DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

SIGLE DU COURS: IFT 1010 (A2000)

NOM DU PROFESSEUR: Nadia El-Mabrouk et Philippe Langlais

TITRE DU COURS: Programmation I

EXAMEN FINAL

Date	: 08 décembre 2000
Heure	: 13:30-16:30
Salle	: B-0245 Pav. 3200

DIRECTIVES PÉDAGOGIQUES:

- Toute documentation est permise.
- **Inscrivez tout de suite votre nom et code permanent dans la case** (Figure 1).
- Répondez **sur le questionnaire** en utilisant l'espace libre qui suit chaque question.
- Le nombre de points accordé à chaque exercice est indiqué. Le total des points est 100.

1. ....
2. ....
3. ....
4. ....
5. ....

Total ....

Figure 1: Inscrivez votre nom et votre code permanent ici

Exercice 1 (29 points):
Dans cet exercice, vous allez compléter le code de deux classes `Etudiant` et `Cours` permettant l'analyse des notes d'un cours donné. Voici un exemple d'utilisation du code que vous devez écrire:

```
public class Final {
  public static void main(String [] args) {
    Cours cours = new Cours("IFT1010",160); // au plus 160 etudiants

    cours.add("LAMONTAGNE",96,true);
    cours.add("PROULX",80,true);

    System.out.println("LAMONTAGNE assiste aux cours ? " +
               cours.assisteAuCours("LAMONTAGNE")); // affiche true
    System.out.println("FORTIN est inscrit au cours ? " +
               cours.estEnregistre("FORTIN")); // affiche -1
    System.out.println("MOYENNE " +
               cours.calculeMoyenneDeCeuxQuiAssistentAuCours()); // 88
    System.out.println("LE MEILLEUR " +
               cours.meilleurDeCeuxQuiNeViennentPasAuCours()); // LAMONTAGNE
  }
}
```

Un étudiant est caractérisé par les informations suivantes:

son nom : stocké dans une chaine de caractères;

sa note : rangée dans une variable entière;

assiduité : un booléen qui vaut `true` si l'étudiant est assidu (assiste au cours auquel il est inscrit) et `false` sinon;

nbAssidus : variable entière qui contient le nombre total d'étudiants assidus (dont la variable `assiduite` vaut `true`).

Un cours est caractérisé par les informations suivantes:

son nom : (ex: IFT1010) stocké dans une chaine de caractères (`identification`);

un tableau d'étudiants : (`etudiants`) qui contiendra tous les étudiants enregistrés dans un cours. La taille maximale de ce tableau est passée en argument au constructeur de la classe;

nb : variable entière qui indique le nombre d'étudiants effectivement rangés dans la table `etudiants`;

Les questions suivantes (de 1-A à 1-C) indiquent comment le code (pages 4 à 6) doit être complété. Pour répondre à ces questions, il s'agit uniquement de compléter ce code.

Question 1-A

Ajouter directement dans le code, pour les deux classes `Etudiant` et `cours`, les modificateurs d'accès (`public` et `private`) de manière à satisfaire au principe d'encapsulation de la programmation orientée objet.
Ajouter si nécessaire le modificateur de portée `static` aux variables et/ou méthodes concernées.

Question 1-B

Vous devez compléter le code des méthodes de la classe `Etudiant`. Suffisamment de place vous a été laissée à même le code. Si cependant vous manquez de place[1], écrivez votre réponse au verso.

Etudiant(String nom, int note, boolean assiduite) : le seul constructeur de la classe, qui prend en argument respectivement le nom de l'étudiant, sa note et son assiduité au cours pour lequel il est enregistré.

int getNote() : méthode qui retourne la note de l'étudiant

String getNom() : méthode qui retourne le nom de l'étudiant.

boolean assisteAuCours() : méthode qui retourne `true` si l'étudiant est assidu (`assiduite` vaut alors `true`) et `false` sinon.

int getNbEtudiantsAssidus() : qui retourne le nombre d'etudiants assidus.

boolean equals(Etudiant ref) : qui réalise la comparaison de la note de l'objet `Etudiant` appelant ($n_{appelant}$) avec la note de l'`Etudiant` dont la référence est passée en argument (n_{ref}). Cette méthode retourne `true` si $n_{appelant}$ est égale à n_{ref}, et `false` sinon.

Question 1-C

Vous devez maintenant compléter directement dans le code (vous avez suffisamment de place) les méthodes de la classe `Cours`:

Cours(String nom, int nbEtudiants) : le seul constructeur de la classe. Prend en argument l'intitulé du cours ainsi que le nombre maximum d'étudiants de ce cours.

int estEnregistre(String nom) : méthode qui retourne l'indice dans la table `etudiants` de l'étudiant dont le nom est spécifié. Si l'étudiant n'existe pas, la méthode retourne `-1`.

boolean add(String nom, int note, boolean assiduite) : méthode qui ajoute dans la table (s'il n'y est pas déjà) l'étudiant dont le nom est spécifié en premier argument. Les deux paramètres suivants sont respectivement sa note et son assiduité.

boolean assisteAuCours(String nom) : prédicat qui retourne `true` si l'étudiant nommé assiste habituellement au cours (`assiduite` vaut `true`) et `false` sinon.

float calculeMoyenneDeCeuxQuiAssistentAuCours() : retourne la moyenne des étudiants assidus (ceux dont la variable `assiduite` est `true`). Note: cette moyenne peut éventuellement contenir une partie décimale (ex: 87.5).

String meilleurDeCeuxQuiNeViennentPasAuCours() : retourne le nom de l'étudiant qui a obtenu la meilleure note parmi les étudiants non assidus (ceux dont la variable `assiduite` est `false`). Note: dans le cas improbable où tous les étudiants enregistrés viennent en cours, alors cette méthode retourne une chaine vide.

---

[1]Dans ce cas, vous faites probablement un mauvais usage des méthodes que vous devez écrire.

Code à compléter

```
class Etudiant {
    private String nom;
    private int note;
    private boolean assiduite;
    private static int nbAssidus = 0;

    public Etudiant(String nom, int note, boolean assiduite) {
        this.nom = nom;
        this.note = note;
        this.assiduite = assiduite
        if (assiduite)
            nbAssidus ++;
    }

    public String getNom() {return nom;}
    public int getNote() {return note;}
    public boolean assisteAuCours() {return assiduite;}
    public static int getNbEtudiantsAssidus() {return nbAssidus;}

    public boolean equals(Etudiant ref) {
        return ( this.note = ref.getNote() );
    }
}

// -----------------------------------------------------------
// classe Cours de l'exercice 1
// -----------------------------------------------------------

class Cours {
    private String identification;
    private Etudiant [] etudiants;
    private int nb;

    public Cours(String nom, int nbEtudiants) {
        identification = nom;
        if (nbEtudiants < 1) nbEtudiants = 1;
        etudiants = new Etudiant[nbEtudiants];
        nb = 0;
    }
```

```java
public int estEnregistre(String nom) {
    for (int i=0; i<nb; i++)
        if (etudiants[i].getNom().equals(nom))
            return i;
    return -1;
}

public boolean add(String nom,int note, boolean assiduite) {
    if ( (estEnregistre(nom) == -1) && (nb<etudiants.length) ) {
        etudiants[nb++] = new Etudiant(nom,note,assiduite);
        return true;
    }
    return false;
}

public boolean assisteAuCours(String nom) {
    int index = estEnregistre(nom);
    if (index != -1)
        return etudiants[index].assisteAuCours();
    return false;
}

public float calculeMoyenneDeCeuxQuiAssistentAuCours() {
    int somme=0;
    int nb = Etudiant.getNbEtudiantsAssidus();
    if (nb == 0) return 0.f;

    for (int i=0; i<nb; i++)
        if (etudiants[i].assisteAuCours())
            somme += etudiants[i].getNote();
    return somme / (float) nb;
}
```

```
// si tous les etudiants d'un cours viennent en cours, alors
// cette methode retourne une chaine vide.
public String meilleurDeCeuxQuiNeViennentPasAuCours() {
    int max_i =-1, max = -1;
    for (int i=0; i<nb; i++)
        if ((!etudiants[i].assisteAuCours()) && (etudiants[i].getNote() > max)) {
            max_i = i;
            max = etudiants[i].getNote();
        }
    return (max_i == -1)? "" : etudiants[max_i].getNom();
}

// pas demande
public String toString() {
    String res = "";
    for (int i=0; i<nb; i++)
        res = res + "[" + etudiants[i].getNom() + " " + etudiants[i].getNote()+ "] ";
    return res;
}
```

Exercice 2 (13 points):

Dans cet exercice, on vous demande d'expliquer de manière concise des erreurs de programmation.

1. Cette méthode n'est pas compilable. Indiquez en une phrase pourquoi.

```
boolean egal4() {
  int a,b;
  if (a == 4) return true;
  return false;
}
```

Réponse:

Erreur de compilation: La variable a n'a pas été initialisée.

2. Cette méthode contient une erreur qui rend impossible sa compilation. Décrivez-la.

```
int div0(int a, int b) {
  if (b == 0) System.err.println("Division par zero");
  else return a/b;
}
```

Réponse:

Si b est égal à 0, il n'y a pas de valeur de retour: manque une instruction "return" dans le cas où $b = 0$.

3. Soit la méthode:

```
void doIt(int a,float b) {System.out.println();}
```

Indiquez pour chacun des appels suivants ceux qui sont corrects et ceux qui ne le sont pas (pas compilables). Indiquez précisement lorsque l'appel est incorrect la raison de l'erreur.

appel	correct? (OUI ou NON)	justification
`doIt(4,3);`	**oui**	**Conversion élargissante de `int` à `float`**
`doIt(4.5,3);`	**non**	**Conversion rétrécissante de `double` à `int`**
`doIt(3,4.5);`	**non**	**Conversion rétrécissante de `double` à `float`**
`doIt(0,0.f);`	**oui**	**Types de données compatibles**

4. Ce code provoque une erreur d'exécution. Décrivez-la.

```
int [][] tab = {{1,2}, {5,8,10}};
System.out.println(tab[0][2]);
```

Réponse: Erreur de compilation: Débordement de capacité. L'élément `tab[0][2]` n'existe pas, étant donné que la première ligne du tableau est de taille 2.

Exercice 3 (14 points) Soit le programme suivant, contenant la méthode récursive queFait.

```
public class AlgoRecursif
{
    public static void main (String[] args)
    {
        int [] tab = {5, 10, 34, 3, 10, 8};
        System.out.println(queFait(tab,10,0));
        System.out.println(queFait(tab,7,0));
    }

    public static int queFait(int[] tab, int val, int n)
    {
        if (n == tab.length)
            return (-1);

        else
            if (tab[n] == val)
                return(n);
            else
                return (queFait(tab,val,n+1));
    }
}
```

1. Que fait cette méthode? Quel est l'affichage du programme?

   Cette méthode retourne le plus petit indice i tel que `tab[i]=val`, et retourne -1 si `val` n'est pas dans le tableau.

   Affichage:

   1

   -1

2. Écrire une méthode itérative qui effectue la même chose que la méthode récursive précédente.

```
public static int chercheValeur(int[] tab, int val)
{
    boolean test = false;
    int i = 0;

    while ( (i<tab.length) && !test )
        {
            if (tab[i] == val)
                test = true;
            else
                i++;
        }

    if ( test )
        return i;
    else
        return -1;
}
```

Exercice 4 (24 points)

Soit le programme:

```
import java.util.Vector;
public class Tableaux
{
    public static void main (String[] args)
    {
        int[] tab1 = {1, 2, 3, 4, 5};
        int[][] tab2 = {{6,7}, {8,9,10}, {11,12}};
        Vector prenom1 = new Vector();
        Vector prenom2 = new Vector();

        prenom1.addElement("Zoe"); prenom1.addElement("Paul");
        prenom1.addElement("Lea");

        prenom2.addElement("Marie"); prenom2.addElement("John");
        prenom2.addElement("George");
```

```
     System.out.print("prenom1, version initiale: ");
     System.out.println(prenom1);
     System.out.print("prenom2, version initiale: ");
     System.out.println(prenom2);

     change1 (prenom1.elementAt(0),prenom2.elementAt(0));
     System.out.print("prenom1, apres le 1er changement: ");
     System.out.println(prenom1);

     change2 (prenom1,prenom2);
     System.out.print("prenom1, apres le 2eme changement: ");
     System.out.println(prenom1);

     affichage("tab1, version initiale:",tab1);
     affichage("tab2, version initiale:",tab2);

     change3 (tab1,tab2[0]);
     affichage("tab1, apres changement:",tab1);
     affichage("tab2, apres changement:",tab2);
 }

 public static void affichage (String texte, int[][] tab) {
     System.out.println(texte);
     for (int i=0; i<tab.length; i++) {
             for (int j=0; j<tab[i].length; j++)
                 System.out.print(tab[i][j]+" ");
             System.out.println();
         }
 }

public static void affichage (String texte, int[] tab) {
    System.out.println(texte);
    for (int i=0; i<tab.length; i++) System.out.print(tab[i]+" ");
    System.out.println();
 }

public static void change1 (Object nom1, Object nom2) {
    Object inter =  nom2;
    nom2 = nom1; nom1 = inter;
}

public static void change2 (Vector nom1, Vector nom2) {
    nom1.addElement (nom2.elementAt(0));
    nom1.insertElementAt (nom2.elementAt(2),2);
}
```

```
   public static void change3 (int[] liste1, int[] liste2) {
       liste1 = liste2;
       liste1[1] = 888;
   }
}
```

Quelle est la sortie de ce programme (donner les affichages successifs)?

```
prenom1, version initiale: [Zoe, Paul, Lea]
prenom2, version initiale: [Marie, John, George]
prenom1, apres le 1er changement: [Zoe, Paul, Lea]
prenom1, apres le 2eme changement: [Zoe, Paul, George, Lea, Marie]
tab1, version initiale:
1 2 3 4 5
tab2, version initiale:
6 7
8 9 10
11 12
tab1, apres changement:
1 2 3 4 5
tab2, apres changement:
6 888
8 9 10
11 12
```

Exercice 5 (20 points):
Écrire une méthode qui prend en argument un tableau `tab` d'entiers, et qui retourne un deuxième tableau résultat `res`. `res` contiendra au plus les `n` plus grandes valeurs de `tab` qui sont supérieures à un entier `seuil`. Noter que le tableau `tab` (et par conséquent `res`) peut contenir moins de `n` entiers supérieurs à `seuil`.

Entête de la méthode à écrire:

```
int [] meilleuresValeurs (int [] tab, int n, int seuil)
```

Algorithme

Voici l'algorithme que vous devez programmer. Cet algorithme nécessite un seul passage sur le tableau `tab`.

1. Créer une table `res` qui contiendra, dans l'ordre décroissant, les `n` plus grandes valeurs de `tab`. `res` est initialement vide et contient, à la fin, au plus `n` éléments.

2. Pour chaque indice i dans `tab`, et `e=tab[i]`

   Si `e` est supérieur ou égal à `seuil`, alors tenter l'insertion de `e` dans `res` comme suit:
   - Chercher dans `res` l'indice j tel que les entiers avant j sont supérieurs ou égaux à `e`, et ceux après j sont inférieurs à `e`.
   - Si nécessaire, décaler vers la droite tous les éléments de `res`, à partir de l'indice j. Dans le cas où `res` contient déjà `n` valeurs, alors le décalage vers la droite provoque l'élimination du dernier élément de `res`.
   - Ajouter `e` dans `res` à l'indice j.

Exemple

Soit le tableau `tab = [ 8 3 5 1 4 ]`, et l'appel suivant: `meilleuresValeurs (tab, 3, 2)`.
Déroulement de l'algorithme:

i	e	j	res	
			vide	
0	8	0	[8]	
1	3	1	[8 3]	
2	5	1	[8 5 3]	
3	1		[8 5 3]	ignoré car 1 est inférieur au seuil (2)
4	4	2	[8 5 4]	3 supprimé car on garde juste les 3 meilleures valeurs

```java
public static int [] meilleuresValeurs (int [] tab, int n, int seuil)
{
    int [] res = new int[n+1];
    int tailleRes = 0;
    int j, k;

    for (int i = 0; i < tab.length; i++)
        if (tab[i] >= seuil)
            {
                j = 0;
                while ( (j<tailleRes) && (res[j]>=tab[i]) )
                    j++;

                for (k = tailleRes - 1; k>=j; k--)
                    res[k+1] = res[k];
                res[j] = tab[i];

                if (tailleRes<n) tailleRes++;

            }
    return res;
}
```