Examen janvier 2006

3 heures - documents écrits autorisés
calculatrice interdite

Vos classes-réponses appartiendront au paquetage `examen.reponse`.

On s'intéresse au système d'information d'une médiathèque dans lequel un ensemble de *documents* est répertorié : des livres, films, ou disques. Ces documents peuvent être empruntés par les clients de la médiathèque.

Les documents sont tous caractérisés par un titre et un auteur. Du point de vue programmation, un *document* est caractérisé par l'interface ci-dessous :

```
package examen.sujet;
import examen.reponse.Auteur;
public interface Document {
   /** @return l'auteur de ce document  */
   public Auteur getAuteur();

   /** @return le titre de ce document */
   public String getTitre();

   /** @return une chaîne de caractères représentant ce document */
   public String toString();

   /** @param l'objet à tester
     * @return true si le paramètre est un document identique à this */
   public boolean equals(Object o);

   /** indique si le document est emprunté ou non
    * @return true ssi le document est emprunté
    */
   public boolean estEmprunte();
   /** permet de marquer comme emprunté ou non un document
    * @param b true si le document est emprunté, false si il est disponible
    */
   public boolean setEmprunte(boolean b);
}
```

Q 1 . La classe `Auteur` permet de réprésenter les auteurs des documents. Un auteur est défini par son nom, de type chaîne de caractères, ses dates de naissance et de décès de type `examen.sujet.Date`. La valeur de la date de décès est `null` si l'auteur est encore vivant. On veut pouvoir accéder aux différentes informations sur un auteur et tester si deux objets auteurs sont égaux (mêmes nom et dates de naissance/décès).

La classe `Date` est définie en annexe.

Donnez un code java complet pour la classe `Auteur`.

Q 2 . On considère le squelette de code suivant de la classe `BaseDocuments` qui permet de gérer la base documentaire de la médiathèque.

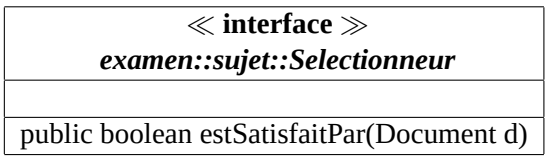
En vous basant sur le code javadoc fourni, donnez un code java **complet** pour la classe `BaseDocuments` (et donc complétez aux endroits mentionnés).

```
COMPLETER

public class BaseDocuments {

   COMPLETER // attribut(s) et constructeur(s)
```

```
    /** ajoute un document dans la base de documents
    * @param d le document à ajouter
    */
       COMPLETER méthode ajoute
    /** supprime un document de la base de documents (si il est plusieurs
    * fois présents, une seule des occurrences est supprimeée).  Il
    * importe peu que le document soit emprunté ou non ,au moment de sa
    * suppression.
    * @param d le document à supprimer
    * @return le document supprimé, null si il n'existe pas
    */
       COMPLETER methode supprime
    /** affiche tous les documents de la mediathèque (pour impression
    * du catalogue par exemple)
    */
       COMPLETER méthode affiche
/** indique si le document donné est disponible (non emprunté)
    * @param d le document concerne
    * @return true si le document donné est disponible (non emprunté)
    * @exception NoSuchElementException si le document n'existe
    * pas dans la mediathèque
    */
       COMPLETER methode estDisponible
    /** permet d'emprunter un document
    * @param d le document concerne
    * @exception NoSuchElementException si le document n'existe
    * pas dans la mediathèque
    */
       COMPLETER méthode emprunte
    /** permet de rendre un document
    * @param d le document concerne
    */
       COMPLETER méthode rend
}
```

Q 3 . On souhaite disposer d'un "moteur de recherches" permettant de réaliser la sélection d'un ensemble de documents de la base de documents selon différents critères.

Pour cela on définit une interface `Selectionneur` qui permet de déterminer les documents qui satisfont un critère :



On dit qu'un document `d` **satisfait** un sélectionneur `s` si et seulement si `s.estSatisfaitPar(d)` vaut `true`.

Pour un sélectionneur donné, il est ensuite facile de récupérer les documents d'une base de documents qui *satisfont* ce `Selectionneur`.

Q 3.1. On ajoute à la classe `BaseDocuments`, la méthode de signature :

```
public Iterator selectionne(Selectionneur s)
```

qui retourne un itérateur sur la collection des documents de la base de documents qui satisfont le sélectionneur `s`.

Donnez un code java pour cette méthode.

Q 3.2. Donnez un code java permettant de définir un sélectionneur satisfait par les documents non empruntés.

Q 3.3. Donnez un code java permettant de définir un sélectionneur satisfait par les documents dont le titre contient un mot `m` (une chaîne de caractères) donné (voir l'annexe pour une documentation sur une méthode utile).

Q 3.4. Donnez un code java permettant de définir un sélectionneur satisfait par les documents dont l'auteur était vivant à une année donnée.

Q 3.5. On veut pouvoir effectuer des sélections multi-critères, c'est-à-dire récupérer les documents qui satisfont **simultanément** plusieurs `Selectionneur`s. On parlera de *sélectionneur composite*. Un *sélectionneur composite* est de type `Selectionneur`. Donnez un code java pour une classe permettant de créer des sélectionneurs composites qui réalisent le "et logique" de plusieurs autres objets `Selectionneur`. Un document satisfait un tel sélectionneur si il satisfait chacun des sélectionneurs qui le composent. On définit un sélectionneur composite par ajouts successifs des sélectionneurs qui le composent (méthode `add(Selectionneur s)`).

Q 3.6. Donnez un code java permettant pour un objet base de documents `bdDoc` (initialisé) de récupérer la sélection des documents dont le titre contient le mot `"Anneaux"` puis pour ce même objet base de récupérer la sélection des documents dont l'auteur était vivant en 1832, enfin de récupérer les documents qui satisfont ces deux critères.

Q 4 . Les clients de la médiathèque ont le droit d'emprunter un et un seul document pendant au plus 15 jours.

La classe `Emprunteur` est définie comme indiqué ci-dessous.

L'attribut `document` correspond au document emprunté et vaut `null` si il n'y en a aucun et l'attribut `dateEmprunt` est la date à laquelle ce document a été emprunté, `null` si aucun document n'est emprunté. Une exception est levée si l'emprunteur tente d'emprunter un second document (méthode `empruntеDoc`).

Chaque client dispose d'une carte sur laquelle se trouve un code barre qui permet d'identifier l'emprunteur. Un code barre est instance de la classe `CodeBarre` dont un extrait est également donné ci-dessous.

Emprunteur
- document : Document - dateEmprunt : Date - nom : String
+Emprunteur(nom : String) + rendDocument() +emprunte(doc : Document) throws Exception +getDocument() : Document +getDateEmprunt():Date +getNom():String

CodeBarre
...
... +hashcode() : int +equals(o : Object)

Q 4.1. Après avoir examiné le squelette de code de la classe `Mediatheque` ci-dessous, quelle est selon vous la structure de données la mieux adaptée pour représenter la donnée `lesEmprunteurs`.

Q 4.2. Complétez le code de la classe `Mediatheque`.

```
package examen.reponse;
import examen.sujet.*;
COMPLETER
public class Mediatheque {

   private BaseDocuments bdDoc;
   private COMPLETER lesEmprunteurs;
   public Mediatheque(BaseDocuments bdDoc) {
      COMPLETER
   }
```

```
    /** ajoute un nouveau client qui sera identifié par le code barre
    * donné
    */
    public void ajouteClient(CodeBarre cb, Emprunteur truc) {
       COMPLETER
    }
    /** le client identifié par le code barre emprunte le document d
    * si c'est possible (document libre et pas d'autre document emprunté)
    * sinon une exception est levée.
    */
    public void empruntеDoc(CodeBarre cb, Document d) throws Exception {
       COMPLETER
    }
    /** le client identifié par cb rend son document, il ne se passe rien
    * si il n'en avait emprunté aucun */
    public void rendDoc(CodeBarre cb) {
       COMPLETER
    }
    /** retourne un itérateur sur les clients qui sont en retard pour
    * rendre leur document au jour d'aujourd'hui*/
    public Iterator enRetard() {
       COMPLETER
    }
}
```

Annexe

La classe `examen.sujet.Date`

La classe `examen.sujet.Mois` est un type énuméré supposé défini (Les valeurs du type sont `Mois.JANVIER`, `Mois.FEVRIER`, ···, `Mois.DECEMBRE`). On supposera que dans le sujet les dates sont toujours valides.
Cette classe implémente l'interface `java.util.Comparable`.
La méthode `differenceEnJours` donne la différence en jours entre "`this`" et le paramètre `d` et la méthode statique `aujourdhui` retourne la date du jour.

examen::sujet::Date
- jour : int - mois : Mois - annee : int
+ Date(j : int, mois : Mois, annee : int) + toString() : String + equals(o : Object) : boolean + getJour() : int + getMois() : Mois + getAnnee() : int + compareTo(o : Object):int + differenceEnJours(d : Date):int + static aujourdhui():Date

Extraits de javadoc de la classe `java.lang.String`

indexOf

`public int indexOf(String str)`

Returns the index within this string of the first occurrence of the specified substring. The integer returned is the smallest value k such that:

```
this.startsWith(str, k)
```

is `true`.

Parameters:

`str` - any string.

Returns:

if the string argument occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring, -1 is returned.

Throws:

`NullPointerException` - if `str` is `null`.

startsWith

`public boolean startsWith(String prefix, int toffset)`

Tests if this string starts with the specified prefix beginning a specified index.

Parameters:

`prefix` - the prefix.

`toffset` - where to begin looking in the string.

Returns:

`true` if the character sequence represented by the argument is a prefix of the substring of this object starting at index `toffset`; `false` otherwise. The result is `false` if `toffset` is negative or greater than the length of this `String` object.

Throws:

`NullPointerException` - if `prefix` is `null`.