DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE

SIGLE DU COURS: IFT 1010 (H2000)

NOM DU PROFESSEUR: Nadia El-Mabrouk

TITRE DU COURS: Programmation I

EXAMEN FINAL

Date	: 13 avril 2000
Heure	: 9:00-12:00
Salle	: B-2245

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.

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

Total ....

Figure 1: Inscrivez votre nom et votre code permanent ici

Exercice 1 (24 points):

On veut développer un programme qui simule le fonctionnement d'un compte banquaire. Le programme sera formé de 2 classes

- Une classe `Banque` contenant la méthode main qui instancie un compte banquaire, et effectue un certain nombre de transactions sur ce compte.
- Une classe `Compte` contenant les méthodes de gestion d'un compte banquaire. Les méthodes service sont les suivantes:
  - Le constructeur Compte;
  - Une méthode `credit` qui rajoute une somme d'argent `valeur` au compte.
  - Une méthode `debit` qui soustrait une somme d'argent `valeur` au compte.
    Dans le cas des deux méthodes `credit` et `debit`, des frais supplémentaires sont retirés au compte. Ces frais correspondent à 2% de `valeur`.
  - Deux méthodes `soldeCompte` et `nomCompte` qui retournent, respectivement, la somme d'argent dans le compte, et le nom du compte ("Tom").
  - Une méthode `soldable` qui renvoie un booléen indiquant si le compte est soldable (i.e. s'il y a encore de l'argent).

  La classe `Compte` contient également une méthode support `frais`.

On donne, ci-dessous, la classe `Banque`, et une esquisse de la classe `Compte`. De plus, dans les deux classes, les modificateurs de visibilité (`public, private, final`) et le modificateur `static` manquent, aussi bien pour les variables que pour les méthodes.

```
public class Banque {
        private final static int SOMME_MIN = 5000;

        public static void main (String[] args) {
        int operations;
        Compte tom = new Compte ("Tom",SOMME_MIN);
        System.out.println("Le solde de " + tom.nomCompte() + " est:"
                                                  + tom.soldeCompte());
        tom.credit(500);
        while (tom.soldable())
            {
                tom.debit(1000);
                System.out.println("Nouveau solde: " + tom.soldeCompte());
            }

        System.out.println(tom.nomCompte() + " est dans le rouge!!");
    }
}

class Compte
{
    private final static int POURCENT = 2;
    private int somme;
    private String nom;
```

```java
    public Compte (String nom, int somme)
    {
        this.nom = nom;
        this.somme = somme;
    }

    private int frais (int valeur)
    {
        return ( (valeur*POURCENT)/100 );
    }

    public void debit (int valeur)
    {
        somme =  somme - valeur - frais (valeur);
    }

    public void credit (int valeur)
    {
        somme = somme + valeur - frais(valeur);
    }

    public int soldeCompte ()
    {
        return (somme);
    }

    public String nomCompte ()
    {
        return (nom);
    }

    public boolean soldable()
    {
        return (somme > 0);
    }
}
```

(a) Compléter le programme précédent avec les modificateurs de visibilité et le modificateur `static` adéquats. Ces modificateurs doivent permettre au programme de fonctionner correctement, et respecter les conventions d'encapsulation de Java. Les écrire aux endroits appropriés dans le programme. Expliquer brièvement:

- La logique d'attribution des modificateurs `public` et `private`.
Pour une encapsulation adéquate, les variables d'instanciation d'une classes sont privées. Pour les méthodes, le modificateur "public" est attribué aux méthodes service, pour qu'elles puissent être accessibles de l'extérieur de la classe. Le modificateur "private" est attribué aux méthodes support, qui ne sont utiles qu'à l'intérieur de la classe. Dans ce programme, il n'existe qu'une seul méthode support: "frais".

- Pourquoi des modificateur `final` ont été attribués, s'il y a lieu?
"final" permet de définir des constantes, i.e. des variables qui ne changent pas de valeurs tout au long du programme.

- Pourquoi des modificateurs `static` ont été attribués, s'il y a lieu?
La méthode "main" doit être déclarée statique pour qu'on puisse y faire appel sans avoir besoin d'instancier un objet de la classe. Étant donné qu'une constante à la même valeur pour tous les objets de la classe, il est préférable de la déclarer statique pour avoir un seul emplacement mémoire pour tous les objets. Cela permet d'économiser de l'espace.

(b) Compléter les méthodes service de la classe `Compte`:
L'espace suivant la signature de chaque méthode dans le programme est suffisant pour écrire les instructions nécessaires.

Exercice 2 (12 points): Les fragments de programme suivants présentent chacun une erreur de compilation. Expliquer le problème et dire comment le résoudre en une phrase courte (on ne parle pas, ici, de modificateurs de visibilé manquants).

(a)
```
int sum (int x, int y)
{
    int result;
    result=x+y;
}
```
Erreur de compilation: "Return" manque à la fin de de la méthode "sum".

Correction: Rajouter l'instruction return(result);

(b)
```
int division ()
{
    int x=7, y=8;
    float result;
    result=x/y;
    System.out.println(" Le produit est : " + result);
    return(result);
}
```
Erreur de compilation: Le type de retour de la méthode avec le type de la variable retournée.

Correction: Le type de retour de la méthode doit être "float", ou alors, result doit être déclaré "int".

(c)
```
public static void main (String[] args)
{
    int y=3;
    int result = cube();
}

static int cube ()
{
    return y*y*y;
}
```
Erreur de compilation: La variable "y" est indéfinie dans la méthode main.

Correction: Définir "y" comme variable statique de la classe, ou passer "y" en paramètre de la méthode.

(d)

```
public static void main (String[] args)
{
    char a;
    a = caracPosition("Exemple",3);
}

static char caracPosition(int pos, String chaine)
{
    return(chaine.charAt(pos))
}
```

Erreur de compilation: L'ordre des paramètres formels de la méthode ne correspond pas à celui des paramètres effectifs lors de l'appel à la méthode.

Correction: Intervertir l'ordre des paramètres, soit dans la signature de la méthode, soit dans l'appel à la méthode.

Exercice 3 (20 points): Soit le programme:

```
public class Test {
    public static void main (String[] args) {
        int valeur;
        int[][] tab1 = {{1,2},{3}};
        int[][] tab2 = {{8,9},{10,11,12},{13}};

        print("Premier Tableau, version originelle:",tab1);
        valeur = change1(tab1[1][0]);
        System.out.println("valeur = " + valeur + ", tab1[1][0] = " + tab1[1][0] );
        change2(tab1,tab2);
        print("Premier Tableau, apres change2: ",tab1);
        change3(tab1,tab2);
        print("Premier Tableau, apres change3: ",tab1);
        print("Deuxieme Tableau, apres change3: ",tab2);

    }

    public static void print (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 int change1(int nb) {
        nb=0;
        return(nb);
    }
```

```
    public static void change2(int[][] list1, int[][] list2) {
        list1[0]=list2[1];
        list1[1]=list2[0];
    }

    public static void change3(int[][] list1, int[][] list2) {
        list1=list2;
        list1[0]=list2[2];
    }
}
```

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

Premier Tableau, version originelle:

1 2

3

valeur = 0, tab1[1][0] = 3

Premier Tableau, apres change2:

10 11 12

8 9

Premier Tableau, apres change3:

10 11 12

8 9

Deuxieme Tableau, apres change3:

13

10 11 12

13

Exercice 4 (20 points):

Soit la méthode récursive:

```
    public static int caFaitQuoi(int a, int b)
    {
        if (b==0)
            return 1;
        else return(a*caFaitQuoi(a,b-1));
    }
```

(a) Que fait cette méthode, si elle est appelée avec $b > 0$? Quel est le résultat pour $a = 2$ et $b = 4$?

Calcule “a” à la puissance “b”. Le résultat pour a=2 et b=4 est 16.0.

(b) Modifier la méthode pour qu'elle produise un résultat correct quelque soit la valeur de b (b positif ou négatif).

```
public static float puissance(float a, int b)
{
    if (b==0)
        return 1;
    else
        if (b>0) return(a*puissance(a,b-1));
        else
            return ((1/a)*puissance(a,b+1));
}
```

(c) Écrire une méthode itérative qui effectue la même chose que la méthode récursive précédente, pour b quelconque.

Important: Vous ne devez utiliser aucune méthode mathématique prédéfinie, à part, peut-être, la méthode statique `Math.abs` pour le calcul de la valeur absolue d'un nombre.

```
public static float puissance_iteratif(float a,int b)
{
    float result=1;
    for (int i=1; i<=Math.abs(b); i++)
        result*=a;
    if (b>0)
        return(result);
    else
        return(1/result);
}
```

Exercice 5 (24 points):

L'exercice consiste à développer un programme qui génère une liste de 10 nombres aléatoires entre 0 et 100, et qui trie cette liste dans un ordre croissant. Les nombres aléatoires seront mis dans un tableau. On décrit, ci-dessous, la façon de générer des nombres aléatoires, et la façon de trier le tableau:

- On considère un générateur de nombre aléatoire spécifique (on n'utilise pas la classe `Random`), utilisant la formule:

  $$r = ((r * 25173) + 13849)\%65536$$

  La programme commence avec une valeur initiale de r (par exemple, $r = 0$), et à chaque itération, la formule est utilisée avec, comme valeur de r, le nombre aléatoire obtenu à l'itération précédente.

  La formule produit un nombre dans l'intervalle $[0, 65535]$. Étant donné que l'on veut des nombres aléatoires dans l'intervalle $[0, 100]$, une conversion doit être effectuée.

- Une fois que le tableau *tab* est remplit avec 10 nombres aléatoires, on le trie en utilisant la procédure suivante:
  - Effectuer un parcours complet du tableau, i.e. un index i prend les valeurs successives $0, 1, \cdots, tab.length$. Pour tout index $i < tab.length$, si $tab[i]$ est supérieur à $tab[i+1]$, alors échanger les deux cases i et $i+1$.
  - Si à la fin du parcours du tableau, au moins un échange a été effectué, alors recommencer un autre parcours complet du tableau.
  - Recommencer ce processus jusqu'à ce qu'un parcours ne provoque plus aucun échange de cases. Cela signifie que le tableau est trié.

    Considérons, par exemple le tableau $[4, 8, 2, 6, 5]$ à 5 éléments. Les échanges successifs effectués lors d'un premier parcours du tableau sont:

    ```
    i=0:    4  8  2  6  5
    i=1:    4  2  8  6  5
    i=2:    4  2  6  8  5
    i=3:    4  2  6  5  8
    ```

    Les ordres obtenus après chaque parcours du tableau sont:

    ```
    1 er parcours:          4  2  6  5  8
    2 eme parcours:         2  4  5  6  8
    3 eme parcours:         2  4  5  6  8
    ```

    Au cours du 3^{eme} parcours, aucun échange n'est effectué. Alors, le programme s'arrête, et le tableau est trié.

On vous donne, ci-dessous, le début du programme, i.e. la classe `ListeAleatoire` contenant la méthode main. La constante `VALEUR_INIT` est la valeur initiale du générateur de nombres aléatoires. Cette méthode instancie un objet d'une classe `Liste`. Cette classe doit définir toutes les variables et méthodes nécessaires au remplissage d'un tableau de nombres aléatoires, et au tri du tableau.

Écrire la classe `Liste`. Les méthodes service à définir sont les suivantes

(a) Le constructeur `Liste`.

(b) Une méthode `print` pour l'affichage du tableau.

(c) Une méthode `remplit` qui remplit le tableau de nombres aléatoires, comme indiqué ci-dessus.

(d) Une méthode `trie` qui trie un tableau, comme indiqué ci-dessus.

Remarque: D'autres méthodes support pourront être développées pour faciliter la définition de certaines des méthodes service précédentes.

```
public class ListeAleatoire {
    public static final int TAILLE=10;
    public static final int VALEUR_INIT=0;
    public static void main (String[] args) {
        Liste tab = new Liste(TAILLE,VALEUR_INIT);
        tab.remplit();
        System.out.println("Tableau de nombres aleatoires:");
        tab.print();
        tab.trie();
        System.out.println("Tableau trie:");
        tab.print();
    }
}

class Liste
{
    private int[] liste_al;
    private int valeur;

    public Liste (int taille,int valeur)
    {
        liste_al = new int[taille];
        this.valeur=valeur;
    }

    public void print()
    {
        for (int i=0; i<liste_al.length; i++)
            System.out.print(liste_al[i]+ "  ");
        System.out.println();
    }

    public void remplit()
    {
        for (int i=0; i<liste_al.length; i++)
            {
                valeur = nbAleatoire(valeur);
                liste_al[i]=valeur;
            }
    }
```

```
    private void permute(int index1, int index2)
    {
        int temp;

        temp=liste_al[index1];
        liste_al[index1]=liste_al[index2];
        liste_al[index2]=temp;

    }

    public void trie()
    {
        int permutations;

        do
            {
                permutations=0;
                for (int i=0; i<liste_al.length-1; i++)
                    if (liste_al[i]>liste_al[i+1])
                        {
                            permute(i,i+1);
                            permutations++;
                        }
            }
        while (permutations>0);
    }

    private int nbAleatoire(int valeur)
    {
        return((((valeur * 25173)+13849)%65536)%100);
    }
}
```