Corrigé du QCM de Programmation 1

Licence d'informatique

Le 01/12/2004

1 Questions générales

Barème : bonne réponse=1pt ; mauvaise réponse=-0.5pt ; pas de réponse=0pt.

1. class B {
   private int x ; private float y ;
   public void setX(int x) {this.x=x ; }
   public void setY(float y) {this.y=y ; }
   }
   Le code ci-dessus est une illustration **de l'encapsulation**.
2. La fonction principale de la JVM consiste à **interpréter le bytecode**.
3. Lequel de ces langages n'est pas orienté objet : **C**. *Par ailleurs, SIMULA est probablement le premier langage OO, tandis que C++ est une extension objet de C.*
4. Toutes les classes de Java héritent de **Object**.
5. Le passage de paramètres en Java est effectué par **valeur**, *même si les variables objet sont des références, et donc leur passage par valeur ressemble à un passage par référence.*
6. Un objet traite un message reçu **en exécutant une méthode d'instance,** *c'est juste une terminologie différente pour dire la même chose*

2 Questions spécifiques

Barème : bonne réponse=3pt ; mauvaise réponse=-1pt ; pas de réponse=0pt

7. Combien d'instances de la classe A sont créées pendant l'exécution du code suivant ? Combien en reste après le passage du *Garbage collector* ?

   ```
   A u,b,c ;
   A a=new A() ;
   b=new A() ; c=b ;
   a=b ;
   ```

   Les deux instances référencées par a (1ère) et b (2ème) sont créées par les deux new. *Ensuite la variable a est réorientée sur la 2ème instance, en laissant la 1ère "orpheline". Par la suite l'instance "orpheline" sera nettoyée par le* Garbage collector. *D'où la réponse* **2 ;1**.
8. Étant donné que la classe Sardine étend la classe Poisson, trouvez une ligne qui passe bien la compilation mais produit une erreur à l'exécution parmi les suivantes
   - Poisson y =new Poisson() ; Sardine x= (Sardine)y ; Poisson z=x ;
     Tout va bien pour le compilateur : une affectation sans transtypage, un downcasting explicite, un upcasting implicite. Par contre, à l'execution la tentative de transtyper un simple Poisson en Sardine donne une erreur. Donc **la première ligne correspond à l'énoncé.**
   - Sardine y =new Sardine() ; Poisson x= y ; Sardine z=(Sardine)x ;
     Tout va bien pour le compilateur : une affectation sans transtypage, un upcasting implicite, un downcasting explicite. A l'execution le downcasting se passe bien, comme l'objet référencé par x est en réalité Sardine. Pas d'erreur.
   - Poisson y =new Sardine() ; Object x= y ; Sardine z=x ;
     La dernière affectation est un downcasting implicite, ce qui est interdit et donne lieu à une erreur à la compilation.
   - Poisson y =new Poisson() ; Sardine z= new Sardine() ; y=z ;
     Pas de downcasting - pas de problèmes.

9. Pour la classe D définie comme suit :

```
class D {
public static int x ;
public int y ;
public static travailler() {x++ ;}
public D() {x++ ; y- - ; }
 }
```

qu'affichera le code suivant ?

```
D.travailler() ; D a=new D() ; D b=new D() ; a.travailler() ;
System.out.println(b.x + " et " + b.y) ;
```

x est ici une variable de classe, elle appartient à toute la classe D et l'écriture b.x signifie D.x. Cette variable est initialisée a 0 au début incrémentée 4 fois (avec 2 appels de travailler et 2 appels du constructeur), donc sa valeur finale est 4. Par contre, y est une variable d'instance, chaque objet de la classe D contient son propre y. La variable b.y est initialisée à 0 et tout de suite decrémentée par le constructeur. Sa valeur finale est -1. D'où la réponse **4 et -1** valant 3 points.

Il y une petite coquille dans cet exercice, j'ai oublié de mettre le type de retour void dans la définition de la fonction travailler. Ceux qui ont choisi la réponse pédantique **ce code ne se compile pas** ont 1 point.

10. Pour les classes Oeuf et Poule définies comme suit :

```
class Oeuf {
public int x ;
public Oeuf() {x=5 ; }
public Oeuf(int y) {x=y ; }
}
```

```
class Poule extends Oeuf {
public Poule() {}
public Poule(int i){this() ; x=x*i ; }
public Poule(String s){super(33) ; x- - ; }
}
```

qu'affichera le code suivant ?

```
Poule b1=new Poule("2004") ; Poule b2 =new Poule(2004) ; Poule b3= new Poule() ;
System.out.println(b1.x + " et " + b2.x + " et encore " + b3.x ) ;
```

Pour obtenir la bonne réponse il suffit de ne pas oublier que le constructeur Poule() contient un appel implicite de Oeuf(). Il faut aussi comprendre que Poule(int i) commence par un this() qui signifie Poule(), et que Poule(int i) commence par un super(33) qui signifie Oeuf(33). En tenant compte de toutes ces considérations on obtient la bonne réponse : **32 et 10020 et encore 5**.

11. Pour les classes A et B définies comme suit :

```
class A {
public int f(int x) {return(x+1) } ;
public static int g(int x) {return (6) ; }
}
```

```
class B extends A {
public int f(int x) {return(x+2) } ;
public static int g( int x) {return (x+4) ; }
}
```

qu'affichera le code suivant ?

```
B b=new B() ; A a =b ;
System.out.println(a.f(2)*a.g(3)) ;
```

La méthode d'instance f est définie dans la classe A et redéfinie dans B. Le choix de la version utilisée s'effectue à l'exécution (liaison tardive) et dépend de la vraie classe de l'objet. Comme la variable a réfère en réalité à un objet de classe B, la JVM utilisera "la f de B". Pour la méthode statique g tout est différent. Cette méthode est associée à une classe (et non à une instance), sa version est choisie statiquement par le compilateur. Bref, a.g() signifie A.g(), c-à-d "la g de A". La bonne réponse sera donc **24**