IFT1176 – Session Automne, Intra Solution

Mohamed Lokbani & Michel Reid

IFT1176 – INTRA - SOLUTION

Nom: _____| Prénom(s): _____|

Signature: _____| Section: _____|

Date : 26 octobre 2002

Durée: 2 heures (de 10h30 à 12h30) Local: Z-110 (UdM) ; 604-x Longueuil.

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.

Soyez **BREFS** et **PRÉCIS** dans vos réponses.

1._____/20	Q1.1 Q1.2 Q1.3 Q1.4 Q1.5
2._____/20	Q2.1 Q2.2 Q2.3 Q2.4 Q2.5
3._____/20	Q3.1 Q3.2 Q3.3
4._____/10	Q4.1 Q4.2
5._____/30	Q5.1

Total:_____/100

Question 1 (20 points)

```
abstract class Document{
   protected String titre, auteur;
   public Document(String leTitre, String lAuteur){
      titre = leTitre;
      auteur = lAuteur;
   }
   abstract public int compare(Document autre);
   public void affiche(){
      System.out.println( "Le titre est " + titre ) ;
   }
}
class Video extends Document{
   public Video (String leTitre, String leDirecteur){
      super(leTitre, leDirecteur);
   }
   public int compare(Document autre){
      return titre.compareTo(autre.titre);
   }
   public void affiche(){
      super.affiche();
      System.out.println("Le directeur est " + auteur);
   }
}
class Audio extends Document{
   public Audio(String leTitre, String lAuteur){
      super(leTitre, lAuteur);
   }
   public int compare(Document autre){
      if(autre instanceof Audio)
         return titre.compareTo(autre.titre);
      else return -1;
   }
   public void affiche(){
      super.affiche();
      System.out.println("L'artiste est " + auteur);
   }
}
class DVD extends Video{
   private String[] langues;
   public DVD(String leTitre, String leDirecteur, String[] lang) {
      super(leTitre, leDirecteur);
      langues = lang;
   }
   public void afficheLangues(){
      System.out.println("\nListe des langues disponibles");
      for(int i = 0; i < langues.length; i++)
         System.out.println(langues[i]);
   }
   public void affiche(){
      super.affiche();
      afficheLangues();
   }
}
```

Après l'exécution des instructions suivantes :

```
Document cd1 = new Audio("Up", "Peter Gabriel");
String[] langues = {"anglais", "francais"};
Video film1 = new DVD("Star Wars", "Georges Lucas",langues);
```

-Q1.1- Que va faire afficher l'instruction suivante :

```
cd1.affiche();
```

Réponse :

Le titre est Up
L'artiste est Peter Gabriel

-Q1.2- Que va faire afficher l'instruction suivante :

```
film1.affiche();
```

Réponse :

Le titre est Star Wars
Le directeur est Georges Lucas

Liste des langues disponibles
anglais
francais

-Q1.3- Est-ce que l'instruction suivante va compiler? Si oui, que va t'elle faire afficher? Si non, pourquoi?

```
((DVD)film1).afficheLangues();
```

Réponse :

Oui elle va compiler. Elle va provoquer l'affichage suivant:

Liste des langues disponibles
anglais
francais

-Q1.4- Quel sera le résultat de l'instruction suivante? Expliquez brièvement pourquoi?

```
System.out.println(cd1 instanceof Document);
```

Réponse :

true

-Q1.5- Quel sera le résultat de l'instruction suivante? La méthode compare exécutée sera celle de quelle classe?

```
System.out.println(cd1.compare(film1));
```

Réponse :

Celle de Audio.

-1

Question 2 (20 points)

Soit le fragment de code suivant :

```
import java.util.*;
class Cours {
      private String nom;
      public Cours(String nom){
            this.nom = nom;
      }
      public String getNom(){
            return nom;
      }
}
public class Exo{
      public static void main(String [] args){

            /* On ajoute un seul bloc à la fois */

      }
}
```

Le corps de la méthode **main** est constitué des blocs d'instructions des questions **Q2.1** à **Q2.5**. Indiquez pour chaque bloc, s'il est correct ou pas (en cochant la bonne case), et dites pourquoi. Si le bloc est incorrect, corrigez l'erreur. Un bloc est dit incorrect, si au moins une des instructions lui appartenant est incorrecte. Par incorrect, nous entendons qu'une des instructions du bloc en question, génère une erreur lors de la compilation et/ou lors d'exécution.

-Q2.1-
```
//bloc -1-
Object o = new LinkedList();
```

Correct ☒ **Incorrect** ☐

Pourquoi? (Courte explication) **ET** correction (si nécessaire)

Object est la classe mère au sommet de toutes les classes. LinkedList dérive de Object, du coup, l'instanciation est correcte.

-Q2.2-
```
//bloc -2-
List l = new LinkedList();
```

Correct ☒ **Incorrect** ☐

Pourquoi? (Courte explication) **ET** correction (si nécessaire)

List est une interface. Cette interface est implantée dans les collections par ArrayList ou LinkedList. Du coup l'instanciation est correcte.

-Q2.3- `//bloc -3-`

```
List l2 = new List();
```

Correct ☐ **Incorrect** ▨

Pourquoi? (Courte explication) **ET** correction (si nécessaire)

List étant une classe abstraite. L'instanciation d'une classe abstraite n'étant pas permise d'où l'erreur à la compilation.

List l2 = new LinkedList();
Ou
List l2 = new ArrayList();

-Q2.4- `//bloc -4-`

```
HashMap h = new HashMap();
Cours c = new Cours("ift1176");
h.put(1176, c);
```

Correct ☐ **Incorrect** ▨

Pourquoi? (Courte explication) **ET** correction (si nécessaire)

Erreur dans l'instruction: h.put(1176, c);

La clé doit être un "Object" ce qui n'est pas le cas des int. Si on veut passer cette valeur comme clé:

h.put(Integer(1176),c);

-Q2.5- `//bloc -5-`

```
Object ol = new ArrayList();
ol.add(new Cours("ift1176"));
```

Correct ☐ **Incorrect** ▨

Pourquoi? (Courte explication) **ET** correction (si nécessaire)

On cherche dans la classe Object la méthode add(Cours), or elle n'existe pas d'où l'erreur. Il existe par contre la méthode add(Object) dans ArrayList. Du coup pour pouvoir l'utiliser, il faudra caster le type de l'objet.

```
((ArrayList) ol).add(new Cours("ift1176"));
```


Question 3 (20 points)

Soit le programme suivant, qui compile et s'exécute correctement:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class UnAutre implements ActionListener {
   public static String s = "";
   public static int courant = 1;
   public static void bouttonAction ()
   {
      // VOIR QUESTION -Q3.3- BLOC À COMPLÉTER
      windowPrint (s);
   }
   public UnAutre() {
      JButton button = new JButton ("Cliquer Moi!");
      button.addActionListener (this);
      label.setText ("----------------------------------------");
      JPanel panel = new JPanel ();
      panel.add (button);
      panel.add (label);
      JFrame frame = new JFrame ("UnAutre");
      frame.getContentPane().add (panel, BorderLayout.WEST);
      frame.addWindowListener(new WindowAdapter() {
         public void windowClosing (WindowEvent e) { System.exit(0); }
      });
      frame.pack ();
      frame.setVisible(true);
   }
   public static JLabel label = new JLabel ();
   public void actionPerformed (ActionEvent e) { bouttonAction(); }
   public static void windowPrint (String s) { label.setText (s); }
   public static void main (String[] args) {
      new UnAutre();
   }
}
```

-Q3.1- Tout en justifiant votre réponse, que va afficher (dessiner les figures correspondantes) l'exécution de la commande suivante: `java UnAutre`

On crée une fenêtre dans laquelle se trouve un Panel qui contient un bouton et un label. Le bouton contient l'étiquette "Cliquez Moi!" et le label une série de -. On a associé aussi à la fenêtre deux types d'événements. Un sur le bouton "Cliquez Moi!", un autre sur le l'onglet de fermeture de la fenêtre.

Si on appuie sur le bouton "Cliquez Moi!", le contenu de la variable s va être affiché à la place des -.

-Q3.2- Que va-t-on obtenir comme résultat (sur l'interface graphique) suite à un seul clic sur le bouton "**Cliquez Moi!**".

Comme la variable s a été initialisée à "" i.e .à une valeur nulle ; un clic sur le bouton "Cliquez Moi!" va faire que la série de – va être remplacée par rien (vide).

-Q3.3- Compléter la méthode "bouttonAction" afin de produire le scénario suivant. Les tableaux suivants représentent le nombre de clics sur le bouton "Cliquer Moi!" et le résultat obtenu (affichage obtenu).

nombre de cliques	résultat
1	1
2	12
3	123
4	1234
5	1245

nombre de cliques	résultat
6	1256
7	1267
8	1278
9	1289
10	1290

(La simplicité de votre code est un élément important de la notation)

```
public static void bouttonAction () {

   if (s.length() >= 4) {
      String debut = s.substring (0,2);
      String fin = s.substring (3,4);
      s = debut.concat (fin);
   }
   s = s.concat (Integer.toString (courant % 10));
   courant ++;
   windowPrint (s);
}
```


Question 4 (10 points)

Qu'allez-vous obtenir suite à la compilation puis l'exécution des programmes suivants (vous devez justifier votre réponse). S'il y a une erreur, indiquez la clairement. Si le programme compile et s'exécute, indiquez le résultat obtenu. Vous devez simuler une exécution donnée si nécessaire.

-Q4.1-

```
public class ThrowsDemo {
      static void throwMethode() {
            System.out.println("Dans throwMethode.");
            throw new IllegalAccessException("demo");
      }
      public static void main(String args[]) {
            try {
                  throwMethode();
            }
            catch (IllegalAccessException e) {
                  System.out.println("Capturé " + e);
            }
      }
}
```

Correct ☐ **Incorrect** ▨

Pourquoi? (Courte explication) **ET** description des erreurs s'il y en a, sinon affichage produit.

```
static void throwMethode() {
```
L'exception doit être capturée ou déclarer qu'elle le sera (throws).
static void throwMethode() ***throws IllegalAccessException****{*

```
catch (IllegalAccessException e) {
```
On capture quelque chose qui a été levée or ce n'est pas le cas! À cause de la précédente erreur.

-Q4.2-

```
import java.io.*;
class Base{
      public static void amethod()throws FileNotFoundException{}
}
public class ExceptDemo extends Base{
      public static void main(String argv[]){
            ExceptDemo e = new ExceptDemo();
      }
      public boolean amethod(int i){
            try{
                  DataInputStream din =
                              new DataInputStream(System.in);
                  System.out.println("Pause");
                  din.readChar();
                  System.out.println("Continue");
                  this.amethod();
                  return true;
            }catch(IOException ioe) {}
            finally{
                  System.out.println("traitement finally");
            }
            return false;
      }
      ExceptDemo(){
            amethod(99);
      }
}
```

Correct **Incorrect**

Pourquoi? (Courte explication) **ET** description des erreurs s'il y en a, sinon affichage produit.

Rien d'anormale toutes les exceptions sont présentes et bien capturées en cas d'erreur.

Affichage en sortie:

Pause
<return>
Continue
traitement finally

Question 5 (30 points)

Une collection Map est une structure dont chaque élément est une paire contenant une clé et une valeur associée à cette clé. Une clé donnée n'est présente qu'une seule fois dans une telle structure. Il existe des applications où il est cependant souhaitable d'associer plusieurs valeurs à une même clé. Par exemple la chanteuse Céline Dion (la clé), et ses différents albums (les valeurs). Ce type d'association est appelé dans le jargon technique Multimap. Or Java n'a pas implanté ce genre de collection. Comment faire alors? C'est le but de cet exercice!

Soit l'échantillon d'un fichier (ascii) contenant des données arrangées comme suit:

```
"Louis Armstrong" "Hot Five"
"Louis Armstrong" "Hot Seven"
"Charlie Parker" "Bird And Diz"
"Charlie Parker" "South Of The Border"
"Charles Mingus" "Pithecanthropus Erectus"
"Charles Mingus" "The Clown"
"Oscar Peterson" "The Gershwin Songbooks"
"Dave Brubeck" "Time Out"
"Miles Davis" "Sketches of Spain"
"Miles Davis" "Birth Of The Cool"
"Miles Davis" "Quintet 1965 - 1968"
```

Chaque ligne contient une paire représentée par le nom d'un seul compositeur et le titre d'un de ses albums. Ces éléments sont entre "" et seront séparés par un espace (blanc). Le fichier peut contenir une infinité de paires.

Soit le programme suivant:

```
import java.io.* ;
import java.util.* ;
public class Codage {
   private SortedMap m; // Une Map triée.
   private String nomfich; // Le nom du fichier de données.

   public Codage(String argument){
         m = new TreeMap();
         nomfich = argument;
         process();
         affiche();
   }
   private void process() {
         //À compléter
   }
   private void affiche(){
         //À compléter
   }
   public static void main (String args[]) {
         new Codage(args[0]);
   }
}
```

- La méthode **`process`**: Cette méthode sert à lire les données du fichier d'entrée, passé comme argument sur la ligne de commande. Les données sont organisées dans une collection du type SortedMap.

- La méthode **`affiche`**: Cette méthode permet d'afficher en sortie la collection en respectant l'affichage suivant (obtenu suite à l'exécution d'un tel programme avec comme argument le fichier jazz.txt) :

java Codage jazz.txt

```
Les oeuvres des compositeurs contenues dans le fichier [jazz.txt] :
[Louis Armstrong] : [Hot Five, Hot Seven]
[Dave Brubeck] : [Time Out]
[Miles Davis] : [Sketches of Spain, Birth Of The Cool, Quintet 1965 - 1968]
[Charles Mingus] : [Pithecanthropus Erectus, The Clown]
[Charlie Parker] : [Bird And Diz, South Of The Border]
[Oscar Peterson] : [The Gershwin Songbooks]
*** fin liste fichier ***
```

On affiche donc les paires: (compositeur, œuvres). Les compositeurs sont affichés en ordre alphabétique selon leur nom de famille, puis de leurs prénoms. Le nom de famille est le dernier paramètre dans la clé: Louis **Armstrong**, Dave **Brubeck**, etc. Il y aura une forme d'écriture unique du nom et prénoms du compositeur dans le fichier de données. Vous n'aurez pas donc le même compositeur écrit de deux (ou plusieurs) manières dans le fichier de données.

(La simplicité de votre code est un élément important de la notation)

-Q5.1- On vous demande d'écrire les deux méthodes **`process`** et **`affiche`**.
Vous pouvez ajouter des méthodes dans la classe Codage, ou d'autres classes si vous jugez cela nécessaire.

Juste pour des raisons de clarté, nous avons décidé d'ajouter les deux méthodes suivantes:

On va inverser l'ordre (prénom1 prénom2 ... prénomN nom_de_famille) afin d'obtenir: (nom_de_famille prénom1 prénom2 ... prénomN). La clé ainsi formée sera donc "triée" d'abord par nom de famille au lieu du 1er prénom.

```
private String inverseOrdre(StringTokenizer chainetok, int nbretok){
    String sx="";
    if (chainetok.hasMoreTokens())
    sx=chainetok.nextToken();
    for (int i=1 ; i<nbretok ; i++) {
        sx =  chainetok.nextToken() + " " + sx;
    }
    return sx;
}
```

L'idée est que la clé inversée sert comme point de repère, donc la paire (prénoms/nom) est inversée, les valeurs sont représentées dans une liste. Le premier élément de la liste c'est la clé dans l'ordre original afin d'obtenir l'affichage demandé.

```
private void fillMultimap(String cle, Map mm, String nom,
                          StringTokenizer chainetok, int nbretok){
    // Est-ce que la clé existe, si ce n'est pas le cas, nous allons
    // l'insérer dans la Map. Nous allons créer la liste l, dont le premier
    // élément n'est autre que la clé (dans le bon ordre).
    List l = (List) mm.get(cle);
    if (l==null){
        mm.put(cle, l=new ArrayList());
        l.add(nom);
    }
    // Si la clé existe déjà ... on a qu'à insérer les différentes valeurs.
    for (int i=1 ; i<nbretok ; i++) {
        if (chainetok.nextToken() != " "){
            l.add(chainetok.nextToken());
            i++;
        }
    }
}
```



```
private void affiche(){

    System.out.println (
        "Les oeuvres des compositeurs contenues dans le fichier ["
         + nomfich + "] :"
    );
    // On veut afficher que les valeurs. La clé telle que représentée,
    // elle nous sert à rien!
    for (Iterator i = m.values().iterator();i.hasNext();){
        // Chaque valeur n'est autre qu'une liste.
        List l = (List) i.next();
        // On se crée deux vues sur cette liste.
        // La première va contenir la clé dans le bon ordre.
        List sl1 = l.subList(0,1);
        // La seconde les valeurs associées à cette clé.
        List sl2 = l.subList(1,l.size());
        // On affiche donc la clé puis les valeurs associées.
        System.out.println(sl1 + " : " + sl2);
    }
    System.out.println ("*** fin liste fichier ***");
}


private void process() {

    try {
        // Ouverture du fichier et définition du buffer en entrée.
        BufferedReader entree =
                  new BufferedReader (new FileReader (nomfich));
        while (true) {
            // Lecture ligne par ligne.
            String ligneLue = entree.readLine();
            // Une ligne null (EOF), lecture terminée.
            if (ligneLue == null) break ;
            // 1er découpage: les guillemets
            StringTokenizer coupeligne = new StringTokenizer (ligneLue, "\"");
            // On récuppère par la même occasion le nombre de
            // tokens dans coupeligne.
            int ncligne = coupeligne.countTokens();
            // Le premier token est le nom de famille.
            String nomfamille = coupeligne.nextToken();
            // Pour se débarrasser de l'espace blanc, on découpe
            // une seconde fois, cette fois-çi sur l'espace séparateur
            StringTokenizer coupealbums =
                        new StringTokenizer (nomfamille, " ");
            // On récupère par la même occasion le nombre de tokens
            // dans coupealbums.
            int ncalbums = coupealbums.countTokens();
            // Inversion de l'ordre prénoms/nom.
            String sx = inverseOrdre(coupealbums,ncalbums);
            // Remplissage de la Map sous le format d'une Mutlimap.
            fillMultimap(sx, m, nomfamille, coupeligne, ncligne);
        }
        entree.close ();
    }
    catch (IOException e){
        System.err.println("Erreur lors du traitement du fichier: "
                                                              + nomfich);
        System.exit(1);
    }
}
```