# Examen 2008-2009 - Durée : 2h

POO : Programmation Orientée Objet

Première année

**Tous documents interdits.**

| NOM : | PRÉNOM : |
|---|---|

## ★ Exercice 1. (10 pt) [ *barème dégressif* ]

### ▷ Question 1. (1 pt)

Considérer la classe suivante et indiquer quelle est l'erreur qui empêche la compilation :

```
1  class Frog {
2    int frogCount = 0;
3    public Frog() {
4      frogCount += 1;
5    }
6    public static void main(String[] args){
7      Frog f1 = new Frog();
8      Frog f2 = new Frog();
9      Frog f3 = new Frog();
10     System.out.println("count = " + frogCount);
11   }
12 }
```

Raison de l'erreur :

### ▷ Question 2. (1 pt)

Considérer les classes suivantes et indiquer la ou les réponses correctes :

```
1  class Vehicule {
2    protected int getSpeed() { return -1; }
3  }
4  class Car extends Vehicule {
5    public int getSpeed() { return 60; }
6  }
7  class RaceCar extends Car {
8    public int getSpeed() { return 120; }
9  }
10 class Main {
11   public static void main(String[] args) {
12     Vehicule v = new RaceCar();
13     System.out.println("speed="
14                        +v.getSpeed());
15   }
16 }
```

- ☐ Une ou des erreurs de compilation ou d'exécution sont détectées
- ☐ Aucune erreur n'est détectée et le programme affiche : `speed = -1`
- ☐ Aucune erreur n'est détectée et le programme affiche : `speed = 60`
- ☒ Aucune erreur n'est détectée et le programme affiche : `speed = 120`

### ▷ Question 3. (1 pt)

Considérer les deux classes `Animal` et `Dog` où la méthode `eat()` a été redéfinie (*overriding*). Indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 13 :

```
1  class Animal {
2    public void eat() throws Exception {
3      System.out.println("eat A");
4    }
5  }
6  class Dog extends Animal {
7    public void eat() {
8      System.out.println("eat D");
9    }
10   public static void main(String[] args) {
11     Animal a = new Dog();
12     Dog d = new Dog();
13     /* à remplacer */
14   }
15 }
```

| | Erreur (compilation) | Affiche `eat A` | Affiche `eat D` |
|---|---|---|---|
| `d.eat();` | ☐ | ☐ | ☒ |
| `a.eat();` | ☒ | ☐ | ☒ |

### ▷ Question 4. (1 pt)

Lors de la surcharge d'une méthode (*overloading*) dans une classe et non pas de la redéfinition de celle-ci (*overriding*) dans une sous-classe :

| | PEUT | DOIT | NE DOIT PAS |
|---|---|---|---|
| La nouvelle définition ⌊...⌋ changer la liste des paramètres (type ou nombre de paramètres). | ☐ | ☒ | ☐ |
| La nouvelle définition ⌊...⌋ changer le type de la valeur de retour. | ☒ | ☐ | ☐ |
| La nouvelle définition ⌊...⌋ changer le modificateur d'accès (private, public, protected) | ☒ | ☐ | ☐ |
| La nouvelle définition ⌊...⌋ lancer de nouvelles exceptions. | ☒ | ☐ | ☐ |

▷ **Question 5.** (2 pt)
Parmi les déclarations de classe et d'interface suivantes, indiquer lesquelles sont valides (certaines déclarations seront ré-utilisées par les déclarations qui les suivent) :

(a) ☒ `class Foo { }`
(b) ☐ `class Bar implements Foo { }`
(c) ☒ `interface Baz { }`
(d) ☒ `interface Fi { }`
(e) ☐ `interface Fee implements Baz { }`
(f) ☐ `interface Zee implements Foo { }`
(g) ☒ `class Yow extends Foo implements Fi { }`
(h) ☐ `interface Zoo extends Foo { }`
(i) ☒ `interface Boo extends Fi { }`
(j) ☒ `class Zoom implements Fi, Baz { }`
(k) ☐ `class Toon extends Foo, Zoom { }`
(l) ☒ `interface Vroom extends Fi, Baz { }`
(m) ☒ `class Yow extends Foo implements Fi { }`

▷ **Question 6.** (1 pt)
Considérer la classe suivante et indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 12 :

```
1  class X {
2    void do1() { }
3  }
4  class Y extends X {
5    void do2() { }
6  }
7  class Chrome {
8    public static void main(String[] args) {
9      X x1 = new X();
10     X x2 = new Y();
11     Y y1 = new Y();
12     /* à remplacer */
13   }
14 }
```

| | Erreur (compilation) | Erreur (exécution) | Ok |
|---|---|---|---|
| `x2.do2();` | ☒ | ☐ | ☐ |
| `(Y) x2.do2();` | ☒ | ☐ | ☐ |
| `((Y) x2).do2();` | ☐ | ☐ | ☒ |

▷ **Question 7.** (2 pt)
Considérer la classe suivante et indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 14 :

```
1  class A {
2    void doTheJob() {  }
3  }
4  class B extends A {
5    void doTheJob() {  }
6  }
7  class C {
8    void doTheJob() {  }
9  }
10 class Main {
11   public static void main(String[] args) {
12     A a = new A();
13     B b = new B();
14     /* à remplacer */
15   }
16 }
```

| | Erreur (compilation) | Erreur (exécution) | Ok |
|---|---|---|---|
| `A x = a;` | ☐ | ☐ | ☒ |
| `A x = b;` | ☐ | ☐ | ☒ |
| `B x = a;` | ☒ | ☐ | ☐ |
| `B x = b;` | ☐ | ☐ | ☒ |
| `A x = (A) b;` | ☐ | ☐ | ☒ |
| `B x = (B) a;` | ☐ | ☒ | ☐ |
| `C x = (C) b;` | ☒ | ☐ | ☐ |

## ★ Exercice 2. (10 pt)

### ▷ Question 1. (5 pt)

Considérer les classes suivantes et réaliser des schémas de la mémoire (états de la pile et du tas) lors de l'exécution de la méthode `main()` de la classe principale `mem.Main` aux points identifiés dans le code source.
Vous préciserez l'affichage obtenu sur la sortie standard pour l'ensemble de l'exécution.

mem/Geek.java

```
1  package mem;
2
3  public class Geek {
4    private int qi;
5    private String name;
6
7    public Geek(String n, int iq) {
8      name = n;
9      qi = iq;
10   }
11   public int getQI() {
12         return qi;
13   }
14   public void learn(int p) {
15     qi += p;
16   }
17   public void setName(String n) {
18     name = n;
19   }
20   public Geek clone(int p) {
21     return new Geek(name, qi + p);
22   }
23   public boolean eq(Geek o) {
24     return (name == o.name && qi == o.qi);
25   }
26   public String toString() {
27     return "Geek[" + name + "," + qi + "]";
28   }
29 }
```

mem/GeeksGroup.java

```
1  package mem;
2
3  public class GeeksGroup {
4    private Geek[] geeks = new Geek[10];
5    private int size = 0;
6
7    public void join(Geek t) {
8      geeks[size] = t;
9      size++;
10   }
11   public Geek getGeek(int i) {
12     return geeks[i];
13   }
14   public void study(int iqi) {
15     for (int i = 0; i < size; i++)
16       geeks[i].learn(iqi);
17   }
18   public GeeksGroup duplicate() {
19     GeeksGroup g = new GeeksGroup();
20     g.size = size;
21     for (int i = 0; i < size; i++)
22       g.geeks[i] = geeks[i];
23     return g;
24   }
25   public boolean eq(GeeksGroup g) {
26     for (int i = 0; i < size && i < g.size; i++) {
27       if (geeks[i] != g.geeks[i])
28         return false;
29     }
30     return true;
31   }
32   public String toString() {
33     String s = "Geeks(" + size;
34     for (int i = 0; i < size; i++)
35       s += "," + geeks[i].toString();
36     s += ")";
37     return s;
38   }
39 }
```

mem/Main.java

```
1  package mem;
2
3  public class Main {
4    public static void main(String[] args) {
5      Geek t1 = new Geek("Leonard", 173);
6      Geek t2 = t1.clone(14);
7      t2.setName("Sheldon");
8      GeeksGroup tg = new GeeksGroup();
9      tg.join(t1);
10     tg.join(t2);
11
12     System.out.println("POINT 1");
13     System.out.println(t1);
14     System.out.println(t2);
15     System.out.println(tg);
16     // POINT 1
17
18     GeeksGroup bbt = tg.duplicate();
19
20     System.out.println("POINT 2");
21     System.out.println(bbt.getGeek(0));
22     System.out.println(bbt.getGeek(1));
23     System.out.println(tg);
24     System.out.println(bbt);
25     System.out.println("same: "+(bbt == tg));
26     System.out.println("eq: "+bbt.eq(tg));
27     // POINT 2
28
29
30     Geek t3 = new Geek("Howard",109);
31     tg.join(t3);
32     bbt.join(t3);
33
34     System.out.println("POINT 3");
35     System.out.println(tg);
36     System.out.println(bbt);
37     System.out.println("same: "+(bbt == tg));
38     System.out.println("eq: "+bbt.eq(tg));
39     // POINT3
40
41     Geek t4 = new Geek("Rajesh", 108);
42     tg.join(t4);
43     bbt.join(t4.clone(0));
44
45     System.out.println("POINT 4");
46     System.out.println(tg);
47     System.out.println(bbt);
48     System.out.println("same: "+(bbt == tg));
49     System.out.println("eq: "+bbt.eq(tg));
50     // POINT4
51
52     bbt.study(10);
53     System.out.println("POINT 5");
54     System.out.println(tg);
55     System.out.println(bbt);
56     System.out.println("same: "+(bbt == tg));
57     System.out.println("eq: "+bbt.eq(tg));
58     // POINT5
59   }
60 }
```

**Réponse**

```
POINT 1
Geek[Leonard,173]
Geek[Sheldon,187]
Geeks(2,Geek[Leonard,173],Geek[Sheldon,187])

POINT 2
Geek[Leonard,173]
Geek[Sheldon,187]
Geeks(2,Geek[Leonard,173],Geek[Sheldon,187])
Geeks(2,Geek[Leonard,173],Geek[Sheldon,187])
same: false
eq: true

POINT 3
```

```
Geeks(3,Geek[Leonard,173],Geek[Sheldon,187],Geek[Howard,109])
Geeks(3,Geek[Leonard,173],Geek[Sheldon,187],Geek[Howard,109])
same: false
eq: true

POINT 4
Geeks(4,Geek[Leonard,173],Geek[Sheldon,187],Geek[Howard,109],Geek[Rajesh,108])
Geeks(4,Geek[Leonard,173],Geek[Sheldon,187],Geek[Howard,109],Geek[Rajesh,108])
same: false
eq: false

POINT 5
Geeks(4,Geek[Leonard,183],Geek[Sheldon,197],Geek[Howard,119],Geek[Rajesh,108])
Geeks(4,Geek[Leonard,183],Geek[Sheldon,197],Geek[Howard,119],Geek[Rajesh,118])
same: false
eq: false
```

**Fin réponse**

▷ **Question 2.** (1 pt)
Indiquer 3 raisons faisant que la méthode `eq(Geek o)` de la classe `mem.Geek` n'implémente pas correctement le contrat habituel de la méthode `equals()` définie dans la classe `Object` (méthode que l'on doit généralement rédéfinir dans toute nouvelle classe).

**Réponse**

- le paramètre doit être de type `Object`
- il faut tester le type du paramètre (avec `instanceof` ou `getClass()` en fonction du comportement souhaité)
- une réference `null` passé en paramètre déclenche une `NullPointerException`
- la comparaison de la variable d'instance `name` doit se faire avec la méthode `equals()` et non pas avec l'opérateur `==`.
- une réference `null` stocké dans la variable d'instance `name` risque de déclencher une `NullPointerException`

**Fin réponse**

▷ **Question 3.** (2 pt)
Écrire le code de la méthode `maxQI()` de la classe `mem.GeeksGroup` qui retourne une réference vers un objet de classe `mem.Geek`. L'objet retourné correspond au `Geek` possèdant le Q.I. le plus élevé. Dans le cas où il y aurait plusieurs `Geek` avec le même Q.I. le dernier `Geek` trouvé ayant le Q.I. le plus élevé sera retourné.

```
 public Geek maxQI() {
       int maxQI = 0;
       Geek genius = null;
       for (int i = 0; i < size; i++) {
              Geek g = getGeek(i);
              if (g.getQI() >= maxQI) {
                     maxQI = g.getQI();
                     genius = g;
              }
       }
       return genius;
 }
```

**Fin réponse**

▷ **Question 4.** (2 pt)
L'implémentation actuelle de la classe `mem.GeeksGroup` ne supporte pas l'ajout de plus de 10 membres au sein d'un groupe. Proposer une solution permettant de supprimer cette limitation. Ecriver le code de cette nouvelle implémentation.

**Réponse**

Deux solutions sont possibles :
- Remplacer le tableau statique de type `Geek[]` par une référence vers une collection (`ArrayList`, `Vector`, `ArrayList<Geek>`, `Vector<Geek>`, ...).
- Implémenter soit même un tableau à taille variable en augmentant sa capacité lorsque c'est nécessaire.

Dans tous les cas, il faut modifier les méthodes en conséquence et surtout faire bien attention au `size` utilisée dans les différentes boucles et au code de la méthode `eq()`.

**Fin réponse**