

UE Conception Orientée Objet

TP Présentation Eclipse

Le logiciel ECLIPSE est un environnement intégré de développement (IDE) pour le langage JAVA (et d'autres langages). Ecrit en Java, ce logiciel est gratuit et peut être téléchargé sur le site www.eclipse.org. Nous utiliserons également un plugin pour ce logiciel qui permet d'éditer des diagrammes UML, il s'agit du plugin eUML2 téléchargeable sur le site www.soyatec.fr (version "Free").

Vous trouverez la séquence de lancement du logiciel ECLIPSE en salle de TP à l'adresse : <http://www.fil.univ-lille1.fr/INTRANET/LOGICIELS/eclipse.html>.

Il faut utiliser la version eclipse-euml2 de Eclipse. Comme indiqué sur la page citée ci-dessus, modifiez (ou créez) le fichier `.bashrc` à la racine de votre dossier.

Dans un nouveau terminal, lancez ECLIPSE par `eclipse &`.

Au premier lancement d'Eclipse vous aurez une fenêtre "Workspace Launcher" qui apparaît, elle propose un emplacement par défaut pour le *Workspace*. Le *Workspace* est un répertoire où ECLIPSE stocke un certain nombre d'informations et de fichiers qui lui sont utiles.

Laissez l'emplacement non modifié et 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, vous arrivez sur une fenêtre "Welcome". Fermez cette fenêtre. Vous arrivez sur l'espace de travail. 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.

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 lors de l'ouverture d'une fenêtre de dialogue (pour la création de classe par exemple) d'examiner les différentes options proposées. De même, explorez les différents menus, certaines commandes fournissent des aides appréciables...

Créer un projet

ECLIPSE fonctionne par projet. Pour créer un projet faites `File→New→Java Project...`, ou dans la fenêtre **Package** (sur la gauche) faites clic droit puis **New**, etc.

Donnez un nom de projet, **test** par exemple.

Dans la partie **Project layout** (vers le bas de la fenêtre) vous pouvez 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** ("plus haut" dans la fenêtre) vous avez la possibilité ne pas placé les fichiers du projet dans un dossier **workspace** géré par ECLIPSE mais de préciser le répertoire de travail. **Prenez cette seconde option** et indiquez un répertoire 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 ajouté¹

Dans l'onglet **Libraries** vous devriez voir apparaître la référence à la librairie du `jdk` utilisé (1.6 priori)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'écran³.

¹Si on il faut l'ajouté en choisissant **Creat new source folder**.

²c'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.

³Si vous n'avez pas de dossier de sources, clic droit sur **test** puis **New** et **Source Folder** vous permet d'en créer un. Dans ce cas, il est possible que votre projet ait été mal configuré et qu'il lui manque le lien vers les bibliothèques Java. Il faudra alors aller dans les **Properties** du projet, puis dans **Java Build Path** et dans l'onglet **Libraries**, si **JRE System Library** n'apparaît pas, l'ajouter via le bouton **Add Library**, puis **JRE System Library** puis **Next** puis **Finish** (ouf!)

Créer un paquetage

Pour créer un paquetage, placez vous sur l'icône du dossier `src`, cliquez droit puis **New** et **Package**. Nommez ce paquetage, `pack1` par exemple.

Créer une classe/interface

Sur l'icône du paquetage, cliquez droit et **New** puis choisissez **Class** ou **Interface**.

Créez une classe **C** puis une interface **I** : 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 f()** pour **I**, et pour la classe **C** un constructeur avec un paramètre entier et une méthode **public void g()** (mettez des corps vides).

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 un autre classe, appelons la **Autre**, qui implémente l'interface **I** et hérite de **C**.

Pour cela, remplissez les champs **Superclass** et **Interfaces** (sans oublier les paquetages ! et éventuellement à l'aide des boutons fournis ; **Browse...** pour la super-classe et **Add...** pour l'interface). 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 `f`. Ajoutez un attribut entier `x` à cette classe. Sauvez. Dans la fenêtre d'édition, 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 `getX` et `setX`. Créez les.

Autre création et code

Créez un second paquetage `truc`. Créez "dedans" une classe **Timoleon** dans laquelle vous créez un attribut de type **C** : `private C att`.

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 **C**). 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 (gauche) sur cette ampoule, le premier choix de correction suggère l'import, c'est ce qu'il faut faire donc appliquez cette correction (double clic). Le code nécessaire est ajouté.

Ajoutez maintenant à la classe **Timoleon** une méthode `public void t(int i)`.

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 `att`. Ensuite tapez `.` (point), à nouveau les complétions apparaissent, choisissez `g`.

D'une manière plus général la séquence de touche `CTRL+ESPACE` permet de faire apparaître les propositions de complétion en fonction du contexte du code (nom d'exception, de méthodes, etc.).

Maintenant, placez votre pointeur de souris sur ce "g()". Faites alors `CTRL-clic gauche` (ou cliquez droit puis **Open declaration** ou encore **F3**), vous accédez alors au code de définition de cette méthode. Il en est de même si vous opérez sur un nom de classe ou d'interface.

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 où 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 `g` de la classe **C**, vous retrouvez son usage dans la classe **Timoleon**.

Javadoc Placez votre curseur dans la signature de la méthode `t`. Cliquez droit puis **Source**, puis **Generate Element Comment** (ou `SHIFT+ALT+J`). Le "template" pour la javadoc est automatiquement inséré.

Dans la classe **Timoleon**, 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 `equals` de **Object**.

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 la classe **Autre** ou avec la méthode **equals** de **Timoleon**. Le menu contextuel (celui obtenu par clic droit) offre beaucoup d'autres fonctionnalités utiles. Elles sont à découvrir par vous-même.

Exécutez le code

Placez une méthode **main** dans la classe **Timoleon** (contentez vous de lui faire afficher un message quelconque). 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** – c'est l'icône en haut à gauche avec un petit “+” jaune. 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.

Refactoring

Des outils vous aident à réorganiser votre projet : déplacer des classes, modifier des noms de méthodes, etc. **g**, cliquez droit puis **Refactor** (**SHIFT+ALT+T**) puis **Rename...** (**SHIFT+ALT+R**) et changez le nom de la méthode en **meth** par exemple et validez. Vous pouvez vérifier dans **Timoleon** que le code de la méthode **t** a été adapté. Vous pouvez également changer la signature d'une méthode. sélectionnez **Autre.java**, cliquez droit puis **Refactor**, puis **Move...** et choisissez le paquetage **truc**. La classe est déplacée et les modifications nécessaires ont été faites, 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 **Timoléon**.

UML

Nous utilisons la version “free edition” du plugin **eUML2** de la compagnie “Soyatec” (<http://www.soyatec.fr/euml2>).

Génération de l'UML à partir du code Java. Commencez par redéplacer la classe **Autre.java** dans le package **pack1**. Dans la fenêtre **Package Explorer**, sélectionnez le paquetage **pack1**, faites clic droit, **eUML2**→**Class diagram editor**. Une fenêtre apparaît proposant de choisir le type d'association à considérer, **inheritance** est proposé par défaut, c'est ce que l'on veut donc validez. Validez également si une fenêtre apparaît à propos d'annotations ajoutées pour favoriser le “reverse engineering”. Vous arrivez à une fenêtre qui vous propose de choisir les types que vous voulez voir apparaître dans le diagramme UML. Déployez les répertoire et sélectionnez toutes les classes et interfaces de **pack1**. Choisissez **ok...** Le diagramme avec les relations entre les entités apparaît. Faites de même avec le paquetage **truc**. Après un clic droit sur un diagramme de classe, le choix **View content selector...** permet de choisir le niveau de détail de présentation du diagramme (les paramètres de méthodes, visibilité des éléments en fonction de leur modificateur d'accès, etc.). Sélectionnez ce qu'il faut pour afficher les attributs avec leurs types, ainsi que les paramètres des méthodes et leurs types ainsi que le **typ** des valeurs de retour. Vous pouvez définir une fois pour toute les propriétés de visibilité des éléments dans vos diagrammes en fixant les paramètres dans **Window**→**Preferences** puis déployez **Soyatec** et **eUML2 Free Edition**→**Class/Package Diagram** et dans le choix **Element views** parcourez les onglets pour activer les éléments à afficher pour

les **Attributes** et **Methods**. On vous suggère dans les méthodes d'activer l'affichage des arguments, nom et type, et types de retour. N'oubliez pas le **Apply**.

Création de code Java à partir de diagramme UML La démarche adoptée dans le paragraphe précédent n'est pas celle d'un (bon) développeur puisqu'a priori les diagrammes de classes doivent précéder l'écriture de code. Nous allons donc maintenant, comme il se doit, commencer par la création des diagrammes puis à partir de ces diagrammes générer le code. Dans la zone de diagramme du paquetage **truc**, faites clic droit, puis **New**→**Class** (ou utilisez l'une des icônes de la barre d'outils). Nommez la classe créée **Scronch** (ou peu importe). Ensuite sélectionnez cette classe, puis faites un clic droit sur le diagramme de classe, la commande **New** permet d'y ajouter des méthodes, attributs, etc. (ou utilisez la barre d'outils qui s'affiche lorsque l'on sélectionne la classe). Ajoutez une méthode **public** **gloups** dont le type de retour est un **int**. Ajoutez un attribut (privé) **dong** de type **String** et pour lequel vous ferez générer les **accesseurs** automatiquement : tout se fait via la fenêtre proposée. Double-cliquez ensuite sur le diagramme de **Scronch**. Un éditeur avec le (squelette de) code de la classe automatiquement généré apparaît. Plus besoin de saisir ce code fastidieux... A partir du diagramme UML, ajoutez un constructeur **public** prenant un objet **String** en paramètre. Le code source est automatiquement généré. De même, dans l'éditeur, ajoutez une méthode et/ou un attribut et sauvegardez, le diagramme est modifié. Il en est de même pour les suppressions ou les changements de type, de signatures, etc. Le diagramme UML et le code source sont synchronisés. est possible via le menu contextuel du diagramme d'une classe de choisir **Model Synchronize** (ou éventuellement vérifier dans le **View Content Selector** si certains éléments ne sont pas désactivés). Créez maintenant (via l'éditeur UML!) dans ce paquetage une interface **Glop** qui définit une méthode **public int abcd()**. Toujours dans le diagramme UML, sélectionnez le bouton représentant la flèche de généralisation (trait plein, flèche à pointe creuse, ce bouton sert donc à la fois pour l'implémentation d'interface et l'héritage), cliquez sur le diagramme de la classe **Scronch** puis sur celui de **Glop**, cela pour indiquer que la classe **Scronch** implémente l'interface **Glop**. Une fenêtre s'ouvre afin de vous proposer d'ajouter dans la classe **Scronch** les méthodes induites par l'implémentation de l'interface **Glop** (ici **abcd()**). Acceptez. Le code de **Scronch** est modifié. Il faut “simplement” compléter le code de la méthode **abcd** dans **Scronch**.

NB : il est possible d'exporter sous forme d'images les diagrammes UML, commande **exporter** du menu contextuel de la zone de diagramme.

Debugger

ECLIPSE dispose d'un debugger offrant de nombreuses possibilités. Vous pouvez exécuter une application en mode debugger par le menu **Run**→**Debug...** ou en cliquant sur le bouton représentant une espèce de **scarabée**. Vous pouvez placer des points d'arrêt dans votre code, pour cela il suffit de cliquer dans la zone d'ascenseur située à gauche de la fenêtre d'éditeur (double clic ou clic droit puis **Add Breakpoint**). En mode debugger le flux d'exécution du programme fait une pause sur ces points. Dans la perspective “debugger” qui s'affiche alors, il est possible d'examiner (zone supérieure droite) l'état de l'objet invoquant et des variables locales de la méthode interrompue. A vous de découvrir plus en détail le fonctionnement de ce debugger lorsque vous en aurez le temps et/ou en éprouverez le besoin. Apprendre à utiliser correctement un debugger est certainement important !

Exercices de manipulation

Utilisez l'environnement pour programmer le sujet de TD sur les **afficheurs** : créez un projet puis un paquetage et les classes étudiées en TD (files **FIFO** et **afficheurs**). Pour cela : **passer** par l'éditeur UML et exploitez les facilités de génération automatique de code ! C'est ainsi qu'il faut procéder : avec l'éditeur UML on crée les diagrammes et leurs dépendances, puis on génère le code associé. Vous pouvez le compléter par la programmation d'une interface graphique pour les afficheurs : une zone de saisie du message, une zone de texte (**JLabel**) pour l'affichage de ce qui est visible et un bouton qui déclenche le “top”. Placez cette IHM dans un sous-paquetage que vous appellerez **gui** par exemple.