

NOM :

PRÉNOM :

▷ **Question 1.** (3 pt)

Indiquer si chaque affirmation est correcte ou non :

Vrai	Faux	
<input type="checkbox"/>	<input type="checkbox"/>	En Java, toutes les classes de base et les autres classes héritent de la classe <code>Object</code>
<input type="checkbox"/>	<input type="checkbox"/>	Une méthode marquée comme <code>private</code> est accessible depuis toutes les autres classes du même paquetage.
<input type="checkbox"/>	<input type="checkbox"/>	L'instruction <code>(new Integer(10)) == (new Integer(10))</code> est évaluée à vraie.
<input type="checkbox"/>	<input type="checkbox"/>	L'instruction <code>(new Float(1.)).equals(new Float(1.))</code> est évaluée à vraie.
<input type="checkbox"/>	<input type="checkbox"/>	Il est toujours possible de définir une sous-classe d'une classe existante.
<input type="checkbox"/>	<input type="checkbox"/>	Dans une classe, on ne peut définir qu'au plus 64 méthodes à deux paramètres de types primitifs portant le même nom.
<input type="checkbox"/>	<input type="checkbox"/>	Pour rendre le code source d'un programme plus facile à maintenir, on cherche toujours à maximiser le couplage entre les classes.
<input type="checkbox"/>	<input type="checkbox"/>	Il est impossible d'instancier une classe abstraite.
<input type="checkbox"/>	<input type="checkbox"/>	Il est interdit de définir un constructeur dans une classe abstraite.
<input type="checkbox"/>	<input type="checkbox"/>	Une variable d'instance déclarée <code>private</code> n'est accessible directement que par cette instance.
<input type="checkbox"/>	<input type="checkbox"/>	On ne peut modifier la valeur d'un objet de type <code>Integer</code> dont la référence est stockée dans une variable d'instance déclarée comme <code>final</code> .
<input type="checkbox"/>	<input type="checkbox"/>	Une classe peut implémenter plusieurs interfaces.

▷ **Question 2.** (1 pt)

Considérer l'extrait de code Java suivant. Que se passe-t-il si `test2()` est évaluée à la valeur `false` ?

```
1 if (test1() && test2() && test3()) {
2   /* corps du bloc if */
3 }
```

- ☐ `test3()` est évaluée, puis le corps du bloc `if` est exécuté.
- ☐ `test3()` est évaluée, puis le corps du bloc `if` n'est pas exécuté comme si l'expression complète était fausse.
- ☐ `test3()` n'est pas évaluée, puis le corps du bloc `if` n'est pas exécuté comme si l'expression complète était fausse.

▷ **Question 3.** (1 pt)

Considérer l'extrait de code Java suivant. Quel sera l'affichage produit par l'exécution de ce code ?

```
1 class Base {
2   void methodA() {
3     System.out.println("base - MethodA");
4   }
5 }
6
7 class Sub extends Base {
8   public void methodA() {
9     System.out.println("sub - MethodA");
10  }
11
12   public void methodB() {
13     System.out.println("sub - MethodB");
14   }
15
16   public static void main(String args[]) {
17     Base b = new Sub();
18     b.methodA();
19     b.methodB();
20   }
21 }
```

- ☐ `sub - MethodA , sub - MethodB.`
- ☐ `base - MethodA , sub - MethodB.`
- ☐ Une erreur à la compilation est détectée ligne 17.
- ☐ Une erreur à la compilation est détectée ligne 18.
- ☐ Une erreur à la compilation est détectée ligne 19.

▷ Question 4. (1 pt)

Considérer l'extrait de code Java suivant :

```
1 String example = "Hi girls!";
2 example.toLowerCase();
3 example.charAt(3);
4 System.out.print(example);
```

Vrai | Faux
☐ | ☐

L'exécution de ce programme affiche : g

▷ Question 5. (1 pt)

Considérer l'extrait de code Java suivant. Quelles sont les deux définitions de méthodes qui peuvent être insérées de manière indépendante à la ligne 9 pour produire un code correct ?

```
1 public abstract class Employee {
2     protected abstract double getSalesAmount();
3     public double getCommision() {
4         return getSalesAmount() * 0.15;
5     }
6 }
7
8 class Sales extends Employee {
9     // insérer votre définition de méthode ici
10 }
```

- ☐ double getSalesAmount() { return 1.45; }.
- ☐ public double getSalesAmount() { return 1.45; }.
- ☐ private double getSalesAmount() { return 1.45; }.
- ☐ protected double getSalesAmount() { return 1.45; }.

▷ Question 6. (1 pt)

Considérer l'extrait de code Java suivant. Quel est le résultat obtenu, si vous essayez de compiler et exécuter ce code ?

```
1 public class A {
2     public int i = 3;
3 }
4
5 public class B {
6     public static void doSomething(int i, A a) {
7         i = 10;
8         a = new A();
9         a.i = 100;
10    }
11 }
12
13 public class C {
14     public static void main(String[] args) {
15         int k = 0;
16         A a = new A();
17         B.doSomething(k, a);
18         System.out.println("k = " + k);
19         System.out.println("a.i = " + a.i);
20    }
21 }
```

- ☐ Une erreur est détectée à la compilation.
- ☐ Une exception est levée à l'exécution.
- ☐ k = 0 et a.i = 3.
- ☐ k = 0 et a.i = 100.
- ☐ k = 10 et a.i = 3.
- ☐ k = 10 et a.i = 100.

▷ Question 7. (1 pt)

Considérer l'extrait de code Java suivant. Quel est le résultat obtenu, si vous essayez de compiler et exécuter ce code ?

```
1 public class Test {
2     private int i;
3
4     public Test() {
5     }
6
7     public Test(int i) {
8         this.i = i;
9     }
10
11     public static void printStatic(Test t){
12         System.out.println(t.i);
13     }
14     public void print(Test t){
15         System.out.println(t.i);
16     }
17
18     public static void main(String[] args) {
19         Test t1 = new Test(3);
20         Test.printStatic(t1);
21         (new Test()).print(t1);
22     }
23 }
```

- ☐ Une erreur est détectée à la compilation
(à cause de l'instruction `Test.printStatic(t1)`).
- ☐ Une erreur est détectée à la compilation
(à cause de l'instruction `(new Test()).print(t1)`).
- ☐ Une exception est levée à l'exécution.
- ☐ La valeur 3 sera affichée deux fois.

▷ Question 8. (3 pt)

a) Corriger le code de la méthode `test()` de la classe `MyClass` ci-dessous pour qu'il ne provoque plus aucune erreur à la compilation. On ne s'intéressera pas aux erreurs pouvant se produire à l'exécution. Vous devriez trouver 4 erreurs. Plusieurs solutions sont possibles, la plus simple consistant à supprimer tout le code. Vos modifications devront donc supprimer le moins de code possible et privilégier le remplacement ou l'ajout d'un mot clé ou d'une variable. Justifier vos modifications.

```

1 class MyClass {
2     void test() {
3         animal Animal = new Canard();
4         if (canard instanceof Canard) {
5             animal.coincoin();
6             animal = new Blaireau();
7         }
8         if (animal instanceof Blaireau) {
9             ((Dindon) animal).glousse();
10            animal = new Animal();
11        }
12        if (animal instanceof Animal) {
13            System.out.println("terminé");
14        }
15    }
16 }
17
18 abstract class Animal {}
19
20 class Blaireau extends Animal {}
21
22 class Canard extends Animal{
23     public void coincoin() {};
24 }
25
26 class Dindon extends Animal{
27     public void glousse() {};
28 }

```

b) En vous basant sur votre version corrigée du code, déterminer quelle(s) est/sont la/les exécution(s) qui affiche(nt) la chaîne de caractères **terminé** si à la ligne 3, on crée un objet du type proposé. Justifier votre réponse

	Affiche terminé	Pas d'affichage	Justification
new Blaireau()	<input type="checkbox"/>	<input type="checkbox"/>	
new Canard()	<input type="checkbox"/>	<input type="checkbox"/>	
new Dindon()	<input type="checkbox"/>	<input type="checkbox"/>	

▷ **Question 9.** (2 pt)

Déterminer l'affichage en sortie standard lors de l'exécution de la méthode `test(int i)`, en fonction de la valeur entière `i` passée en paramètre.

```

1 void test(int i) {
2     System.out.print("A");
3     try {
4         System.out.print("B");
5         try {
6             System.out.print("C");
7             if (i < 0)
8                 throw new MyException1();
9             System.out.print("D");
10            if (i > 0)
11                throw new MyException2();
12            System.out.print("E");
13            if (i == 0)
14                System.out.print("F");
15            System.out.print("G");
16        } catch (MyException1 e) {
17            System.out.print("H");
18        } finally {
19            System.out.print("I");
20        }
21        System.out.print("J");
22    } catch (MyException2 e) {
23        System.out.print("K");
24    } finally {
25        System.out.print("L");
26    }
27    System.out.print("M");
28 }
29
30 class MyException1 extends Exception {};
31 class MyException2 extends Exception {};

```

▷ **Question 10.** (2 pt)

Considérer le code suivant.

```

1 class Foret {
2     void planter(){
3         SapinDeNoel sap1 = new SapinDeNoel(10);
4         System.out.println(sap1);
5         SapinDeNoel sap2 = new SapinDeNoel(4,8);
6         System.out.println(sap2);
7     }
8 }
9
10
11 class Arbre {
12     protected double taille;
13
14     public Arbre(double taille) {
15         this.taille = taille;
16         System.out.print("arbre de "+taille+"m");
17     }
18 }
19
20 class Sapin extends Arbre {
21     public Sapin() {
22         this.taille = 3.20;
23         System.out.print(" de type : sapin");
24     }
25
26     public Sapin(double taille) {
27         super(taille);
28         System.out.print(" de type : sapin");
29     }
30 }
31

```

```

1 class SapinDeNoel extends Sapin {
2     protected int nbBoules;
3
4     public SapinDeNoel(int nb) {
5         this.nbBoules = nb;
6         System.out.print(" de Noël avec "+nbBoules+" boules");
7     }
8
9     public SapinDeNoel(double taille) {
10        super(taille);
11        this.nbBoules = 15;
12        System.out.print(" de Noël avec "+nbBoules+" boules");
13    }
14
15    public SapinDeNoel(double taille,int nb) {
16        this(nb);
17        this.taille = taille;
18    }
19
20    public String toString() {
21        return "ce sapin de Noël mesure "+taille+
22            " mètres et est décoré de "+nbBoules+" boules.";
23    }
24 }

```

a) Préciser l'affichage, si la compilation n'échoue pas, de l'exécution de la méthode `planter()`. Si la compilation échoue, indiquer la cause.

b) Même question en rajoutant le constructeur ci dessous dans la classe `Arbre` :

```

1 public Arbre() {
2     this.taille = 3.50;
3     System.out.print("arbre de " + taille + "m");
4 }

```

c) Même question en supprimant le constructeur `SapinDeNoel(int nb)` de la classe `SapinDeNoel`. L'ajout précédent est conservé.

➤ **Question 11.** (2 pt)

Considérer la classe suivante et indiquer **la/les réponses correcte(s)** si l'instruction proposée est insérée en ligne 30. Si il n'y a pas d'erreur, préciser l'affichage.

```

1 class Astre {
2     protected int id;
3
4     public Astre(int myid) {
5         this.id = myid;
6     }
7     public boolean equals(Astre a) {
8         return (a.id == this.id);
9     }
10 }
11
12 class Planete extends Astre {
13     private String name;
14     public Planete(int myid, String name) {
15         super(myid);
16         this.name = name;
17     }
18     public boolean equals(Planete d) {
19         return (d.id == this.id
20             && this.name.equals(d.name));
21     }
22 }
23
24 class Main {
25     public static void main(String args[]) {
26         boolean b = false;
27         Planete mars = new Planete(29, "Mars");
28         Planete jupiter = new Planete(29, "Jupiter");
29         Astre saturne = new Planete(29, "Saturne");
30         // instruction a inserer
31         System.out.println(b);
32     }
33 }
```

	Erreur	Ok	Affichage
b = mars.equals(jupiter);	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
b = jupiter.equals(mars);	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
b = mars.equals(saturne);	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
b = saturne.equals(mars);	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>

▷ **Question 12.** Schéma mémoire (2 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 `Main` aux points identifiés dans le code source (indiqués par le marqueur `// POINT_? schema mémoire à cet instant`).

Vous préciserez l'affichage obtenu sur la sortie standard à chaque étape.

```

1 interface Expr {
2     int evaluer() ;
3     String decompiler();
4 }
5
6 class Value implements Expr {
7     private int val;
8
9     public Value(int v) {
10         this.val = v;
11     }
12
13     public int evaluer() {
14         return this.val;
15     }
16
17     public String decompiler() {
18         return ""+this.val;
19     }
20 }
21
22 class Add implements Expr {
23     private Expr g;
24     private Expr d;
25
26     public Add(Expr g, Expr d) {
27         this.g = g;
28         this.d = d;
29     }
30
31     public void setG(Expr g) {
32         this.g = g;
33     }
34
35     public void setD(Expr d) {
36         this.d = d;
37     }
38
39     public int evaluer() {
40         return this.g.evaluer() + this.d.evaluer();
41     }
42
43     public String decompiler() {
44         return "("+this.g.decompiler()+
45             ""+this.d.decompiler()+")";
46     }
47 }

```

```

1 class Main {
2     public static void main(String args[]) {
3         Expr v1 = new Value(42);
4         Expr v2 = new Value(17);
5
6         Add a = new Add(v1, v2);
7         // POINT_1 schema mémoire à cet instant
8         System.out.println(v1.decompiler() + " = " + v1.evaluer());
9         System.out.println(v2.decompiler() + " = " + v2.evaluer());
10        System.out.println(a.decompiler() + " = " + a.evaluer());
11
12        v2 = v1;
13        v1 = new Value(13);
14        // POINT_2 schema mémoire à cet instant
15        System.out.println(v1.decompiler() + " = " + v1.evaluer());
16        System.out.println(v2.decompiler() + " = " + v2.evaluer());
17        System.out.println(a.decompiler() + " = " + a.evaluer());
18
19        a.setG(v1);
20        // POINT_3 schema mémoire à cet instant
21        System.out.println(v1.decompiler() + " = " + v1.evaluer());
22        System.out.println(v2.decompiler() + " = " + v2.evaluer());
23        System.out.println(a.decompiler() + " = " + a.evaluer());
24
25        a.setD(a);
26        // POINT_4 schema mémoire à cet instant
27        System.out.println(v1.decompiler() + " = " + v1.evaluer());
28        System.out.println(v2.decompiler() + " = " + v2.evaluer());
29        System.out.println(a.decompiler() + " = " + a.evaluer());
30    }
31 }

```