Auteur(s) : *C. Vrain, B. Dao, J-P. Prost, M. Bougeret*

Objectif(s) : *Héritage simple*

Médiathèque

Exercice 1. Médiathèque

On souhaite réaliser un programme permmettant de gérer les adhérents d'une médiathèque et leurs emprunts. On considère les règles de gestion suivantes :

- les documents gérés sont de deux types : livre ou CD ;
- un document est identifié par un numéro ;
- un document a un titre ;
- un CD est un document qui a un interprète et un compositeur ;
- un livre est un document qui a un auteur ;
- certains documents sont empruntables, d'autres non (ils sont seulement consultables sur place).
- Un adhérent peut avoir jusqu'à cinq emprunts en cours qui seront indifféremment des livres et des CDs. Il ne peut emprunter de documents que s'il n'est pas en retard pour l'un de ses emprunts ;
- Un adhérent peut garder un livre pendant trois semaines mais ne peut garder un CD que deux semaines.

Les documents

Question 1.1. Construire une classe `Document` qui satisfait les règles ci-dessus. Penser à respecter le principe d'*encapsulation* (cf. cours). Redéfinir la méthode `toString()`, qui retourne un objet de type `String` décrivant l'*état* de l'objet au moment de l'appel.

Question 1.2. Construire les classes `Livre` et `CD` qui dérivent de la classe `Document`, et satisfont les régles de gestions ci-dessus. Penser toujours à respecter le principe d'encapsulation. Tester ces classes.

Question 1.3. Ajouter les méthodes suivantes (attention au choix de la classe de définition) :

- `public boolean estEmpruntable()`
- `public boolean emprunter()` qui retourne `TRUE` ou `FALSE` selon que l'emprunt a pu être effectué ou pas.

Question 1.4. Construire une classe `Mediatheque` qui permet de gérer un ensemble de documents, en utilisant un tableau. Cette classe doit être munie des fonctionnalités publiques suivantes :

- ajout d'un document quelconque
- recherche d'un document par titre
- recherche de tous les livres d'un auteur donné
- recherche de tous les CD d'un interprète ou compositeur donné

Un peu de tri...

On souhaite pouvoir trier les documents par type (d'abord les cds), puis par titre (alphabétiquement), et enfin par numéro (croissant) identifiant. Autrement dit, on souhaite trier les documents selon l'ordre lexicographique $<_{lex}$ sur les documents (on a donc $d_1 <_{lex} d_2$ ssi ..)

Question 1.5. Munir la classe `Document` d'une méthode permettant de comparer deux documents avec $<_{lex}$ (on rappelle que la classe `String` possède une méthode `public int compareTo(String t)` qui renvoit un entier négatif ssi la chaîne est plus petite que t)

Question 1.6. Munir la classe `Document` d'une méthode permettant l'affichage à l'écran de tous les documents à disposition, triés

Les adhérents (subsidiaire)

Question 1.7. Construire une classe `Emprunt` caractérisée par un attribut `dateEmprunt` de type `Calendar` (cf Annexe), un constructeur ayant un paramètre `dateEmprunt` de type `Calendar`, et une méthode de signature et retour `public abstract boolean estEnRetard()` permettant de déterminer si un emprunt est en retard.

Question 1.8. Construire deux classes dérivant de la classe `Emprunt` :

1. `EmpruntLivre` caractérisée par un attribut `leLivre` de type `Livre`, un constructeur avec deux paramètres `dateEmprunt` et `livre`, et la méthode `estEnRetard()`.
2. `EmpruntCD` caractérisée par un attribut `leCD` de type `CD`, un constructeur avec deux paramètres `dateEmprunt` et `leCD` et la méthode `estEnRetard()`.

Question 1.9. Redéfinir la méthode `toString()` pour chacune des trois classes ci-dessus.

Question 1.10. Construire une classe `Adherent` caractérisée par :

1. les attributs suivants :
   (a) `numAdh` de type `int`,
   (b) `nom` de type `String`,
   (c) `lesEmpruntsEnCours` de type tableau d'objets de type `Emprunt`, et
   (d) `nbEmpruntsEnCours` de type `int` ;
2. le constructeur `public Adherent(int numAdh, String nom)`
3. les méthodes suivantes :
   (a) `public boolean estEnRetard()` qui retourne `vrai` si au moins un des emprunts de cet adhérent est en retard ;
   (b) `public boolean empruntPossible()` qui retourne `vrai` si cet adhérent peut emprunter, c'est-à-dire s'il n'est pas en retard et s'il a moins de 5 emprunts en cours ;
   (c) `public boolean emprunterLivre(Livre, Calendar)`
   (d) `public boolean emprunterCD(CD, Calendar)`
   (e) `private void ajouterEmprunt(Emprunt)`
   (f) `public toString()`, qui retourne un objet de type `String` décrivant l'*état* de l'objet au moment de l'appel.