Consolidations 2008-2009 - Durée : 2h

POO : Programmation Orientée Objet

Première année

Nancy-Université — Université Henri Poincaré

Nom :	Prénom :

▷ **Question 1.** (1 pt)

Indiquer la/les réponses correcte(s) :

☒ Une classe abstraite (déclarée `abstract`) ne peut pas être instanciée

☐ Le mot clé `final` indique que le corps de la méthode doit se trouver dans une sous-classe

☒ Un champ `static` indique que le champ est commun à toutes les instances de la classe

☐ Une méthode définie `private` est accessible dans toutes les classes appartenant au même package

▷ **Question 2.** (1 pt)

Considérer le code suivant. Quel est le résultat affiché quand le code est compilé et executé ?

```
1  public class test {
2      public static void main(String args[]) {
3        String s1 = "abc";
4        String s2 = new String("abc");
5
6        if(s1 == s2)
7          System.out.print("1 ");
8        else
9          System.out.print("2 ");
10       if(s1.equals(s2))
11         System.out.println("3 ");
12       else
13         System.out.println("4 ");
14     }
15 }
```

☐ 1 3

☐ 1 4

☒ 2 3

☐ 2 4

▷ **Question 3.** (1 pt)

Considérer les deux fichiers suivants. Quel est le résultat ? (indiquer la/les réponses correcte(s)) :

```
1  package pkgA;
2  public class Foo {
3      int a = 5;
4      protected int b = 6;
5      public int c = 7;
6  }
```

```
7   package pkgB;
8   import pkgA.*;
9   public class Baz {
10     public static void main(String[]args) {
11         Foo f = new Foo();
12         System.out.print(" " + f.a);
13         System.out.print(" "+ f.b);
14         System.out.println(" " + f.c);
15     }
16  }
```

☐ 5 6 7

☐ 5 suivi par une exception

☐ Erreur de compilation à la ligne 11

☒ Erreur de compilation à la ligne 12

☒ Erreur de compilation à la ligne 13

☐ Erreur de compilation à la ligne 14

▷ **Question 4.** (1 pt)

Considérer la classe suivante et indiquer quelle est/sont la ou les erreur(s) qui empêche(nt) la compilation :

```
1  class Donut {
2  }
3
4  class DonutFactory {
5    int donutCount = 0;
6    public DonutFactory() {
7    }
8
9    public static Donut cookDonut() {
10     donutCount = donutCount + 1;
11     return new Donut();
12   }
13
14   public static int getDonutCount() {
15     return donutCount;
16   }
17
18   public static void main(String[] args){
19     Donut d1 = DonutFactory.cookDonut();
20     Donut d2 = DonutFactory.cookDonut();
21     System.out.println("count = " +
22                        DonutFactory.getDonutCount());
23   }
24 }
```

Raison(s) de ou des erreur(s) :

```
Test.java :10 : non-static variable donutCount cannot be
referenced from a static context
donutCount = donutCount + 1 ;
Test.java :10 : non-static variable donutCount cannot be
referenced from a static context
donutCount = donutCount + 1 ;
Test.java :15 : non-static variable donutCount cannot be
referenced from a static context
return donutCount ;
3 errors
```

▷ **Question 5.** (1 pt)
Considérer les classes suivantes et indiquer la ou les réponses correctes :

```
1  class Equipment {
2    public int getWeight() { return -1; }
3  }
4  class Tool extends Equipment {
5    public int getWeight() { return 5; }
6  }
7  class Hammer extends Tool {
8    public int getWeight() { return 17; }
9  }
10 class Main {
11   public static void main(String[] args) {
12     Tool t = new Hammer();
13         Equipment e = new Hammer();
14     System.out.println("w1="+t.getWeight()+
15                        " w2="+e.getWeight());
16   }
17 }
```

☐ Des erreurs sont détectées à la compilation
☐ Des erreurs sont détectées à l'exécution
☒ Aucune erreur n'est détectée, le programme affiche : `w1=17 w2=17`
☐ Aucune erreur n'est détectée, le programme affiche : `w1=5 w2=-1`
☐ Aucune erreur n'est détectée, le programme affiche : `w1=-1 w2=-1`
☐ Aucune erreur n'est détectée, le programme affiche un autre résultat que ceux proposés

▷ **Question 6.** (2 pt)
Considérer les deux classes `Animal` et `Cat` où la méthode `speak()` a été redéfinie (*overriding*). Indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 14 :

```
1  class Animal {
2    public void speak() throws Exception {
3      System.out.println("hello!");
4    }
5  }
6  class Cat extends Animal {
7    public void speak() {
8      System.out.println("salut!");
9    }
10   public static void main(String[] args) {
11     Animal a = new Cat();
12     Cat c = new Cat();
13     Animal d = new Animal();
14     /* à remplacer */
15   }
16 }
```

	Erreur (compilation)	Affiche `hello !`	Affiche `salut !`
`d.speak() ;`	☐	☒	☐
`a.speak() ;`	☒	☐	☐
`c.speak() ;`	☐	☐	☒

▷ **Question 7.** (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 { }`
(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 8.** (2 pt)
Considérer la classe suivante et indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 14 :

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

	Erreur (compilation)	Erreur (exécution)	Ok
`Person x = b ;`	☐	☐	☒
`Person x = a ;`	☐	☐	☒
`Student x = b ;`	☐	☐	☒
`Student x = a ;`	☒	☐	☐
`Dog x = (Dog) b ;`	☒	☐	☐
`Student x = (Student) a ;`	☐	☒	☐
`Person x = (Person) b ;`	☐	☐	☒

▷ **Question 9.** (2 pt)
Considérer la classe suivante et indiquer la/les réponses correcte(s) si l'instruction proposée est insérée en ligne 12 :

```
1  class Weapon {
2    void attack() { }
3  }
4  class Gun extends Weapon {
5    void shot() { }
6  }
7  class Armory {
8    public static void main(String[] args) {
9      Weapon w1 = new Weapon();
10     Weapon w2 = new Gun();
11     Gun g1 = new Gun();
12     /* à remplacer */
13   }
14 }
```

	Erreur (compilation)	Erreur (exécution)	Ok
`w2.shot();`	☒	☐	☐
`((Gun) w2).shot();`	☐	☐	☒
`(Gun) w2.shot();`	☒	☐	☐
`(Gun) g1.shot();`	☒	☐	☐
`g1.attack();`	☐	☐	☒

▷ **Question 10.** (1 pt)
Considérer le code suivant et indiquer quelle est l'erreur :

```
1  class C {
2    public int x;
3
4    public static void main(String[] args) {
5      C[] a = new C[10];
6      for (int i=0; i<a.length; i++)
7        a[i].x = i;
8    }
9  }
```

Raison de l'erreur :

les éléments a[i] n'ont pas été initialisés

▷ **Question 11.** (2 pt)
Quel est le résultat généré par le code suivant :

```
1  class Emu {
2    static String s = "->";
3
4    public static void main(String[] args) {
5      try {
6        throw new Exception();
7      } catch (Exception e) {
8        try {
9          try {
10           throw new Exception();
11         } catch (Exception ex) {
12           s += "ici ";
13         }
14         throw new Exception();
15       } catch (Exception x) {
16         s += "bas ";
17       } finally {
18         s += "par ";
19       }
20     } finally {
21       s += "la ";
22     }
23     System.out.println(s);
24   }
25 }
```

- ☐ `->ici la`
- ☐ `->par la`
- ☐ `->bas par`
- ☐ `->ici par la`
- ☒ `->ici bas par la`
- ☐ `->ici bas la par`
- ☐ Erreur de compilation

▷ **Question 12.** (2 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 13.** (2 pt)

Considérer le code suivant et indiquer quelle est le résultat obtenu lors de son exécution.

```
1  class I {
2    int i=3;
3
4    I() {
5      this.i();
6    }
7
8    int i() {
9      return i++;
10   }
11 }
12
13 class II extends I {
14   int i=5;
15
16   int i() {
17     return i+super.i();
18   }
19
20   public static void main(String[] i) {
21     System.out.println(new II().i());
22   }
23 }
```

Résultat lors de l'exécution :

9