Ecole Nationale des Sciences Appliquées - Al Hoceima

Département Mathématiques et Informatique

Devoir Surveillé N°1

Module : Programmation Orientée Objet en Java

*Première Année Ingénierie des données. Le 07 Avril 2022. 9h-11h (**Durée 02h**)*

Enseignant : Pr. Tarik BOUDAA

Les documents imprimés sont autorisés. L'utilisation du correcteur « Blanco » et du crayon est interdite. Il est interdit d'échanger vos fournitures ou vos documents avec d'autres étudiants. Le barème est donné à titre indicatif, et il est susceptible d'être légèrement modifié. Le sujet se compose de 3 pages et de deux exercices indépendants.

Exercice 1 : 10 points

Répondre aux questions (indépendantes) suivantes :

1. Quel est le résultat de l'exécution du programme ci-dessous :

```
public class Main {
public static void main(String[] args) {
        System.out.println(2*(1011/2001 * 5000/1000  + 3.0 / 2));

        }
}
```

2. On considère la suite d'instructions suivantes :

```
Personne o1 = new Personne ();
Personne o2 = o1;
Personne o3;
Personne o4=new Personne ();
Personne o5=o4;
```

Combien d'instances de la classe Personne sont créés ?

3. Qu'affichera le code suivant ?

```
class C {
        public static int i = 1;
        public static int j;
        public int k = 1;
        public C() {
                i++;
                k++;
                j = i + k;
        }
        public static void main(String[] args) {
         C x = new C();
         C y = new C();
         C z = new C();
         C u = x;
         System.out.println(x.i + u.j + y.j + z.j + z.k + C.j);
        }
}
```

4. Quel est le résultat de l'exécution du programme ci-dessous :

```
public class M {
        public static int c(int i, int j) {
                return i * j + 1;
        }
        public static double c(double i, int j) {
                return i * j + 2;
        }
        public static double c(double i, double j) {
                return 2 * (i - j);
        }
        public static void main(String[] args) {
        System.out.println(M.c(1, M.c(1, 1.5)) * M.c(1.5, 1));
        }
}
```

5. Qu'affiche le programme suivant :

```
public class Prog {
        private static int i;
        public Prog() {
                this(i/2);
        }
        public Prog(int a) {
                i += a;
                System.out.println(i);
        }
        public static void main(String[] args) {
                new Prog(2);
                new Prog(3);
                new Prog();
        }
    }
```

6. Qu'affiche le programme suivant :

```
public class C {
  String test;
  public C(String t) {
   if (t.charAt(0) == t.charAt(t.length() - 1)) {
          test = t.substring(1, 4).toUpperCase();
   } else {
   test = t.replace(t.charAt(0), t.charAt(t.length()-1));
   }
  }
  public static void main(String[] args) {
    C c1 = new C("AbcdA");
    C c2 = new C("aBaA");
    System.out.println(c1.test + c2.test);
   }
}
```

7. Qu'affiche le programme suivant :

```
public class C {
        private int i;
        private int j;
        public C(int a, int b) {
                i = a;
                j = b;
        }
        public void tester() {
                System.out.println("TESTER");
        }
        public static void main(String[] args) {
                C x = null;
                x.tester();
        }
    }
```

8. Qu'affiche le programme suivant :

```
public class C {
    private int i;
    private int j;
    public C(int a, int b) {
      i = a;
      j = b;
     }
     public static void main(String[] args) {
      C x = new C(1, 2);
      C y = new C(1, 2);
      C u = new C(2, 1);
      int a=(x == y || u == y) ? (x.i*y.j):(4*u.i*u.j);
      System.out.println(a);
     }
}
```

9. Qu'affiche le programme suivant :

```
import java.util.ArrayList;
public class Main {
public static void main(String[] args) {
   int[][] tabInt1 = { { 2, 3 }, { 4, 5, 6 }, { 7, 8, 9, 10 } };
    ArrayList<String> l = new ArrayList<String>();
    l.add("A");
    l.add("B");
    l.add("C");
    l.add("D");
    int i = 0;
   for (String it : l) {
        if (i < 3) {
                System.out.println(it + tabInt1[i][i]);
        } else {
        System.out.println(it + tabInt1[2][0]);
        }
        i++;
   } //fin for
  }//fin méthode main
}//fin classe
```

10. Qu'affiche le programme suivant :

```
public class Main {
 public static void main(String[] args) {
  int[][] tabInt1 = { { 1 }, { 2, -1, -2 }, { 1, 3 } };
  int[][] tabInt2 = { { -1 }, { 3, 2, 2 }, { 5, 2 } };
  int[][] tabInt3 = tabInt2;
  for (int i = 0; i < tabInt1.length; i++) {
     for (int j = 0; j < tabInt1[i].length; j++) {
          System.out.println(tabInt3[i][j] / tabInt1[i][j]);
      }//fin for interne
    }//fin for externe
 }//fin méthode main
}//fin classe
```

Exercice 2 : 10 points

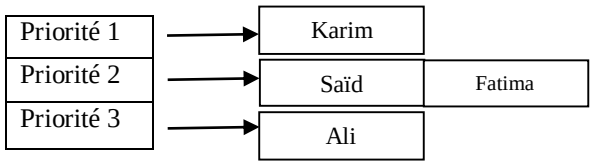
Un patient est caractérisé par son nom, son prénom, son numéro de la carte national CIN et sa priorité qu'est un entier compris entre 1 et 3. Avec 3 est la priorité maximale (3 : Très urgent, 2 : Urgence moyenne, 1 : Peut attendre).

1. Ecrire la classe ***Patient*** qui dispose des éléments suivants :
   - Attributs qui représentent les différentes informations d'un patient (nom, prénom, cin et priorité).
   - Un constructeur adéquat qui permet d'initialiser tous les attributs d'un objet de type Patient.
   - Une méthode *affichePatient* pour afficher toutes les informations d'un patient.
   - Une redéfinition de la méthode *equals* sachant que deux objets de type ***Patient*** sont considérés égaux si et seulement si ils ont le même numéro de la carte national.
   - Une méthode *getPriorite* qui retourne la priorité d'un patient.
2. A l'aide de la classe ***ArrayList***, écrire la classe ***FilePatient*** qui permet de construire une file de patients. Cette classe a les méthodes décrites par leur JavaDoc dans le squelette du code ci-dessous :

```
import java.util.ArrayList;
public class FilePatient {
       private ArrayList<Patient> list =  new ArrayList<Patient> () ;
       /** ajoute un patient à la fin de la file */
       public void enfiler(Patient p){
           //Ecrire le code de cette méthode
       }
```

```
        /** Enlève le patient le plus prioritaire de la file. Si la file est vide la méthode
        retourne false sinon true*/
        public boolean defiler(){
           //Ecrire le code de cette méthode
        }
         /** Affiche le contenu de la file dans l'ordre en commençant par la sortie de la file*/
        public void afficher() {
        //Ecrire le code de cette méthode
        }
    }
```

3. On veut gérer la liste d'attente dans un service d'urgences à l'aide d'une file de priorité de patients présentée par la classe ***FilePatientPrio***. Les patients sont insérés et enlevés de la file selon leur priorité et leur ordre d'arrivée. Si un patient avec priorité k (k=1, 2, ou 3) arrive, il sera rangé en dernier parmi les patients de même priorité. On enlève un patient de la file s'il est le premier arrivé parmi tous ceux ayant la plus grande priorité.
   Exemple : Supposons que 3 patients arrivent dans l'ordre suivant : Ali de priorité 3, Saïd de priorité 2, Fatima de priorité 2 et Karim de priorité 1. L'organisation de la file de priorité constituée de trois sous-files correspondantes à chacune des 3 priorités, sera comme illustrée sur la figure ci-dessous :



   Le patient de priorité maximale donc est Ali dans ce cas. Si on affiche la liste de patients dans l'ordre dans lequel ils seront traités cela devra donner : Ali, Saïd, Fatima, Karim

   Compléter le code de la classe ***FilePatientPrio*** qui devra être implantée à l'aide d'un tableau de 3 cases, chacune avec une sous-file (objet de type ***FilePatient***) de patients pour l'une des 3 priorités. Cette classe a les méthodes décrites par leur JavaDoc dans le squelette du code ci-dessous :

```
public class FilePatientPrio {
       /**
        data[0] contient les patients de priorité 1 (les moins prioritaires).
        data[1] contient les patients de priorité 2.
        data[2] contient les patients de priorité 3 (les  plus prioritaires) */
   private FilePatient[] data = new FilePatient[3];

   /** Constructeur permettant d'initialiser les sous-files */
   public FilePatientPrio() {
          //Ecrire le code du constructeur
   }
   /** Permet d'ajouter un patient dans la file de priorité en prenant en compte sa priorité et son
   * ordre d'arrivée
   * @param p le patient à insérer
   */
   public void enfiler(Patient p) {
          //Ecrire le code de cette méthode
   }
   /** Permet d'enlever le patient le plus prioritaire de la file et retourne true
    * si l'opération s'est effectuée avec succès et false sinon*/
   public boolean defiler() {
          //Ecrire le code de cette méthode
   }
   /** Méthode qui affiche le contenu de la file dans l'ordre de priorité*/
   public void affiche() {
          //Ecrire le code de cette méthode
   }
}
```

4. Un programme principal qui crée les 4 patients de l'exemple, crée un objet de type ***FilePatientPrio***, ajoute ces patients dans cette file de priorité, et enfin affiche le contenu de la file de priorité.