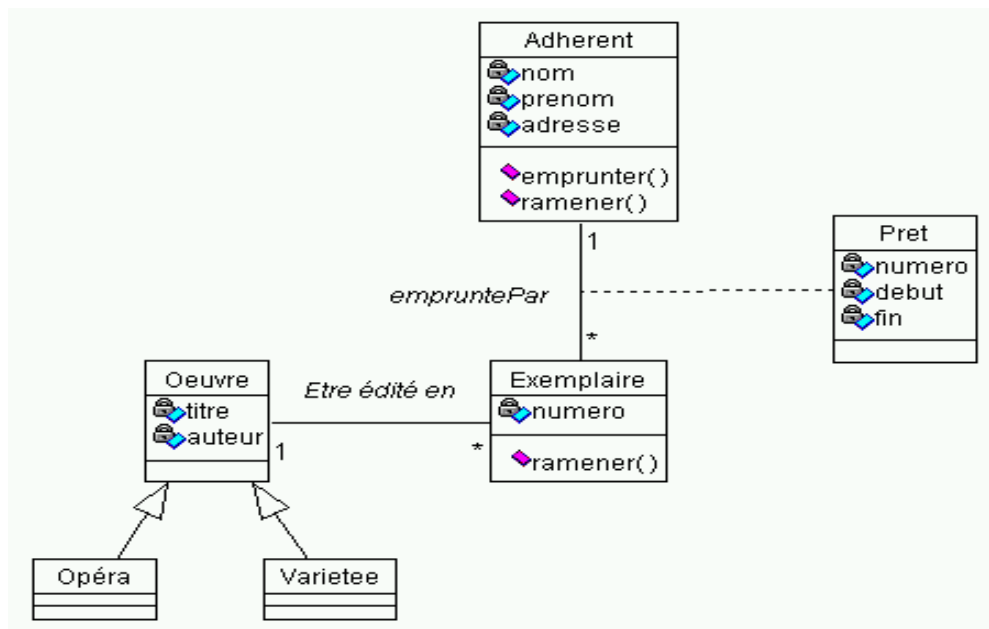


Projet : Gestion d'une médiathèque

Remarque préliminaire : les différentes parties de Java que vous allez faire forment un tout (un programme de gestion d'une médiathèque). Il est donc important d'arriver « au bout » de chaque partie.

Diagramme de classes V1



Le diagramme des classes ci-dessus modélise le système informatique simplifié d'une médiathèque : un adhérent peut emprunter un ou plusieurs exemplaires parmi deux types d'œuvres (des musiques d'opéras et des musiques de variétés).

En vous aidant du programme principal suivant, écrire l'ensemble des classes de diagramme ci-dessus :

```
public static void main( String [] args ){
    try{
        Adherent adherent1 = new Adherent( "Albert", "Durant", "2 allée tataouine" );
        Adherent adherent2 = new Adherent( "Joseph", "Beauve", "14 rue de la grange" );

        String titre = "BestOfLouisMariano" ;
        String auteur = "Louis Mariano" ;
        int nombreExemplaire = 3 ;
        Varietee varietee1 = new Varietee( titre, auteur, nombreExemplaire ) ;

        System.out.println( "Adherent 1 avant emprunt = " + adherent1 ) ;
    }
}
```

```
Exemplaire exemplaire = adherent1.emprunter( varietee1 );
System.out.println( "Adherent 1 apres emprunt = " + adherent1 );

adherent1.ramener( exemplaire );
System.out.println( "Adherent 1 apres restitution = " + adherent1 );

} catch( Exception e ){
    e.printStackTrace();
}
}
```

L'association « être édité en » entre les classes Œuvre et Exemplaire doit être initialisée quand une œuvre d'un type donné est créée. Par exemple l'instruction suivante :

```
Varietee varietee1 = new Varietee( titre, auteur, nombreExemplaire );
précise que l'œuvre varietee1 est édité en nombreExemplaire exemplaires.
```

Pour coder l'association « emprunterPar » entre les classes Adherent et Exemplaire, vous pouvez utiliser une table de hachage (java.util.Hashtable) :

```
public class Adherent{
    Java.util.Hashtable dicoExemplairePret = new java.util.Hashtable();
    // ...
}
```

où les clefs de la table de hachage sont des exemplaires et les valeurs des prêts. Cette association doit être mise à jour quand un adhérent emprunte ou ramène un exemplaire comme dans :

```
Exemplaire exemplaire = adherent1.emprunter( varietee1 );
adherent1.ramener( exemplaire );
```

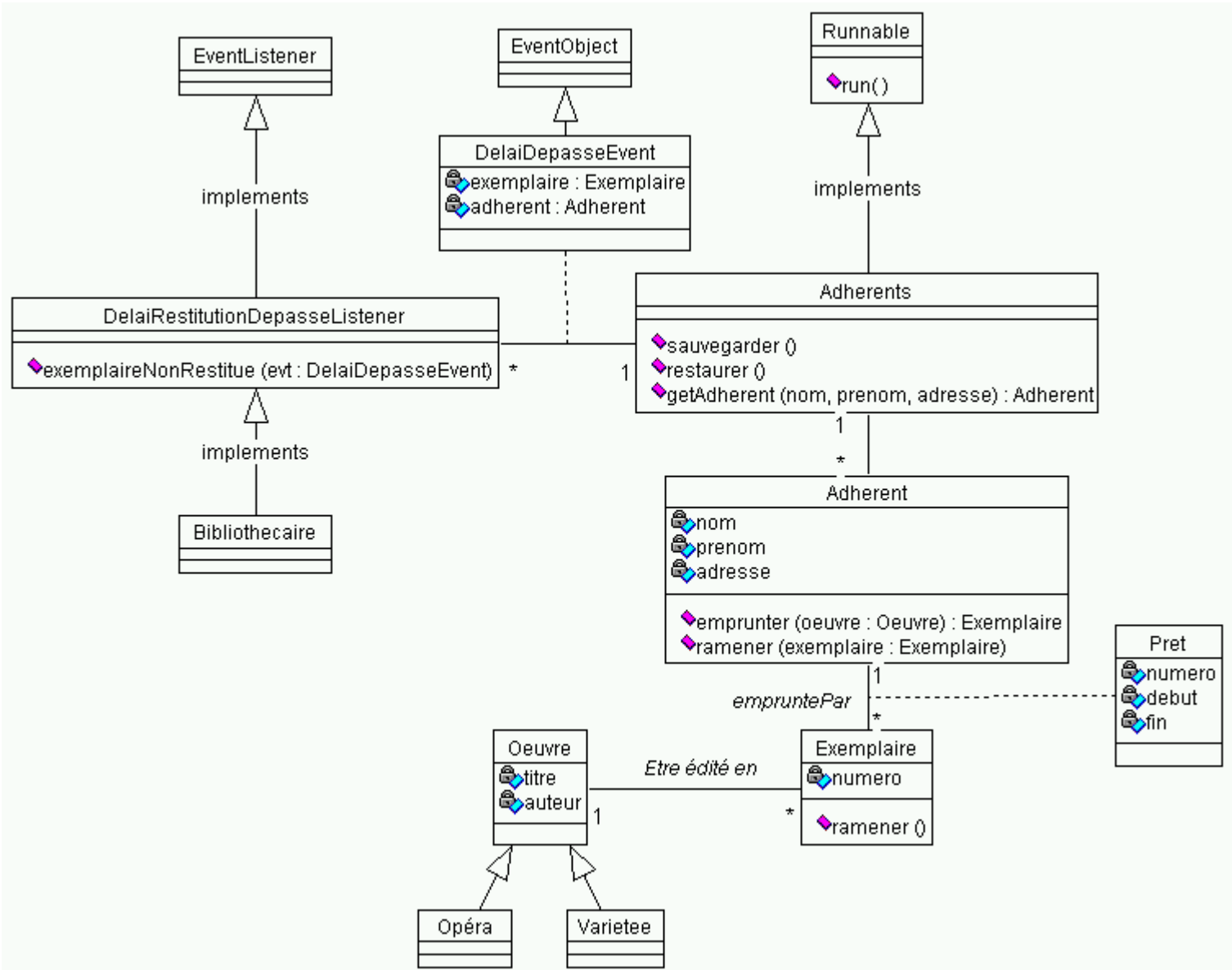
Lors d'un emprunt, il ne faut pas oublier de décrémenter le nombre d'exemplaires associés à une œuvre signifiant ainsi qu'il y a un exemplaire en moins dans la médiathèque. A l'inverse, il faut incrémenter le nombre d'exemplaires associés à une œuvre quand un adhérent ramène un exemplaire.

Pour pouvoir utiliser un objet dans une instruction telle que
System.out.println("Adherent 1 avant emprunt = " + adherent1);

vous devez surcharger la méthode
public String toString();

de la classe Object dans chacune des classes dont une instance peut être utilisée pour un affichage.

Diagramme de classes V2



Gestion objet !

Le but de cette partie est d'implanter un mécanisme pour pouvoir prévenir automatiquement le bibliothécaire quand un adhérent a conservé un exemplaire au-delà du temps normal d'un prêt. Pour cela vous allez utiliser une technique issue des Java Beans (un modèle à composants de Java). Cette technique est illustrée sur le diagramme des classes précédant. Il s'agit de pouvoir appeler une méthode de la classe Bibliothecaire (exemplaireNonRetitue) dès qu'un prêt excède le délai habituel.

Pour stocker l'ensemble des adhérents, vous pouvez ajouter une classe supplémentaire à votre modèle (la classe Adhérents qui est en association avec la classe Adhérent). Ces classes en association peuvent s'utiliser avec les instructions suivantes :

```
Adherents adherents = new Adherents() ;

Adherent adherent1 = new Adherent( "Albert", "Durant", "2 allée tataouine" ) ;
Adherent adherent2 = new Adherent( "Josef", "Beauve", "14 rue de la grange" ) ;

adherents.addAdherent( adherent1 ) ;
adherents.addAdherent( adherent2 ) ;
```

C'est la classe Adherents qui doit se charger d'appeler la méthode exemplaireNonRestitue de la classe Bibliothecaire. Pour ce faire, ces deux classes doivent être associées (voir le schéma ci-dessus). Ceci peut être réalisé avec les instructions suivantes :

```
Bibliothecaire bibliothecaire = new Bibliothecaire() ;
Adherents adherents = new Adherents() ;
adherents.addDelaiRestitutionDepasseListener( bibliothecaire ) ;
```

où le squelette de la méthode addDelaiRestitutionDepasseListener s'écrit :

```
class Adherents{
    public void addDelaiRestitutionDepasseListener( DelaiRestitutionDepasseListener l ){
        // ...
    }
}
```

Comment la méthode addDelaiRestitutionDepasseListener peut-elle recevoir un bibliothecaire alors que son argument est de type DelaiRestitutionDepasseListener ? Quelle est l'intérêt de procéder ainsi ?

Après avoir réalisé l'association, un objet du type Adherents peut prévenir le bibliothécaire en appelant sa méthode ExemplaireNonRestitue. Cette méthode reçoit en argument un objet de type DelaiDepasseEvent sensé contenir des informations sur l'exemplaire en litige. C'est pourquoi la classe DelaiDepasseEvent a deux attributs, exemplaire et adhérent, qui sont l'adhérent et l'exemplaire en litiges.

A présent, il ne reste plus qu'à rechercher parmi tous les adhérents ceux qui ont des exemplaires non rendus. Idéalement, cette recherche peut se faire en tâche de fond en utilisant un thread (classe Thread). Voilà pourquoi la classe Adherents implémente Runnable. Une première version pourra être simpliste en utilisant une méthode de vérification.

Les informations sur les prêts en cours pour un adhérent donné sont stockées dans la classe d'association Prêt codée par une table de hachage. L'algorithme de recherche d'un exemplaire en litige peut s'écrire comme suit :

```
Pour tous les adhérents d'une instance de la classe Adhérents :  
    pour tous les prêts d'un adhérent :  
        S'il existe des prêts en litige :  
            Appeler la méthode ExempleNonRestitue de la classe Bibliothecaire.  
        Fin de si.  
    Fin de pour.  
Fin de pour.
```

Les prêts étant codés dans une table de hachage, vous allez devoir la parcourir. Cela peut se faire avec la boucle :

```
for( Enumeration e=dico.elements() ;e.hasMoreElements(); ){  
    Pret pret = (Pret)e.nextElement() ;  
}
```

Où dico est la table de hachage.

Remarques :

- Il y a deux classes dont on n'a pas encore parlé : EventListener qui est en fait une interface, et EventObject. Ces classes sont utilisées pour se conformer aux spécifications des Java Beans ;
- Le modèle à composants de Java (les Java Beans) est bien plus riche que ce que vous avez utilisé dans ce module.

Gestion de la Persistance Objet

Dans cette partie, vous allez programmer la persistance de votre application. Il s'agit de pouvoir arrêter l'application à n'importe quel moment, la sauvegarder sur un support persistant, un fichier en l'occurrence, puis il doit être possible de la restaurer en état de marche : c'est à dire restaurer tous les objets, avec toutes les valeurs de leurs attributs mais aussi toutes les relations entre les objets ! Un exemple d'utilisation de la persistance est donné par le programme suivant :

```
Adherents adherents = new Adherents() ;  
adherents.sauvegarder() ;  
adherents.restaurer() ;
```

où toutes l'application est sauvegardée et restituée en un seul appel de méthode !

Rassurez-vous, Java est là pour vous aidez ! cela se fait en quelques lignes de codes. Commencez par réaliser la sauvegarde des attributs de la classe Adherents dans la méthode sauvegarder. Tentez d'exécuter l'application ainsi modifiée. Une exception doit alors être générée à l'exécution. Faites les corrections nécessaires pour éliminer l'exception, puis lancez l'exécution de votre programme à nouveau. Procédez ainsi de suite, et petit à petit, jusqu'à ce que tous les objets soient sauvegardés.

Maintenant que vous avez écrit la sauvegarde, la restauration ne doit pas vous poser de gros problèmes.

A présent vous allez vérifier que la persistance est correcte. Pour ce faire utilisez le programme principal suivant, où un adhérent emprunte un exemplaire avant que l'application soit sauvegardée. L'exemplaire ne sera rendu qu'après avoir restauré l'application, preuve que tous les objets mais aussi toutes les relations entre objets ont été rétablies. Vous remarquerez cependant dans le programme que les références d'objets ne sont pas persistantes : en effet la restauration est suivie de l'appel de la méthode getAdherent qui retourne une nouvelle référence sur un exemplaire. Quelle-en est selon vous la raison ?

```
// ...  
Adherents adherents = new Adherents( 2000 ) ;  
Adherent adherent1 = new Adherent( "Albert", "Durant", "2 allée tataouine" ) ;  
  
String titre = "BestOfLouisMariano" ;  
String auteur = "Louis Mariano" ;  
int nombreExemplaire = 3 ;  
Variete varietee1 = new Variete( titre, auteur, nombreExemplaire ) ;  
  
Exemplaire exemplaire = adherent1.emprunter( varietee1 ) ;  
  
adherents.sauvegarder() ;  
adherents.restaurer() ;
```

```
Adherent adherentReconstruit = adherents.getAdherent( "Albert", "Durant", "2 allée tataouine" );  
Vector exemplairesEmpruntes = adherentReconstruit.getExemplairesEmpruntes();  
for( int i=0; i<exemplairesEmpruntes.size(); i++ ){  
    exemplaire = (Exemplaire)exemplairesEmpruntes.elementAt( i );  
    adherentReconstruit.ramener( exemplaire );  
}
```

Conseil : pour pouvoir exécuter correctement le programme ci-dessus, vous aurez besoin de redéfinir la méthode

```
public boolean equals( Object object )
```

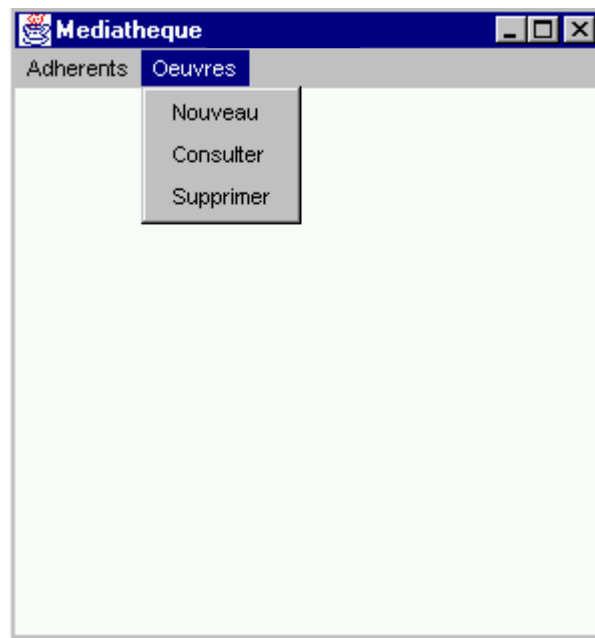
de la classe Object dans certaines de vos classes.

Remarque : une application plus réaliste utilisera une base de données pour assurer la persistance d'une application ;

Interface Graphique

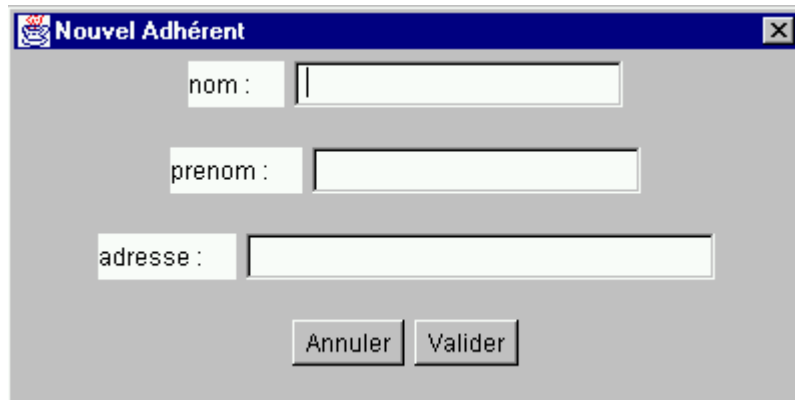
Vous allez concevoir l'interface graphique de votre application. Reportez-vous souvent au cours (chapitre AWT) où tout est déjà quasiment fait, mais de façon parcellaire.

La fenêtre d'accueil de la médiathèque doit avoir l'apparence suivante :



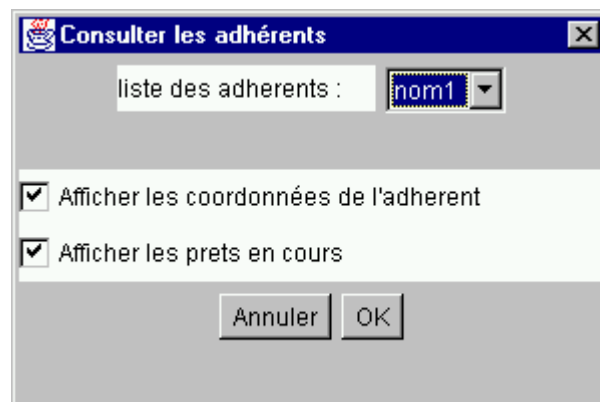
où les deux menus Œuvres et Adherents ont un sous-menu ayant la même apparence. Un clic sur le sous-menu Nouveau du menu Adherents doit produire la fenêtre suivante :

JAVA 2



A Java Swing dialog box titled "Nouvel Adhérent" with a standard window icon and a close button (X). The dialog contains three text input fields: "nom :", "prenom :", and "adresse :". Below the fields are two buttons: "Annuler" and "Valider".

La fenêtre suivante doit apparaître quand on sélectionne le sous-menu Consulter du menu Adherents :



A Java Swing dialog box titled "Consulter les adhérents" with a standard window icon and a close button (X). The dialog contains a label "liste des adherents :" followed by a dropdown menu showing "nom1". Below this are two checked checkboxes: "Afficher les coordonnées de l'adherent" and "Afficher les prêts en cours". At the bottom are two buttons: "Annuler" and "OK".

La suppression d'un adhérent se fait dans la fenêtre ci-dessous qui doit apparaître en cliquant sur le sous-menu Supprimer du menu Adherents :



Concevez aussi les fenêtres associées au menu Œuvre en vous inspirant de ce qui a été fait précédemment.

Intégrez les objets métiers dans l'interface graphique.