Tutoriel de Compléments en Programmation Orientée Objet : Prise en main d'Eclipse

Ce TP est à voir comme une série d'exercices simples permettant de faire le tour de quelques fonctionnalités très utiles de l'IDE [1] Eclipse. Il est possible que tout ne soit pas toujours expliqué en détails, auquel cas, utilisez votre moteur de recherche (notez qu'il y a un moteur intégré dans la barre d'outils d'Eclipse !) ou bien n'hésitez pas à me demander de l'aide.

Au fur et à mesure que vous explorez les menus, notez les raccourcis clavier qui vous semblent utiles. Vous gagnerez du temps pour la suite.

Exercice 1 : Avant de commencer

1. Commencez par ouvrir Eclipse (commande `eclipse`).
2. Créez éventuellement un espace de travail (*workspace*), ou bien ouvrez en un existant.
3. Fermez l'éventuel écran de bienvenue.
4. Vérifiez que la perspective Java est sélectionnée (ń Java ż à droite de la barre d'outils).

Dans la suite, pensez à enregistrer les fichiers (Ctrl+S) après chaque modification, afin qu'elle soit aussitôt prise en compte dans l'ensemble du projet.

Exercice 2 : Créer le projet et ses composants

1. Créez un projet Java (peut se faire depuis le menu ń File ż puis ń New ż, ou bien depuis la première icône de la barre d'outils, ou bien depuis le menu contextuel de l'explorateur de paquetages.).
2. Créez quelques paquetages dans le projet (mêmes endroits dans l'interface graphique). Remarque : les paquetages ont par convention des noms tout en minuscules. [2]. Supposons pour la suite que vous avez créé `package1` et `package2`.

   Remarquez qu'on peut créer un package à l'intérieur d'un autre package.
3. Créez des classes (toujours en passant par les menus) : en dehors des paquetages (c.-à-d. directement dans `src`) et dans chacun des paquetages. Supposons que vous avez créé `MaClasse` dans `src`, puis `MaClassei` dans `packagei` (i=1,2).
4. Regardez comment Eclipse pré-remplit chaque fichier créé. À quoi sert la ligne ń `package blabla;` ż au début du fichier ?

Exercice 3 : Remplir les classes

1. Ouvrez la classe `MaClasse`.
2. Dans les accolades de la classe, écrivez ń `main` ż, puis faites Ctrl+espace. Eclipse vous propose alors de genérer la méthode `main()`. Choisissez cette option.
3. Si votre code est mal indenté, faites ctrl+maj+F. Eclipse réarrange le code.
4. Ajoutez un affichage (`println()`) à l'intérieur du main, puis exécutez (ctrl+F11, ou bien menu ń Run ż, ń Run ż, ou bien l'icône en forme de rond vert avec triangle blanc). Eclipse vous propose de sauvegarder avant d'exécuter (toujours une bonne idée !).

1. Integrated Development Environment
2. Pour les conventions de nommage en Java, vous pouvez consulter le document suivant : `http://www.loribel.com/java/normes/nommage.html`.

5. Avez-vous remarqué qu'Eclipse propose toute une liste de choix dans un menu surgissant, si vous faites une pause après avoir écrit un des ń . ż de `System.out.println()` ? (Pratique pour retrouver les noms des méthodes et gagner du temps pour les écrire.). À tout moment, en cours de saisie, vous pouvez aussi lancer l'autocomplétion avec Ctrl+espace.
6. Déclarez (à la main) des attributs dans les différentes classes.
7. Générez leurs accesseurs (menu ń Source ż, ń generate getters and setters ż).
8. Dans `MaClasse`, déclarez un attribut statique `champ1` de type `MaClasse2`.
9. Remarquez qu'Eclipse souligne `MaClasse1` en rouge. Survolez la zone soulignée avec la souris, et regardez ce qu'Eclipse propose. Choisissez ń import MaClass1 (package1) ż.
10. Remarquez que le problème est maintenant réglé, et que le fichier a maintenant une ligne `import package1.MaClasse1;` dans son préambule.
11. Dans le `main()` (de `MaClasse`), initialisez l'attribut `champ1` en instanciant un objet (`new MaClasse1()`).
12. Ajoutez l'affichage `System.out.println(champ1);`. Exécutez. Que voyez-vous ?
13. Allez dans `MaClasse1`. Ajoutez quelques attributs non-statiques.
14. Allez dans ń Source ż, ń Generate Constructor using Fields... ż, validez. Remarquez le nouveau constructeur ajouté.
15. Retournez dans `MaClasse`, remarquez le problème, analysez et corrigez-le (on peut survoler et choisir la première solution). Exécutez à nouveau, rien ne devrait vraiment avoir changé.
16. Retournez dans `MaClasse1`, faites ń Source ż, ń Generate toString()... ż. Validez, exécutez à nouveau ? Est-ce mieux ?
17. Maintenant, on veut renommer ń `MaClasse1` ż. Placez votre curseur sur le mot `MaClasse1`, allez dans le menu ń Refactor ż, faites ń Rename... ż (ou bien faites un clic-droit sur le mot, puis idem, ou bien faites ctrl+alt+R), choisissez un autre nom. Observez que toutes les occurrences de ce nom ont été changées en conséquence (y compris le nom du fichier `.java`).

Exercice 4 : Déboguer

1. Dans votre main, ajoutez l'instructions suivante :

```
1    for (int i=10; i >= 10; i+=3) System.out.println(i);
```

   Exécutez. Que se passe-t-il ? Arrêtez l'exécution en appuyant sur le carré rouge en haut à droite du cadre où s'exécute le programme (sinon ń Run ż, ń Terminate ż).
2. Faites un clic droit dans la marge (sur le numéro de ligne) sur la ligne de l'instruction `for`. Choisissez ń Toggle Breakpoint ż.
3. Maintenant, exécutez le programme en mode débug : c'est l'icône de la barre d'outils à gauche de celle qui exécute normalement (sinon, menu ń Run ż, ń Debug ż ou F11). Ceci va ouvrir au passage la perspective de déboguage (remarquez le mot ń Debug ż en haut à droite. On peut revenir vers la perspective ń Java ż en cliquant dessus).
4. Le programme est maintenant en pause sur l'instruction qui a le breakpoint. Pour avancer pas-à-pas dans l'exécution, on peut utiliser les touches F5 (qui fait les pas en entrant dans les méthodes) et F6 (qui saute jusqu'au retour de méthode, si jamais l'instruction courante en appelle une). Appuyez plusieurs fois sur F6 pour voir le programme avancer.

5. Dans le cadre en haut à droite, regardez l'onglet ń Variables ż, et remarquez qu'on peut observer la valeur de chaque variable dans la portée courante, afin d'aider à déceler l'origine du bugx.