DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

SIGLE DU COURS: IFT 1010 (A2002)

NOMS DES PROFESSEURS: Pierre Poulin et Balázs Kégl

TITRE DU COURS: Programmation I

EXAMEN FINAL

Date : 10 décembre 2002
Heure : 8:30 - 11:30
Salle : B2285

DIRECTIVES PÉDAGOGIQUES:

- Toute documentation est permise.
- **Inscrivez tout de suite votre nom et code permanent dans l'encadré.**
- Répondez **sur le questionnaire** en utilisant l'espace libre qui suit chaque question.
- Si vous désirez changer votre réponse, des feuilles blanches sont disponibles et seront brochées à votre copie.

NOTATION:

Question 1:	/10
Question 2:	/20
Question 3:	/15
Question 4:	/10
Question 5:	/5
Question 6:	/10
Question 7:	/10
Question 8:	/20
Total:	/100

Inscrivez votre nom et votre code permanent ici

CONSEIL:
Les programmes demandés sont relativement courts et ont presque tous la même pondération. Ne consacrez donc pas trop de temps à un seul numéro, avant d'avoir bien répondu aux autres.

Question 1 (10 points):

Ecrivez une méthode qui lit trois entiers à partir d'une chaîne de caractères (`String`), où chaque entier est séparé par un seul espace (caractère blanc), et qui *retourne* (spécifiez le type de retour) ces trois entiers à la méthode l'appelant.

```
public static int[] lire3Entiers (String s)
{
  int[] entier = new int[3];
  int pos = 0;
  int indice = 0;
  while (pos < s.length())
  {
    char c = s.charAt(pos);
    if (Character.isDigit(c))
      entier[indice] = entier[indice] * 10 +
               Integer.parseInt(s.substring(pos,pos+1));
    else
      indice++;

    pos++;
  }

  return entier;
}
```

Question 2 (20 points):

Soit un fichier en format texte où chaque ligne est formée de trois éléments séparés par un seul espace, dont un numéro d'identification d'employé, suivi d'une heure dans le format: heure minute. Chaque entrée correspond au moment d'arrivée ou de sortie d'un employé.
Ecrivez un programme (classe avec une méthode `main` et potentiellement d'autres méthodes) qui ouvre ce fichier (passé en argument à votre méthode `main`), lit les données (en utilisant votre méthode écrite à la question 1, ou en supposant qu'une telle méthode existe), et calcule et imprime en sortie (avec `System.out.println`) le temps passé au travail par chaque employé qui a travaillé durant cette journée.
Notez que le fichier est en ordre chronologique, et que tous les employés qui travaillent arrivent après 6:00 et repartent avant 20:00. Cependant un employé peut partir et revenir plusieurs fois dans la même journée. Aucun employé n'a de numéro d'identification supérieur à 100.

Exemple de fichier:

```
22 6 25
61 8 33
61 11 55
61 13 14
38 15 11
22 18 5
38 19 10
61 19 17
```

```
import java.io.*;

public class Temps
{
  public static void main(String[] args)
    throws IOException
  {

    int[][] time = new int[100][2];

    BufferedReader in = new BufferedReader (new FileReader (args[0]));

    boolean done = false;

    String s = null;
    int employee = 0;
    int hour = 0;
    int minute = 0;
    int[] values = new int[3];
```

Question 2 (suite):

```
    while (!done)
    {
      s = in.readLine();
      if (s != null)
      {
        values = lire3Entiers(s);
        employee = values[0];
        hour = values[1];
        minute = values[2];
        if (time[employee][1] == 0)
          time[employee][1] = hour*60 + minute;
        else
        {
          time[employee][0] += (hour*60+minute) - time[employee][1];
          time[employee][1] = 0;
        }
      }
      else
      {
        if (in != null) in.close();
        done = true;
      }
    }

    for (int i = 0; i < 100; i++)
    {
      if (time[i][0] != 0 || time[i][1] != 0)
        System.out.println("Employee " + i + " worked " +
                           (time[i][0]/60) + " hours and " +
                           (time[i][0] - ((time[i][0]/60)*60)) + " minutes");
    }
  }
}
```

Question 3 (15 points):

Ecrivez le code de la méthode suivante. Vous devez ***tout*** vérifier au sujet des tableaux `v1` et `v2`. Par exemple, un tableau 2D en Java n'a pas nécessairement un nombre constant de lignes. Il faut que votre méthode soit efficace.

```
final static double MAX_DIFF = 0.00001;

/**
   Compare deux à deux les éléments des tableaux, pour les mêmes indices.
   Lance un IllegalArgumentException si les tailles des tableaux sont
   différentes, sauf si une paire d'éléments ayant une différence > MAX_DIFF
   a été découverte avant que ne soit détectée une taille différente.
   @return false si une paire d'éléments a une différence > MAX_DIFF
   @return true  si toutes les paires d'éléments ont une différence <= MAX_DIFF
 **/
boolean compare(double[][] v1, double[][] v2)
   throws IllegalArgumentException
{
    if (v1 == null || v2 == null)
      throw new IllegalArgumentException();

    if (v1.length != v2.length)
      throw new IllegalArgumentException();

    for (int i = 0; i < v1.length; i++)
    {
      if (v1[i].length != v2[i].length)
        throw new IllegalArgumentException();

      for (int j = 0; j < v1[i].length; j++)
      {
        if (Math.abs (v1[i][j] - v2[i][j]) > MAX_DIFF)
          return false;
      }
    }

    return true;
}
```

Question 4 (10 points):

Expliquez en vos propres termes ce qui ne fonctionne pas dans les morceaux de code suivants. Supposez que les instructions remplacées par `// ...` sont correctes.

```
try
{
  // exécute des instructions
}
catch (ArrayIndexOutOfBoundsException e)
{
  int nombreErreurs = 1;
}
finally
{
  if (nombreErreurs > 0)
    // corrige la situation
}
```

`nombreErreurs` est déclaré dans le bloc `catch`, et donc il n'existera pas hors de ce bloc. Ce bout de code ne compilera même pas.

```
int ouch (int n)
{
  if (n == 1)
    return 1;
  else
    return (n * ouch(n-2));
}
```

Lorsque `n` est pair, cette méthode bouclera à l'infini.

Question 5 (5 points):

Ecrivez une seule méthode récursive `puissance` qui évalue a^b, donc qui élève un entier positif `a` à la puissance `b`, un autre entier positif. Exemple: $3^4 = 3 \times 3 \times 3 \times 3$.

```
public static int puissance (int a, int b)
{
  if (b == 1)
    return a;

  return (a * puissance(a, b-1));
}
```

Question 6 (10 points):

Ecrivez une seule méthode récursive qui prend un entier de type (`long`) en argument et retourne une chaîne de caractères (`String`) représentant cet entier, mais dont les milliers sont séparés par des virgules. Exemple: l'entier 1234567890 est converti en la chaîne de caractères 1,234,567,890

```
public static String formatEntier (long n)
{
  if (n < 1000)
    return "" + n;

  long reste = n % 1000;
  String zeros = null;

  if (reste > 99)
    zeros = "" + reste;
  else if (reste > 9)
    zeros = "0" + reste;
  else
    zeros = "00" + reste;

  return (formatEntier(n / 1000) + "," + zeros);
}
```

Question 7 (10 points):

Soient les deux méthodes suivantes, dites ce que chacune imprimera si on les appelle dans la méthode `main` avec les deux commandes ci-bas.

```
public class Recursive
{
  public static int reci(int i)
  {
    if (i < 0) return 0;
    if (i < 1) return 1;

    System.out.println(i);
    return (reci(i-1) + reci(i-2));
  }

  public static int reco(Integer i)
  {
    if (i.intValue() < 0) return 0;
    if (i.intValue() < 1) return 1;

    System.out.println(i);
    return (reco(new Integer(i.intValue()-1)) + reco(new Integer(i.intValue()-2)));
  }

  public static void main(String[] args)
  {
    ...
  }
}
```

reci (4);

4
3
2
1
1
2
1

reco (new Integer(4));

4
3
2
1
1
2
1

Question 8 (20 points):

Dans une file, les éléments sont ajoutés à la fin et ils sont enlevés au début. Un tampon circulaire permet de simuler une file avec un tableau de dimension fixe, avec des indices pour les premier et dernier éléments qui s'incrémentent lorsque les éléments sont ajoutés et enlevés, respectivement. Ecrivez le code des méthodes suivantes de la classe `TamponCirculaire` dont les éléments sont des références à des objets de type `Object`.

```
public class TamponCirculaire
{
  public TamponCirculaire (int taille)
  {
    // votre code pour allouer et initialiser le tampon

    tampon = new Object[taille];
    tete = 0;
    queue = 0;
    length = 0;
  }

  /**
     Ajoute un élément à la fin du tampon
     @return true si l'élément a pu être inséré dans le tampon
   **/
  public boolean ajoute (Object obj)
  {
    // votre code pour ajouter un élément

    if (length == tampon.length)
      return false;

    tampon[queue] = obj;
    queue++;
    length++;

    if (queue == tampon.length)
      queue = 0;

    return true;
  }
```

```
  /**
     Enlève un élément au début du tampon
     @return Object une référence au premier élément Object dans le tampon
             null si le tampon est vide
   **/
  public Object enleve ()
  {
    // votre code pour enlever un élément

    Object obj = tampon[tete];
    tete++;
    length--;

    if (tete == tampon.length)
      tete = 0;

    return obj;
  }

  /**
     @return String contenu de chaque Object du tampon séparé par un blanc
     @prec Assume que la méthode toString a été définie aussi dans Object
   **/
  public String toString()
  {
    String s = "";
    int j = tete;

    for (int i = 0; i < length; i++)
    {
      s += " " + (Double)tampon[j];
      j++;
    }
    return s;
  }

  private int tete;   // indice du premier élément du tampon
  private int queue;  // indice du dernier élément du tampon
  // autres variables de classe?

  private Object[] tampon;
  private int length;    // nombre d'éléments dans le tampon



}
```