

DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

SIGLE DU COURS : IFT 1010 (A2001)

NOM DU PROFESSEUR : N. Chabini, P. Poulin et M. Reid

TITRE DU COURS : Programmation I

EXAMEN FINAL

Date : Mercredi le 12 décembre 2001

Heure : 17 :30 à 20 :30

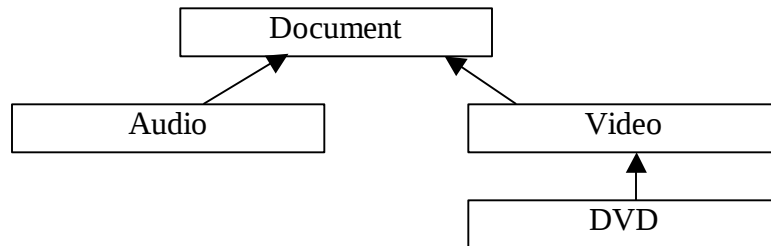
Salle : Z-110

DIRECTIVES PÉDAGOGIQUES :

- ?? Toute documentation est permise
- ?? Répondez dans le cahier d'examen
- ?? Inscrivez tout de suite votre nom et code permanent sur le cahier d'examen
- ?? Le nombre de points accordés à chaque exercice est indiqué entre parenthèse.
- ?? Le total des points est 100.

1. POO, héritage (30 points) :

Vous devez réaliser la hiérarchie de classes suivantes :



Vous disposez comme information sur ces classes de tous les attributs et méthodes (hérités ou non) accessibles à celles-ci :

??Audio :

La chaîne de caractères retournée par la méthode `toString()` devra contenir les informations suivantes :

Titre, langue, éditeur, auteur, ainsi que la liste des titres des chansons.

Le constructeur de la classe `Audio` est déclaré comme suit :

Audio	
Identificateur	Type
titre	String
auteur	String
nbChansons	int
langue	String
tempsTotal	double
tabChansons	Chanson[]
editeur	String
toString()	String

```
public Audio(String leTitre, String lAuteur, String langage,
             Chanson[] tab, String lEditeur);
```

Celui-ci initialise aussi les valeurs de `nbChansons`, et `tempsTotal` à partir des méthodes et informations disponibles par le tableau de `Chanson`. La description de cette classe est à la page suivante.

??Video :

La chaîne de caractères retournée par la méthode `toString()` devra contenir les informations suivantes :

Titre, langue, éditeur, directeur, ainsi que la liste des acteurs :

Le constructeur de la classe `Video` est déclaré comme suit :

Video	
Identificateur	Type
titre	String
directeur	String
acteurs	String[]
langue	String
tempsTotal	double
editeur	String
toString()	String

```
public Video(String leTitre, String leDirecteur, String langage,
             String lEditeur, double temps, String[] lesActeurs);
```

?DVD :

Voici un exemple de la chaîne de caractères retournée par la méthode `toString()` de la classe DVD:

La chaîne de caractères retournée par la méthode `toString()` devra contenir les informations suivantes :

Titre, langue, éditeur, directeur, la liste des acteurs, ainsi que la liste des langages disponible.

DVD	
identificateur	Type
titre	String
directeur	String
langue	String
acteurs	String[]
tempsTotal	double
editeur	String
languesSup	String[]
changeLangue()	void
toString()	String

Le constructeur de la classe DVD est déclaré comme suit :

```
public DVD(String leTitre, String leDirecteur, String[] langSup,  
           String lEditeur, double temps, String[] lesActeurs);
```

Note : `langue` sera initialisée avec le premier élément de `languesSup`

Soit la déclaration de la méthode suivante :

```
public void changeLangue(int indice);
```

Cette méthode change la valeur de `langue` en faveur du langage situé à la position `indice` dans le tableau `languesSup`.

La classe **Chanson** est définie (en partie) comme suit :

```
public class Chanson{  
    private String titre;  
    private double temps;  
    ...  
    public double getTemps() { return temps; }  
    public String toString() { return titre; }  
}
```

En utilisant l'héritage, en satisfaisant le principe d'encapsulation et en incorporant toutes les informations précédentes, vous devez :

- Écrire la classe de base **Document** qui regroupe tout ce qui est commun à ses classes enfants **(7 points)**
- Écrire la classe **Audio** qui est dérivée de **Document** **(9 points)**
- Écrire la classe **Video** qui est dérivée de **Document** **(8 points)**
- Écrire la classe **DVD** qui est dérivée de **Video** **(6 points)**

Solution :

a)

```
public class Document{
    protected String titre;
    protected String langue;
    protected double tempsTotal;
    protected String editeur;

    public Document(String leTitre, String laLangue, double leTemps, String lEditeur){
        titre = leTitre;
        langue = laLangue;
        tempsTotal=leTemps;
        editeur = lEditeur;
    }

    public String toString(){
        String resultat = "Le titre est " + titre + "\n" ;
        resultat += "L'editeur : " + editeur + "\n";
        resultat += "La langue est : " + langue + "\n";

        return resultat;
    }
}
```

B)

```
public class Audio extends Document{
    private String auteur;
    private int nbChansons;
    private Chanson[] tabChansons;

    public Audio(String leTitre, String lAuteur, String langage, Chanson[] tab, String lEditeur){
        super(leTitre, langage, 0.0, lEditeur);
        nbChansons = tab.length;
        tabChansons= tab;
        auteur = lAuteur;
        for(int i = 0; i<nbChansons; i++)
            tempsTotal += tabChansons[i].getTemps();
    }
}
```

```

    public String toString(){
        String resultat = super.toString();

        resultat += "l'auteur est " + auteur + "\n\nLa liste des chansons\n\n" ;
        for (int i = 0; i<nbChansons; i++)
            resultat += tabChansons[i].toString() + "\n";
        resultat += "\n";
        return resultat;
    }
}

```

C)

```

public class Video extends Document{
    protected String directeur;
    protected String[] acteurs;

    public Video (String leTitre, String leDirecteur, String langage, String lEditeur,
String[] lesActeurs, double temps){
        super(leTitre, langage, temps, lEditeur);
        directeur = leDirecteur;
        acteurs = lesActeurs;
    }

    public String toString(){
        String resultat = super.toString();

        resultat += "Le directeur est " + directeur + "\n\nLa distribution\n\n";
        for(int i = 0; i < acteurs.length; i++)
            resultat += acteurs[i] + "\n";
        resultat += "\n";
        return resultat;
    }
}

```

D)

```

public class DVD extends Video{
    private String[] languesSup;

    public DVD( String leTitre, String leDirecteur, String[] langSup, String lEditeur,
String[] lesActeurs, double temps){
        super(leTitre, leDirecteur, langSup[0], lEditeur, lesActeurs, temps);
        languesSup = langSup;
    }
}

```

```

    public void changeLangue(int indice){
    if (indice >= 0 && indice < languesSup.length)
        langue = languesSup[indice];
    }

    public String toString(){
    String resultat = super.toString();
    resultat += "\nLes langages disponibles sont :\n\n";
    for(int i = 0; i < languesSup.length; i++)
        resultat += languesSup[i] + "\n" ;

    resultat += "\n";

    return resultat;
    }
}

```

2. Tableau (20 points) :

Attention, le nombre de colonnes dans un tableau à deux dimensions peut varier d'une ligne à l'autre.

a) Écrivez une méthode : (4 points)

```
public boolean estCarre (int[][] tab, int dim);
```

qui vérifie si un `tab` est de dimensions `dim x dim`

b) Écrivez une méthode : (7 points)

```
public boolean estTriangulaire(int[][] tab, int dim);
```

qui vérifie si un tableau est triangulaire, c'est-à-dire si le tableau est carré et que pour tous les éléments de `tab`, `tab[i][j] == tab[j][i]`.

Important : Évitez de faire des comparaisons inutiles.

c) Écrivez la méthode suivante : (9 points)

```
public double[] moyColonnes (int[][] tab, int ln, int maxCol);
```

où `ln` est le nombre de lignes de `tab` et `maxCol` est le nombre maximum de colonnes de `tab`.

Chaque élément du tableau retourné par la méthode contient la moyenne des valeurs de sa colonne respective de `tab`.

réponses:

```
public boolean estCarre( int[][] tab, int dim)
{
    for (int i = 0; i < dim; i++)
        if( tab[i].length != dim )
            return false;
    return true;
}
```

```
public boolean estTriangulaire( int[][] tab, int dim)
{
    int i,j;
    if( estCarre( tab, dim))
    {
        for(i = 0; i < dim ; i++)
            for(j = i + 1; j < dim; j++)
                if(tab[i][j] != tab[j][i])
                    return false;

        return true;
    }
    return false;
}
```

```

public static double[] moyColonnes( int[][] tab, int ln, int maxCol)
{
    double[] tabMoy = new double[maxCol];
    double moy;
    int nb;

    for(int i = 0; i < maxCol; i++)
    {
        moy = 0.0;
        nb = 0;
        for( int j = 0; j < ln; j++)
            if( i < tab[j].length)
            {
                moy += tab[j][i];
                nb++;
            }

        if(nb > 0)
            tabMoy[i] = moy/nb;
    }
    return tabMoy;
}

```

3. Récursivité (12 points)

Soit la méthode récursive suivante :

```

public int queFait (int n) {
    if (n==0) return 0;
    else if (n==1) return 1;
    else return (n + queFait (n-2));
}

```

- a) Donner les valeurs retournées par les appels queFait(5) et queFait(6). (6 points)
réponse : 9 12
- b) Donner une version itérative de ce que fait la méthode queFait. (6 points)

Réponse :

```

public int queFait_iter (int n) {
    int i, sum;

    if (n %2 == 0) sum = 0;
    else sum = 1;
    for (i=n; i>1; i = i-2)
        sum = sum + i;
    return sum;
}

```

4. Récursivité (12 points)

La fonction d'Ackermann, dénotée Ack, est définie comme suit :

```

Ack(i, j) = j+1 si i=0
Ack(i, j) = Ack(i-1, 1) si i>0 et j=0
Ack(i, j) = Ack(i-1, Ack(i, j-1)), dans tous les autres cas de i et j.

```

Les variables i et j sont entiers. Donner une implantation récursive de la fonction d'Ackermann.

réponse :

```

public int Ack (int i, int j) {
    if (i == 0) return (j+1);
    else if (i>0 && j == 0) return (Ack(i-1, 1));
    else return (Ack(i-1, Ack(i, j-1)));
}

```

5. Exceptions (10 points)

- a) Quel est l'objectif de mettre des `try` imbriquées? Utiliser le morceau de code suivant pour votre explication. Toutes les variables sont déclarées. (3 points)

```
try {
    // Pour ouvrir le fichier file en mode lecture.
    FileReader      fr = new FileReader (file);
    BufferedReader  inFile = new BufferedReader (fr);

    line = inFile.readLine ();
    while (line != null) {
        tokenizer = new StringTokenizer (line);
        first_name = tokenizer.nextToken ();
        last_name  = tokenizer.nextToken ();
        try {
            mark1 = Integer.parseInt (tokenizer.nextToken ());
            mark2 = Integer.parseInt (tokenizer.nextToken ());
            items[count++] =
                new Item (first_name, last_name, mark1, mark2);
        }
        catch (NumberFormatException exception) {
            System.out.println ("Error in input. Line ignored:");
            System.out.println (line);
        }
        line = inFile.readLine ();
    }
    // Pour fermer le fichier file.
    inFile.close ();

    // Pour ouvrir le fichier file en mode écriture.
    // Le contenu du fichier file sera écrasé.
    FileWriter      fw = new FileWriter (file);
    BufferedWriter   bw = new BufferedWriter (fw);
    PrintWriter      outFile = new PrintWriter (bw);
    outFile.println ("Je serai le nouveau contenu du fichier " + file);
    outFile.close();
}
catch (FileNotFoundException exception) {
    System.out.println ("The file " + file + "was not found.");
}
catch (IOException exception) {
    System.out.println (exception);
}
```

réponse :

Même si une exception se produirait dans une zone du code, le programme continuera son exécution des instructions qui ne font pas partie de cette zone. Par exemple, le `TRY` le plus interne du code donne permet au programme de continuer à lire un fichier même si une exception se produirait pendant la lecture d'une de ses lignes. Si on avait omis ce `TRY`, le `TRY` le plus externe ne permet pas au programme de continuer à lire le fichier si une exception se produirait pendant la lecture d'une de ses lignes.

b) Soit la méthode suivante : (4 points)

```
public void EstCeQue_A_EstLeDoubleDe_B (int A, int B) {  
    if (A/B == 2 && A % B == 0)  
        System.out.println (A + " est le double de " + B);  
    else  
        System.out.println (A + " n'est le double de " + B);  
}
```

Quel est le problème principal avec la méthode `EstCeQue_A_EstLeDoubleDe_B`?

Réponse : Si `B` vaut 0 il y aura division par zéro

Soit la méthode `TriEnOrdreCroissant` suivante : (3 points)

```
public void TriEnOrdreCroissant (int[] Tab) {  
    int i, j, temp;  
    for (i=0; i<=Tab.length-1; i++) {  
        for (j=i+1; j<=Tab.length; j++) {  
            if (Tab[i] > Tab[j]) {  
                temp = Tab[i];  
                Tab[i] = Tab[j];  
                Tab[j] = temp;  
            }  
        }  
    }  
}
```

Quel est le problème principal dans la méthode `TriEnOrdreCroissant`?

Réponse : la condition du deuxième `for`, `j<=Tab.length` implique que l'on va tenter d'accéder un indice invalide du tableau.

6. Fichier (10 points)

Supposons que le fichier `f.txt` contient les informations suivantes :

```
4  
a  
b  
c  
d
```

où la première ligne contient le nombre total des lignes suivantes, à raison d'un caractère par ligne. On vous demande d'écrire une méthode en Java pour ajouter une ligne à la fin du fichier `f.txt`. Cette ligne doit contenir le caractère ``X``. Votre méthode doit être de la forme `public void MiseAJourFichier (String NomFichier)`. Après avoir appelé votre méthode `MiseAJourFichier`, le fichier `f.txt` doit contenir ce qui suit :

```
4  
a  
b  
c
```

d
X

Vous n'avez pas à gérer les exceptions.

Réponse :

```
public class Q6 {
    public static void main (String [] args) {
        char []      items;
        StringTokenizer tokenizer;
        String        line, file = "f.txt";
        int           max, count = 0;

        try {
            FileReader      fr = new FileReader (file);
            BufferedReader   inFile = new BufferedReader (fr);

            FileWriter      fw = new FileWriter (file);
            BufferedWriter   bw = new BufferedWriter (fw);
            PrintWriter      outFile = new PrintWriter (fw);

            line = inFile.readLine ();
            tokenizer = new StringTokenizer (line);
            max = Integer.parseInt (tokenizer.nextToken ());
            items = new char [max];

            System.out.println ("max =====:" + max);

            line = inFile.readLine ();
            while (line != null) {
                tokenizer = new StringTokenizer (line);
                try {
                    // items[count++] = Integer.parseInt (tokenizer.nextToken ());
                    items[count++] = tokenizer.nextToken ().charAt(0);
                }
                catch (NumberFormatException exception) {
                    System.out.println ("Error in input. Line ignored:");
                    System.out.println (line);
                }
                line = inFile.readLine ();
            }
            inFile.close ();

            outFile.println (max);
            for (int i=0; i<count; i++)
                outFile.println (items[i]);
            outFile.println ("X");
            outFile.close();
        }
        catch (FileNotFoundException exception) {
            System.out.println ("The file " + file + "was not found.");
        }
        catch (IOException exception) {
            System.out.println (exception);
        }
    }
}
```

7. Liste chaînée (6 points)

Nous avons vu comment créer une liste linéaire simplement chaînée en utilisant l'exemple du code en Java qui se trouve à la page suivante (ce code est une copie du code se trouvant dans les pages 500 à 503 du livre *JAVA Software Solutions* de *John Lewis & William Loftus*, 2ème édition). Dans ce code, la méthode `add` de la classe `BookList` permet d'insérer un livre à la fin de la liste. On vous demande d'écrire une autre méthode `add` qui permet d'insérer un élément dans la liste de livres à une certaine position x . L'élément se trouvant à la tête de cette liste est à la position 1. L'élément occupant la fin de la liste est à la position t où t est le nombre d'éléments de la liste. Pour faciliter votre tâche, on suppose que votre méthode `add` ne sera appelée que si la liste contient au moins deux éléments et que x vérifie $1 < x < t$. Votre méthode doit être de la forme : `public void add (Book newBook, int x)`

réponse :

```
public void add (Book newBook, int x){
    BookNode node = new BookNode (newBook);
    BookNode current;
    int position;

    current = list;
    position = 1;
    while (position < x-1) {
        current = current.next;
        position++;
    }
    node.next = current.next;
    current.next = node;
}
```

Annexe pour la question sur l'insertion d'un élément dans une liste

```
// Book.java      Author: Lewis and Loftus    Pages 500 to 503
public class Book {
    private String title;

    public Book (String newTitle){
        title = newTitle;
    }

    public String toString (){
        return title;
    }
}

public class BookList {
    private BookNode list;

    BookList(){
        list = null;
    }

    public void add (Book newBook){
        BookNode node = new BookNode (newBook);
        BookNode current;

        if (list == null) list = node;
        else {
            current = list;
            while (current.next != null)
                current = current.next;
            current.next = node;
        }
    }

    public String toString (){
        String result = "";
        BookNode current = list;

        while (current != null) {
            result += current.book.toString() + "\n";
            current = current.next;
        }
        return result;
    }

    private class BookNode {
        public Book book;
        public BookNode next;

        public BookNode (Book theBook) {
            book = theBook;
            next = null;
        }
    }
}

public class Library {
    public static void main (String[] args){
        BookList books = new BookList();

        books.add (new Book("The Hitchhiker's Guide to the Galaxy"));
        books.add (new Book("Jonathan Livingston Seagull"));
        books.add (new Book("A Tale of Two Cities"));
        books.add (new Book("Java Software Solutions"));

        System.out.println (books);
    }
}
```

Fin de l'examen. Bonne chance et bonne année!