TP 2&3 - Prise en main Eclipse et 1er exo "long"

Exercice 1 – Prise en main d'Eclipse

En lançant le logiciel Eclipse, vous pouvez remarquer qu'il y a énormément de menus, de fenêtres et d'outils qui vous sont proposés (certains disent qu'il y en a beaucoup trop) : c'est un logiciel complexe et très configurable pour les programmeurs dans l'objectif de concevoir des projets informatiques longs et complexes. Nous n'allons voir ici que quelques aspects de ce logiciel.
Notez bien que pour coder ou exécuter un code Java, vous pouvez continuer comme au TP1 avec un éditeur texte et les commandes javac et java. Un IDE n'est qu'un aide au développement : la compilation et l'exécution ont lieu dans le logiciel à partir des fichiers .java.

Eclipse est un IDE (Integrated Development Environment), un logiciel d'aide au développement logiciel. Il propose en interne le compilateur Java (javac) et la machine virtuel pour exécuter le code Java (JRE pour Java Runtime Environment). Il contient aussi d'autres utilitaires comme l'archiveur (jar), le générateur de documentation automatique (javadoc) et le débogueur (jdb).

Au lancement d'Eclipse, le logiciel vous demande de choisir un répertoire comme étant emplacement de travail (workspace), vous pouvez laisser celui par défaut ou en proposer un autre : les fichiers des codes Java seront alors dans des sous-répertoires de ce workspace.

Dans le répertoire fichier, on a la possibilité de créer des nouvelles choses avec "New" ou "Nouveau". Sous un IDE comme Eclipse, on crée **en premier** un projet Java (Java project) en lui donnant un nom : ceci va créer un répertoire avec ce nom (attention, il doit être unique dans votre workspace).
Suivant les versions, Eclipse peut vous demander si vous souhaitez créer un module ou un package : pour l'instant, refusez ces options : nous verrons plus tard les packages.

Suivant les versions d'Eclipse, vous avez alors devant vous plusieurs fenêtres ouvertes au sein de votre logiciel Eclipse. Si en revanche, il vous manque des fenêtres de cette liste, allez dans le menu "Fenêtre" (Window) puis Visualisation (ShowView) : une liste des fenêtres à ouvrir vous sera alors proosée. Voici une liste des fenêtres utiles, n'hésitez pas à fermer les autres :

- Package Explorer : la liste de vos projets Java avec en jouant sur les petits triangles devant les répertoires, l'accès aux fichiers contenus dans les répertoires
- Un ou plusieurs fichiers Java : ils sont mis en onglet dans la même fenêtre
- Console : il y a différents onglets pour visualiser les erreurs de debug et l'exécution

Dans le "Package Explorer", pour un projet donné, vous pouvez alors faire click droit et utilisez le menu New/Nouveau pour créer des fichiers .java de votre projet.
Si vous avez plusieurs projets en cours, vérifiez bien que votre fichier est créé dans le bon projet !
Vous pouvez alors taper des lignes de code dans le fichier .java. Vous pouvez remarquer qu'Eclipse vous propose de manière intuitive de l'aide à la frappe, ainsi qu'un mode débugage syntaxique tout au long de la frappe (avec des petits avertissement en début de ligne). **Attention**, les propositions

de correction d'Eclipse ne sont pas toujours à suivre... mais cela peut souvent aider et surtout faire gagner du temps.

Q 1.1 Ecrivez une classe `HW` retournant "Hello World" dans sa méthode toString(). Ecrivez une autre classe `testHW` qui utilise la classe `HW` pour afficher "Hello World" à l'écran.

Une fois que vous avez créé un ou plusieurs fichiers chacun contenant une classe ; et que l'une des classes contient une fonction main, alors on peut exécuter le code :

- soit avec menu Run puis Run ou le triange vert des icônes ou ctr+F11 : l'exécution a alors lieu dans la fenêtre console
- soit dans le menu Run puis "Run configuration" qui vous permet d'indiquer des paramètres en ligne de commande
- il existe aussi un mode d'exécution debug pas à pas (F11) qui peut être très utile ! Il est possible d'ajouter des breakpoints et le programme se déroule pas à pas jusqu'au breakpoint. Vous pouvez alors visualiser vos variables : soit elles sont automatiquement proposées, soit vous les ajouter en tant qu'"Expression".

A la fin, vous pouvez toujours récupérer vos fichiers java qui auront été placés dans le sous-répertoire `scr` : ces fichiers peuvent être compilés avec javac puis exécutés avec java à part d'Eclipse. Nous verrons aussi comment les rassembler en un package que vous pourrez exécuter comme un logiciel complet.

Q 1.2 Tester votre programme.

Exercice 2 – Solidarité villageoise

Un énorme rocher est tombé dans la nuit sur un petit village de l'ouest de la France bloquant l'unique axe routier sortant du village. Il est décidé de former une équipe de villageois pour tenter de déplacer le rocher.

Q 2.1 Créer la classe `Villageois` qui contient les attributs `private` suivants

- poids en kg du villageois (nombre entier)
- malade : champs indiquant si le villageois est malade ou non

et les méthodes suivantes

- une méthode `private` permettant de fixer aléatoirement un poids entre 50 et 150 kg (150 exclus)
- une méthode `private` qui attribue aléatoirement la variable malade à true dans 20% des cas et à false sinon.
- un constructeur par défaut `public` qui initialise un villageois en utilisant les deux méthodes précédentes pour initialiser les autres valeurs
- ajouter des accesseurs `public` aux deux attributs ainsi qu'une méthode `toString()` retournant une chaîne du format : "villageois : poids : 95 kg, malade : non".

Q 2.2 Dans une nouvelle classe `TestVillage`, ajouter une méthode main, qui crée une instance de la classe `Villageois` et affiche les données d'un villageois.

Q 2.3 Créer une classe `Village` contenant un attribut tableau de type `Villageois` et le poids du rocher. Un constructeur ayant en paramètre le nombre de villageois disponibles pour soulever le rocher et le poids du rocher. Ce constructeur crée les villageois en utilisant le constructeur de la classe

`Villageois`. Créer dans `TestVillage` et afficher un objet `Village` en lui ajoutant une méthode `toString()`.

Q 2.4 Ajouter dans la classe Villageois la méthode `public double poidsSouleve()` qui retourne le poids soulevé par ce villageois : le tiers de son poids s'il est en bonne santé, le quart s'il est malade.

Q 2.5 Ajouter dans la classe `Village`, une méthode `hoHisse()` qui calcule le poids total que peuvent soulever les villageois et qui affiche un message pour indiquer s'ils réussissent à soulever le rocher ou pas. Testez-là (plusieurs fois) pour 4 villageois et un rocher de 100kg.

Exercice 3 – Gestion des notes dans une université

Cet exercice est un exercice long et complexe. Il est **nécessaire** de prendre une feuille et un crayon pour dessiner des figures afin de concevoir les classes et les méthodes.

On désire concevoir un ensemble de classe permettant de gérer les notes dans une université. Une université est composée d'étudiants et d'UEs. L'université peut créer et inscrire un nouvel étudiant, ainsi que créer et ajouter une nouvelle UE. On peut inscrire un étudiant dans une UE, attribuer la note d'un étudiant dans une UE et récupérer la moyenne de l'UE et de l'étudiant.
Afin de réaliser toutes ces opérations, un étudiant est associé à un numéro qui est une clef unique (il ne doit pas y avoir d'étudiants avec un même numéro) ; et une UE est aussi associée à un codeUE qui est une clef unique.

Afin de simplifier l'énoncé de cet exercice, on va supposer que toutes les entrées d'inscription d'étudiants et d'ajouts d'UE sont définitives et donc nous ne regarderons pas les suppressions d'éléments. On utilisera uniquement des tableaux qui seront définis dés leur initialisation avec des tailles maximales, puis seront accompagnés de valeurs donnant le nombre de cases utilisées.

Partie 1 : inscriptions des étudiants

Q 3.1 Créer une classe `TestUniversite` qui contient une fonction main et qui vous permettra **tout au long de l'exercice** de tester unitairement chaque méthode. Créer une classe `Etudiant` et une classe `UE` pour l'instant vide.

Q 3.2 Ajouter une classe `Etudiant` comportant des attributs `private` un nom, un numero (nombre entier), un nombre d'UE suivis et un tableau des UEs suivis. Ce tableau contient des références sur des objets `UE`. Tous les étudiants partagent le fait d'avoir un gabarit de 10 UEs maximum où il peuvent être inscrits : ce sera défini par l'attribut `static final int NB_gabarit=10`.
Un étudiant est créé à partir d'un nom et d'un numéro. Proposer des accesseurs pour le nom et le numéro ainsi qu'une méthode toString() donnant le nom, numéro et liste des UEs suivis. Ajouter une méthode testant si l'étudiant a déjà choisi toutes ses 10 UEs ou non. Testez votre classe dans TestUniversite.

Q 3.3 Ajouter une classe `UE` comportant des attributs `private` un intitulé, un codeUE (un String), un nombre d'inscrits dans l'UE et un tableau tab_inscrits référençant les étudiants inscrits.
Une UE est créée à partir d'un intitulé, un codeUE et le nombre maximal d'inscrits possibles dans l'UE. Pensez à proposer des accesseurs pour l'intitulé et le codeUE ainsi qu'une méthode toString()

donnant l'intitulé et le code UE. Ajouter une méthode testant si l'UE est pleine en atteignant son nombre max d'inscrits.

Q 3.4 Créer dans la classe `UE` une méthode private int indice_(int numero_etudiant) qui retourne l'indice de l'étudiant dans le tableau des inscrits de l'UE. Cette méthode renvoie -1 si l'étudiant n'est pas inscrit dans l'UE. Tester la méthode.

Q 3.5 Créer une méthode inscrire_etudiant(Etudiant e) dans la classe `UE` qui prend en paramètre une référence sur Etudiant et l'ajoute dans le tableau des inscrits dans l'UE. Cette méthode doit :
- assurer que l'étudiant n'est pas déjà inscrits dans l'UE
- assurer l'association en ajoutant l'UE dans la liste des UE suivies par l'étudiant. Attention à bien tester si les tailles maximales sont respectées (sinon afficher de messages d'erreurs).

Tester la méthode dans `TestUniversité`.

Q 3.6 Créer une classe `Universite` qui contient
- deux constantes de classe NB_max_etudiants et NB_max_UE qui seront initialisé par le constructeur : il s'agit des nombre maximaux d'étudiants et d'UE possibles.
- un tableau tab_UEs de toutes les UE et le nombre des UEs de l'université
- un tableau tab_etudiants de tous les étudiants et le nombre d'étudiants de l'université

Ajouter un constructeur, une méthode affichant toutes les UEs avec la taille de leurs effectifs et une méthode affichant tous les étudiants de l'université avec la liste des UEs choisies par l'étudiant.

Q 3.7 On veut assurer l'unicité des clefs des code d'UEs et des numéros d'étudiants. Pour cela créer dans la classe `Universite`
- une méthode indice_etudiants(int numero) qui retourne l'indice dans le tableau tab_etudiants de l'étudiant ayant le numéro passé en paramètre ; ou retourne -1 si ce numéro est absent.
- une méthode indice_UE(String codeUE) qui retourne l'indice dans tab_UEs de l'UE ayant le code passé en paramètre ; ou retourne -1 si ce code est absent.

Testez vos méthodes.

Q 3.8 Ajouter dans la classe `Universite` une méthode inscrire_etudiant(Etudiant e) permettant d'ajouter un étudiant dans l'université en assurant l'unicité de son numéro ; et une méthode ajouter_UE(UE ue) permettant d'ajouter une UE dans l'université en assurant l'unicité de son code. Tester vos méthodes.

Q 3.9 Ajouter dans la classe `Universite` une méthode inscrire_dans_UE(int numero_etu, String codeUE) qui inscrit un étudiant dans une UE. Tester votre méthode sur plusieurs étudiants et UE.

Partie 2 : Gestion des notes et des moyennes

Q 3.10 Ajouter dans la classe `UE` un attribut `private double[] tab_notes` dont une case tab_notes[i] va contenir la note de l'étudiant tab_inscrits[i]. Pensez à l'initialiser dans le constructeur. Ajouter dans `UE` une méthode set_notes(int numero_etu, double note) qui attribue la note à l'étudiants (et affiche une erreur si l'étudiant n'est pas dans l'UE).

Q 3.11 Ajouter dans la classe `Universite` une méhtode attribuer_note(int numeor_etu, String codeUE, double note) qui affecte une note dans une UE pour un étudiant. Tester votre méthode.

Q 3.12 Créer, dans la classe `UE`, une méthode calculant la moyenne d'une UE et une méthode affichant la moyenne de l'UE (avec un joli affichage). Créer dans la classe `Universite` une méthode affichant les moyennes de toutes les UEs.

Q 3.13 Créer, dans la classe `Etudiant`, une méthode calculant la moyenne d'un étudiant et une méthode affichant la moyenne de l'étudiant (avec un joli affichage). Créer dans la classe `Universite` une méthode affichant les moyennes de toutes les étudiants ainsi que la moyenne de toutes les moyennes !

Partie 3 : Chargement et sauvegarde fichiers

Le site du module propose un fichier expliquant comme lire et écrire dans un fichier texte.

Q 3.14 Inventer un format de fichier texte permettant de sauvegarder toute l'université. Créer un fichier avec quelques étudiants et des UEs.
Indication : Il est utile de sauvegarder d'abord les infos générales de l'université, puis les étudiants (en remplissant l'université), puis les UEs (lors de la création des UEs, on inscrit les étudiants dans les UEs).

Q 3.15 Ajouter à la classe `Universite` un constructeur qui intialiser une université à partir d'un fichier. Tester la sur votre fichier.

Q 3.16 Ajouter à la classe `Universite` une méthode de sauvegarde sur fichier. Cette méthode va s'appuyer sur des méthodes de sauvegarde de la classe `Etudiant` et de la classe `UE` (bien entendu, on sauvergarde les numéros des étudiants et les code des UEs, pas les références). Tester votre fonction.

Partie "Bonus" : allons un peu plus loin

Q 3.17 Créer une méthode générant aléatoirement des UEs, puis des étudiants en les inscrivant aléatoirement dans les UEs, puis en leur attribuant aléatoirement des notes. Tester vos méthodes en sauvegardant, puis en lisant ensuite votre fichier avec de grande taille de données aléatoire.

Q 3.18 Quelle est la valeur obtenue pour la moyenne des moyenne pour un très grand nombre d'étudiants ?

Q 3.19 Vous pouvez remarquer que le calcul de la moyenne d'un étudiant est assez lent. Quelle est sa complexité ? Comment faire pour améliorer la vitesse de ce calcul ?