QCM de POO-2005 corrigé

Section 1. Questions générales bonne réponse=1pt ; mauvaise réponse=-0.5pt ; pas de réponse=0pt.

Bonne réponse=1pt ; mauvaise réponse=-0.5pt ; pas de réponse=0pt.

1. L'interprétation des programmes Java est effectuée par

   (a) API
   (b) JDK
   [(c)] JVM
   (d) AWT

   La machine virtuelle Java (JVM) interprète le bytecode des programmes Java

2. Trouver la phrase qui **n'est pas** une caractérisation correcte de polymorphisme :

   [(a)] le P. est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet.
   (b) le P. signifie que la même opération peut se comporter différemment sur différentes classes de la hiérarchie.
   (c) le P. offre la possibilité à plusieurs objets de natures différentes d'exposer une interface identique au système, et ainsi répondre à un même message d'une manière qui leur est propre
   (d) le P. consiste à autoriser le même code à être utilisé avec différents types, ce qui permet des implémentations plus abstraites et générales.

   La première phrase est en fait une définition de l'encapsulation, les trois autres décrivent divers aspects du polymorphisme

3. Laquelle des opérations ci-dessus est interdite en Java ?

   (a) le upcasting implicite
   (b) le upcasting explicite
   [(c)] le downcasting implicite
   (d) le downcasting explicite

   Le downcasting est une opération risquée et le programmeur doit la demander explicitement

4. Quelle classe n'a pas de classe mère ?

   (a) Orpheline
   (b) String
   [(c)] Object
   (d) une classe abstraite

   La classe Object est l'ancêtre de toutes les autres classes.

5. Qu'est-ce qui est **faux** pour les interfaces ?

   (a) Une I. peut être le type d'une référence
   (b) Une I. déclare des méthodes sans les implémenter
   (c) Une I. peut être implémentée
   [(d)] Une I. peut être instanciée

   L'instanciation est impossible pour les interfaces, tout le reste est autorisé.

Section 2. Questions spécifiques bonne réponse=3pt ; mauvaise réponse=-1pt ; pas de réponse=0pt

1. Pour les classes A et D définies comme suit :

```
class A {
public static int f(int x) {return(x+5); };
public int g(int x) {return (3); }
}
```

```
class D extends A {
public static int f(int x) {return(x+4); };
public int g( int x) {return (x+8); }
}
```

qu'affichera le code suivant ?

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

(a) 18
(b) 21
(c) 66
[(d)] 77

La méthode f() est statique, la variante utilisée est déterminée par la classe de la référence x (c'est à dire A), donc a.f(2)= 2+5. La méthode g() est dynamique, la variante utilisée est déterminée par la vraie classe de l'objet référé par x (dans notre cas D), donc a.g(3)=3+8. D'où la réponse 77.

2. On définit la méthode permuter

```
public static void permuter (String s1, String s2, int x1, int x2){
String tmp1=s1; s1=s2; s2=tmp1;
int tmp2=x1; x1=x2; x2=tmp2; }
```

On l'applique dans le contexte suivant :

```
String a="bon" ; String b="jour" ; int c=3; int d =4;
permuter(a,b,c,d);
```

Quelles seront les valeurs de a,b,c,d après l'exécution de ce code ?

[(a)] "bon", "jour", 3, 4
(b) "jour", "bon", 3, 4
(c) "bon", "jour", 4, 3
(d) "jour", "bon", 4, 3

Avec le passage de paramètres par valeur cette méthode ne change rien.

3. Pour la classe D définie comme suit :

```
class D {
public int x;
public D() {x=3; };
public D( int a){this(); x=x+a ;};
public D( int a, int b){this(b); x= x-a ;}}
```

qu'affichera le code suivant ?

```
D a=new D(5,6);
System.out.println(a.x);
```

(a) 1
(b) 2
(c) 3
[(d)] 4

Il suffit de comprendre que l'appel de constructeur D(5,6) commence par appeler D(6) qui appelle à son tour D().

4. Étant donné que la classe Triangle étend la classe Figure, trouvez une ligne correcte parmi les suivantes

   (a) `Triangle x= new Triangle(); Object y = (Object)x; Triangle z=y;`

   La troisième instruction est un downcasting implicite qui mène à une erreur de compilation.

   (b) `Figure y =new Figure(); Triangle x= (Triangle)y; Figure z=x;`

   La compilation se fait sans erreurs, mais à l'exécution le downcasting explicite de la deuxième instruction produit une ClassCastException, comme une simple Figure référée par y n'est pas un Triangle.

   (c) `Triangle x= new Triangle(); Figure y = x; Triangle z=(Triangle)y;`

   Ici tout va bien : l'objet construit par la première instruction est un Triangle, la deuxième instruction est un upcasting implicite, la troisième instruction est un downcasting explicite qui se déroule bien.

   (d) `Figure y =new Figure(); Triangle x= (Triangle)y; Figure z=(Figure)x;`

   La compilation se fait sans erreurs, mais à l'exécution le downcasting explicite de la deuxième instruction produit une ClassCastException, comme une simple Figure référée par y n'est pas un Triangle.

5. Pour la classe définie comme suit :

   `public class Bidon {int x;};`

   lequel des programmes est faux ?

   (a) `Bidon a=new Bidon(3);`

   (b) `Bidon a=new Bidon(); String s=a.toString();`

   (c) `Bidon a=new Bidon(); boolean b= a.equals("bonjour");`

   (d) `Bidon a=new Bidon(); boolean b= a.equals(a);`

   `new Bidon(3)` fait appel à un constructeur inexistant. Toutes les autres lignes utilisent des méthodes héritées de la classe Object ce qui est toujours possible.