Introduction (rapide) à Eclipse

Le logiciel Eclipse est un environnement intégré de développement (IDE) pour le langage Java (et d'autres langages). Gratuit, il peut être téléchargé sur le site `www.eclipse.org`. Nous utiliserons également le plugin JUnit4. De nombreux autres plugins existent. Vous pourrez par exemple peut-être être intéressé par le plugin ObjectAid qui permet de générer des diagrammes UML à partir de votre code ou Eclemma qui calcule la couverture de tests unitaires.
Il n'est pas question de présenter exhaustivement les fonctionnalités de l'outil Eclipse. Seules les principales fonctionnalités seront présentées ici, charge à vous de découvrir les autres possibilités offertes afin d'accroître encore votre efficacité dans votre travail de développement. L'intérêt d'un tel outil, quand il est bien utilisé, est de permettre au programmeur de se concentrer sur l'essentiel de son travail et de se dégager de certains petits détails techniques sans importance. Pour cela Eclipse fournit des outils facilitant et encourageant la bonne écriture de code et la bonne conception d'application. A vous d'explorer les différents menus et différentes possibilités proposées par les fenêtres de dialogue.

Au premier lancement d'Eclipse vous aurez probablement une fenêtre "Workspace Launcher" qui apparaît, elle propose un emplacement par défaut pour le *Workspace*. Le *Workspace* est un dossier où Eclipse stocke un certain nombre d'informations et de fichiers qui lui sont utiles.
Vous pouvez ou non modifier l'emplacement proposé puis cochez la case située en dessous et intitulée Use this as default and do not ask again. Vous pouvez ensuite valider par OK. Cette fenêtre ne devrait plus apparaître lors des lancements ultérieurs d'Eclipse.
Toujours uniquement pour le premier lancement, si vous arrivez sur une fenêtre "Welcome". Fermez cette fenêtre. Vous arrivez alors sur l'espace de travail. Si elles apparaissent, fermez la fenêtre "Tasks list" (en bas à droite) et "Font and Colors" (sur la droite) qui ne nous seront pas utiles, vous n'en aurez pas besoin.

Créer un projet avec Eclipse

> *Vous n'aurez pas besoin de cette fonctionnalité pour vos projets puisque vous devez les créer avec Maven. La section suivante vous présentera l'importation de projets existants.*

Eclipse fonctionne par projet. Pour créer un projet directement depuis Eclipse faites File→New→Java Project..., ou dans la fenêtre Package Explorer (sur la gauche) faites clic droit puis New, etc. Choisissez un nom de projet.

Une fenêtre de dialogue s'ouvre. Dans la seconde moitié de celle-ci la partie vous trouvez une zone Project layout. Vous pouvez y demander à distinguer les dossiers pour les fichiers de sources et de classes. C'est évidemment ce qu'il faut faire. Si ce n'est pas déjà fait, sélectionnez donc Create separate source and output folders, puis cliquez sur Configure Defaults... et dans les Folders nommez les dossiers de source et de classes (par exemple src et classes). Cliquez Ok.

De retour dans la fenêtre "New Java Project", en décochant Use default location ("en haut" dans la fenêtre) vous avez la possibilité de ne pas placer les fichiers du projet dans un dossier workspace géré par Eclipse mais de préciser le dossier de travail. **Prenez cette seconde option** et indiquez un dossier dans votre espace de travail où seront rangés les fichiers de ce projet (vous pouvez créer un nouveau dossier via Eclipse via Browse...).
Faites Next, vous pouvez alors vérifier dans l'onglet Source que le dossier src a bien été ajouté[1]. Vous pouvez aussi ajouter un dossier `test` à vos sources pour y placer vos classes de tests.
Dans l'onglet Libraries vous devriez voir apparaître la référence à la librairie du jdk utilisé [2].
Ces données peuvent être modifiées par la suite en accédant aux Properties d'un projet (clic droit sur le nom du projet).
Enfin cliquez Finish. Si vous avez un message vous proposant de passer en mode java perspective, acceptez.
Le projet est créé et vous le voyez apparaître, ainsi qu'un dossier src dans la zone de gauche de l'écran.

Importer un projet existant

Vous pouvez également importer dans Eclipse des projets existants qui n'auraient pas été créés dans cet environnement, que ce soit de « simples » projets java ou des projets générés avec Maven.
Dans le cas où vous voulez créer un projet dans Eclipse en partant de code java existant, il vous suffit de procéder comme pour la création de projet décrite ci-dessous. Simplement, lors de cette création vous préciserez que le dossier de travail est le dossier parent du dossier `src` existant. Ce dossier sera automatiquement considéré comme dossier source.
Pour importer un projet Maven existant dans Eclipse, il vous suffit de sélectionner File→Import... puis dans la fenêtre qui s'ouvre de choisir Maven et enfin sélectionner le choix Existing Maven Project. Il ne vous reste plus qu'à préciser où se trouve le dossier racine de ce projet.

[1] Sinon il faut l'ajouter en choisissant Create new source folder.

[2] c'est à cet endroit que l'on définit le `CLASSPATH` du projet, en ajoutant éventuellement des références vers d'autres libraires tels que des "jars externes" par exemple.

Créer un paquetage

Pour créer un paquetage, placez-vous sur l'icône du dossier src, cliquez droit puis New et Package. Nommez ce paquetage, `faune` par exemple.

Créer une classe/interface

Sur l'icône du paquetage, cliquez droit et New puis choisissez Class ou Interface.
Créez une classe `Animal` puis une interface `Predateur` : il suffit de donner le nom et de cliquer sur Finish. Prenez le temps de jeter un œil aux possibilités offertes dans cette fenêtre de dialogue de création de classe.
Des éditeurs pour ces entités s'ouvrent automatiquement. Créez une signature de méthode `public void chasse()` pour `Predateur`, et pour la classe `Animal` deux attributs, `poids` un entier et `nom` de type `String`, un constructeur avec deux paramètres représentant le nom de l'animal et son poids et une méthode `public void crie()`[3].
Dans la fenêtre d'édition de cette classe, cliquez droit puis Source (ou Shift+Alt+S), puis Generate Getter and Setter, une fenêtre s'ouvre vous permettant de provoquer la génération automatique au choix des méthodes `getNom` et `setNom`. Créez les. Faites de même pour l'attribut `poids` mais en ajoutant que le getter.
Dans la partie droite de l'IDE vous pouvez consulter l'Outline du type édité (attributs, méthodes, etc.), il est possible via cette fenêtre d'accéder directement à un élément en le sélectionnant.
Sauvegardez les fichiers édités.

Nous allons créer une classe `Loup` qui implémente l'interface `Predateur` et hérite de `Animal`.
Pour cela, lors de la création, à l'aide des boutons fournis ; Browse... pour la super-classe et Add... pour l'interface remplissez les champs Superclass et Interfaces. Dans la partie inférieure, activez Constructors from superclass et vérifiez que Inherited abstract methods est activée. Cliquez Finish.
Vous remarquez la génération automatique du constructeur et de la méthode `chasse`.
Recommencez en créant une classe `Mouton` qui hérite d'animal et dispose d'une méthode `tondre` et une classe `Requin` qui hérite d'`Animal` et implémente `Predateur` et une autre interface `Aquatique` qui impose une méthode `nage()`.

Dans la classe `Requin`, faites clic droit puis Source et Override/Implement methods... qui vous permet de choisir parmi les méthodes des super-types celles que vous souhaitez définir ou surcharger. Par exemple choisissez la méthode `crie` de `Animal`. Vous pouvez alors modifier le code pour qu'un requin crie en silence....

Autre création et code

Créez un second paquetage `humain`. Créez "dedans" une classe `Enfant` dans laquelle vous créez un attribut de type `Animal` : `private Animal animalPrefere`. Ajoutez les getter et setter pour cet attribut.
Vous remarquez un symbole qui apparaît dans la marge gauche du code. Celui-ci signale une erreur (la croix blanche sur fond rouge). Le code que vous saisissez est analysé au fur et à mesure et en cas d'erreur la source d'erreur (estimée) est soulignée de rouge dans le code (ici `Animal`). En plaçant le curseur au-dessus de ce signe le message d'erreur supposée est affiché.
La petite ampoule jaune dans la marge mentionne qu'une proposition d'aide de correction est disponible. Cliquez sur l'ampoule en vis-à-vis de la ligne de déclaration de l'attribut `animalPrefere`, les suggestions de correction s'affichent. Le premier choix de correction suggère l'`import`, c'est ce qu'il faut faire donc appliquez cette correction en la sélectionnant. Le code nécessaire est alors ajouté.

Ajoutez maintenant à la classe `Enfant` une méthode `public void ecoute()`.
Dans le corps de cette méthode tapez "`this.`"(avec le point). Si vous patientez un (très) bref instant après la saisie du point, les complétions possibles (càd autorisées dans ce contexte donc pour le type de `this`) apparaissent par ordre alphabétique, vous pouvez les parcourir et choisir celle sélectionnée par la touche entrée. Sinon, au fur et à mesure que vous tapez des lettres les propositions se réduisent.
Ici choisissez `animalPrefere`. Ensuite tapez `.` (point), à nouveau les complétions apparaissent, choisissez `crie`.

Exploration de code

Dans un fichier de classe, en se plaçant sur le nom de la classe ou un nom de méthode, un clic droit puis le choix Quick Type Hierarchy, ou son raccourci Ctrl+T, fait apparaître pour une classe la hiérarchie des classes (super et sous–classes) et pour une méthode les éventuelles surcharges qui lui sont associées. Il est alors possible d'accéder directement aux éléments proposés. Essayez avec le mot `Loup` dans l'entête de déclaration de la classe ou avec la définition de la méthode `crie` de `Requin`.

Maintenant, placez votre pointeur de souris sur l'appel à la méthode "`crie()`" dans la méthode `ecoute`. Faites Ctrl-clic gauche[4], vous accédez alors directement au code de définition de cette méthode. Il en est de même si vous opérez sur un nom de classe ou d'interface.

[3] Mettez quelque chose comme `System.out.println(this.nom+" crie")` dans le corps...
[4] ou cliquez droit puis Open declaration ou encore F3.

Après un clic droit sur un élément du code (classe, méthode, attribut, etc.), le menu qui s'ouvre offre différentes possibilités. Notamment le choix References→Workspace permet de connaître tous les endroits du code ou cet élément apparaît (cette commande a pour raccourci SHIFT+CTRL+G). Ces occurrences sont affichées dans une fenêtre à part de nom Search (en bas de la fenêtre de l'IDE généralement) et sont accessibles par un clic. Essayez sur la définition de la méthode `crie` de la classe `Animal`, vous retrouvez son usage dans la classe `Enfant`.

Javadoc Placez votre curseur dans la signature de la méthode `ecoute`. Cliquez droit puis Source, puis Generate Element Comment (ou SHIFT+ALT+J). Le "template" pour la javadoc est automatiquement inséré. Testez également avec une méthode qui a des paramètres ou une valeur de retour.
Le menu contextuel (celui obtenu par clic droit) offre beaucoup d'autres fonctionnalités utiles. Elles sont à découvrir par vous-même.

JUnit

Création de classes de tests. Pour générer une classe de tests, il suffit d'un clic droit sur le nom de la classe à tester puis de choisir New puis JUnit Test Case. Dans la fenêtre de dialogue qui apparaît vous pouvez définir différents éléments pour la classe de test, notamment son nom, mais aussi le dossier dans laquelle vous la définissez.
Pour pouvoir placer les classes dans le dossier `test`, il vous faut préalablement avoir ajouter ce dossier aux sources de votre projet. Si ce n'est pas encore le cas, il vous faut cliquer droit sur votre projet, choisir Properties (dernier choix en bas de la liste), sélectionner la rubrique Java Build Path et dans l'onglet Source, ajouter le dossier `test` grâce à Add Folder... (vous pouvez même créer le dossier à ce moment là).
Après avoir vérifié que les informations dans cette fenêtre vous conviennent, cliquez sur Next. Vous pouvez alors choisir les méthodes de la classe pour lesquelles vous souhaitez écrire des tests. Les squelettes des méthodes de tests seront alors automatiquement générés une fois que vous aurez cliqué sur Finish.
Lors de la création de la première classe de test du projet, l'ajout de la bibliothèque `junit4` au projet est proposé, il faut évidemment accepter.
Vous pouvez maintenant modifier cette classe pour définir le contenu de vos méthodes de test.

Exécution des tests. Pour exécuter les tests d'une classe, il suffit de cliquer droit sur le nom de la classe de test (ou du fichier), de choisir Run As puis Junit Test. Un nouvel onglet JUnit apparaît alors à côté de l'onglet Package Explorer. Vous pouvez y observer le rapport de vos tests, notamment la barre verte ou rouge selon que vos tests ont été passés avec succès ou non. Vous y trouvez également des boutons pour manipuler vos tests.

Exécutez le code

Créez une classe Main et dans la partie inférieure de la fenêtre de création de classe cochez la demande création de la méthode `main`, ensuite complétez le code de cette méthode : contentez-vous de créer un animal et de lui faire invoquer la méthode `crie` puis créer un enfant et dont l'animal créé sera l'animal préféré puis d'invoquer la méthode `ecoute`.

Dans la zone des fichiers, vous pouvez simplement cliquer droit sur cette classe `Main` puis dans le menu choisir Run As puis Java Application.

Vous pouvez également créér une cible permanente. Pour cela, sélectionnez dans le menu Run→Run Configurations (ou icône "lecture" – flèche blanche dans un rond rouge – dans la barre du bouton), choisissez Java Application.
Cliquez sur le bouton New[5]. Le champ Project: doit être déjà à jour. Cliquez sur Search... , les classes du projet contenant un main sont proposées (ici il n'y en a qu'une, elle a donc dû être proposée par défaut).
Validez et cliquez Run. Le programme est alors exécuté. La trace apparaît dans une fenêtre console dans la partie inférieure de la fenêtre de l'IDE.

Pour exécuter une cible Maven, il vous suffit de sélectionner le fichier `pom.xml`, de cliquer droit sur ce fichier puis dans le menu choisir Run As puis Maven build... et enfin de préciser dans la fenêtre qui s'ouvre dans Goals que votre cible est `package`. Vous avez également la possibilité de créer une configuration (Run configurations...) pour cette cible.

Refactoring

Des outils vous aident à réorganiser votre projet : déplacer des classes, modifier des noms de méthodes, etc.
Dans le code de `Animal.java`, sélectionnez la méthode `crie`, cliquez droit puis Refactor (SHIFT+ALT+T) puis Rename... (SHIFT+ALT+R) et changez le nom de la méthode en `parle` par exemple et validez. En recherchant les occurrences de cette méthode (SHIFT+CTRL+G dans la déclaration) vous pouvez vérifier dans `Enfant` que le code de la méthode `ecoute` a été adapté, ainsi que dans la méthode `main`.
Vous pouvez également changer la signature d'une méthode.
Dans l'explorateur de paquetages sur la gauche de l'IDE, sélectionnez `Loup.java`, cliquez droit puis Refactor, puis Move... et choisissez le paquetage `humain`[6]. La classe est déplacée et les modifications nécessaires ont été faites,

[5] c'est l'icône en haut à gauche avec un petit "+" jaune
[6] L'homme est un loup pour l'homme...

notamment la mise à jour des `import`. Un glisser/déposer des icônes de fichiers d'un paquetage à l'autre est également possible.

Jar

La génération de jar se fait via la commande Export... du menu File. Dans la rubrique Java choisissez JAR File. Les différents écrans successifs (via Next>) qui sont proposés permettent de définir le contenu de l'archive, notamment la Main-Class.
Explorez cette fonctionnalité en générant un jar exécutable avec le `main` de `Main`. Testez-le.
Evidemment avec Maven, votre jar est généré par la cible `package`.