Premiers pas en Java et découverte d'éclipse

Objectifs *: manipuler l'IDE Eclipse et l'environnement Java (compilation, exécution, javadoc, API...), aborder concrètement les concepts de classe et d'objet : attributs, méthodes, constructeurs et accesseurs.*

Le logiciel Eclipse est un environnement intégré de développement (IDE) pour le langage JAVA. Ce logiciel est gratuit, vous pourrez le télécharger chez vous sur le site www.eclipse.org. Pour le lancer, tapez `eclipse` dans un terminal. Soyez patient au lancement, cela peut prendre 20 a 30 secondes. Il n'est pas question de présenter exhaustivement les fonctionnalités de l'outil Eclipse. Seules les principales fonctionnalités seront présentées ici, charge a vous de découvrir les autres possibilités offertes afin d'accroître encore l'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, de se dégager de problèmes techniques de peu d'importance et de fournir des outils facilitant et encourageant la bonne écriture de code et la bonne conception d'application.

1 Créez un projet

Eclipse fonctionne par projet. Pour créer un nouveau projet on sélectionne *file* → *new* → *project*. Après avoir choisi *java project*, une fenêtre permet de créez un nouveau projet. Dans la partie *jdk compliance*, on peut choisir quelle version de Java sera utilisée. Ensuite dans la partie *Project layout* vous pouvez demander à distinguer les dossiers pour les fichiers sources et les fichiers compilés.

➳ Créez un nouveau projet nommé TPJAVA utilisant Java 5.0 et classant les fichiers sources et les fichiers compilés dans deux répertoires différents.

Il n'est pas recommandé d'utiliser le paquetage par défaut dans Eclipse. Pour chaque nouveau projet on Créera donc un paquetage portant le nom du projet. Un clic droit sur le projet fait apparaître un menu contextuel, le sous-menu *new* propose notamment de créez un nouveau paquetage.

➳ Créez un paquetage nommé `exo`.

2 Créez une nouvelle classe

On crée une nouvelle classe de la même manière qu'un nouveau paquetage. La fenêtre de création d'une nouvelle classe propose de nombreuses options. On veillera à ce que le champ *source folder* soit bien `nom_projet/src`, de même le champ *package* doit contenir le nom du paquetage que vous venez de définir. Le champ nom contient le nom de votre classe, il doit commencer par une majuscule. La case `public static void main` permet de générer automatiquement l'entête de la méthode *main*. L'option *generate comments* permet de générer les commentaires utilisés par *javadoc* pour créer la documentation.

➳ Créez une classe avec une méthode `main`.

➳ Complétez le corps de la méthode `main` par l'instruction suivante :

```
System.out.println("Hello world !");
```

3 Exécuter un programme

Eclipse cache le processus de compilation/exécution du code. Pour exécuter un programme, sélectionnez le menu *run* puis l'option *run...*. Double-cliquez sur *Java Application* pour préciser le type d'exécution de votre programme, vous pouvez alors modifier tous les paramètres de lancement de celui-ci. Cliquez ensuite sur le bouton *run*. Le résultat de l'exécution du code s'affiche dans l'onglet console tout en bas de la fenêtre Eclipse.

➳ Exécutez votre programme.

Pour travailler indépendamment d'Eclipse, "à la main", on doit compiler et exécuter le programme séparément. Pour compiler on tape en ligne de commandes `javac MaClasse.java`, le compilateur Java crée alors un fichier `MaClasse.class`. Pour exécuter votre programme, on tape `java MaClass` (attention, sans extension !). Le résultat devrait être identique à celui proposé par Eclipse dans la fenêtre *console*.

Dans le cas où l'on a utilisé des paquetages, la compilation peut se faire dans le répertoire du paquetage. Par contre, pour lancer la classe principale, il faut remonter au répertoire de plus haut niveau qui contient les répertoires de paquetages. Ensuite, il faut donner à java le nom complet de la classe, comprenant le paquetage. Par exemple : `java monpaquetage.MaClass`

➳ Allez dans le répertoire `src` de votre projet (qui ressemble probablement à `~/workspace/TPJAVA/exo/src`), puis compilez-le. Lancez votre programme depuis le répertoire TPJAVA.

Vous remarquerez que ces étapes sont souvent compliquées et sources d'erreur... d'où l'intérêt d'employer des outils automatisant cette étape (ce que fait Eclipse).

4 Conventions d'écriture

4.1 Organisation des fichiers

Chaque fichier source Java contient une seule classe ou interface *publique*. Le nom du fichier (sans le suffixe `.java`) est *obligatoirement* le nom de cette classe ou interface publique.

Documentation, licence, copyright, tag CVS...	`/** .... */`
Définition de paquetage	`package org.exemple.test`
Importation de paquetage	`import java.util.* ;`
Déclaration de classe ou d'interface	`public class Banque`
Commentaire de programmation	`/* mon commentaire */`
Déclaration des variables	les `public`, puis les `protected`, et enfin les `private`
Méthodes	Les regrouper de préférence par grandes fonctionnalités, pour rendre le tout le plus lisible possible.

4.2 Identificateurs

Choisissez toujours des noms significatifs et non ambigus quant au concept qu'il désigne. On met une majuscule à chaque nouveau mot d'un nom composé : ageDuCapitaine. Evitez les mots contenant des accents (cela permet une meilleure portablilité du code source).

Type	Règle de nommage	Exemple
Classe	La première lettre doit être une majuscule Éviter les acronymes, choisir des noms simple et descriptifs.	`class Image` `class ClientPrivilegie`
Interface	même règle que pour les classes.	`interface ImageAccesDirect`
Méthodes	Les noms des méthodes doivent refléter une action. Choisir de préférence des verbes. Première lettre en minuscule.	`afficher()` `getValue()`
Variables	Première lettre en minuscule. Nom relativement court mais descriptif (mnémoniques). Éviter les noms à une lettre sauf pour les compteurs internes et les variable temporaires.	`int i ;` `float largeur ;`
Constante	En majuscule, le séparateur devient forcément un souligné.	`int VAL_MAX = 999 ;`

5 La banque

Dans ce TP, vous allez implémenter des objets bancaires (compte, client, banque...) qui vont interagir entre eux. Pensez à tester dès que possible le bon fonctionnement de chacun d'entre eux.

➳ Créez un nouveau paquetage `banque`, vous y ajouterez les classes décrites ci-après.

5.1 La classe Date

Eclipse permet de générer du code automatiquement à partir des attributs. On peut notamment créer des constructeurs et des accesseurs (menu *Source* puis *generate constructor using fields* et *generate getters and setters*).

➳ Créez une nouvelle classe Date (sans méthode `main` avec les commentaires pour la javadoc).

➳ Déclarez les attributs (`int`) *privés* jour, mois, an, heure, minute et seconde.

➳ Générez (à l'aide d'Eclipse) le constructeur permettant d'initialiser les attributs jour/mois/an, les autres seront initialisés à 0 (avec les commentaires pour la javadoc).

➳ Créez (à l'aide d'Eclipse) les accesseurs en lecture des différents attributs ainsi que leurs commentaires.

➳ Codez la méthode `toString()` qui renvoie une chaîne de caractères au format suivant : [heure:minute:seconde jour/mois/an].

➳ Testez votre classe, par exemple avec un main qui contient le code suivant :

```
System.out.println(new Date(14,02,2007).toString());
```

➳ Générez la javadoc de la classe Date. (*Project → Generate Javadoc*).

5.2 La classe Compte

La classe compte contient un attribut privé, le solde du compte de type float initialisé à zéro. Un compte dispose de cinq méthodes :

```
void depot(float valeur);    /*pour faire un depot sur le compte*/
void retrait(float valeur); /*pour faire un retrait sur le compte*/
float getSolde();           /*pour obtenir la valeur du solde*/
void afficherSolde();       /*pour afficher le solde*/
void virer(float valeur, Compte destinataire)/*pour virer de l'argent vers le compte destinataire*/
```

➳ Codez la classe `Compte`.

5.3 La classe Client

Chaque client possède un nom, une date de naissance et au moins un compte, il peut avoir au maximum cent comptes. La classe `Client` aura donc pour attributs une chaîne de caractère, une date, un entier représentant le nombre de comptes et un tableau pouvant contenir cent comptes. Le constructeur a pour paramètres le nom du client, une date, et crée le premier compte. La classe `Client` contient les méthodes suivantes :

```
String getNom()       /* renvoie le nom du client */
Date getDate()        /* renvoie la date de naissance du client */
Compte getCompte(int numero) /* renvoie le compte correspondant a numero*/
void afficherBilan() /* affiche le solde de chaque compte */
float soldeTotal()   /* calcule le solde total */
void afficherSolde() /* affiche le solde total */
int ajouterCompte () /* ajoute un nouveau compte */
```

➳ Codez la classe `Client`.

5.4 La classe Banque

La classe Banque a deux attributs, son nombre de clients et un tableau de clients. Une banque a au maximum cent clients. On peut ajouter un client à la banque et afficher le bilan de la banque (ce qui revient à afficher le bilan de tous les clients).

➳ Codez la classe `Banque`.

➳ Ecrivez une méthode main permettant de tester toutes vos classes en interaction.

6 Paiement par carte bancaire

Vous allez maintenant enrichir la banque par des objets permettant le paiement par carte bancaire. Il faudra pour cela modifier certains objets (ajout de méthodes ou d'attributs par exemple).

6.1 Carte

Une carte bancaire contient un code (tableau de 4 entiers) et est associée à une banque, un numéro de client et un numéro de compte (entier) qui relie le propriétaire de la carte et un compte. Elle a également une date de validité (utiliser la classe Date). Enfin, elle contient un compteur initialisé à 3, qui décroît à chaque erreur d'entrée du code. Lorsque celui-ci est égal à 0, la méthode `codeValide` renverra toujours faux. Tous ces attributs sont bien entendu privés. La classe `Carte` contient les méthodes suivantes :

```
boolean codeValide(int[] code); /* renvoie vrai si le code est bon */
void payer(Banque b, int numClient, int numCompte, float montant) /* virement sur un autre compte */
Banque getBanque() /* retourne la banque qui a emis la carte */
Date getDateValid() /* retourne la date de fin de validite de la carte */
```

➳ Codez la classe `Carte`.

6.2 Terminal (TPE)

Pour manipuler une carte bancaire, vous allez ajouter un terminal de paiement électronique (classe `Terminal` dont le rôle est de sécuriser la transaction.

Cette fois-ci, vous devez identifier vous-même les méthodes et attributs à écrire dans vos classes. Le principe de fonctionnement doit être celui représenté dans le diagramme UML suivant (fig. 1) :

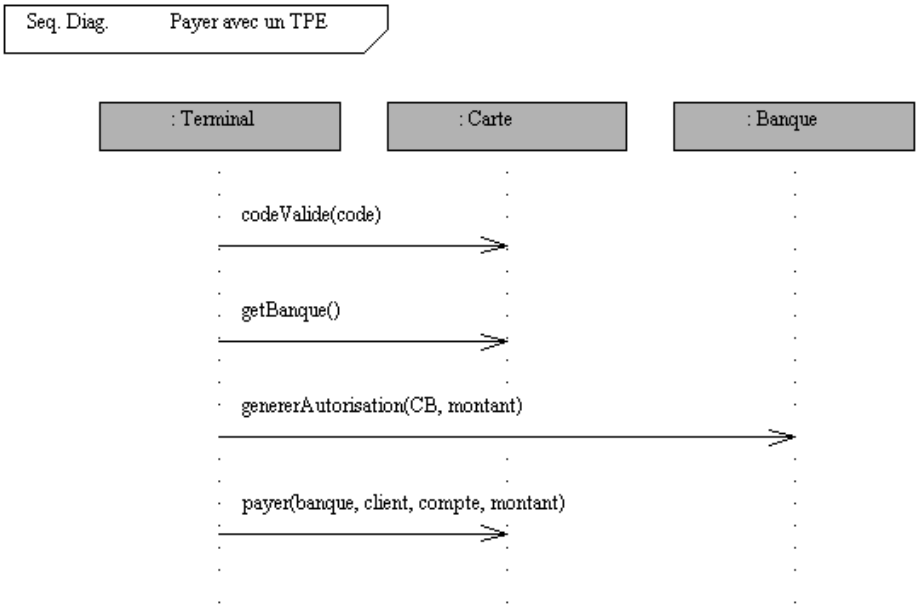

FIG. 1 – Diagramme de séquence du TPE

L'autorisation générée par la banque est un message textuel qui contient "OK" si la transaction peut être faite, ou un message d'erreur si le montant dépasse 1000 euros ou que la carte utilisée est périmée (date invalide).

➳ Ajoutez les éléments nécessaires à la mise en place du protocole décrit dans le diagramme de séquence, testez soigneusement tous vos éléments. Ajoutez des messages d'erreur si le code de la carte ou l'autorisation sont invalides.

➳ Faites contrôler votre travail par votre enseignant de TP.