Travaux pratiques d'informatique Nº 13

Le but principal de cette séance est de vous permettre de tester vos connaissances sur l'héritage, le polymorphisme, les tableaux, la classe String, les classes enveloppes (wrapper classes) ainsi que la classe racine Object.

1. Choisir et encercler la (ou les) réponse(s) correcte(s) :

1.1 Soit le programme suivant :

```
1.  public class ThreeConst {
2.  public static void main(String [ ] args) {
3.  new ThreeConst( );
4.  }
5.  public void ThreeConst(int x) {
6.  System.out.print(" " + (x * 2));
7.  }
8.  public void ThreeConst(long x) {
9.  System.out.print(" " + x);
10. }
11. public void ThreeConst( ) {
12. System.out.print("no-arg ");
13. }}
```

Quel est le résultat du programme ci-dessus ? (Choisir une réponse.)

A. no-arg
B. 8 4 no-arg
C. no-arg 8 4
D. Erreur à la compilation
E. Aucun résultat affiché
F. Exception à l'exécution

1.2 Soit la méthode suivante :

```
1.  long test( int x, float y) {
2.
3.  }
```

Quelles instructions, introduites séparément à la ligne 2, produisent une erreur de compilation? (Choisir deux réponses).

A. return x;
B. return (long) x / y;
C. return (long) y;
D. return (int) 3.14d;
E. return (y / x);
F. return x / 7;

1.3 Soit le programme suivant :

```
1.  import java.util.*;
2.  class Ro {
3.  public static void main(String [ ] args) {
4.  Ro r = new Ro( );
5.  Object o = r.test(  );
6.  }
7.  Object test( ) {
8.
9.
10. }}
```

Quels fragments de code, introduits séparément aux lignes 8 et 9, font intervenir le mécanisme d'autoboxing ? (Choisir deux réponses).

A. *return null;*
B. *Object t = new Object( );*
   return t;
C. *int [ ] a = new int [2];*
   return a;
D. *char [ ] [ ] c = new char [2][2];*
   return c[0] [1];
E. *char [ ] [ ] c = new char [2][2];*
   return c[1];
F. *return 7;*

1.4 Soit le programme suivant :

```
1.  class Test {
2.  public static Foo f = new Foo( );
3.  public static Foo f2;
4.  public static Bar b = new Bar( );
5.  public static void main(String [ ] args) {
```

```
6.  for (int x=0; x<6; x++) {
7.  f2 = getFoo(x);
8.  f2.react( );
9.  }}
10. static Foo getFoo(int y) {
11. if ( 0 == y % 2 ) {
12. return f;
13. } else {
14. return b;
15. }}}
16. class Bar extends Foo {
17. void react( ) { System.out.print("Bar "); }
18. }
19. class Foo {
20. void react( ) { System.out.print("Foo "); }
21. }
```

Quel est le résultat du programme ci-dessus ? (Choisir une réponse.)

A. Bar Bar Bar Bar Bar Bar

B. Foo Bar Foo Bar Foo Bar

C. Foo Foo Foo Foo Foo Foo

1.5 Soient les instructions suivantes :

```
1. String x = "xyz";
2. x.toUpperCase( );
3. String y = x.replace('Y', 'y');
4. y = y + "abc";
5. System.out.println(y);
```

Quel est le résultat de l'exécution des instructions ci-dessus ? (Choisir une réponse.)

A. abcXyZ B. abcxyz

C. xyzabc D. XyZabc

E. Erreur à la compilation F. Exception à l'exécution

1.6 Soient les instructions suivantes :

```
1. String x = new String("xyz");   String y = "abc";   x = x + y;
```

Combien d'objets String ont été créés ? (Choisir une réponse.)

A. 2 B. 3

C. 4 D. 5

1.7 Soit le programme suivant :

```
1.  public class WrapTest3 {
2.  public static void main(String [ ] args) {
3.  String s = "98.6";
4.  // introduire le code ici
5. }}
```

Quelles instructions, introduites séparément à la ligne 4, produisent une erreur à la compilation ? (Choisir deux réponses).

A. *float f1 = Float.floatValue(s);*

B. *float f2 = Float.valueOf(s);*

C. *float f3 = new Float(3.14f).floatValue( );*

D. *float f4 = Float.parseFloat(1.23f);*

E. *float f5 = Float.valueOf(s).floatValue( );*

F. *float f6 = (float) Double.parseDouble("3.14");*

1.8 Soit le programme suivant :

```
1.  public class ObjComp {
2.  public static void main(String [ ] args ) {
3.  int result = 0;
4.  ObjComp oc = new ObjComp( );
5.  Object o = oc;
6.
7.  if (o == oc) result = 1;
8.  if (o != oc) result = result + 10;
9.  if (o.equals(oc) ) result = result + 100;
10. if (oc.equals(o) ) result = result + 1000;
11. System.out.println( result );
12. }}
```

Quel est le résultat du programme ci-dessus ? (Choisir une réponse.)

A. 1 B. 10 C. 101

D. 1001 E. 1101

1.9 Indiquer les affirmations vraies. (Choisir deux réponses.)

A. Si x et y sont des références vers des instances des classes enveloppes (wrapper classes) différentes, alors l'appel x.equals(y) va produire une erreur de compilation.

B. Si x et y sont des références vers des instances des classes enveloppes (wrapper classes) différentes, alors *x == y* peut être parfois vrai.

C. Si x et y sont des références de type String et x.equals(y) est vrai, alors *x == y* est toujours vrai.

D. Si x, y et z sont des références vers des instances des classes enveloppes (wrapper classes) et x.equals(y) est vrai et y.equals(z) est vrai, alors z.equals(x) est toujours vrai.

E. Si x et y sont des références de type String et *x == y* est vrai, alors y.equals(x) est toujours vrai.

2. Créer un nouveau projet Eclipse appelé **PrTP13Exo2**, en suivant la démarche suivante :

- depuis le bureau virtuel du CMS, copier les fichiers **UneDate.java**, **DateNaissance.java** et **CP_TP13Exo2.java** dans le package **cms_tp13** du projet ;
- dans les fichiers sources appropriés, compléter les corps de la classe de base **UneDate** et de la classe dérivée **DateNaissance**, en respectant les indications données sous forme de commentaires ;
- anticiper les résultats affichés suite à l'exécution de la classe principale **CP_TP13Exo2** contenant la méthode **main( )** ;
- exécuter le projet et comparer les résultats obtenus avec ceux anticipés.

Indications :

*La classe de base **UneDate** est conçue afin d'instancier des objets encapsulant des dates indiquées par le jour, le mois et l'année.*

*La classe dérivée **DateNaissance** hérite de la classe **UneDate** et réalise une spécialisation des dates par l'encapsulation supplémentaire de l'heure et de la minute, ainsi que d'un nom de personne.*

*La classe mère **UneDate** et la classe fille **DateNaissance** doivent toutes les deux redéfinir les méthodes **equals( )**, **hashCode( )** et **toString( )**.*

*Les définitions des méthodes **equals( )** et **hashCode( )** doivent être cohérentes : si **ref1.equals(ref2)** retourne **true**, alors **ref1.hashCode( )** et **ref2.hashCode( )** doivent retourner la même valeur.*

Remarque > Voici ci-dessous un diagramme UML pour le deuxième exercice de cette série :