Premiers diagrammes de classes

1 Une classe Rectangle

Proposez en UML puis en Java une classe `Rectangle` dans le cadre d'un éditeur graphique, de manière à pouvoir disposer des caractéristiques suivantes :

- la longueur et la largeur,
- l'aire,
- le périmètre,
- le nombre de côtés,
- la couleur,
- l'angle entre deux côtés,
- la position,
- le plus grand rectangle construit.

Proposez un diagramme d'instances pour un rectangle particulier (avec des valeurs de votre choix).

2 Une classe Tortue

Proposez en UML puis en Java une classe `Tortue`, de manière à pouvoir disposer des caractéristiques suivantes :

- l'âge d'une tortue,
- l'espérance de vie de l'espèce,
- le nom commun de l'espèce (tortue d'Herman, tortue de Caroline, etc.),
- la particularité d'hiberner,
- le type de nourriture (feuilles, fruits, etc.),
- l'habitat (garrigues, maquis, etc.),
- la mise en hibernation,
- l'action de manger,
- le sexe,
- la ponte.

Introduction au « Java Development Kit »
Et quelques instructions en Java

Le *Java Development Kit* offre un ensemble d'outils de développement d'applications Java. Pour utiliser ces outils, JDK ne propose pas d'interface utilisateur, on doit donc écrire des lignes de commandes dans une fenêtre « terminal ». Des environnements plus sophistiqués existent, bien sûr, que vous étudierez ultérieurement, mais cette manière plus primitive de travailler vous permettra de bien comprendre les mécanismes mis en jeu lors de l'élaboration et de l'exécution des programmes.

1 Création des répertoires d'accueil des programmes Java

Soyez très soigneux dans toute la suite en ce qui concerne les noms de répertoires et de fichiers, principalement pour les majuscules et minuscules.
Vous devez créer les répertoires suivants pour accueillir le travail que vous ferez pendant les TPs de Java (aux feuilles de l'arbre, apparaissent certains des premiers fichiers que nous placerons dans les répertoires).

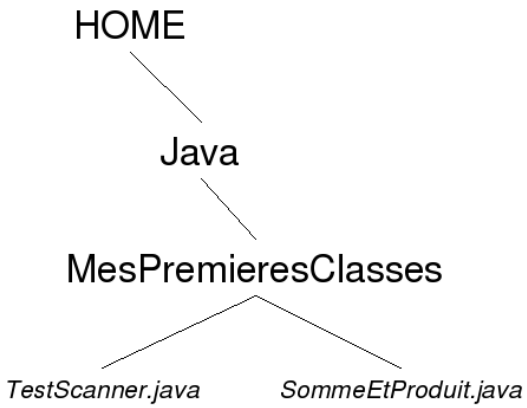

FIGURE 1 – Organisation des répertoires

Vous trouverez dans le dossier TP1 à l'adresse :

`https://moodle.umontpellier.fr/course/view.php?id=1584`

deux fichiers que vous copierez dans le répertoire `MesPremieresClasses` :
- `SommeEtProduit.java`,
- et `TestScanner.java`.

2 Modification des codes sources

Nous utiliserons xemacs ou kate pour l'édition des sources. Le lancement de xemacs se fait par la commande `xemacs &` ou en utilisant les menus ou icônes de votre environnement.
Ouvrez le code source du programme `SommeEtProduit.java` que nous vous rappelons au listing 1. Vous pouvez remarquer que le nom de la classe emballant ce programme est `SommeEtProduit`, c'est-à-dire le nom du fichier sans l'extension `.java`, ce n'est pas une coïncidence, c'est obligatoire.

Listing 1 – SommeEtProduit.java

```
1  package MesPremieresClasses;
2
3  import java.util.Scanner;
4
5  public class SommeEtProduit {
6    public static void main(String[] a) throws java.io.IOException
7    {
8        Scanner sc = new Scanner(System.in);
9
10       double x, y;
11       System.out.println("entrez deux doubles ");
12       x = sc.nextDouble();
13       y = sc.nextDouble();
14       double somme = x+y;
15       double produit = x*y;
16       System.out.println("somme = "+somme+" produit = "+produit);
17    }
18
19 }
```

3 CLASSPATH

Vous devez également mettre à jour une variable d'environnement nommée CLASSPATH. Dans votre fichier nommé `.bashrc`, placez la ligne suivante (n'oubliez pas de la faire suivre d'un « saut de ligne » si c'est la dernière ligne de votre fichier, sinon elle ne sera pas prise en compte) :
`export CLASSPATH=${HOME}/Java:.`

Pour prendre en compte cette modification, tapez la commande :
`. ~/.bashrc`
ou encore la commande
`source ~/.bashrc`

Java permet de désigner les classes de manière unique. Pour cela, Java concatène deux chemins : celui que contient `CLASSPATH` et celui qui correspond au nom du paquetage, par exemple, avec le programme qui précède : `${HOME}/Java` est concaténé avec `MesPremieresClasses`, ce qui donne le chemin absolu pour accéder à `SommeEtProduit`

$$\underbrace{\$HOME/Java/}_{\text{CLASSPATH}}\ \underbrace{MesPremieresClasses/}_{\text{Paquetage}}\ \text{SommeEtProduit}$$

La résolution d'une directive d'import fonctionne sur le même modèle.

4 Compilation

Le compilateur, qui va analyser le programme source et produire le programme en bytecode s'appelle `javac`.
Placez-vous dans une fenêtre terminal, et dans le répertoire `MesPremieresClasses`, la commande de compilation sera la suivante :
`javac SommeEtProduit.java`

Vous pouvez constater qu'un nouveau fichier `SommeEtProduit.class` est apparu dans votre répertoire `MesPremieresClasses`.

5 Exécution de programmes et entrée des données

5.1 Exécution

Notez que l'on ne peut exécuter (interpréter) que les classes qui contiennent une méthode `main`. La méthode `main` est la méthode principale de votre programme. Votre application commence par le début de cette méthode. Elle se terminera quand l'exécution sortira du bloc `main`.
Pour la classe `SommeEtProduit` par exemple, cela se fait par la commande :
`java MesPremieresClasses.SommeEtProduit`
à exécuter depuis le répertoire Java, qui contient le répertoire `MesPremieresClasses`.

5.2 Entrée des données

Lors de cette exécution, vous devez saisir deux nombres réels au clavier. Cela peut se révéler fastidieux lorsque le programme attend beaucoup de données.
Quatre autres solutions s'offrent à vous pour entrer des données.
Les deux premières consistent à créer un fichier contenant les données, sous la même forme que vous les auriez saisies au clavier. Par exemple, vous créez le fichier appelé `entree` qui contient (il se termine par une ligne vide) :

```
2
7,3
```

Puis vous pouvez appeler votre précédent programme de cette manière qui consiste à rediriger l'entrée du programme vers le fichier `entree` :
`java MesPremieresClasses.SommeEtProduit < entree`

Une autre solution pour utiliser le fichier consiste à l'ouvrir et à le parcourir à l'intérieur du programme avec des fonctions Java particulières. Nous la verrons plus tard.

La troisième solution consiste à utiliser les paramètres que l'on passe à la fonction `main`. Dans ce cas, on écrit un programme un peu différent, qui vous est présenté ci-après. Vous pouvez y observer que le paramètre `String[] a`, qui est un tableau de chaînes de caractères est exploité pour y récupérer successivement les deux nombres réels. Comme il s'agit d'un tableau de chaînes, la fonction `valueOf` de la classe `Double` est utilisée pour la conversion de `String` vers `double`.

```
package MesPremieresClasses;
public class SommeEtProduitEntreeParamMain {
  public static void main(String[] a) throws java.io.IOException
  {
      double x = Double.valueOf(a[0]);
      double y = Double.valueOf(a[1]);
      double somme = x+y;
      double produit = x*y;
      System.out.println("somme = "+somme+" produit = "+produit);
   }
}
```

Le programme est alors appelé de la manière suivante en ligne de commande (les paramètres sont sur la ligne de commande à la fin) :
`java MesPremieresClasses.SommeEtProduitEntreeParamMain 2 7.3`

Une quatrième solution consiste à écrire les valeurs à tester directement dans le programme. C'est une solution qui peut être utile pour des petits tests.

```
package MesPremieresClasses;
public class SommeEtProduitEnDur {
  public static void main(String[] a) throws java.io.IOException
  {
      double x = 2;
      double y = 7.3;
      double somme = x+y;
      double produit = x*y;
      System.out.println("somme = "+somme+" produit = "+produit);
   }
}
```

Vous pourrez remarquer le changement d'écriture des réels : on écrit 7,3 pour l'interpréteur (qui est francophone) et 7.3 pour le compilateur (qui est anglophone) avec notre installation actuelle.

6 Scanner

Regardez d'un peu plus près le programme `TestScanner.java`. Il utilise une instance de la classe `Scanner`, qui se trouve dans le paquetage `java.util` de l'API [1] Java fournie avec le `JDK`. La bibliothèque de classes fournie par Java est très grande et variée : elle permet la lecture et l'écriture dans différents médias, la communication réseau, la réalisation d'interfaces graphiques, fournit beaucoup de structures de données classiques, etc. Il est à noter que le chemin vers l'emplacement de la bibliothèque n'est pas à ajouter dans le `CLASSPATH`, il est connu grâce à une autre variable d'environnement. Quand on souhaite utiliser une classe de la bibliothèque, il suffit donc de connaître dans quel paquetage elle se trouve, ce qui permet soit d'importer la classe (par exemple : `import java.util.Scanner;` puis : `Scanner sc= ...`), soit d'utiliser le nom absolu de la classe (`java.util.Scanner sc=...`).
La classe `Scanner` permet l'analyse de texte, notamment de texte provenant de l'entrée standard (texte tapé sur la console). Quand on crée une instance de `Scanner`, on passe en paramètre du constructeur le texte à analyser : on peut lui passer un nom de fichier par exemple, ou bien un flux de texte comme `System.in`, qui permet l'analyse de l'entrée standard. L'analyseur permet de lire les éléments d'un texte les uns après les autres, pour cela, il est nécessaire de connaître le caractère séparateur des éléments dans le texte. Par défaut, il s'agit de l'espace.
L'analyseur permet aussi de traduire les éléments (suites de caractères) lus vers le type que l'on lui fournit : on peut demander à lire le prochain entier, et on récupère alors un élément de type entier, et pas de type chaîne. Bien sûr, l'analyse échoue si l'on demande la lecture d'un entier et que l'on fournit des caractères non traduisibles en entiers.
La classe `Scanner` dispose notamment des méthodes :

- `next()` qui retourne la prochaine chaîne lue dans le texte
- `nextInt()` qui retourne le prochain entier lu dans le texte
- `nextFloat()` qui retourne le prochain flottant lu dans le texte. Le caractère séparant la partie entière de la partie décimale est la virgule.

Compilez le fichier `TestScanner.java`, exécutez-le, et modifiez-le si vous le souhaitez. Vous remarquerez que la méthode *main* contient à la suite de sa déclaration une instruction qui peut paraître ésotérique : throws `IOException`. Ceci est dû au fait que les méthodes de la classe `Scanner` qui sont utilisées dans le main peuvent déclencher des erreurs (par exemple si on demande à lire un flottant et que la syntaxe des flottants n'est pas respectée par l'utilisateur (3.1 au lieu de $3,1$). Ces erreurs sont déclenchées sous la forme d'exceptions, vous verrez l'année prochaine en quoi cela consiste. En Java, quand on appelle une méthode `m` qui déclenche des exceptions, on doit :

- soit les propager (c'est-à-dire ne rien en faire et les laisser remonter soit vers l'utilisateur, soit vers une méthode appelante). Dans ce cas, la méthode `m` devient à son tour émittrice d'exceptions, et doit le signaler

1. Application Programming Interface

par l'instruction throws suivie du nom de l'exception (ou des exceptions). C'est ce qui est fait dans notre exemple : on propage l'exception de type `IOException`.
— soit d'attraper l'exception, ce qui arrête sa propagation.

7 Paquetages et visibilité

Ecrivez en Java le code correspondant au diagramme donné à la figure 2.

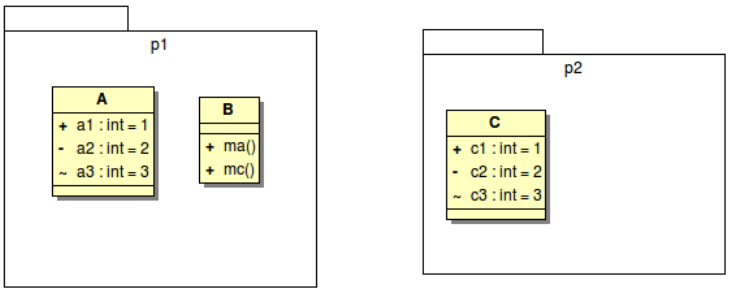

FIGURE 2 – Paquetages et visibilité

Compilez la classe A.
Dans le corps de la méthode `ma`, écrivez les lignes suivantes :

```
A a=new A();
System.out.println(a.a1);
System.out.println(a.a2);
System.out.println(a.a3);
```

Compilez la classe B. Quelle erreur est produite ? Pourquoi ? Commentez la ligne qui bloque la compilation. Recompilez. Prévoyez maintenant un paquetage appelé test et contenant une classe M avec une méthode main, qui sera le point d'entrée de notre programme. Pour exécuter, écrivez dans M.java :

```
package test;

import p1.B;

public class M {
/**
 * @param args pas d'arguments
 */
 public static void main(String[] args) {
  B b=new B();
  b.ma();
 }
}
```

Exécutez ce programme.

Dans le corps de la méthode `mc`, écrivez la ligne suivante :

```
C c=new C();
```

Compilez la classe B. Quelle erreur est produite ? Pourquoi ? Proposez une solution résolvant le problème. Recompilez.
Dans le corps de la méthode `mc`, ajoutez les lignes suivantes :

```
System.out.println(c.c1);
System.out.println(c.c2);
System.out.println(c.c3);
```

Compilez la classe B. Quelles erreurs sont produites ? Pourquoi ? Commentez les lignes qui bloquent la compilation. Recompilez.
Exécutez en ajoutant à la méthode main la ligne b.mc() ;

8 Application

Ecrivez en Java un programme qui demande la saisie sur l'entrée standard de 2 entiers e1 et e2, et 2 flottants f1 et f2, puis qui affiche le résultat de e1/e2 et f1/f2. Exécutez votre programme avec notamment :

- e1=6 et e2=4
- e1=6 et e2=0
- f1=6f et f2=4f

Qu'en déduisez-vous sur l'opérateur / ? Arrivez-vous à comprendre ce qui s'affiche lors d'une division par 0 ?

Classes, attributs, et méthodes en UML et Java

Exercice 1 *Classe Étudiant*

La scolarité souhaite modéliser puis développer (en Java) une classe Etudiant commune à ses divers systèmes d'information. On s'intéresse ici à une version préliminaire de cette classe, où devront apparaître les informations suivantes :

- l'âge
- la date de naissance (dans un premier temps, elle ne contiendra que l'année de naissance)
- un attribut `codeIns` déterminant si c'est la première inscription de l'étudiant ou s'il s'agit d'une réinscription
- un attribut `codePays` permettant de distinguer les étudiants français, les étudiants étrangers non francophones et les étudiants étrangers francophones
- 3 notes obtenues à 3 examens
- les accesseurs associés aux attributs précédents
- un constructeur dont les arguments permettent de garnir les attributs précédents
- une méthode permettant le calcul de la moyenne obtenue
- une méthode permettant le calcul de la mention obtenue
- une méthode `ligneResultats` qui retourne une chaîne d'une ligne précisant le nom, la moyenne et la mention, et, seulement s'il est ajourné, les modules obtenus (c'est-à-dire ceux où il a obtenu la moyenne)

Question 1. Proposez une modélisation UML de la classe Etudiant (qui nécessitera peut-être d'introduire de nouvelles classes que vous modéliserez également).

Question 2. Proposez une implémentation en Java de votre modélisation.
On veillera à bien respecter les règles suivantes :

- les attributs sont privés et sont accédés grâce à un couple d'accesseurs
- les méthodes et les attributs sont tous commentés
- le code est bien indenté
- le code est soigneusement testé en développant une classe dédiée au test, et qui manipule autant d'instances d'étudiants que vous le jugerez nécessaire
- les noms des classes commencent par une majuscule, les noms des packages, attributs et méthodes par une minuscule.

Question 3. Toutes les classes disposent implicitement d'une méthode `String toString()` qui retourne une chaîne de caractères dont le rôle est de représenter une instance ou son état sous une forme lisible et affichable. Si on ne définit pas de méthode `toString` dans

une classe, la méthode par défaut est appelée, elle retourne une désignation de l'instance. Il est conseillé de définir une méthode `toString` pour chaque classe.

a- Utilisez la méthode `toString` présente par défaut dans la classe Etudiant. Que renvoie-t-elle ?

b- Écrivez pour chacune des classes que vous avez implémentées une méthode `toString`. Testez ces méthodes.

Question 4. Gestion plus fine de la date de naissance et de l'âge. Vous terminerez par la gestion plus fine de l'âge et de la date de naissance. Un nouveau package de gestion "du temps" (dates, heures, durées, etc) a été introduit avec Java8 : java.time.

Version avant Java 8 Pour un attribut ou une variable représentant une date, on utilisera la classe Date présente dans le package java.util. Pour tester la classe Etudiant, vous aurez besoin de créer des dates, correspondant aux dates de naissance des étudiants. Pour cela, on utilisera la classe DateFormat du package java.text. Plus particulièrement, on pourra s'inspirer des lignes suivantes :

```
// on récupère un formatteur de date, qui gère des formats longs
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG);
// on met dans d la date correspondant au 03/07/04.
Date d=df.parse(‘‘03 juillet 2004’’);
```

Pour récupérer la date courante, on utilise simplement le constructeur sans paramètre de la classe Date. Pour calculer l'âge, il sera utile de récupérer l'année d'une date, le mois d'une date, et le jour d'une date. On s'inspirera pour cela du code :

```
// création d'un calendrier grégorien
// permettant de récupérer les informations d'une date
Calendar cal=new GregorianCalendar();
cal.setTime(d); // On remplit le calendrier avec la Date d
int annee=cal.get(Calendar.YEAR); // on récupère l'année
int mois=cal.get(Calendar.MONTH); // on récupère le mois
int jour=cal.get(Calendar.DAY_OF_MONTH); // on récupère le jour
```

Version à partir de Java 8 Pour un attribut ou une variable représentant une date, on utilisera la classe `LocalDate` présente dans le package `java.time`. Pour tester la classe `Etudiant`, vous aurez besoin de créer des dates, correspondant aux dates de naissance des étudiants. Pour cela, on utilisera la méthode statique `of(int year, int month, int dayOfMonth)` qui retourne une instance de la classe `LocalDate` avec les valeurs spécifiées en paramètre pour l'année, le mois et le jour. Pour récupérer la date courante, on utilise la méthode statique `now` de la classe `LocalDate` qui retourne la date courante. Pour calculer l'âge, on pourra utiliser la méthode `until` de la classe `LocalDate` qui prend en paramètre une autre date, et calcule la période de temps séparant les 2 dates (où le début de la période

est `this`, et la fin la date en paramètre). Pour une période, on obtient le nombre d'années avec la méthode `getYears`.

Exercice 2 *Pour ceux qui ont fini en avance ...*

Implémentez les classes `Rectangle` et `Tortue/Espece` vue en TD, et testez votre implémentation.

Héritage – La poste en Laponie

1 Objets postaux

Les lapons envoient plusieurs sortes d'objets postaux, des *lettres*, des *colis* et des *colis express*.

▷ Tous les objets postaux ont les caractéristiques suivantes :

- une origine,
- une destination,
- un code postal,
- un poids (en grammes),
- un volume (en m^3),
- un taux de recommandation (égal à 0, 1 ou 2).

On doit pouvoir calculer leur tarif d'affranchissement, leur tarif de remboursement, et afficher une chaîne qui décrit l'objet.

▷ Les *lettres* peuvent en outre avoir un certain caractère d'urgence.
Le tarif d'affranchissement d'une *lettre* se calcule de la manière suivante. Le tarif de base est de 0.5 euro auquel s'ajoutent cumulativement :

- 0.5 euro si le taux de recommandation est 1, 1.5 euros si le taux de recommandation est 2,
- 0.30 euro si c'est une lettre urgente.

Le tarif de remboursement est de :

- 0 euro si le taux de recommandation est égal à 0,
- 1.5 euros si le taux est égal à 1,
- 15 euros si le taux est égal à 2.

La chaîne qui décrit une lettre a la forme suivante (en italiques les données spécifiques d'une lettre) :
`Lettre` *code postal/destination/taux de recommandation/caractère d'urgence*
Par exemple : `Lettre` *7742/famille Kirik, igloo 5 banquise nord/1/ordinaire*

▷ Les *colis* possèdent les caractéristiques complémentaires suivantes :

- une déclaration de contenu (texte)
- une valeur déclarée (en euros)

Le tarif d'affranchissement d'un *colis* s'obtient par cumul des sommes suivantes :

- 2 euros dans tous les cas (tarif de base),
- 0.5 euro si le taux de recommandation, est 1, 1.5 euros si le taux de recommandation est 2 (comme pour les lettres),
- 3 euros de surtaxe si le colis dépasse 1/8 de m^3.

Le tarif de remboursement d'un *colis* est de :

- 10% de la valeur déclarée si le taux de recommandation est 1,
- 50% de la valeur déclarée si le taux de recommandation est 2,
- 0 si le taux de recommandation est 0.

La chaîne qui décrit un colis a la forme suivante (en italiques les données spécifiques d'un colis) :
`Colis` *code postal/destination/taux de recommandation/volume/valeur déclarée*
Par exemple : `Colis` *7854/famille Kaya, igloo 10, terres ouest/2/0.02/200*

▷ Les *colis express* sont des colis qui possèdent en plus les caractéristiques complémentaires suivantes :

- le poids doit être inférieur à 30kg,
- on stocke la date d'envoi (date du moment de l'appel du constructeur),
- on crée un numéro de suivi sur internet qui est défini au moment de la construction, (une variable statique `numeroColisExpress` augmente à chaque appel de constructeur),
- le tarif d'affranchissement est de 30 euros,
- le colis peut être muni de l'emballage proposé par la poste laponaise, dans ce cas le tarif d'affranchissement est augmenté de 3 euros.

La chaîne qui décrit un colis a la forme suivante (en italiques les données spécifiques d'un colis) : `Colis Express` *code postal/destination/taux de recommandation/volume/valeur déclarée/poids/numéro de suivi*
Par exemple : `Colis Express` *7855/famille Artick, igloo 90, baie des vents/2/0.02/2/20/20160128*

Question 1. Proposez un diagramme de classes pour les objets postaux.

Question 2. Dessinez une instance de colis express.

Question 3. Ecrivez les classes Java correspondantes.

Question 4. Proposez une application mettant en œuvre quelques instances de `lettre`, de `colis` et de `colis express`.

Associations et collections

1 Promotions

Vous avez déjà modélisé et implémenté une classe Etudiant. Nous allons étudier une classe Promotion, qui utilisera la classe Etudiant.

Question 1. Une promotion est un ensemble d'étudiants, pour une année donnée. Nous représentons cette relation entre la classe `Etudiant` et la classe `Promotion` par une association (voir Figure 1).

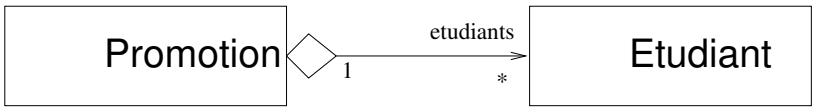

FIGURE 1 – Classes Etudiant et Promotion

a- Mettez en place la classe `Promotion` et l'association entre `Promotion` et `Etudiant`. Pour cela, on placera dans la classe `Promotion` une collection d'étudiants, on choisira la collection ArrayList. Cette collection aura une visibilité privée. On veillera à bien mettre dans la classe Promotion un attribut représentant l'année, les accesseurs associés, et deux constructeurs : un constructeur sans paramètres, et un constructeur prenant une année en paramètre. Les constructeurs initialiseront l'année et créeront la collection. La documentation de la classe ArrayList de Java est disponible via la documentation de l'API Java.

b- Écrire deux méthodes publiques manipulant la collection : une retournant le ième étudiant de la collection, et une retournant le nombre d'étudiants de la promotion.

c- Mettez en place une classe TestPromotion (avec un `main`) qui vous permet de tester votre classe `Promotion`.

Question 2. Complétez la classe `Promotion` avec les méthodes suivantes (on veillera à bien tester chacune des méthodes au fur et à mesure, grâce à la classe `TestPromotion`) :

- une méthode `inscrire` qui permet d'inscrire un étudiant dans la promotion ;
- une méthode `moyenneGenerale` qui retourne la moyenne générale de la promotion ;
- une méthode `afficheResultats` qui affiche une ligne pour chaque étudiant (correspondant au résultat de la méthode `ligneResultat`) ;
- une méthode `recherche` qui permet de retrouver un éudiant d'après son nom. On suppose qu'il n'y a pas d'homonymes ;
- une méthode `admis` qui retourne l'ensemble des étudiants admis ;
- une méthode `nouveauxInscritsNonFrancophones` qui retourne l'ensemble des nouveaux inscrits non francophones, elle est utilisée pour connaître les étudiants susceptibles de suivre des cours de soutien en français ;
- une méthode `majors` qui retourne les étudiants dont la moyenne est la plus élevée.

2 Utilisation de la classe Promotion

Écrivez une classe avec un `main` permettant d'effectuer les opérations suivantes (cette classe doit fortement ressembler à votre classe de test) :

- création d'une promotion vide d'étudiants,
- inscription des étudiants dans cette promotion (les étudiants doivent bien sûr être créés auparavant),

— affichage du nombre des nouveaux inscrits non francophones,
— attribution des notes aux étudiants,
— affichage du nom des majors de la promotion,
— affichage des résultats.

3 Transport du courrier

Revenons sur les objets postaux introduits lors des TD-TP précédents. Nous allons maintenant nous intéresser à la description des *sacs postaux*. Un *sac postal* peut contenir un certain nombre d'objets postaux, et dispose d'une capacité maximale. Cette capacité est de $0,5m^3$ pour un sac ordinaire. On peut fabriquer des sacs d'une autre capacité, sur demande précisant la capacité voulue.
On doit pouvoir ajouter un objet dans un sac, s'il y rentre. On veut connaître le volume occupé par un sac (compter forfaitairement $5dm^3$ pour la toile du sac), et sa valeur de remboursement en cas de perte. Enfin, on désire aussi pouvoir remplir un nouveau sac avec tous les objets de même code postal extraits d'un autre.

Question 3. Complétez le diagramme de classes déjà réalisé pour y intégrer la notion de sac postal.

Question 4. Ecrivez et testez la classe Java correspondante.

4 Pour ceux qui auraient fini en avance : Histogrammes

Complétez la classe Promotion de manière à pouvoir :
— afficher un histogramme des moyennes de la promotion. Par exemple, si les étudiants ont obtenu les moyennes suivantes : `Jacques 16,3, Justine 18, Germain 15,7, Hugues 12,2, Sylvia 15,9, Gaston 11,4, Astrid 11, Kim 11,1` , on affiche le diagramme suivant (limité entre 10 et 20 ici) :

```
[10 − 11[
[11 − 12[ ***
[12 − 13[ *
[13 − 14[
[14 − 15[
[15 − 16[ **
[16 − 17[ *
[17 − 18[
[18 − 19[ *
[19 − 20]
```

— connaître les moyennes les plus fréquentes,
— en utilisant l'histogramme, déterminer combien de personnes ont une certaine moyenne (arrondie) donnée,
— afficher le même histogramme qu'à la première question mais pivoté de 90° vers la droite.

```
[10 − 11[ [11 − 12[ [12 − 13[ [13 − 14[ [14 − 15[ [15 − 16[ [16 − 17[ [17 − 18[ [18 − 19[ [19 − 20]
          ***       *                     **      *             *
```

On veillera à ce qu'il n'y ait pas de duplication de code entre les deux méthodes d'affichage d'histogramme.

Instructions simples et introduction à l'environnement Eclipse

1 Eclipse : quelques manipulations de base

Eclipse est un environnement de développement gratuit, écrit en Java, et dédié à Java, ainsi qu'à d'autres langages grâce à la notion de plugin que nous n'aborderons pas ici. Nous indiquons ici des manipulations de base. Si vous connaissez déjà Eclipse, lisez ce qui suit et assurez-vous que vous connaissez toutes les manipulations indiquées. Il peut y avoir des variantes suivant les versions d'Eclipse.

2 Lancer Eclipse

Lancez Eclipse Luna ou Eclipse Neon par le menu Application/Développement de votre bureau (clic droit souris). Vous êtes invité à préciser votre workspace. Le workspace correspond à l'espace de travail d'Eclipse, c'est en fait un répertoire où seront notamment stockés vos fichiers sources. Vous pouvez avoir plusieurs workspaces si vous le souhaitez.

2.1 Création d'un projet

2 solutions :

1. Dans le package explorer (partie gauche de la fenêtre) : clic droit → new → project
2. Dans le menu : File → New → Project

Un wizard s'ouvre.

— Choisissez Java Project (Next)
— Nommez le projet (*e.g.* TP1). Eclipse crée dans le workspace un répertoire de ce nom.
— Garder la case Use Default Location cochée
— Choisir Create Separate folders for sources and class files, afin de séparer vos fichiers sources (d'extension .java) des fichiers de bytecode (d'extension .class). Eclipse va créer dans votre projet un répertoire src et un répertoire bin
— Dans la partie `JRE`, vérifiez que la JRE est bien 1.7 ou 1.8.
— Si vous faites Next, vous arrivez à des configurations fines qu'il n'est pas nécessaire de modifier pour l'instant. (faites Finish)

Dans l'explorateur de gauche, on voit le nouveau projet avec un répertoire src, et la référence à la librairie système JRE (cliquer sur le triangle pour voir s'ouvrir votre projet et observer ce répertoire).

2.2 Création d'un package

Dans l'explorateur de gauche, sur l'icône src dans votre projet, faire un clic droit → new → package. Dans le wizard, donnez un nom à ce package. le nom commence usuellement par une minuscule, (par exemple : `laPoste`).

2.3 Création d'une classe

Dans l'explorateur de gauche, sur l'icône du paquetage, faire un clic droit → new → class. S'ouvre un wizard "New Java Class".

— ne pas changer le source folder qui doit être correct, ni le nom du package, qui doit l'être aussi (vérifier)
— nommer la classe (ObjetPostal par exemple)
— ne pas toucher aux parties "superclass", "interfaces" pour le moment
— cocher la case `public static void main` si vous souhaitez mettre un `main` dans la classe.

Cela crée dans l'Explorer la classe à l'intérieur du package. Dans l'outline (à droite), s'ouvre un explorateur sur la classe faisant apparaître ses propriétés. Au centre, se trouve le code source de la classe. Vous pouvez contrôler ce qui a été généré (notez qu'il y a un embryon d'annotation pour la génération de la documentation). Notez un commentaire TODO qui signale qu'il faut compléter le code de la méthode main. Les TODO sont générés automatiquement ou ajoutés par le programmeur. L'ensemble des tâches à faire peut être visualisé (menu Window puis Show view puis Tasks).

Nota. Si vous perdez des fenêtres et vous trouvez perdu, allez dans le menu "Window" et choisissez reset perspective, vous retrouverez le package explorer, la fenêtre de code centrale et la fenêtre d'outline à droite.

2.4 Quelques manipulations du code Java

Vous pouvez maintenant écrire le code de la classe. Vous remarquerez qu'il est directement indenté. Si vous copiez du code non indenté ou si vous voulez rafraîchir l'indentation : sélectionner la partie concernée, clic droit → source → correct indentation (ou crtl +I).

Eclipse corrige quelques erreurs de compilation Créer une variable `clavier` de type `Scanner` :

- une petite croix apparaît en début de ligne pour signaler un problème (qui est explicité en passant la souris sur la croix),
- une petite lumière signale une solution,
- cliquer sur la lumière (clic gauche) et choisir la solution appropriée (ici : import java.util.Scanner).

Eclipse crée les éléments (classes, méthodes, ...) nécessaires Créer dans la classe ObjetPostal un attribut destination de type Adresse. La classe Adresse n'existant pas, Eclipse signale une erreur, et propose de la créer. Lancez effectivement la création de la classe Adresse. La classe Adresse est bien sûr vide. Eclipse peut également vous proposer de créer une méthode si vous faites appel à une méthode non encore développée. Dans ce cas, il génère juste l'entête de la méthode avec une note TODO, et pas son code, bien évidemment. Placez dans cette classe une méthode publique (par exemple : getCodePostal).

Complétion sémantique Définir la méthode getCodePostalDestination (`public float getCodePostalDestination()...`). Notez que la compilation se fait au fil de la saisie, et vous signale les erreurs en les soulignant en rouge. Quand on saisit un délimiteur ouvrant (ex. { ou ")"), il ajoute automatiquement le délimiteur fermant. Dans le corps de votre méthode, tapez `destination.` ("destination" suivi de '.'). La liste des méthodes définies pour l'objet destination est proposée ; il suffit de choisir celle qui nous intéresse (ici : getCodePostal) en double-cliquant dessus.

Génération des accesseurs et des constructeurs Créer un attribut dans la classe ObjetPostal `private float poids`. Sur le code, faire un clic droit → source → generate getters and setters. Les attributs sont proposés dans un wizard, où on choisit les attributs et les accesseurs désirés.
On peut générer des constructeurs à partir des attributs. Sur le code, faire un clic droit → source → generate constructors using fields. Sélectionner dans le wizard qui apparaît les champs que l'on veut prendre en paramètre. Pour l'instant, cocher la case *omit call to default constructor super()*.

Renommage Il existe une fonction de renommage des attributs et méthodes, classes, etc. Par exemple, on peut renommer "destination". Pour cela, dans l'outline, on fait un clic droit sur l'élément ("destination") → refactor → rename. Cela renomme toutes les occurrences de l'élément dans le code.

2.5 Exécution d'un programme

Finissez de développer votre premier programme. Pour exécuter le programme (qui contient un `main`), choisir menu run → run as → java application (ou utiliser l'icône avec la flèche blanche dans un disque vert).
Si la fenêtre "Console" n'apparaît pas en bas de votre environnement Eclipse avec le résultat de l'exécution, choisissez dans les menus `Window -> Show View -> Console`.

Les associations UML et les collections Java

Exercice 1 *Bibliothèque*

Une bibliothèque commence l'informatisation de son catalogue.

Une notice bibliographique a un identifiant unique qui est son ISBN. Une notice est caractérisée par son titre, éventuellement son sous-titre et ses contributeurs. Un contributeur est une personne qui a participé à la réalisation de l'ouvrage . Une même personne peut être contributrice dans plusieurs ouvrages, par exemple comme rédacteur, illustrateur, traducteur, rédacteur de la préface, etc. Par exemple, Daniel Pennac est auteur de "La fée Carabine" et rédacteur de la préface de "Une langue venue d'ailleurs" de Akira Mizubayashi. Une notice a un public cible qui est soit enfant, soit junior, soit adulte.

Un abonné possède un numéro d'abonné qui lui est communiqué lors de l'inscription. Il existe des abonnés particuliers qui sont les mineurs (abonné de moins de 18 ans).

La bibliothèque peut disposer de plusieurs exemplaires pour un livre donné. Un exemplaire peut être emprunté puis rendu par un abonné.

On désire également qu'une notice ait un attribut ou une méthode `estDisponible?` précisant si un des exemplaires de la notice est disponible à l'emprunt. Un exemplaire est indisponible quand il est emprunté ou quand le bibliothécaire l'a mis en réparation (l'exemplaire redevient disponible après réparation). Un catalogue est une collection de notices bibliographiques.

Un abonné peut emprunter jusqu'à 5 livres à la fois. Les abonnés mineurs ne peuvent emprunter que des ouvrages pour enfants ou juniors.

Question 1. Proposez un diagramme UML où vous ferez apparaître les classes, les attributs et les associations qui vous semblent judicieux. On s'intéressera particulièrement aux classes d'associations introduites pour l'emprunt et la contribution. Proposez un diagramme d'instance significatif.

Question 2. Proposez une traduction en Java.

Question 3. Modifiez votre modélisation et votre implémentation pour avoir, depuis la bibliothèque, un accès aux livres indexé par l'ISBN.

Exercice 2 *Relations maritales*

On s'intéresse une fois à la modélisation et l'implémentation d'un petit logiciel gérant les contrats de vie commune pouvant être établis entre deux personnes : le mariage, et le PACS (Pacte civil de solidarité). Dans la loi française, ces deux contrats de vie commune peuvent être contractés par deux personnes, quel que soit leur sexe. Un contrat de vie commune a une date de début et une date de fin (pour simplifier ici, on considèrera la date comme un entier et dans l'implémentation, quand un contrat de vie commune est encore en cours, la date de fin sera positionnée à -1). On souhaite conserver pour chaque personne connue du logiciel à la fois son contrat de vie commune courant (s'il existe un tel contrat) et les éventuels anciens contrats de vie commune. Un PACS est contracté dans un tribunal dont on veut stocker le nom, un mariage est contracté dans une mairie, dont on veut conserver le nom, et également le nom de la personne ayant célébré le mariage. De plus, un mariage est associé à un type de contrat de mariage (Communauté, séparation, autre).

Question 4. On s'intéresse tout d'abord au mariage seulement, avant l'existence du PACS et du temps où le mariage était réservé aux couples mixtes. Proposez une modélisation des couples par mariage, avec tour à tour une association réflexive, une classe d'association puis une classe à part entière. Discutez.

On se place maintenant dans le cadre plus général décrit plus haut (celui des lois françaises actuelles).

Question 5. Dans un diagramme de classes placez une classe `Personne`, une classe `ContratVieCommune`, et de quoi représenter également les mariages et les PACS. De plus prévoyez une classe pour représenter les organismes gérant ces aspects de l'état civil, qui seront dans notre contexte soit des mairies, soit des tribunaux.

Question 6. Complétez le diagramme afin de représenter :

- les dates de début et fin des contrats de vie commune (mariage ou PACS),
- le fait qu'un mariage ou un PACS lie 2 personnes, et qu'on conserve à la fois le contrat courant d'une personne, et ses contrats résiliés,
- le fait qu'un mariage est associé à un type de contrat de mariage, qu'on garde mémoire pour un mariage de la mairie de célébration, et du nom de la personne ayant célébré le mariage,
- le tribunal dans lequel est contracté un PACS.

Question 7. Ajoutez au diagramme l'ensemble des constructeurs pertinents.

Question 8. Donnez un diagramme d'objets représentant 2 personnes de nom A et B, ayant contracté un PACS en 2012 à la mairie de Montpellier, et n'ayant jamais été ni mariées ni pacsées auparavant.

Question 9. Ecrivez en Java l'ensemble du code nécessaire à l'implémentation d'une méthode permettant de `résilier` un contrat de vie commune. Quand on résilie un tel contrat, on s'assure que le contrat à résilier est bien le contrat courant des 2 personnes impliquées dans le contrat (si ce n'est pas le cas, on affiche un message d'erreur). On fixe la date de fin de contrat à la date courante . L'année courante peut être obtenue avec l'expression `Calendar.getInstance().getWeekYear();`.

Question 10. Ecrivez en Java l'ensemble du code nécessaire à l'implémentation d'une méthode permettant de `contracter` un contrat de vie commune. Cette méthode positionnera les deux personnes impliquées dans le contrat, et fera en sorte que l'instance courante devienne le contrat courant de ces deux personnes. Dans le cas d'un mariage, il faudra au préalable vérifier qu'aucune des deux personnes impliquées n'est mariée (si l'une ou les deux sont pacsées, les PACS contractés doivent automatiquement être résiliés). Dans le cas d'un PACS, il faudra au préalable s'assurer qu'aucune des deux personnes impliquées n'a de contrat de vie commune en cours.

Question 11. Ecrivez en Java une méthode qui permet de détecter les erreurs de dates dans les anciens contrats, c'est-à-dire qui vérifie qu'aucun ancien contrat n'en chevauche un autre. On pourra pour cela trier les contrats par date de début croissante.

Notes sur le débogueur Eclipse

1 Principes généraux

Un débogueur permet d'exécuter un programme en effectuant des pauses pour en étudier le comportement (par quelles instructions l'exécution passe) et l'état (c'est-à-dire les valeurs des variables ou attributs à un instant donné). Pour travailler avec le débogueur :

- on place des points d'arrêt pour faire des pauses dans l'exécution (on peut les supprimer à tout moment),
- on lance le débogueur pour aller de pause en pause (on peut à tout moment sortir du mode de débogage),
- on peut également avancer de ligne en ligne, ou descendre dans un appel de méthode,
- à chaque pause, on peut inspecter les variables ou les attributs.

Pour placer un point d'arrêt se placer sur la marge gauche (bande bleue) en face de la ligne avant laquelle on veut mettre le programme en pause, clic droit pour appeler le menu contextuel, puis choisir `Toggle Breakpoint` ou bien double-cliquer. Un point bleu apparaît alors en cet endroit.

Mettez en place le programme proposé en fin de sujet. Placez un point d'arrêt devant les lignes `FileAttente f = new FileAttente(); f.entre(p1); System.out.println("age moyen "+f.ageMoyen());` *du main, et* `m+=p.getAge();` *et* `return m/listePersonnes.size();` *de la méthode* `ageMoyen`

Pour supprimer un point d'arrêt se placer sur le point bleu, clic droit pour appeler le menu contextuel, puis choisir `Toggle Breakpoint` ou bien double-cliquer. Plutôt que de supprimer un point d'arrêt, on peut le désactiver.

Pour lancer le débogueur cliquer sur l'icône de la barre du haut représentant un petit insecte. Une fenêtre peut s'ouvrir pour vous prévenir que vous changez de perspective, acceptez. Cela a pour effet de passer dans une nouvelle présentation de votre programme, la `Debug perspective`. Plusieurs fenêtres vous présentent l'exécution et de nouvelles icônes sont apparues en haut :

- en haut à gauche, la fenêtre vous présente, au milieu du `Thread [main]` la pile des méthodes appelées (l'exécution s'est arrêtée dans la plus haute que vous observez), vous connaissez ainsi l'état d'exécution du programme
- en haut à droite, vous pouvez inspecter les variables et leur contenu. Pour les objets, si vous cliquez sur la ligne les représentant, la méthode `toString` est appelée et vous voyez son résultat. Si vous cliquez sur le triangle à gauche de l'objet, cela ouvre sa représentation et vous voyez toutes les valeurs de ses attributs
- en haut à droite, un autre onglet permet de voir les points d'arrêt placés. Vous pouvez leur donner des propriétés particulières en cliquant dessus comme :
  - le nombre de fois où on passera sur ce point d'arrêt sans l'activer (Hint count)
  - une condition disant que l'on va s'arrêter quand une condition devient vraie ou quand une valeur change (à remplir dans la case qui s'ouvre au-dessous)
- au centre à gauche, vous observez le code source (Java) de vos programmes en cours de test
- au centre à droite se trouve un aperçu synthétique sur le programme qui est en premier plan à gauche
- en bas, se trouve la console qui montre les résultats de l'exécution

Pour suivre l'exécution pas à pas Les différentes icônes apparues vont vous permettre d'effectuer ce suivi. Commencez par passer la souris sur ces icônes pour les reconnaître. Les principales sont :

- `resume` permet d'avancer de point d'arrêt en point d'arrêt
- `suspend` permet d'arrêter manuellement le programme

- `terminate` permet de stopper le programme au point où il est (utile s'il itère indéfiniment par exemple)
- `step into` fait entrer dans l'exécution d'une méthode s'il y en a une appelée à la ligne courante
- `step over` fait passer à la ligne suivante, même si la ligne courante contient un appel de méthode
- `step return` provoque le retour direct depuis un appel de méthode

Exécutez pas à pas le programme, en observant à chaque étape ce qui se passe. Corrigez au fur et à mesure les erreurs que vous trouvez grâce à l'inspection des variables (elles sont indiquées par des commentaires pour vous aider à les trouver). Notamment, lorsque votre programme s'arrêtera sur `f.entre(p1);`, regardez le contenu de la `ArrayList`. Inspectez une personne et un voyageur pour voir comment apparaissent les attributs hérités. Observez l'évolution des valeurs de la `ArrayList` et de la variable `m`.

Pour revenir dans la perspective Java classique Choisissez dans le menu `Window`, l'item `open perspective` puis `Java`

```
package testDebogueur;

public class Personne {
   private String nom, prenom;
   private int age;
   public Personne() {}

   public Personne(String nom, String prenom, int age) {
      this.nom = nom;
      this.prenom = prenom;
      this.age = age;
   }

   public String getNom() {return nom;}
   public void setNom(String nom) {this.nom = nom;}
   public String getPrenom() {return prenom;}
   public void setPrenom(String prenom) {this.prenom = prenom;}
   public int getAge() {return age;}
   public void setAge(int age) {this.age = age;}

   @Override
   public String toString() {return nom;}
}

public class Voyageur extends Personne {
   private String numCarte;

   public Voyageur() {}

   public Voyageur(String nom, String prenom, int age, String numCarte) {
      super(nom, prenom, age);
      this.numCarte = numCarte;
   }

   @Override
   public String toString() {return super.toString()+" " + numCarte;}
}
```

```
public class FileAttente {
   private ArrayList<Personne> listePersonnes; //correction: = new ArrayList<>();
   public FileAttente(){}
   public void entre(Personne p){listePersonnes.add(p);}
   public Personne sort(){
      Personne p = listePersonnes.get(0);
      listePersonnes.remove(p);
      return p;
   }
   @Override
   public String toString() {
      return "FileAttente [listePersonnes=" + listePersonnes + "]";
   }

   public double ageMoyen(){
      double m = 0;
      for (Personne p : listePersonnes)
         m+=p.getAge();
      return m/listePersonnes.size(); // attention division par 0 possible
      // correction: intégrer un test préliminaire
   }
}

public class ProgrammePersonne {
   public static void main(String[] arg){
      System.out.println("----");
      Personne p1 = new Personne("jean","tigra",27);
      Personne p2 = new Personne("Abdel","Martin",50);
      Voyageur v1 = new Voyageur("Anne","Foret",34,"ddf23h");
      FileAttente f = new FileAttente();
      f.entre(p1); f.entre(p2); f.entre(v1);
      System.out.println(f);
      System.out.println("----");
      System.out.println("age moyen "+f.ageMoyen());
      System.out.println("----");
      FileAttente f2 = new FileAttente();
      System.out.println("age moyen "+f2.ageMoyen());
      System.out.println("----");
      }
}
```

Interfaces

1 Extraction d'une interface

Question 1. On suppose connue une classe `Personne` et ci-dessous on vous donne le code d'une classe représentant des files d'attente de personnes. Proposez une interface décrivant le type de cette classe, c'est-à-dire ce qu'elle peut exposer de manière publique aux autres classes qui veulent l'utiliser.

```
public class FileAttente {
	private String nomFile;
	private static String reglementationFile = "sans priorité";
	private ArrayList<Personne> contenu;
	public FileAttente(){contenu=new ArrayList<Personne>();}
	public void entre(Personne p){contenu.add(p);}
	public Personne sort()
	{
		Personne p=null;
		if (!contenu.isEmpty())
			{p=contenu.get(0);
			contenu.remove(0);}
			return p;
			}
	public boolean estVide(){return contenu.isEmpty();}
	public int taille(){return contenu.size();}
	public String toString(){return ""+descriptionContenu();}
	private String descriptionContenu()
	{
		String resultat = "";
		for (Personne p:this.contenu)
			resultat += p + " ";
		return resultat;
	}
	public void vider(){
		int taille = taille();
		for (int i=0; i<taille; i++)
			this.sort();
	}
}
```

Question 2. Comment modifiez-vous la classe `FileAttente` pour qu'elle implémente l'interface que vous avez créée ?

Question 3. Proposez une interface pour représenter les files d'attente avec des statistiques. Des opérations supplémentaires permettent de connaître le nombre d'entrées et le nombre de sorties qui ont eu lieu depuis la création de la file.

Question 4. Proposez une classe qui implémente cette nouvelle interface (file d'attente avec statistiques).

Question 5. Dessinez le diagramme UML de vos classes et interfaces avec leurs relations (associations, spécialisation, implémentation).

2 Interfaces `Comparable` et `Comparator`

2.1 Ordre naturel entre objets avec l'interface `Comparable`

En Java, une interface de l'API permet de représenter des objets *comparables*. Elle est munie d'une opération de comparaison qui retourne 0 si les deux objets sont égaux (`equals` est appelée), un nombre négatif si le receveur précède l'argument, et un nombre positif si le receveur est un successeur de l'argument.

```
public interface Comparable<T>
{
   int compareTo(T o);
}
```

Elle est implémentée par un grand nombre de classes, dont la classe `String`, ce qui veut dire que cette dernière contient les éléments suivants. Remarquons que l'on déclare que les `String` sont comparables avec d'autres `String`.

```
public final class String extends Object implements Comparable<String> (...)
{
     ....
     public int compareTo(String anotherString){
     // ici code qui compare this et anotherString par ordre lexicographique
     }
     ....
}
```

Lorsqu'une classe implémente l'interface `Comparable`, l'ordre total défini par la méthode `compareTo` est appelé l'ordre naturel pour les objets de cette classe.
Cette interface permet de réaliser différents traitements sur une collection, tels que trouver le minimum, le maximum ou trier la collection. De telles méthodes se trouvent dans la classe `Collections`.
Pour vous faire comprendre son principe, nous montrons ci-dessous une méthode qui imite l'une des méthodes existantes. Elle retourne l'élément maximum pour toutes les collections vérifiant *l'interface List* et paramétrées par des éléments implémentant *l'interface Comparable*.

```
public static<E extends Comparable<E>> E max(List<E> c)
{
    if (c.isEmpty())
        return null;
    E max = c.get(0);
    for (E e : c)
        if (e.compareTo(max)>0)
            max = e;
    return max;
}
```

Le code ainsi écrit n'utilise que des interfaces. On remarque aussi que l'algorithme n'est pas une méthode d'instance. On peut s'en servir sur des ArrayList, des LinkedList, etc, pourvu qu'elles implémentent l'interface `List`. On a écrit un code "générique" au sens où il s'applique à un large ensemble de collections, comme le montre le programme ci-dessous (testez-le).

```
ArrayList<Integer> listeEntiers = new ArrayList<Integer>();
listeEntiers.add(4);  listeEntiers.add(8);
System.out.println(max(listeEntiers));

LinkedList<String> listeChaines = new LinkedList<String>();
listeChaines.add("galette");
listeChaines.add("crèpes");
listeChaines.add("bugnes");
System.out.println(max(listeChaines));
```

Question 6.
On veut appliquer la méthode `max` sur une liste de personnes (disposant d'un nom et d'un âge). On compare les personnes d'après l'ordre lexicographique de leur nom (ce serait l'ordre naturel entre les personnes).

- Comment devez-vous compléter la classe représentant des personnes (vous pouvez utiliser votre classe `Etudiant`) pour que ce soit possible ?
- Ecrivez un programme qui crée une liste de personnes, puis affiche une personne dont le nom est le plus grand dans l'ordre lexicographique.
- Recherchez dans la documentation de la classe `Collections` la méthode `max`, regardez sa signature et appliquez-la à votre liste de personnes pour obtenir (en principe) le même résultat que la méthode `max` de cet énoncé.
- Recherchez dans la documentation de la classe `Collections`, les méthodes `sort`, identifiez celle qui peut s'appliquer ici et appliquez-la à votre liste de personnes pour obtenir une liste triée (testez en affichant la liste après le tri).

2.2 Définitions de comparaisons entre objet avec l'interface `Comparator`

Maintenant imaginons que l'on ne veuille pas utiliser l'ordre naturel sur les objets, mais un autre ordre pour rechercher le maximum dans une liste ou pour trier celle-ci.
Pour cela, on définit une classe qui implémente une interface `Comparator` qui demande de définir une méthode `compare`. On ne se préoccupera pas de la méthode `equals` que contient cette interface.

```
public interface Comparator<T>
{
    int compare(T o1, T o2);
    boolean equals(Object obj);
}
```

Par exemple, pour définir un autre opérateur de comparaison entre chaînes de caractères, basé sur la longueur des chaînes plutôt que sur leur ordre lexicographique, on définit une classe implémentant l'interface `Comparator<String>` comme suit.

```
public class comparateurTailleChaines implements Comparator<String>
{
    public int compare(String s1, String s2) {
        boolean egal = (s1.length() == s2.length());
        boolean inf = (s1.length() < s2.length());
        if (egal) return 0;
        else
        if (inf) return -1;
            else return 1;
    }
}
```

On peut à présent définir une autre version de la méthode `max` admettant en paramètre une liste et un comparateur (cette méthode imite également une méthode de la classe `Collections`).

```
public static<E> E max(List<E> c, Comparator<E> comp)
{
    if (c.isEmpty())
        return null;
    E max = c.get(0);
    for (E e : c)
    if (comp.compare(e, max)>0)
        max = e;
    return max;
}
```

On peut alors l'utiliser pour trouver une plus longue chaîne dans une liste de chaînes (testez le programme ci-dessous).

```
LinkedList<String> listeChaines = new LinkedList<String>();
listeChaines.add("galette"); listeChaines.add("crèpe"); listeChaines.add("bugne");
listeChaines.add("crèpe dentelle");
System.out.println(max(listeChaines));
System.out.println(max(listeChaines,new comparateurTailleChaines()));
```

Question 7.

- Créez une classe comparateur de personnes, qui compare deux personnes suivant leur âge.
- Recherchez, dans la liste de personnes, une personne qui est l'une des plus âgées à l'aide de la méthode `max` de l'énoncé.
- Réalisez le même traitement avec la méthode `max` apparentée de la classe `Collections`.
- Trier la liste de personnes suivant leur âge, en recherchant une méthode `sort` appropriée (vérifiez en affichant la liste après le tri).

Étude de cas : quincaillerie

1 Cahier des charges

Un fabricant de quincaillerie souhaite informatiser son catalogue, et commencer la construction d'un portail de vente sur la toile. Nous ne nous intéresserons pas ici à ce qui concerne la partie interface web du portail de vente, mais à sa logique interne. On ne s'intéresse pas non plus à la gestion des stocks, qui peuvent donc être vus ici comme illimités.

1.1 Fonctionnement de l'application

Les clients peuvent choisir des pièces parmi le catalogue de pièces. Ces pièces sont ajoutées au panier d'achats. Quand le client valide son panier, il paye en ligne. S'il le souhaite, il peut obtenir une facture, éditée en texte brut (la facture contiendra les coordonnées du client, celles de la quincaillerie, la liste des achats avec leur prix individuel, et le prix total). On ne s'intéressera pas ici aux mécanismes à mettre en œuvre pour le paiement en ligne.

1.2 Gestion des clients

Les clients sont des entreprises ou des particuliers. Ces 2 types de client ont un nom, une adresse postale et électronique. Les particuliers ont en plus un prénom, une civilité (Mme, Mlle, M) et un sexe (M, F), alors que les entreprises ont un numéro de SIRET. Les clients restent anonymes lors de leur sélection d'achat, puis, au moment de la validation, ils doivent s'identifier. Ils peuvent être inconnus du système si c'est leur premier achat (dans ce cas on leur fera entrer leur coordonnées), ou connus s'ils ont déjà effectué un achat (dans ce cas, on demandera juste une confirmation des coordonnées). On ne s'intéressera pas à la sécurisation de l'identité des clients (pas de mot de passe notamment). L'adresse mail est l'identifiant d'un client.

1.3 Mise en place de la carte de fidélité

On souhaite mettre en place une carte de fidélité. La carte de fidélité est gratuite, et donc est automatiquement créée pour chaque nouveau client. Chaque achat rapporte des points de fidélité : 1 point par tranche de 5 euros d'achat (1 point pour un achat entre 5 et 10 euros, 2 points entre 10 et 15 euros, etc.).

Puis quand on atteint un nombre de points suffisant, on obtient des réductions sur une commande.

— Pour 10 points, le taux de réduction est 4%.
— Pour 20 points, le taux de réduction est 9%.
— Pour 30 points, le taux de réduction est 15%.

Quand on décide d'utiliser ses points pour obtenir une réduction, le nombre de points correspondant est retranché de la carte de fidélité. L'achat avec réduction permet lui aussi d'acquérir de nouveaux points de fidélité.

1.4 Le panier d'achat

Le panier représente les articles choisis. Quand on choisit plusieurs pièces de même référence (3 boulons identiques par exemple), on ne souhaite pas que le panier contienne directement ces pièces, mais plutôt la pièce, et la quantité souhaitée. On peut ajouter un élément au panier (on devra alors vérifier s'il existe déjà une pièce de même référence). On peut retirer une référence du panier, ou juste décrémenter le nombre de pièces souhaité pour une certaine référence. On peut aussi incrémenter le nombre de pièce souhaité pour une certaine référence. À tout moment, on peut visualiser le panier.

1.5 Description des pièces

Trois sortes de pièces se distinguent essentiellement, les pièces de base, les pièces composites en kit et les pièces composites montées.

Les *pièces de base* correspondent à des éléments de quincaillerie simples (vis, clou, rayon de roue, chambre à air, etc ...). Elles sont décrites par une référence de préfixe `00`, un prix, une durée de garantie (en mois), une durée de fabrication (en jours). On doit pouvoir éditer une fiche caractéristique sous le format suivant (les données propres à la pièce apparaissent en caractères italiques) :

`nom :` *vis*
`référence :` *007152*
`prix :` *0.01* euros
`garantie :` *12* `mois`
`durée de fabrication :` *1* `jour(s)`

Les *pièces composites* correspondent à des éléments de quincaillerie construits à partir d'autres éléments, simples ou eux-mêmes composites.

Les pièces composites *en kit* sont livrées en pièces détachées avec une notice de montage. Elles se caractérisent par une référence de préfixe `01` et une durée moyenne de montage par un particulier (en minutes). Leur prix se calcule en prenant la somme des prix de leurs composants. Leur durée de garantie s'obtient en prenant la plus courte durée de garantie parmi celles de leurs composants et en la divisant par deux (on ne fait pas confiance aux montages effectués par les particuliers). Leur durée de fabrication s'obtient en prenant la durée de fabrication la plus longue d'un composant. La fiche caractéristique a la forme suivante :

`nom :` *roue de brouette en kit*
`référence :` *011512*
`prix :` *24* euros
`garantie :` *10* `mois`
`durée de fabrication :` *4* `jour(s)`
`durée de montage particulier :` *15* `mn`
`composants :`
- *pneu - 004741*
- *chambre à air - 004565*
- *jante - 014541*
  - *disque de jante - 001214*
  - *rayon - 004748*
  - *rayon - 004748*
  - *rayon - 004748*

Les pièces composites *montées* se caractérisent par une référence de préfixe `02`, un prix de montage et une durée de montage en atelier (en jours). Leur prix se calcule en prenant la somme des prix de leurs composants à laquelle on ajoute le prix de montage. Leur durée de garantie s'obtient en prenant la plus

courte durée de garantie parmi celles de leurs composants et en lui ajoutant un bonus de garantie de 6 mois. Leur durée de fabrication s'obtient en prenant la durée de fabrication la plus longue d'un composant augmentée de la durée de montage en atelier. La fiche caractéristique a la forme suivante :

`nom :` *roue de brouette*
`référence :` *021512*
`prix :` *39* euros
`garantie :` *26* `mois`
`durée de fabrication :` *5* `jour(s)`
`durée de montage atelier :` *1* `jour(s)`
`prix du montage :` *15* euros
`composants :`
pneu - 004741
chambre à air - 004565
jante - 024541
disque de jante - 001214
rayon - 004748
rayon - 004748
rayon - 004748

2 Travail à réaliser

Question 1. Proposez un diagramme de classes UML pour décrire les pièces, en ne s'intéressant qu'à la structure des pièces. Donnez également le diagramme d'instances de la roue de brouette en kit.

Question 2. Faire apparaître dans la hiérarchie :
- les constructeurs,
- des méthodes `toString` retournant une chaîne de caractères décrivant succinctement les objets,
- une méthode `ajoute(p:Piece)` permettant de rajouter une pièce aux pièces composites.
- une méthode `prix():float`, qui retourne le prix d'une pièce quelconque.
- une méthode `dureeGarantie():float`, qui retourne la durée de garantie d'une pièce quelconque,
- une méthode `dureeFabrication():float`, qui retourne la durée de fabrication d'une pièce quelconque,
- une méthode `affiche()` qui imprime la fiche caractéristique d'une pièce quelconque en respectant strictement le format indiqué dans l'énoncé.

Question 3. Écrivez les classes Java correspondant à cette hiérarchie et créez une petite application qui montre le fonctionnement de vos classes.

Question 4. Complétez le diagramme de classes réalisé pour les pièces, de façon à prendre en compte le reste de l'énoncé.

Question 5. On souhaite que le panier puisse être trié par ordre de prix croissants ou décroissants des items contenus dans le panier, ou par ordre alphabétique du nom des pièces. Ajoutez de quoi trier les paniers.

Question 6. Implémentez l'ensemble de l'application. On veillera à ce que les clients enregistrés soient indexés par leur adresse mail.