DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

SIGLE DU COURS: IFT1010 (Hiver 2002)

ENSEIGNANTS: Michel Reid & Philippe Langlais

TITRE DU COURS: Programmation I

SOLUTIONNAIRE DE L'EXAMEN FINAL

Date : 18 avril 2002
Heure : 17:30 - 20:30

DIRECTIVES PÉDAGOGIQUES:

- Vous pouvez utiliser vos documents (lire: il faudrait mieux que vous n'ayez pas à la faire trop souvent).
- **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 manquez de place, écrivez au dos des feuilles, en l'indiquant clairement.

NOTATION:

Exercice 1:	/20	Exercice 4:	/30
Exercice 2:	/15	Exercice 5:	/20
Exercice 3:	/15		
Total:	/100		

Inscrivez votre nom et votre code permanent ici

Exercice 1 — Tableaux — (20 points):

Dans cet exercice, nous considérons uniquement des tableaux dont les éléments de base sont tous des entiers. On vous demande d'écrire une méthode `existe` qui permet de chercher si un tableau (t_1) à une dimension existe dans une ligne d'un tableau (t_2) à deux dimensions. Si t_1 existe dans t_2, alors votre méthode doit retourner un entier qui est calculé selon la formule: $l * 1000 + c$ où l est la ligne de t_2 qui contient t_1 et c la colonne de t_2 où débute t_1. Si t_1 n'existe pas dans t_2, votre méthode doit retourner -1.

Voici tout de suite quatre exemples d'appels de la méthode que vous devez écrire, accompagnés de leur valeur de retour respective.

```
int [][] table = {{1,2,3,4,5,6},{8,2,3},{13,14,15},{22,23,25}};
int [] t1 = {23,25};
int [] t2 = {14,15,16};
int [] t3 = {3,4,5};
int [] t4 = {8};

// cet appel affichera 3001 car {23,25} existe en ligne 3, colonne 1
// ---> 3 * 1000 + 1
System.out.println("t1 existe dans table ? " + existe(t1,table));

// cet appel affichera -1 car {14,15,16} n'existe pas dans table
System.out.println("t2 existe dans table ? " + existe(t2,table));

// cet appel affichera 2 car {3,4,5} existe en ligne 0, colonne 2
// ---> 0 * 1000 + 2
System.out.println("t3 existe dans table ? " + existe(t3,table));

// cet appel affichera 1000 car {8} existe en ligne 1, colonne 0
// ---> 1 * 1000 + 0
System.out.println("t4 existe dans table ? " + existe(t4,table));
```

Note 1: Dans le cas où un tableau existe plusieurs fois dans la table de dimension 2, la première occurrence trouvée est retournée.

Note 2: La table à deux dimensions passée en second argument n'est pas nécessairement une matrice. Dans l'exemple reporté, ce n'est d'ailleurs pas le cas (la première ligne possède 6 colonnes, la seconde ligne en possède 3, etc.).

Note 3: Bien que cela ne soit pas nécessaire, vous pouvez si vous le souhaitez décomposer votre code en plusieurs méthodes.

Note 4: Votre méthode ne doit pas faire d'hypothèse sur les tableaux passés en argument (ce ne sont pas nécessairement ceux de l'exemple).

Le code suivant est une réponse tout à fait acceptable à la question. L'idée à développer est: “passe chaque ligne de t_2 en revue; pour chaque ligne, recherche la table t_1 au commencement de chaque colonne. Dès que t_1 est trouvé, faire le return approprié. Si t_1 n'a pas été trouvé, retourner finalement -1”.

```
public static int existe(int [] t1, int [][] t2) {
  int i,j;
  for (int l=0; l<t2.length; l++)
    for (int c=0; c<t2[l].length; c++)
       if ((c + t1.length) <= t2[l].length) { // cette ligne n'est meme pas obligatoire
         for (j=c,i=0;
             (j<t2[l].length) && (i<t1.length) && (t1[i] == t2[l][j]);
         i++,j++);
         if (i == t1.length) return l * 1000 + c;
       }
  return -1;
}
```

Exercice 2 – Récursivité – (15 points):

Considérez la méthode suivante:

```
int methode(String s, int d, int f) {
  if (d <= f) {
    switch (s.charAt(d)) {
    case '(':
      if (s.charAt(f) == ')') {
        int recurs = methode(s,d+1,f-1);
        return (recurs == -1)? -1: 1 + recurs;
      }
      return methode(s,d,f-1);
    case ')':
      return -1;
    default:
      return methode(s,d+1,f);
    } // switch
  } else
      return 0;
}
```

Cette méthode est-elle récursive (répondez par **oui** ou par **non**)? .OUI (regardez le titre de l'exercice !)

Indiquez la valeur de n après l'exécution de chacune des instructions suivantes (on suppose n déclaré entier):

`n = methode("abc",0,2);` 0

`n = methode("()",0,1);` 1

`n = methode("(())",0,3);` 2

`n = methode("a(a(aaa)a)aa",0,3);` 0

`n = methode("(()())",0,5);` ....-1

`n = methode("(((aaa)))",0,5);` 0

Énoncez en une phrase **simple**[1] ce que retourne cette méthode:

Cette méthode retourne le nombre de parenthèses imbriquées dans une expression ne contenant qu'un foyer d'imbrication. Elle retourne `-1` dans tous les autres cas.

Exercice 3 – Vecteurs – (15 points):

Indiquez l'affichage produit par le programme suivant:

```
class MyInt{
    private int v;
    public void setValue(int v) {this.v = v;}
    public MyInt(int v) {this.v=v;}
    public String toString() {return v+"";}
}

class Exercice3 {
    private static Vector v = new Vector();
    private static MyInt entier = new MyInt(10);

    public static void question1() {
        v.add(new MyInt(1));
        v.add(new MyInt(2));
        v.add(new MyInt(3));
        System.out.println("vecteur 1 " + v);
        v.add(entier);
        v.remove(0);
        v.add(0,entier);
        System.out.println("vecteur 2 " + v);
        ((MyInt) v.get(0)).setValue(12);
        System.out.println("vecteur 3 " + v);
        System.out.println("est-ce que 3 est present dans v? " +
                        ((v.contains(new MyInt(3)))? "oui":"non"));
    }
}

public class FinalH02 {
    public static void main(String [] args) {
       Exercice3.question1();
    }
}
```

vecteur 1 [1, 2, 3] ....
vecteur 2 [10, 2, 3, 10] ....
vecteur 3 [12, 2, 3, 12] ....
est-ce que 3 est present dans v? non ....

[1]Seule la réponse exacte sera considérée. Inutile d'essayer quelque chose du genre: cette méthode retourne un entier qui si le caractère d'indice *d* est une parenthèse ouvrante, alors lorsque le caractère de fin est une parenthèse fermante, alors on incrémente de 1, puis sinon, etc...

Si nous remplaçons le code de la classe `MyInt` par le code qui suit, quel est l'affichage (complet) produit par le programme `FinalH02`?

```
class MyInt{
    private int v;
    public void setValue(int v) {this.v = v;}
    public MyInt(int v) {
      System.out.println("Creation d'un MyInt("+ v + ")");
      this.v=v;
    }
    public String toString() {return v+"";}
    public boolean equals(Object o) {
      if (o instanceof MyInt)
        return ((MyInt) o).v == this.v;
      return false;
    }
}
```

Creation d'un MyInt(10) ....
Creation d'un MyInt(1) ....
Creation d'un MyInt(2) ....
Creation d'un MyInt(3) ....
vecteur 1 [1, 2, 3] ....
vecteur 2 [10, 2, 3, 10] ....
vecteur 3 [12, 2, 3, 12] ....
Creation d'un MyInt(3) ....
est-ce que 3 est present dans v? oui ....

Exercice 4 – POO et Héritage – (30 points):

Considérez la classe `Logement` suivante où `cout` indique le coût mensuel du logement associé:

```
public abstract class  Logement{
    protected double cout;
    protected String adresse;

    public Logement(double cout, String adresse) {
        this.cout = cout;
        this.adresse = adresse;
    }
    public double calculCoutMensuel(){return cout;}
}
```

Lisez attentivement la description de la hiérarchie de classes qui suit puis répondez aux questions qui suivent.

La classe `Location` dérive de la classe `Logement`. Une instance de cette classe représente un logement habité par un locataire.

- Le constructeur reçoit deux paramètres, le montant du loyer mensuel et l'adresse.
- la méthode `toString` retourne une chaîne contenant l'adresse et le montant mensuel du loyer.

La classe `Propriete` dérive de la classe `Logement`. Une instance de cette classe représente une propriété habitée uniquement par le propriétaire (propriété unifamiliale).

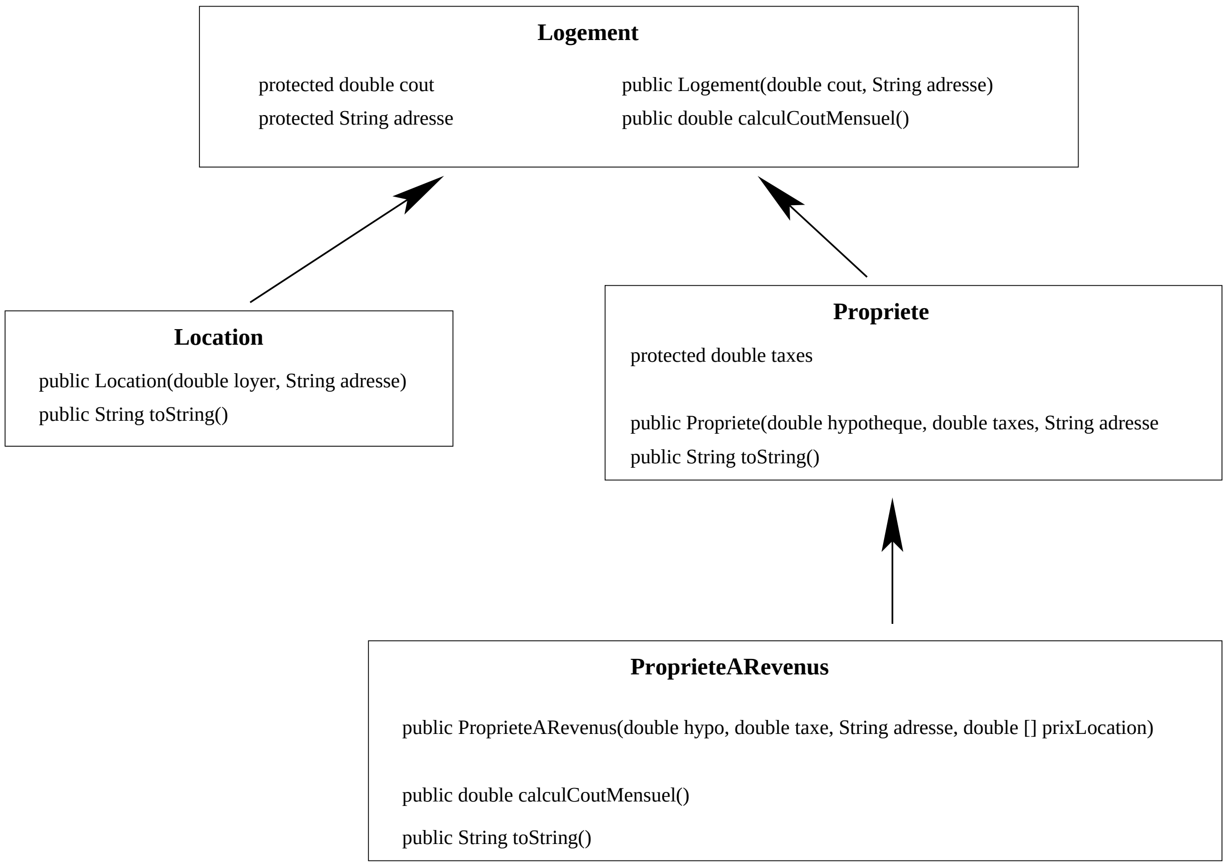
- Elle contient une variable d'état `taxes`: un `double` représentant le montant de la taxe annuelle de la propriété.
- Le constructeur reçoit trois paramètres: le montant mensuel du remboursement de l'hypothèque, le montant annuel des taxes et l'adresse de la propriété. Le constructeur initialise le coût mensuel de la propriété à la valeur de l'hypothèque.
- La méthode `calculCoutMensuel()` retourne le coût mensuel de l'hypothèque et des taxes (hypothèque + un douzième des taxes).
- La méthode `toString()` retourne une chaîne contenant l'adresse et le coût mensuel de la propriété, le montant annuel des taxes ainsi que le résultat de l'appel à la méthode `calculCoutMensuel()`.

La classe `ProprieteARevenus` dérive de la classe `Propriete`. Une instance de cette classe représente une propriété découpée en plusieurs appartements dont un est habité par le proriétaire (propriété à revenus de locations).

- La classe comporte une variable d'état `prixLocations` qui est un tableau de `double` où chaque élément représente le montant perçu par le propriétaire pour le loyer mensuel d'un logement de la propriété.
- Le constructeur reçoit quatre paramètres: le montant mensuel pour le remboursement de l'hypothèque de la propriété, l'adresse de la propriété, le montant annuel des taxes et un tableau de loyers perçus. Le constructeur initialise le coût mensuel de la propriété à la valeur de l'hypothèque.

- La méthode `calculCoutMensuel()` retourne le coût mensuel de l'hypothèque et des taxes moins la somme des loyers percus par le propriétaire pour le mois (hypothèque plus un douzième des taxes moins la somme des éléments du tableau `prixLocations`)
- La méthode `toString()` retourne une chaîne contenant l'adresse, le coût mensuel de l'hypothèque, le montant des taxes, le résultat de `calculCoutMensuel()` et la liste de tous les loyers perçus.

Question A: Dessinez la hiérarchie des classes décrites précédemment en indiquant dans chaque classe les variables d'état et méthodes (membres).



Question B: Compte-tenu de ce qui est décrit, l'instruction suivante est-elle compilable ? Répondez par **oui** ou par **non** et justifiez brièvement votre réponse.

```
Logement logement = new Logement(720.0,"25, Saint Laurent");
```

Non ! Il est impossible de créer un objet d'une classe abstraite. ....

Question C: Écrivez les classes `Location`, `Propriete` et `ProprieteARevenus` tout en respectant le principe de l'encapsulation.

```
public class  Location extends Logement {
   public Location(double loyer, String adresse) {
     super(loyer, adresse);
   }
   public String toString() {
     return  "adresse du logis : " + adresse + "\nloyer mensuel: " + cout ;
   }
}
```

```
public class Propriete extends Logement {
   protected double taxes;

   public Propriete(double hypotheque, double taxes, String adresse) {
     super(hypotheque,adresse);
     this.taxes = taxes;
   }

   public double calculCoutMensuel() {return cout+taxes/12;}

   public String toString() {
     return "adresse : " + adresse + "\nhypotheque : " +
            cout + "\ntaxes annuelles: " + taxes +
            "\ncout mensuel net: " + calculCoutMensuel();
   }
}
```

```java
public class ProprieteARevenus extends Propriete {

   private double[] prixLocations;

   public ProprieteARevenus(double hypotheque, double taxes,
                            String adresse,  double[] prixLocations) {
      super(hypotheque,taxes,adresse);
      this.prixLocations = prixLocations;
    }

    public double calculCoutMensuel() {
      double coutNet = super.calculCoutMensuel();
      for(int i = 0; i < prixLocations.length; i++)
         coutNet -= prixLocations[i];
      return coutNet;
    }

    public String toString() {
      String resultat = super.toString( )+
                        "\nrevenus de locations pour les  " +
                        prixLocations.length + " logements\n";

      for(int i = 0; i < prixLocations.length; i++)
        resultat += prixLocations[i] + "\n";
      return resultat;
    }
}
```

Question D: Soit la méthode `main`:

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

   Logement[] tabLogement = new Logement[10];

   // une location de loyer 500$ au 28 Villeneuve Ouest
   tabLogement[0] = new Location(500,"28 Villeneuve Ouest");

   /* une propriete dont l'hypotheque mensuelle est 900$, la taxe
    * annuelle de 2000$ au 29 Villeneuve Ouest */
   tabLogement[1] = new Propriete(900, 2000, "29 Villeneuve Ouest");

   /* une propriete a revenus dont l'hypotheque mesuelle est 2500$,
    * la taxe annuelle est de 7800$ et pour laquelle le proprietaire
    * percoit 3 loyers (350$, 450$ et 720$) au 42 Villeneuve Ouest */
   tabLogement[2] = new ProprieteARevenu(2500,7800,
                                  "42 Villeneuve Ouest", {350,450,720});
   // d'autres logements sont ajoutes de maniere similaire

   double moyenne = coutMoyen(tabLogement);
   double var = variance(tabLogement, moyenne);
}
```

Écrivez la méthode `variance` qui retourne la variance[2] du coût mensuel de tous les logements du tableau passé en argument.

```
public static double variance(Logement[] t, double moy) {
   double var = 0;

   if (t.length == 0)    return 0.;

   for(int i = 0; i<t.length; i++)
      var += Math.pow((t[i].calculCoutMensuel()-moy),2.0);

   return var/t.length;
}
```

[2] On rappelle que la variance d'une distribution de N valeurs $(x_1 \dots x_N)$ est: $\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$ où μ est la moyenne de la distribution.

Exercice 5 – Structure de données – (20 points):
Considérez la classe `Element` suivante:

```
public class Element{

   private double valeur;
   private Element suivant;

   public Element(double valeur){
     this.valeur = valeur;
     suivant = null;
   }

   public double getValeur() {return valeur;}
   public Element getSuivant() {return suivant;}
   public void setValeur(double valeur) {this.valeur = valeur;}
   public void setSuivant(Element suivant) {this.suivant = suivant;}
   public String toString() {return "" + valeur;}
}
```

On vous demande dans cet exercice d'implémenter une `Queue` à l'aide d'une liste chaînée (que vous programmerez vous même, du moins les méthodes utiles à ce problème) dont les noeuds sont des objets de la classe `Element`.

Question A: Écrivez la classe `Queue` qui contient les méthodes suivantes:

enqueue méthode qui ajoute un élément à la fin de la liste

dequeue méthode qui enlève un élément du début de la liste et retourne une référence vers cet objet

empty méthode qui retourne `true` si la liste est vide (et `false` sinon)

Écrivez les variables d'instances ainsi qu'un constructeur, si nécessaire. Attention, il ne vous est pas demandé d'utiliser la classe `LinkedList` de la librairie standard, mais bien d'écrire votre propre implémentation d'une liste chaînée (du moins la partie du code qui correspond à vos besoins).

Question B: Écrivez la méthode `main` qui:

(a) crée un objet de la classe `Queue`,
(b) insère dans l'ordre dans cette queue les valeurs *25.32, 123.40, 13.00* et *234.56*,
(c) appelle 2 fois *dequeue* (depuis cette queue), en affichant à chaque fois l'élément retourné par *dequeue*,
(d) insère la valeur *250.00*, et finalement,
(e) vide la liste en appelant `dequeue` autant de fois que nécessaire.

La réponse à la question A ressemble étrangement à ce que nous avons vu en cours ...

```
public class Queue{
  private Element tete, fin; // mis a null a la construction

  public void enqueue(Element unElement){
    if (tete == null)
       fin = tete = unElement;
    else {
       fin.setSuivant(unElement);
       fin = unElement;
       }
  }

  public Element dequeue() {
    if(tete == null) return null;
    else {
      Element tempo = tete;
      tete = tete.getSuivant();
      return tempo;
      }
  }

  public boolean empty() {return tete == null;}
}
```

La question B pouvait être faite sans pour autant avoir répondu à la question A et ne nécessitait que la compréhension de la notion de pile:

```
public class TestQueue{
  public static void main(String args[]) {

    Queue liste = new Queue();

    liste.enqueue(new Element(25.32));
    liste.enqueue(new Element(123.40));
    liste.enqueue(new Element(13.00));
    liste.enqueue(new Element(234.56));

    System.out.println(liste.dequeue());
    System.out.println(liste.dequeue());

    liste.enqueue(new Element(250.00));

    while(!liste.empty())
       liste.dequeue();
  }
}
```