Programmation Objet en Java - Feuille de TP 4

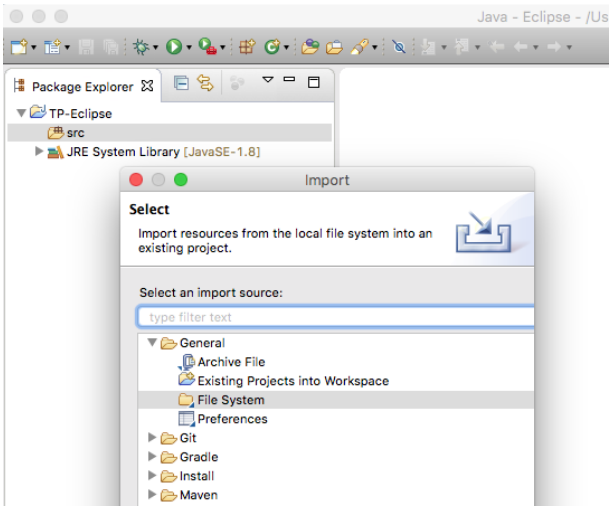
Exercice 1. (∼ 20 minutes)
Commencez par trouver Eclipse (light) et lancez-le.

a) Choisissez comme *workspace* le dossier où vous stockez vos TPs de Java.

b) Créez un nouveau **projet Java** nommé "TP-Eclipse". Eclipse va vous demander si vous souhaitez créer un module : refusez.

c) Nous allons maintenant importer des fichiers existants dans ce projet. Placez-vous dans le projet "TP-Eclipse" (vous pouvez voir son contenu dans le panneau de gauche en utilisant la petite flèche grise).

À l'intérieur du projet se trouve un dossier nommé *src*. Faites un clic droit dessus et choisissez *Import...* pour obtenir ceci :



d) Sélectionnez le dossier *General* puis *File sytem*. Retrouvez vos fichiers du TP précédent : `Book.java` et `BookStore.java` et importez-les.

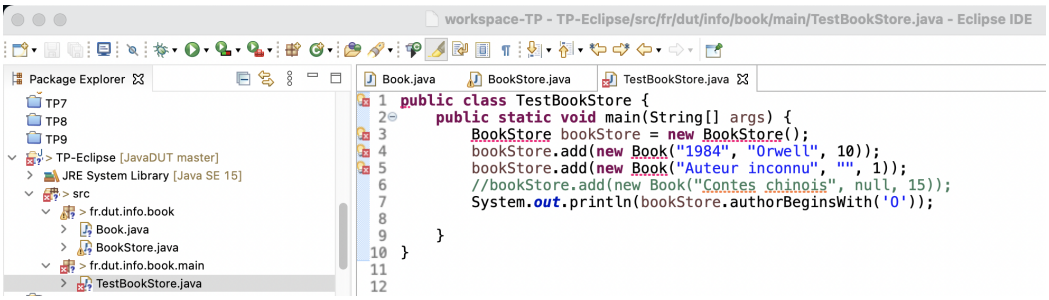
e) Notez que vos fichiers se trouvent désormais dans un package. Il est probablement nommé *default package*. **Vous ne devez pas laisser vos fichiers dans ce package par défaut.** Quel que soit le package dans lequel se trouve vos fichiers, vous allez en créer un nouveau, nommé *fr.dut.info.book* en faisant un clic droit sur le dossier *src* et en choisissant *New...* puis *Package*. Attention les points sont importants dans le nom du package.

Une fois le package créé, sélectionnez vos deux fichiers `.java`, faites un clic droit et choisissez *Refactor...* puis *Move* pour les déplacer dans le package que vous venez de créer. Vous pouvez supprimer le package par défaut s'il existe toujours.

f) Double-cliquez sur `Book.java` et regardez la première ligne du fichier... Allez également voir ce que contient le dossier "TP-Eclipse" dans le gestionnaire de fichiers. L'organisation en dossiers et sous-dossiers que vous obtenez est obligatoirement celle que vous devez avoir lorsque vous manipulez des packages en Java.

g) Revenons dans Eclipse. Créez maintenant un deuxième package nommé *fr.dut.info.book.main* et importez dedans le fichiez `TestBookStore.java` que vous aviez écrit pour tester vos classes précédentes. Attention à l'emplacement où vous faîtes le clic droit pour importer!

h) Ouvrez `TestBookStore.java`. Voici ce que vous obtenez :



i) La petite croix rouge qui se trouve devant la déclaration de la classe signifie qu'il y a une erreur à cette ligne-là. À votre avis, quel est le problème?

Placez votre souris sur cette petite croix rouge et cliquez. Eclipse vous propose différentes façons de régler le problème : double-cliquez sur celle qui est judicieuse. Notez que si vous sauvegardez le fichier, le problème devant la déclaration de la classe a disparu.

j) Il reste malgré tout quelques problèmes liés au fait que la classe de test se trouve dans un package différent des classes `Book.java` et `BookStore.java`. Cliquez sur la croix suivante et importez le package nécessaire.

k) Finissez de corriger les erreurs et lancez votre programme avec le bouton `Run` (flèche blanche sur rond vert, dans la barre d'outils). Allez chercher dans vos dossiers où se trouvent les fichiers `.class` générés.

Exercice 2. (∼ 40 minutes) On souhaite pouvoir ajouter un avis (*review*) pour chaque livre, afin d'aider les lecteurs à faire leur choix.

a) Créez une nouvelle classe `Review` dans le package *fr.dut.info.book* avec comme champs le nom de la personne qui donne l'avis et une note sur 5.

b) Utilisez les touches `Ctrl + Space` dans la classe `Review`... Profitez-en pour créez le constructeur par défaut. En fait, on préférerait créer le constructeur utilisant les 2 champs ; cherchez dans les menus si Eclipse ne peut pas vous aider à le faire. Ajoutez également un constructeur qui prend uniquement une note en paramètre.

c) Réutilisez `Ctrl + Space` pour vous aider à écrire la méthode d'affichage.

d) Ajoutez une méthode `changeNote` qui permet de changer la note d'un `Review`. Attention à ce que le note soit toujours valide.

e) Créez maintenant une nouvelle classe `ReviewTest` dans le package *fr.dut.info.book.main* pour testez vos `Reviews`. Pensez à demander à Eclipse d'inclure la méthode `main` au moment de la création de la classe.

f) Testez la classe `Review`. Que se passe-t-il si vous tapez `sysout` suivit de `Ctrl + Space` dans la méthode `main` ? On obtient, par exemple :

```
System.out.println(new Review("Joey", 5));     // affiche Joey (5)
System.out.println(new Review(1));             // affiche ___ (1)
```

g) Ajouter à `BookStore` une méthode `numberOfAnonymousBooks` permettant de compter le nombre de livre dont l'auteur est "no author".

h) Comment pouvez-vous faire pour ne pas être obligé de re-calculer ce nombre à chaque fois que vous utilisez cette méthode ? Modifiez votre code en conséquence.

i) Revenez à la classe `Book` et ajoutez un champs pour un review. Modifiez le constructeur pour que les livres puissent être créés avec un `Review`. **Au passage, testez ce que fait *Format*, dans le menu *Source*.**

j) Ajoutez une méthode `changeReview` qui permet de remplacer le `Review` d'un livre.

k) ★ Dans la classe `BookStore`, on souhaite maintenant obtenir la liste des livres les mieux notés (plusieurs livres différents peuvent avoir la meilleure note). Écrire une méthode `bestBooks` permettant de faire cela. De quelle autre méthode avez-vous besoin ?

Pouvez-vous utiliser la même technique que pour `numberOfAnonymousBooks` pour ne pas être obligé de re-calculer la meilleure note à chaque fois que vous utilisez cette méthode ? Réfléchissez bien et justifiez votre réponse.

l) En fait, "note" n'est pas le bon terme en anglais pour la note donnée dans le `Review`, il faudrait remplacer le nom du champs par "rank", par exemple. Faites un clic droit sur le champs *note* et choisissez *Refactor...* puis *Rename...* afin de modifier le nom du champs. Quel est l'intérêt de cette façon de faire ? Quelles sont ses limites ?

m) On souhaite également changez le nom de la classe `BookStore` en `BookShop` et du package *fr.dut.info.book.main* en *fr.dut.info.book.test*. Faites-le en utilisant *Refactor* et observez l'effet sur les noms de dossiers et fichiers dans le répertoire "TP-Eclipse".

Exercice 3. Le retour des dictionnaires (∼ 1 heure)

On considère la définition de l'interface `Dico` (package `fr.dut.info.dictionnary`) qui décrit le comportement d'un dictionnaire (on verra plus tard que cette structure de données existe déjà en Java avec une implémentation beaucoup plus efficace que celles que nous allons écrire aujourd'hui). Pour simplifier, on fixe les types des clés (`String`) et des valeurs (`int`). On rappelle que, dans un dictionnaire, les clés ne peuvent pas apparaître plusieurs fois.

```
public interface Dico {
  /**
   * @return le nombre de clés du dictionnaire
   */
  int size();

  /**
   * @return true si le dictionnaire est vide et false sinon
   */
  boolean isEmpty();

  /**
   * @param key
   * @return true si le dictionnaire contient la clé key et false sinon
   */
  boolean containsKey(String key);

  /**
   * @param value
   * @return true si le dictionnaire contient la valeur value et false sinon
   */
  boolean containsValue(int value);

  /**
   * @param key
   * @return la valeur associée à key si elle existe et null sinon
   */
  Integer get(String key);

  /**
   * Associe la valeur value à la clé key dans le dictionnaire.
   *
   * @param key
   * @param value
   */
  void put(String key, int value);

  /**
   * Remplace la valeur associée à la clé key par value, si key existe et sinon ne fait rien.
   *
   * @param key
   * @param value
   */
  void replace(String key, int value);
}
```

a) Quelles sont les méthodes qui peuvent être déclarées `default` ? Écrire leur code.

b) Pourquoi choisir `Integer` et pas `int` comme type de retour de la méthode `get` ?

c) Quelle précaution faut-il prendre pour le code de la méthode `put` ?

d) Écrire une classe `ListOfPairsDico` qui implémente l'interface `Dico` sous la forme d'une liste contenant des couples de type `(String, int)`.

e) Écrire une classe `PairOfListsDico` qui implémente l'interface `Dico` sous la forme d'un couple de listes, l'une contenant des `String` et l'autre des `int`.