IFT1170 - Session Été, Intra-Solution

Mohamed Lokbani

IFT1170 - INTRA

Nom:___________________________________| Prénom(s):______________________________|

Signature:_______________________________| Code perm:______________________________|

Date:14 juin 2001

Durée: 2 heures (de 20h30 à 22h30) Local: Z-110

Directives:

- Il vous est permis d'utiliser un livre de votre choix.
- Les documents de cours ne sont pas autorisés.
- Ordinateurs personnels prohibés.
- Calculatrice (simple) permise.

- Répondre directement sur le questionnaire.

1.______________________/24

2.______________________/09

3.______________________/17

4.______________________/20

5.______________________/30

Total:___________________/100

Question 1 (24 points)

-Q1.1- Quels sont les avantages et les inconvénients de la programmation orientée objet? Expliquez brièvement.

<u>Réponse</u>

Les objets sont plus stables que les spécifications qui définissent leurs interactions. De ce fait les application sont plus simples à écrire et à faire évoluer.

Cependant:
- il est impossible de prévoir toutes les actions que l'ont peut avoir à effectuer sur un objet. L'héritage peut palier à ce problème.
- le choix des classes dépend de l'application en question. Ce choix ne peut être réalisé qu'à partir d'une étude approfondie de l'application.

-Q1.2- Définissez ce qu'est l'instanciation d'un objet?

<u>Réponse</u>

Fabrication à partir du modèle de la classe, d'un objet particulier, élément de cette classe.

Objet = instance de la classe.

-Q1.3- Si un constructeur explicite est défini dans une classe, a-t-on accès au constructeur par défaut de cette classe? (encerclez la réponse correcte)

Oui, nous avons accès au constructeur par défaut.
Non, nous n'avons pas accès au constructeur par défaut.

Pourquoi?

<u>Réponse</u>

Si, dans une classe, il a été défini un constructeur, ce dernier masque le constructeur par défaut. Ainsi, le constructeur par défaut cesse d'exister et devient donc inaccessible.

-Q1.4- En plus d'être **public**, la méthode **main** doit-elle être aussi **static**? (encerclez la réponse correcte)

Oui, la méthode main doit être aussi **static**.
Non, la méthode main n'a pas à être **static**.

Pourquoi?

<u>Réponse</u>

Une méthode statique peut être appelée même si aucun objet n'a été crée. Par ailleurs, la machine virtuelle accède à un programme donné à travers un point d'entrée. Ce point d'entrée est représenté par la méthode main. Pour que cette méthode soit accessible de l'extérieur (pour la MV) sans qu'un objet ait été déjà crée, il faudra que la méthode main soit static.

-Q1.5- Dans l'instruction:

```
Test tst = new Test();
```

À quoi réfère le **`Test`** à gauche du "**`=`**" ? Et celui à droite du "**`=`**"?

<u>Réponse</u>

à gauche: nom de la classe

à droite: constructeur.

-Q1.6- Combien d'objets et de références à des objets sont crées dans le fragment de code suivant et pourquoi (expliquez votre réponse)?

```
Test premier, second;
premier = new Test();
Test exemple = new Test();
```

(une ou plusieurs réponses)

a) un objet a été crée
b) deux objets ont été crées
c) trois objets ont été crées
d) une référence a été créée
e) deux références ont été créées
f) trois références ont été créées

<u>Réponse</u>

Les réponses correctes sont: b & f.

La commande new a permis de créer les 2 objets premier et exemple.

3 références ont été créés et sont associées à : premier, second et exemple.

Il faut distinguer entre:

Test exemple; cette instruction permet de créer une référence à un objet.

exemple = new Test(); cette instruction créé physiquement l'objet.

-Q1.7- Si vous avez le programme **`test.java`** et vous devez l'exécuter, laquelle des instructions suivantes permettra de le faire et pourquoi (expliquez aussi pourquoi les autres possibilités sont incorrectes).

a) `java Test`
b) `javac test`
c) `java test.java`
d) `java test.class`
e) `java test`
f) `java test.main()`

<u>Réponse</u>

La réponse correcte est: e.

a) incorrecte: Java fait la distinction entre majuscule et minuscule.

b) incorrecte: la commande javac est utilisée pour la compilation mais pas pour l'exécution d'un programme.

c) incorrecte: le programme doit exécuter un .class et non pas .java (code source).

d) incorrecte: le programme va chercher la classe test dans le paquetage class.

e) correcte: test.java étant le nom du programme source. test.class sera le nom du programme après compilation. Pour exécuter ce programme, il faut donc utiliser le nom du fichier sans extension avec la commande java.

f) incorrecte: la même réponse que d, si nous faisons abstraction de l'emplacement des ().

-Q1.8- Expliquez l'intérêt d'avoir la méthode **`toString`** dans une classe.

<u>Réponse</u>

Chaque classe même sans l'utilisation de l'héritage dérive en réalité de la classe: Object

Dans cette classe, il est défini une fonction qui convertie les champs de X vers String si nécessaire.

En réalité si le mécanisme de conversion n'est pas la, tout ce que cette fonction va réaliser c'est d'afficher le type de l'objet qui fait l'appel et son emplacement mémoire (en réalité dans une hashtable: une table spécialisée contenant des clés. Il est associé à chaque objet une clé).

Il est conseillé de définir dans chaque classe que vous allez définir, une méthode toString pour masquer celle par défaut et afficher par conséquence des informations que vous jugez plus utiles, que celles fournies par la méthode par défaut.

Question 2 (9 points)

Que vont afficher en sortie les expressions suivantes et dans chacun des cas expliquez brièvement pourquoi?

-Q2.1- `System.out.println(6 + 7 - 1);`

<u>Réponse</u>

12

L'addition d'entiers: 6+7-1 = 12

-Q2.2- `System.out.println(-5 +7- -6);`

<u>Réponse</u>

8

L'addition d'entiers: -5 + 7 - -6= -5+7+6=8

-Q2.3- `System.out.println(13 % 3);`

<u>Réponse</u>

1

Le reste de la division de deux entiers: 13/3 = 4 et le reste = 1.

-Q2.4- `System.out.println(13.5 % 3.5);`

<u>Réponse</u>

3.0

Le reste de la division de deux réels: 13.5/3.5 = 3 et le reste = 3.0.

-Q2.5- `System.out.println(8 + 4 + "9");`

<u>Réponse</u>

129

L'addition de deux entiers puis la concaténation avec une chaîne de caractères:

8+4 = 12 + "9" = 129

-Q2.6- `System.out.println("8" + 4 + 9);`

<u>Réponse</u>

849

Concaténation directe car la chaîne se trouve en premier:

"8"+4+9=849

-Q2.7- `System.out.println(8 + 1.0f);`

<u>Réponse</u>

9.0

La somme d'un entier et un flottant c'est un flottant.

-Q2.8- `System.out.println(55/7);`

<u>Réponse</u>

7

La division de deux entiers est un entier.

-Q2.9- `System.out.println('c' + 2);`

<u>Réponse</u>

101

Le caractère est converti en un entier. Le code ascii de 'c' est 99. Puis 99+2 = 101.

Question 3 (17 points)

-Q3.1- Expliquez le sens du paragraphe suivant :

« En java, le passage d'un objet dans une méthode se fait par référence. La méthode appelée ne peut pas modifier la valeur de la référence, associée à l'objet, de la méthode appelante, bien qu'elle puisse manipuler directement l'objet. »

<u>Réponse</u>

En java, les arguments sont passés aux méthodes de 2 manières: le passage par valeur et le passage par référence. Le choix de l'une des deux manières se fait de manière automatique en fonction de l'argument. Ainsi le passage par référence est automatiquement choisi pour passer des objets. Le passage par référence signifie que l'appelant permet à la méthode appelé d'accéder directement à ses données et les modifier au besoin. Par contre, dans la méthode appelée, la référence associée à l'objet ne peut pas être modifiée. La référence quand elle est attribuée, elle l'est pour toujours.

-Q3.2- Que va afficher en sortie le programme suivant et pourquoi?

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

      int i = 0; int j = 0;
      int [] tab = new int[1]; tab[0] = i;
      increment1(i); increment2(tab);
      System.out.println("i=" + i + " j=" + j + " tab[0]=" + tab[0]);
    }
    public static void increment1(int x) {x++;}
    public static void increment2(int[] x) {x[0]++;}
}
```

<u>Réponse</u>

i=0 j=0 tab[0]=1

<u>Explications</u>

```
      int i = 0; int j = 0;
      int [] tab = new int[1]; tab[0] = i;
```

i = 0, j=0 et tab[0]=i=0

appel de increment1(0)

passage par valeur. La valeur de i ne va pas changer:

x=i=0; x++; x=1; mais i=0;

appel de increment2(tab)

un tableau est passé par référence.

x[0]=0; x[0]++; x[0]=1; à ce niveau la valeur de tab[0]=1 à cause du passage par référence.

```
      System.out.println("i=" + i + " j=" + j + " tab[0]=" + tab[0]);
```

i=0 j=0 tab[0] = 1

Question 4 (20 points)

Le programme suivant contient plusieurs erreurs de syntaxe ; identifiez-les et corrigez-les pour qu'il puisse compiler et exécuter correctement. Les numéros de ligne qui figurent au début de chaque ligne ne font pas partie du code et servent à identifier clairement les erreurs.

```
1   PUBLIC CLASS Erreur {
2
3       PUBLIC void main(string args) {
4
5         double fahrenheit = 55.5;
6
7         double celsius = convertion(fahrenheit);
8
9         */ convertion /*
10
11
12        System.out.println(fahrenheit + ’F = ’ + celsius + ’C’);
13      }
14
15      double convertion(float fahrn) {
16        RETURN (fahr - 32) * 5 / 9;
17      }
18
19  }
```

Format de réponse : Ligne : xxx ; Erreur : ; Courte explication : ; Correction :

<u>Réponse</u>

Ligne -1-

java fait la distinction entre majuscule et minuscule. Deux erreurs:
PUBLIC → public & CLASS → class

Ligne -3-

PUBLIC → public (idem que -1-)
main est une méthode static (voir Q1.4)
string → String : la classe String.
args → [ ] args : un tableau d'arguments.

Ligne -5-

Ok. Attention au type double de la variable fahrenheit.

Ligne -7-

Ok.

Ligne -9-

/ convertion / → /* convention */
les commentaires en java entre /* et */ ou // pour une seule ligne.

Ligne -12-

System.out.println(fahrenheit + 'F = ' + celsius + 'C'); →
System.out.println(fahrenheit + "F = " + celsius + 'C');

' ' utilisée pour afficher un seul caractère. Dans le cas de 2 caractères et plus, il faut utiliser " ". On peut utiliser " " pour afficher aussi un seul caractère.

Ligne -15-

Vu que main est une méthode static, la méthode convention doit l'être aussi.
elle accepte un argument du type double.

Ligne -16-

RETURN → return
fahr → fahrn

Question 5 (30 points)

Écrivez un programme **`stats.java`** qui calcule la fréquence d'apparition de chaque mot dans une phrase passée en argument. Cette phrase peut contenir plusieurs mots alphanumériques écrits en majuscules et/ou minuscules. Ce programme devra afficher en sortie, les mots et leur fréquence respective, dans un ordre décroissant par rapport à la fréquence d'apparition. Dans le cas ou deux mots ont la même fréquence, affichez ces mots dans un ordre croissant alphabétique.
Nous considérons que deux mots sont identiques s'ils ne diffèrent que par le fait que certains de leur caractères sont en majuscule et minuscule. Par exemple : HOMME, Homme et homme sont considérés ici comme identiques, i.e. le même mot. Aussi, afin de faciliter l'exercice, la ponctuation est à ignorer ainsi que les caractères accentués.

Exemple, pour la phrase (citation de Albert Camus):

« Certains hommes parlent pendant leur sommeil Les conferenciers parlent pendant le sommeil des autres »

Séparateurs de mots : espace ou fin de phrase ou début de phrase. Le résultat doit être :

mot	fréquence
parlent	2
pendant	2
sommeil	2
autres	1
certains	1
conferenciers	1
des	1
hommes	1
le	1
leur	1
les	1

<u>Réponse</u>

```
class Stat {

    int nbre_mots; // nbre de mots dans la phrase
    String [] TAB; // la phrase en minuscule
    int [] frequence; // fréquence des mots dans la phrase

    // calcul de la fréquence d'apparition de chaque mot dans la phrase

    void calcul_frequence(){
      for (int i = 0 ; i< nbre_mots; i++) {
           for (int j =0 ; j< nbre_mots; j++){
             if (TAB[i].compareTo(TAB[j])==0) {
                  frequence[i]++;
             }
           }
      }
    }
```

```
// tri
// un peu lourd ..... mais "basic"!

void tri(){

  for (int i =0 ; i<= nbre_mots-2; i++){
      int Ind_Min = i;
      for (int j=i+1 ; j< nbre_mots; j++) {
        // tri sur la frequence ordre decroissant
        if (frequence[j] > frequence[Ind_Min]){
            Ind_Min = j;
        }else {
            // dans le cas fréquences identiques, tri sur les mots,
            // ordre croissant alphabétique
            if ((frequence[j] == frequence[Ind_Min])
              && (TAB[j].compareTo(TAB[Ind_Min])<=0)) {
              Ind_Min = j;
            }
        }
      }
      // operation de swap
      if (Ind_Min != i) {
        String tempo = TAB[i];
        TAB[i] = TAB[Ind_Min];
        TAB[Ind_Min] = tempo;

        int tempo2 = frequence[i];
        frequence[i] = frequence[Ind_Min];
        frequence[Ind_Min] = tempo2;
      }
  }
}


// affichage en sortie …

void affichage(){
  for (int i =0 ; i< nbre_mots; i++){
      if (frequence[i] != 0) {
        System.out.println(TAB[i] + "\t" + frequence[i]);
        for (int j=0 ; j< nbre_mots; j++){
            if (TAB[i].compareTo(TAB[j])==0) {
              frequence[i]=0;
              frequence[j]=0;
            }
        }
      }
  }
}
```

```
// constructeur et calcul

    Stat(String [] MaPhrase) {

      nbre_mots = MaPhrase.length;

      TAB = new String[nbre_mots];

      for (int i=0;i<nbre_mots;i++)
          TAB[i] = MaPhrase[i].toLowerCase();

      frequence = new int[nbre_mots];

      for (int i =0 ; i< nbre_mots; i++) frequence[i] = 0;

      calcul_frequence();

      tri();

      affichage();

    }
}

// pour faire tourner la moulinette …

public class exo5 {

    // fonction principale

    public static void main(String [] args) {

      if (args.length == 0) {
          System.err.println("Vous avez oublié d'inclure\
                              votre phrase sur la ligne de commande!");
          System.err.println("Usage: ...");
          System.err.println("java exo5 votre_phrase");
          System.exit(1);
      }
      Stat Phrase_in = new Stat(args);

      System.exit(0);

    }

}
```