans l'utilisation manuelle il y a généralement trois étapes :

1. Insertion d'un ou de plusieurs points d'arrêt (*breakpoint*) dans le code source et sur la ligne que vous voulez observer. Voir la figure 1 pour un exemple, le point d'arrêt apparaît dans

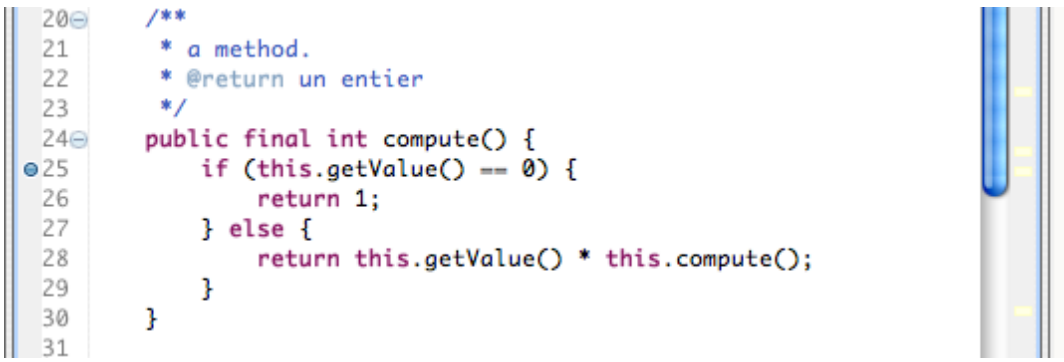

FIGURE 1 – Un point d'arrêt (*breakpoint* ligne 25, dans la méthode `compute`

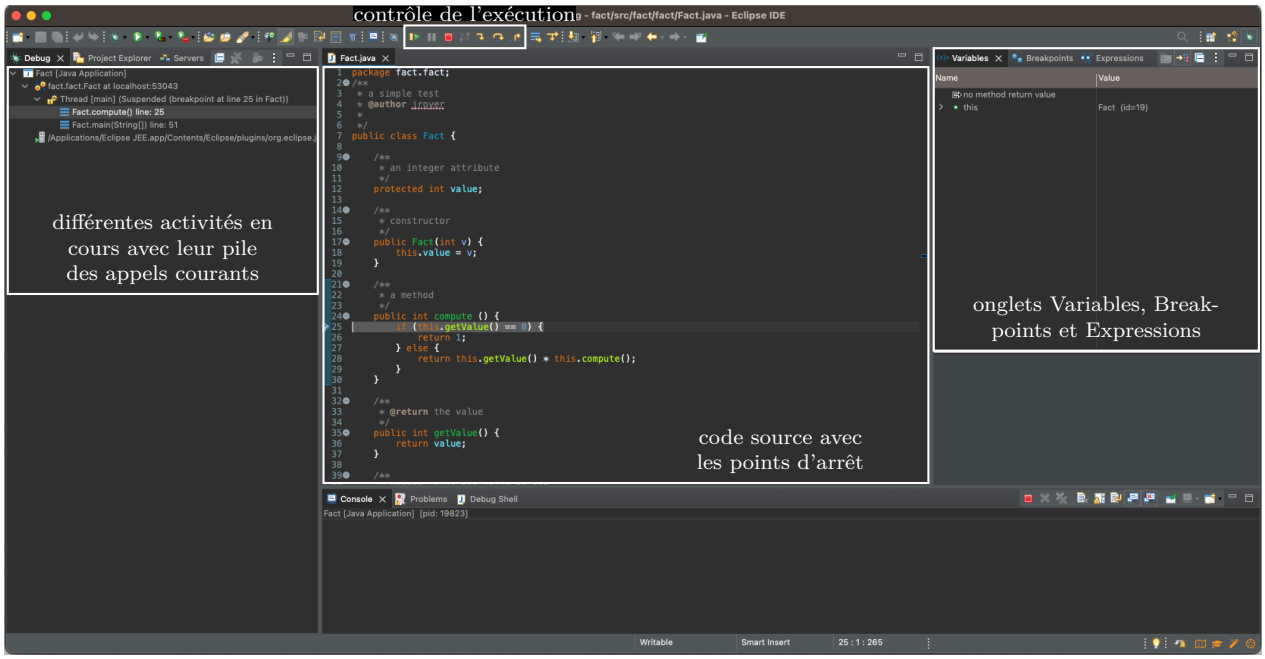

FIGURE 2 – La vue debug

la colonne de gauche de l'éditeur sous la forme d'un point bleu. On les ajoute en cliquant dans la colonne avant ligne de code sur laquelle on souhaite mettre un point d'arrêt.

Lors de l'utilisation du dévermineur le programme s'arrêtera à cet endroit et l'utilisateur peut alors faire certaines actions. Les options disponibles sur un *breakpoint* le sont via un menu contextuel : suppression, désactivation ou pose d'une condition.

2. Dans un deuxième temps il faut lancer l'application de débogage. L'accès au mode `debug` se fait en sélectionnant le point d'entrée de l'application à observer et soit par le menu contextuel item `debug` soit par la barre d'outils en utilisant le bouton avec un petit insecte :

3. Après l'exécution si un point d'arrêt est atteint alors la perspective debug devient active et diverses opérations sont possibles. La figure 2 montre cette perspective :
   - Tout en haut, la barre des outils contient plusieurs flèches permettant d'exécuter en mode pas à pas notamment.
   - En haut à gauche, les différentes activités en cours et leur pile des appels courants : i.e. les différentes activités qui ont conduit jusqu'à ce point d'arrêt.
   - Au centre, votre source code avec les éventuels point d'arrêts visible sur ce code dans la colonne de gauche.
   - En haut à droite, trois onglets :
     - `Variables` permet d'observer les objets et les variables et de poser des conditions d'observation (*watchpoints*).
     - `Breakpoints` montre les points d'arrêt et les points d'observation.
     - `Expressions` est une calculatrice symbolique permettant d'évaluer des expressions dans le contexte courant.
   - Tout en bas, la console habituelle.

2 Un petit essai

1. Récupérer sur Moodle le fichier `Fact.java` qui définit une classe simple pour calculer factorielle de 3.
2. L'exécuter et constater un « débordement de pile » (*stack overflow*).
3. Double-cliquer dans la colonne de gauche sur la ligne `if (this.getValue() == 0)` (ligne 25). Si vous voyez apparaître un point bleu, vous avez gagné car vous avez placé un point d'arrêt, sinon il faut recommencer !
4. Lancer le mode debug et utiliser plusieurs fois la touche `step over`  .
5. Que constate-t-on sur le déroulement dans la pile d'exécution et sur la valeur de l'attribut dans l'objet `this` ?
6. Dans la vue d'affichage on peut visualiser des valeurs primitives, des objets et leurs attributs ou des collections et leurs éléments.
7. En déduire ce qui manque, corriger le programme et vérifier à nouveau avec le dévermineur que le comportement est acceptable.

3 Explications

Dans la perspective debug, il y a trois boutons importants :

— *step into*  : continue l'exécution de la ligne et s'arrête sur la première ligne de code de la première méthode contenu dans la ligne. Vous pouvez constater un empilement dans la pile des appels. Permet de voir le code en profondeur.

— *step over*  : exécute la ligne courante et s'arrête à la suivante. Sans empilement.

— *step return*  : exécute la ligne et ce qui suit jusqu'à la prochaine instruction **`return`** de la méthode. Dépile.

4 Mise en application

Vous trouverez sur Moodle une archive `bad4debug.zip` qui contient un modèle du réseau de Petri pour le Fil Rouge. Mais ce n'est pas une bonne solution à votre problème, juste pour expérimenter le dévermineur. Mais le résultat n'est pas vraiment conforme aux prévisions, je n'arrive pas à trouver mes erreurs. *Help me, you're my only hope !*

1. Récupérer le code et l'installer sous Eclipse.
2. Essayer de trouver ce qui ne vas pas en utilisant le dévermineur.
3. Corriger et tester.
4. Compléter le fichier README avec votre nom et vos remarques sur les erreurs trouvées.

4.1 À vous de jouer !

Pour commencer une première erreur est facile à trouver.

1. Exécuter le `Test2Debug`, vous obtenez une erreur d'index dans un tableau.
2. Placer un point d'arrêt ligne 106 de la classe `PetriNet`.

3. Lancer le dévermineur.
4. C'est l'occasion de jouer avec les boutons "steps" mais clairement le "step into" va vous mener dans tous les méandres du code y compris les classes de la bibliothèque standard. Le *step over* avance ligne par ligne ce qui est plus intéressant pour vous et après quelques clics vous devriez avoir une idée du problème. Finalement le *step return* permet d'aller un peu plus vite pour atteindre l'erreur.
5. Après quelques *step over* regardez la valeur de "`t`", comprendre et corriger

Mais vous constatez cette fois que le message de la dernière ligne du test est suspect.

1. Pourquoi la réponse `true` vous semble fausse ?
2. Placer un point d'arrêt quelque part dans la méthode `testEffect`.
3. Trouver ce qui ne va pas et corriger le problème.
4. Une lecture attentive du code aurait dû vous mettre sur la piste...

Et il y a sûrement bien d'autres erreurs, n'hésitez pas à les chercher et à les corriger.

5 Pour aller plus loin

Pour aller plus loin, essayez d'expérimenter les items *facultatifs* suivants.

— Le dévermineur s'arrête sur un point d'arrêt ou une exception non capturée.
— Il est possible de définir un point d'arrêt sur une exception interceptée, non intercepté ou dans les deux cas. Pour cela il faut utiliser le bouton *Add Java Exception Breakpoint* dans l'onglet des points d'arrêt.
— Sur un point d'arrêt, il est possible de configurer le nombre d'occurrences avant que celui-ci ne soit pris en compte ou encore une condition d'activation. Pour cela, il faut éditer les propriétés du point d'arrêt ou regarder dans la vue des points d'arrêt la description de celui-ci.
— La vue « Expressions » vous permet de calculer la valeur d'une expression en fonction des variables dans les traitements en cours de débogage.