Introduction à Eclipse

Ce document présente le minimum pour développer en Java avec Eclipse [1], une plateforme de développement libre et gratuite, développée dans le cadre d'une association fondée par IBM.

Exercice 1 : Démarrage Eclipse

1.1. *Lancer Eclipse.* Pour lancer Eclipse à l'ENSEEIHT, choisir *Eclipse JEE* dans le menu *Applications / Programmation*.

Lors du premier lancement, Eclipse affiche plusieurs icônes (*Overview*, *Tutorials*, *Samples*, *What's New*) qui donnent accès à différentes documentations. Sélectionner *Workbench*. Le *workbench* correspond à l'espace de travail dans lequel seront conservés les projets créés sous Eclipse. Il correspond à un dossier dans le système de gestion de fichiers. On peut garder le dossier proposé (`workspace_jee` dans le dossier racine). La fenêtre de la plateforme Eclipse apparaît.

1.2. *Sélectionner une perspective.* Une perspective décrit une configuration d'Eclipse, c'est-à-dire les vues affichées par Eclipse. Le nom de la perspective courante est affiché en haut à droite.

Pour changer la perspective, on peut faire *Window / Open Perspective*. On peut également cliquer sur l'icône à côté du nom de la perspective courante. Les perspectives disponibles incluent *Java* pour l'édition de programmes Java, *Java Browsing* pour la consultation de projet Java, *Debug* pour la mise au point de programmes...

1.2.1. Sélectionner successivement les perspectives *Java Browsing*, *Debug* puis *Java* et constater que les éléments affichés changent.

1.2.2. Sélectionner la perspective *Java*. Elle est composée des éléments suivants (figure 1) :

- un *Package Explorer* (à gauche) : il liste les projets créés et leur contenu sous la forme d'une arborescence ;
- une *Outline* (à droite) : il donne la vue synthétique d'un élément par exemple, les attributs et les méthodes de la classe sélectionnée ;
- des onglets (*Problems*, *javadoc* et *Declaration*) dans la partie inférieure ;
- une partie centrale utilisée pour éditer les fichiers Java.

Exercice 2 : Faire un projet Java simpliste

Voyons comment créer un projet Java, y ajouter une classe, la compiler et l'exécuter.

2.1. *Créer un projet Java.* Faire *File / New / Project* [2], sélectionner *Java Project* et faire *Next*. Saisir le nom du projet : *bonjour*. Faire *Next*. Décocher *create module-info.java file*. Faire *Finish*. Répondre *Don't Create* à la boîte de dialogue *Create moudle-info.java*. Le projet est créé et apparaît dans l'explorateur (partie gauche d'Eclipse).

2.2. *Créer une classe Bonjour.* Pour créer une classe, on peut sélectionner le projet et choisir *New* puis *Class* dans le menu contextuel. On doit alors donner le nom de la classe, ici `Bonjour`. Comme cette classe contiendra une méthode principale, on peut cocher l'option correspondante.

1. `http://www.eclipse.org`
2. On peut aussi utiliser le menu contextuel (clic droit) de l'explorateur et faire *New / Project*.

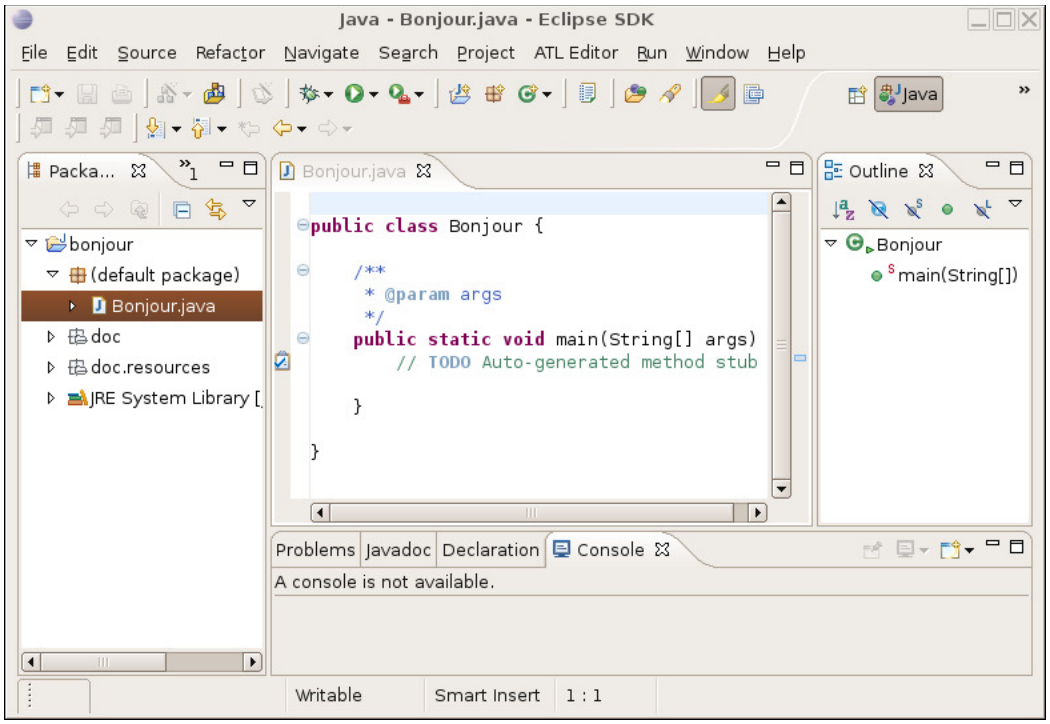

FIGURE 1 – Eclipse 3.1 avec la perspective Java.

Faire *Finish*. Le fichier `Bonjour.java` apparaît dans le projet (dans le paquetage par défaut) et dans l'éditeur (partie centrale d'Eclipse) comme sur la figure 1.

Le squelette de la classe est engendré (y compris la documentation).

Notons également que la classe `Bonjour` et la méthode `main` apparaissent dans la *Outline*.

On peut aussi créer une classe en sélectionnant le bouton marqué C dans la barre de boutons.

2.3. *Compléter le code de `main`.*

2.3.1. Compléter le code de `main` avec l'instruction `Sytem.out.println("Bonjour")` (en oubliant [3] le « s » de `System` et le point-virgule).

Eclipse vérifie le programme et signale une erreur en mettant une croix dans la ligne. Un message d'erreur est affiché dans l'onglet *Problems* (mis à jour lors de la sauvegarde du fichier).

2.3.2. L'erreur signale le point-vigule manquant. Corriger l'erreur.

Une erreur est encore signalée. On remarque que derrière la croix rouge, une ampoule apparaît. C'est que Eclipse a des solutions à proposer pour corriger cette erreur. Cliquer sur l'ampoule (ou le texte souligné) permet de les voir. Attention à lire attentivement ces propositions pour choisir la bonne, voire constater qu'aucune ne convient ! Ici, c'est l'avant dernière qui convient (« Rename in file »).

Après cette correction, il ne doit plus y avoir d'erreur !

2.4. *Compiler l'application.* L'application est compilée à chaque sauvegarde. Il n'est donc pas nécessaire de lancer explicitement une compilation.

3. Eclipse propose la *completion* automatique avec CTRL-Espace pour éviter ce type d'erreur.

2.5. *Exécuter l'application.* Pour exécuter le programme, il est conseillé de sélectionner la classe que l'on souhaite exécuter (et qui doit contenir une méthode principale). On peut la sélectionner dans l'explorateur, l'éditeur ou le *outline*. Ici, nous n'avons que la classe `Bonjour`.

En utilisant ensuite le menu contextuel, sélectionner *Run As / Run*. Sélectionner *Java Application*. Ceci lance l'exécution. Le résultat s'affiche dans la vue « Console » (en bas).

Ceci a créé une configuration. On peut la voir en faisant *Run As / Run* puis *Run configurations...*. On constate qu'il y a une configuration qui s'appelle *Bonjour* (nom de la classe). Le zone de saisie *Main class* est initialisée avec le nom de cette classe. On peut utiliser l'onglet *Arguments* pour définir les arguments de la ligne de commande du programme (récupérés par main) ou de la machine virtuelle (par exemple -ea).

Pour les exécutions suivantes, il suffit de choisir la configuration souhaitée et lancer son exécution.

Exercice 3 : Exécuter avec le raccourci

Une fois les configurations créées, il est possible de relancer l'exécution de la dernière configuration en cliquant sur la flèche « lecture » dans le bouton vert de la barre de boutons. En cliquant sur la flèche à droite, on peut sélectionner une autre configuration ou faire *Run* pour obtenir la boîte de dialogue consacrée aux configurations.

Exercice 4 : Importer les fichiers sources d'une archive

Pour intégrer dans un projet Eclipse les fichiers sources d'une archive (.tar, .jar ou .zip), il faut commencer par créer un projet Java (s'il n'existe pas), sélectionner ce projet, puis dans le menu contextuel faire *Import*, sélectionner *General*, puis *Archive file*, faire *Next*, sélectionner l'archive (avec *Browse*), faire *Finish*. Les fichiers de l'archive ont été ajoutés au projet.

Attention : Si l'archive contient des dossiers, ils sont conservés et traités comme des paquetages dans le projet Java. Si le dossier n'était là que pour regrouper les fichiers de l'archive, il faudra déplacer son contenu dans le paquetage par défaut (ou la racine du projet).

Exercice 5 : Utiliser dans un projet Eclipse les fichiers d'un dossier existant

Si les sources existent déjà dans un dossier, par exemple un dossier SVN[4], la solution la plus simple est, lors de la création d'un projet, de faire *Next* depuis le premier écran de l'assistant, sur le second écran choisir *Link additional source* et ajouter le dossier externe.

Dans un projet Eclipse existant, on peut utiliser en dossier externe en faisant grâce à un clic droit sur le projet *Import... / File System / Next*. On sélectionne alors le dossier (*From directory*), la destination (*Into filder*), on clique sur *Advanced* puis *Create links in workspace*. Il faut sélectionner les ressources (les fichiers) à considérer avant de cliquer *Finish*. S'il s'agit d'un dossier contenant des sources Java, il faudra les ajouter au *SOURCEPATH* : on fait un clic droit sur le nouveau dossier contenant les sources Java, puis on sélectionne *Add to source path*.

Exercice 6 : Écrire et exécuter des tests JUnit

6.1. *Préparer le projet Java.* Créer un nouveau projet et y importer les fichiers du TP 2.

6.2. *Le corriger.* Le programme de test JUnit contient des erreurs. Il manque la bibliothèque JUnit. On peut ouvrir le fichier et dans les corrections proposées choisir *Fix Project Setup*.

4. Une meilleure solution serait bien sûr d'utiliser un greffon pour connecter le dépôt SVN au projet Eclipse.

6.3. *Exécuter un test unitaire.* Exécuter le programme en le sélectionnant dans le *Package Explorer*, puis par un clic droit en faisant *Run as* > *Junit Test*.
6.4. *Créer un nouveau test unitaire.*
6.4.1. Écrire une nouvelle classe de test en faisant *New* > *JUnit Test Case*. Choisir `MathTest` pour le nom de la classe (*Name*) et choisir la classe `Math` [5] pour la classe à tester (*Class under test*). Faire *Next*, sélectionner la méthode `sqrt(double)` et faire *Finish*. La classe de test est créée.
6.4.2. *Voir les erreurs.* Lire la classe `MathTest`, puis l'exécuter. Elle contient des erreurs et la barre d'exécution du test est rouge. Quand on sélectionne un test, la trace des exceptions apparaît en bas à gauche (vue *Failure Trace*). Quand on clique sur l'icône la plus à gauche de cette vue (*Show Stack Trace in Console View*), la trace des appels s'affiche dans la console. La première ligne explique pourquoi le test a échoué.
6.4.3. *Corriger le test.* Modifier le test `testSqrt` pour vérifier que la racine carrée de 4 vaut 2 en utilisant `assertEquals` sur les réels. L'exécuter (et corriger si le test [6] échoue).

Exercice 7 : Configurer le Build Path
Le build path indique à Eclipse où trouver les bibliothèques ou, plus généralement, les fichiers .class. Prenons deux exemples.
7.1. *Utiliser une bibliothèque connue d'Eclipse.* Pour ajouter JUnit, par exemple, on peut faire un clique droit sur le projet, sélectionner *Build Path* puis *Add Libraries....* Dans celles proposées, il y a JUnit. La sélectionner et faire *Next*. Choisir la version et faire *Finish*.
Remarque : Lors de la création d'une classe de test (JUnit Test Case), soit par le raccourci, soit par le menu New, la bibliothèque JUnit sera rajoutée si nécessaire.
7.2. *Ajouter une bibliothèque à partir de son .jar.* Quand la bibliothèque est disponible sous la forme d'un fichier d'archive (.jar), on peut l'ajouter en faisant un clique droit sur le projet, puis *Build Path* > *Add External Archives....* Il suffit alors de sélectionner le fichier .jar qui sera ajouté dans *Referenced Libraries*.

Exercice 8 : Faire du *refactoring* avec Eclipse
Le *refactoring* [7] consiste à modifier une partie d'une application en répercutant les modifications sur l'ensemble de l'application. Par exemple, lorsque l'on change le nom d'une classe, il faut répercuter ce changement dans toutes les classes qui l'utilisent. Ce type de modification est pris en charge par Eclipse.
8.1. *Renommer.* Pour renommer un élément, on le sélectionne, on fait un clic droit, on choisit *Refactor* > *Rename*. Renommer la classe Point en Point2D, l'attribut x en abscisse.
8.2. *Ajouter un paramètre.* L'opération *Refactor* > *Change Method Signature...* permet de modifier la signature d'une méthode ou d'un constructeur.

Ajouter un paramètre couleur au constructeur de Point2D avec pour nom couleur, type Color et valeur par défaut Color.green. Ne pas oublier de corriger le code du constructeur !

Vérifier que la modification a bien été répercutée sur le programme de test.

5. Cliquer sur *Browse* et taper `Math` dans la zone de saisie pour faire apparaître la classe.
6. L'erreur est bien dans la classe de test et non dans la classe Math !
7. La traduction française pourrait être restructuration, réorganisation du code.