Cartouche du document

Année : ING 1
Matière : UML
Activité : Travail dirigé

Objectifs

L'objectif principal de ce travail dirigé est d'aborder le passage de l'analyse à la conception.

Nous imposons comme choix de conception de générer du code Java.

Nous étudierons des règles de passages ou de Mapping des classes d'analyse.

- **vers des classes Java.**

Les régles concernent
- La correspondance entre classe d'analyse et classe Java (attributs et opérations)
- Les méthodes get et set pour récupérer et mettre à jour les attributs d'un objet
- L'information contenue dans les associations
  - - La navigabilité : qui connait qui ?
  - - Les cardinalités : en quelles quantités ?
  - - Les liens d'existence entre les objets associés : qui dépend de qui ?
- L'information contenue dans les automates d'états des objets

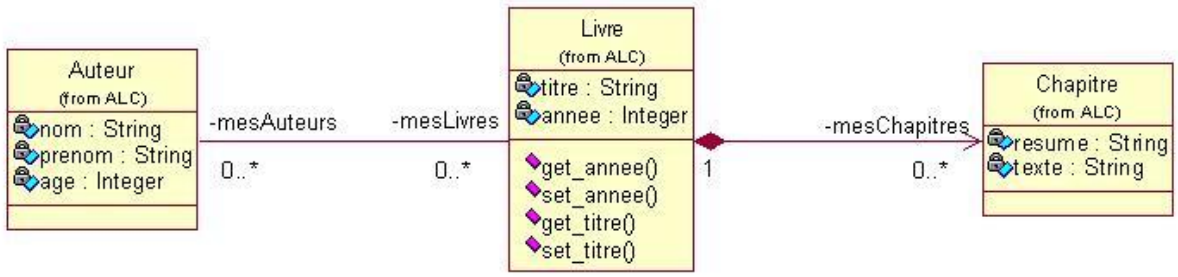
Sommaire des exercices

Corps des exercices

1 - UML vers Java : Association, composition et agrégation

Énoncé :

Dans le cadre d'une gestion de bibliothèque, on vous demande de traduire le schéma d'analyse ci-dessous en classe Java.



On vous demande de traduire en Java, ce diagramme de classes

Question 1)

Énoncé de la question

Que déduisez de ce schéma pour les cardinalités ?

Solution de la question

Pour les cardinalités, on en déduit :

- Un auteur peut avoir participé à la création de plusieurs livres
- Un livre peut être composé de plusieurs chapitres
- Un livre peut avoir été écrit par plusieurs auteurs.

Question 2)

Énoncé de la question

Que déduisez-vous de ce schéma pour la vie des objets ?

Solution de la question

Les auteurs et les livres ont une vie indépendante, mais si un objet Livre disparaît, il faut mettre à jour tous les objets Auteur concernés et réciproquement.

Un objet Javahapitre appartient à un objet Livre. Un objet Javahapitre ne peut donc exister que dans un objet Livre. On en déduit la durée de vie d'objet Javahapitre par rapport à la durée de vie d'un livre comme l'indique le schéma ci-desous.

Question 3)

Énoncé de la question

Que déduisez-vous de ce schéma pour la navigabilité ?

Solution de la question

Pour la navigabilité, on en déduit qu' :

- un livre connaît ses chapitres

- un auteur connaît ses livres
- un livre connaît ses auteurs

Question 4)

Énoncé de la question

Faire le mapping des classes d'analyse en Java ?

On vous donne ci-dessous quelques éléments de code Java à compléter :

- Le fichier

```
/**
 Le fichier auteur.java
*/
public class Auteur1 {
  private String nom;
  private String prenom;
  private int age;

  // Pour gérer les différents livres d'un objet Auteur


  public Auteur1(String pnom, String pprenom, int page) {
    nom = pnom;
    prenom = pprenom;
    age = page;
  }

  public int getAge() { return age; }

  String getNom() { return nom; }
  String getPrenom() { return prenom; }

  // je dois pouvoir associer des livres à un auteur

  // je dois pouvoir dissocier des livres d'un auteur

}
```

- Le fichier

```
/* Le fichier Livre1.java */
public class Livre1 {
  private String titre;
  private int annee;
```

```
/* Pour gérer les différents auteurs d'un objet livre */


/* Pour gérer les différents chapitres d'un objet livre */

public Livre1(String ptitre, int pannee) {
  titre = ptitre;
  annee = pannee;
}

public int getAnnee() { return annee; }
String getTitre() { return titre; }

// Il faut pouvoir associer des auteurs à un objet Livre

// Il faut pouvoir créer des objets Chapitre à un objet Livre

// Il faut pouvoir supprimer des chapitres à un objet Livre

// Il faut pouvoir dissocier des auteurs à un objet Livre

}
```

- Le fichier

```
/* Le fichier Chapitre1.java */
public class Chapitre1 {
  private String resume;
  private String texte;

  public Chapitre1(String presume, String  ptexte) {
   resume = presume;
   texte = ptexte;
  }

  String  getResume() { return resume; }
  String  getTexte() { return texte; }
}
```

Solution de la question

Voici les fichiers complétés :

- Le fichier

```
/**
```

```
 Le fichier auteur.java
*/
public class Auteur {
  private String nom;
  private String prenom;
  private int age;

  // Pour gérer les différents livres d'un objet Auteur
  Livre [] mesLivres;


  public Auteur(String pnom, String pprenom, int page) {
    nom = pnom;
    prenom = pprenom;
    age = page;

    mesLivres = null;
  }

  public int getAge() { return age; }

  String getNom() { return new String(nom); }
  String getPrenom() { return new String(prenom); }

  /**
   Il faut pouvoir associer des Livres à un objet Auteur.
   @param a : une référence sur l'Livre à associer
   On stocke la valeur de la référence dans le tableau mesLivres
   On ne crée pas de nouvel Livre. Car l'Livre a une vie indépendante du Livre
  */
  public void associerLivre(Livre a) {
    if(mesLivres == null) {
      mesLivres = new Livre[1];
      mesLivres[0] = a;
    }
    else {
      Livre [] mesLivresTmp = new Livre[mesLivres.length + 1];

      for(int i = 0; i < mesLivres.length; i++) {
        mesLivresTmp[i] = mesLivres[i];
      }
      mesLivresTmp[mesLivres.length] = a;
      mesLivres = mesLivresTmp;
    }
  }
```

```
/**
  Il faut pouvoir dissocier des livres à un objet Auteur
  @param l : une référence sur un livre
*/
public void dissocierLivre(Livre l) {
  if(mesLivres == null)
    return;

  int i;
  for(i = 0; i < mesLivres.length; i++)
    if(mesLivres[i] == l) break;

  /**
    Si l'auteur existe dans le tableau alors la boucle précédente
    s'est arrêtée avant la fin
  */
  if( i < mesLivres.length) {
    if(mesLivres.length == 1)
      mesLivres = null;
    else {
      Livre []  mesLivresTmp = new Livre[mesLivres.length - 1];

      int j;
      for(j = 0; i < i; i++)
        mesLivresTmp[j] = mesLivres[j];
      for(j = i+1; j < mesLivres.length; j++)
        mesLivresTmp[j-1] = mesLivres[j];
      mesLivres = mesLivresTmp;
    }
   }
  }
}
```

- Le fichier

```
/**
 Le fichier de la classe Livre
*/
public class Livre {
  private String titre;
  private int annee;

  /* Pour gérer les différents auteurs d'un objet livre */
  Auteur [] mesAuteurs;
```

```
/* Pour gérer les différents chapitres d'un objet livre */
Chapitre [] mesChapitres;

/**
Le constructeur de la classe Livre
@param ptitre : Le titre du livre
@param ptitre : L'année de parution
pas d'auteur et pas de chapitre
*/
public Livre(String ptitre, int pannee) {
titre = ptitre;
annee = pannee;

mesAuteurs = null;
mesChapitres = null;
}

public int getAnnee() { return annee; }

/**
L'acesseur du titre.
On renvoie une copie pour préserver l'encapsulation
*/
String getTitre() { return new String(titre); }

/**
 Il faut pouvoir associer des auteurs à un objet Livre.
 @param a : une référence sur l'auteur à associer
 On stocke la valeur de la référence dans le tableau mesAuteurs
 On ne crée pas de nouvel auteur. Car l'auteur a une vie indépendante du livre
*/
public void associerAuteur(Auteur a) {
  if(mesAuteurs == null) {
    mesAuteurs = new Auteur[1];
    mesAuteurs[0] = a;
  }
  else {
    Auteur [] mesAuteursTmp = new Auteur[mesAuteurs.length + 1];

    for(int i = 0; i < mesAuteurs.length; i++) {
      mesAuteursTmp[i] = mesAuteurs[i];
    }
    mesAuteursTmp[mesAuteurs.length] = a;
    mesAuteurs = mesAuteursTmp;
  }
}
```

```
/**
 Il faut pouvoir créer des objets Chapitre pour l'objet Livre
 @param resume : le résumé du chapitre
 @param texte : le texte du chapitre
 Un chapitre appartient à un livre, il est donc créé par un livre.
 Il n'est référencé que dans le livre.
*/
public void creerChapitre(String resume, String texte) {
  /**
  On crée le chapitre dans une méthode du livre
  */
  Chapitre ch = new Chapitre(resume,texte);

if(mesChapitres == null) {
    mesChapitres = new Chapitre[1];
    mesChapitres[0] = ch;
  }
  else {
    Chapitre [] mesChapitresTmp = new Chapitre[mesChapitres.length + 1];

    for(int i = 0; i < mesChapitres.length; i++) {
      mesChapitresTmp[i] = mesChapitres[i];
    }
    mesChapitresTmp[mesChapitres.length] = ch;
    mesChapitres = mesChapitresTmp;
  }
}

/**
 Il faut pouvoir supprimer des chapitres à un objet Livre

*/
public void supprimerChapitre(int no) {
  // pas de chapitre
  if(mesChapitres == null)
    return;

  // mauvais numéro de chapitre
  if(no <= 0 || no > mesChapitres.length)
    return;

  // s'il restait un chapitre alors il n'y a plus de chapitres
  if( mesChapitres.length == 1)
    mesChapitres = null;
```

```
    // Le cas général : on crée un nouveau tableau d'une case en moins
    Chapitre []  mesChapitresTmp = new Chapitre[mesChapitres.length - 1];

    int i,j;
    for(i = 0; i < (no-1); i++)
      mesChapitresTmp[i] = mesChapitres[i];
    for(i = no; i < mesChapitres.length; i++)
      mesChapitresTmp[i-1] = mesChapitres[i];
    mesChapitres = mesChapitresTmp;
  }


  /**
    Il faut pouvoir dissocier des auteurs à un objet Livre
    @param a : une référence sur un auteur
  */
  public void dissocierAuteur(Auteur a) {
    if(mesAuteurs == null)
      return;

    int i;
    for(i = 0; i < mesAuteurs.length; i++)
      if(mesAuteurs[i] == a) break;

    /**
      Si l'auteur existe dans le tableau alors la boucle précédente
      s'est arrêtée avant la fin
    */
    if( i < mesAuteurs.length) {
      if(mesAuteurs.length == 1)
        mesAuteurs = null;
      else {
        Auteur []  mesAuteursTmp = new Auteur[mesAuteurs.length - 1];

        int j;
        for(j = 0; i < i; i++)
          mesAuteursTmp[j] = mesAuteurs[j];
        for(j = i+1; j < mesAuteurs.length; j++)
          mesAuteursTmp[j-1] = mesAuteurs[j];
        mesAuteurs = mesAuteursTmp;
      }
    }
  }
}
```

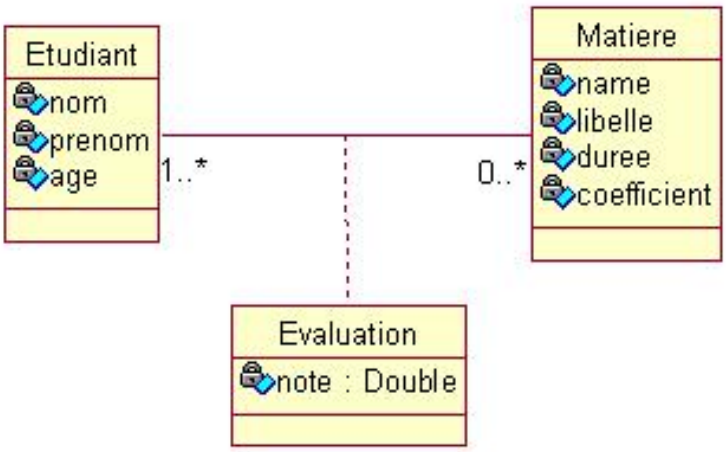
- Le fichier

```
/* Le fichier Chapitre.java */
public class Chapitre {
  private String resume;
  private String texte;

  public Chapitre(String presume, String  ptexte) {
   resume = presume;
   texte = ptexte;
  }

  String  getResume() { return new String(resume); }
  String  getTexte() { return new String(texte); }
}
```

2 - UML vers Java : Classe d'association : Des étudiants, des matières et des notes

Énoncé :

Dans le cadre d'une gestion de la scolarité, on s'intéresse au diagramme suivant :



Question 1)

Énoncé de la question

Faire le mapping des classes d'analyse en Java ?

Solution de la question

- Le fichier

```
/** Le fichier Note.java */


public class Note {
  private Etudiant et;
  private Matiere m;
  double val;

  public Note(Etudiant et , Matiere m, double val) {
    this.et = et;
    this.m = m;
    this.val = val;
  }

  public Etudiant  getEtudiant() {
    return et;
  }

  Matiere getMatiere() {
    return m;
  }

  double getNote() {
    return val;
  }
}
```

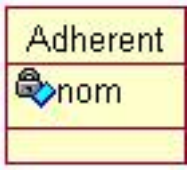
3 - UML vers Java : Diagramme d'états d'un objet adhérent

Énoncé :

Dans cet exercice, on étudie comment intégrer le diagramme d'état dans la conception d'une classe.

Dans le cadre d'une gestion de bibliothèque, on s'intéresse à la classe Adherent. L'analyse nous fournit les diagrammes suivants :

- Un premier diagramme de classe sommaire



Ebauche de la classe Adherent

- Le diagramme d'états/transitions de la classe Adherent

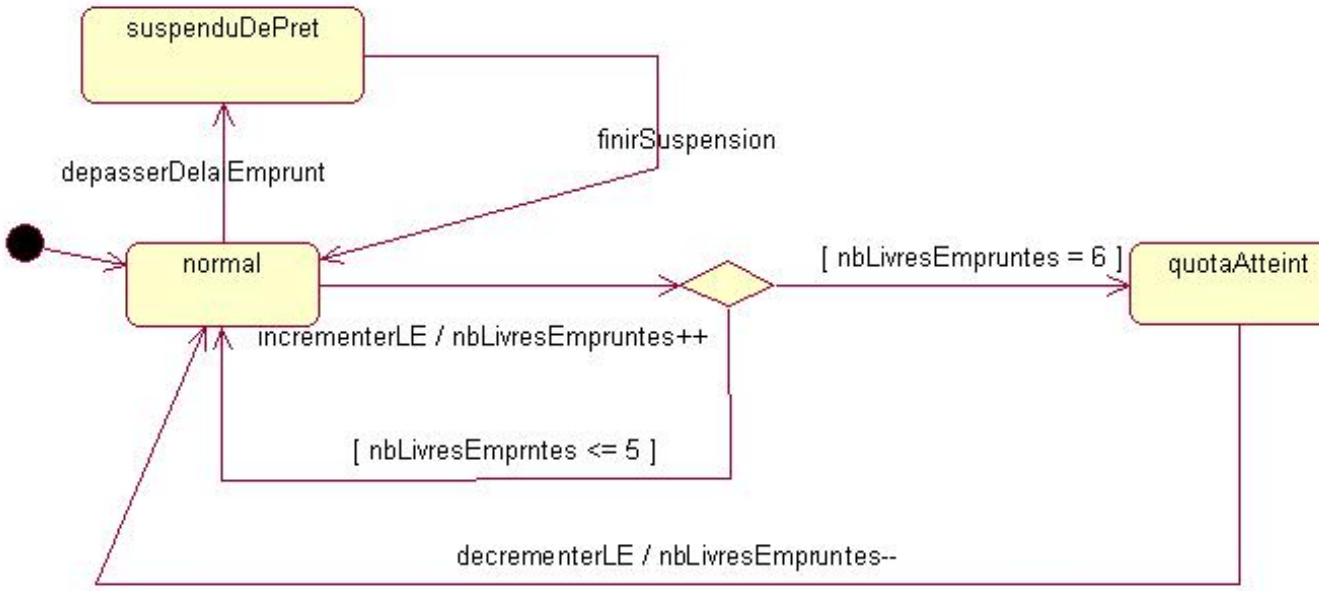


Diagramme d'états d'un objet Adherent

Question 1)

Énoncé de la question

Faire le mapping de la classe Adherent en tenant compte du diagramme d'état ? On définira dans la classe Adherent en plus des attributs pour le nom et le nombre de livres empruntés, un autre attribut pour gérer l'état de l'objet. Cet attribut sera de type int.

Dans cette même classe, on définira autant de constantes qu'il y a d'états possibles.

Solution de la question

- Le fichier

```
public class Adherent {
  public static final int SUSPENDU_DE_PRET = 0;
  public static final int NORMAL = 1;
  public static final int QUOTA_ATTEINT = 2;

  private String nom;
  private int etat;
  private int nbLE;

  public Adherent(String nom) {
    this.nom = nom;
    this.etat = NORMAL;
    this.nbLE = 0;
```

```
  }

  public void depasserDelaiEmprunt() throws AdherentException {
    if(etat != SUSPENDU_DE_PRET)
      etat= SUSPENDU_DE_PRET;
    else
      throw new AdherentException(AdherentException.DEJA_SUSPENDU);
  }

  public void finirSuspension()  throws AdherentException  {
    if(etat == SUSPENDU_DE_PRET)
      etat= NORMAL;
    else
      throw new AdherentException(AdherentException.PAS_SUSPENDU);
  }

  public void incrementerLE()  throws AdherentException {
    if(etat == NORMAL) {
      nbLE++;
      if(nbLE == 6)
        etat = QUOTA_ATTEINT;
    }
    else
      throw new AdherentException(AdherentException.INCREMENT_IMPOSSIBLE);
  }

  public void decrementerLE() throws AdherentException  {
    if((etat == NORMAL || etat == QUOTA_ATTEINT)int && nbLE > 0) {
      nbLE--;
      etat = NORMAL;
    }
    else
      throw new AdherentException(AdherentException.DECREMENT_IMPOSSIBLE);
  }
}

class AdherentException extends Exception { // classe
  public static final int DEJA_SUSPENDU = 0;
  public static final int PAS_SUSPENDU = 1;
  public static final int INCREMENT_IMPOSSIBLE = 2;
  public static final int DECREMENT_IMPOSSIBLE = 3;

  private int errorCode;

  public AdherentException(int errorCode) {
    this.errorCode = errorCode;
```

```
  }

  int getErrorCode() {
    return errorCode;
  }

  public String toString() {
    return new String(" " + errorCode);
  }
}
```

Question 2)

Énoncé de la question

Dans la solution précédente, on a un code truffé de structures conditionnelles portant sur l'état de l'objet. Si on veut faire évoluer cette classe, on sera probablement obligé de tout recoder.

Il existe une technique plus élégante et donc plus facilement modifiable. L'attribut d'état est une référence sur une interface. Cette interface est constituée de tous les prototypages de la classe Adherent. Le code de chaque méthode f1 de la classe Adherent consiste entre autre à demander à l'objet d'état d'exécuter sa propre méthode f1. Enfin, la classe Adherent aura une méthode qui permettra de changer l'attribut d'état.

Il faut coder autant de classes qu'il y a d'états possibles. Chacune de ces classes implémente l'interface et doit avoir un attribut qui référence l'objet Adherent. Grâce à cette référence, l'objet d'état lors de l'exécution d'une de ces méthodes pourra mettre à jour l'etat courant de l'objet Adherent.

Il est évident que dans un premier temps, cette solution paraît plus lourde mais le gain évident se trouve (comme souvent) dans l'évolution de la classe.

Faire le mapping de la classe Adherent en tenant compte du diagramme d'état avec cette nouvelle technique ?

Solution de la question

- Le fichier

```
interface IAdherent {
  public void depasserDelaiEmprunt() throws AdherentException;
  public void finirSuspension()  throws AdherentException;
  public void incrementerLE()  throws AdherentException;
  public void decrementerLE() throws AdherentException;
}

public class AdherentPE {
```

```
  private IAdherent etatCourant;
  private String nom;
  private int nbLE;

  public AdherentPE(String nom,IAdherent premierEtat) {
    this.nom = nom;
    this.etatCourant = premierEtat;
    this.nbLE = 0;
  }

  public void depasserDelaiEmprunt() throws AdherentException {
    etatCourant.depasserDelaiEmprunt();
  }

  public void finirSuspension()  throws AdherentException  {
    etatCourant.finirSuspension();
  }

  public void incrementerLE()  throws AdherentException {
    etatCourant.incrementerLE();
    nbLE++;
  }

  public void decrementerLE() throws AdherentException  {
    etatCourant.decrementerLE();
    nbLE--;
  }

  public void affEtatCourant(IAdherent etatCourant) {
    this.etatCourant = etatCourant;
  }

  public int recNbLE() {
    return nbLE;
  }
}

class AdherentNormal implements IAdherent { // classe
  private AdherentPE contexte;

  public AdherentNormal(AdherentPE contexte) {
    this.contexte = contexte;
  }

  public void depasserDelaiEmprunt() throws AdherentException {
    contexte.affEtatCourant(new AdherentSuspendu(contexte));
```

```
  }

  public void finirSuspension()  throws AdherentException  {
    throw new AdherentException(AdherentException.PAS_SUSPENDU);
  }

  public void incrementerLE()  throws AdherentException {
    if(contexte.recNbLE() == 5) {
        contexte.affEtatCourant(new AdherentQuotaAtteint(contexte));
    }
  }

  public void decrementerLE()  throws AdherentException {
    if(contexte.recNbLE() == 0) {
      throw new AdherentException(AdherentException.PAS_DE_LIVRE_EMPRUNTE);
    }
  }
}

class AdherentQuotaAtteint implements IAdherent { // classe
  private AdherentPE contexte;

  public AdherentQuotaAtteint(AdherentPE contexte) {
    this.contexte = contexte;
  }

  public void depasserDelaiEmprunt() throws AdherentException {
    throw new AdherentException(AdherentException.QUOTA_ATTEINT);
  }

  public void finirSuspension()  throws AdherentException  {
    throw new AdherentException(AdherentException.PAS_SUSPENDU);
  }

  public void incrementerLE()  throws AdherentException {
    throw new AdherentException(AdherentException.QUOTA_ATTEINT);
  }

  public void decrementerLE()  throws AdherentException {
    contexte.affEtatCourant(new AdherentNormal(contexte));
  }
}

class AdherentSuspendu implements IAdherent { // classe
  private AdherentPE contexte;
```

```
  public AdherentSuspendu(AdherentPE contexte) {
    this.contexte = contexte;
  }

  public void depasserDelaiEmprunt() throws AdherentException {
    throw new AdherentException(AdherentException.DEJA_SUSPENDU);
  }

  public void finirSuspension()  throws AdherentException  {
    contexte.affEtatCourant(new AdherentNormal(contexte));
  }

  public void incrementerLE()  throws AdherentException {
    throw new AdherentException(AdherentException.INCREMENT_IMPOSSIBLE);
  }

  public void decrementerLE()  throws AdherentException {
    throw new AdherentException(AdherentException.DECREMENT_IMPOSSIBLE);
  }
}

class AdherentException extends Exception { // classe
  public static final int DEJA_SUSPENDU = 0;
  public static final int PAS_SUSPENDU = 1;
  public static final int INCREMENT_IMPOSSIBLE = 2;
  public static final int DECREMENT_IMPOSSIBLE = 3;
  public static final int QUOTA_ATTEINT = 4;
  public static final int PAS_DE_LIVRE_EMPRUNTE = 5;

  private int errorCode;

  public AdherentException(int errorCode) {
    this.errorCode = errorCode;
  }

  int getErrorCode() {
    return errorCode;
  }

  public String toString() {
    return new String(" " + errorCode);
  }
}
```