

NOM :

PRÉNOM :

Question 1. (2 pt)

Indiquer si l'affirmation est correcte ou non :

Vrai	Faux	
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Deux classes peuvent hériter de la même classe mère.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Il est interdit de définir un constructeur dans une classe abstraite (abstract).
<input type="checkbox"/>	<input checked="" type="checkbox"/>	En Java, la méthode qui sera appelée est toujours déterminée à la compilation.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	En Java, seul le type dynamique d'une référence est utilisée pour déterminer la méthode à appeler.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Une méthode privée (private) ne peut pas être abstraite (abstract).
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Une méthode abstraite (abstract) peut être déclarée (final).
<input type="checkbox"/>	<input checked="" type="checkbox"/>	L'opérateur de transtypage (cast) permet de changer le type dynamique d'une référence.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	L'opérateur de transtypage (cast) permet uniquement de changer le type statique d'une référence.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Une classe peut implémenter deux interfaces définissant la même méthode.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	En Java, l'encapsulation est une encapsulation de classe.

Question 2. (1 pt)

Considérer les classes suivantes et indiquer la réponse correcte :

```

1 class Counter {
2     public static int counter = 0;
3
4     public void increment() {
5         counter++;
6     }
7     public int getValue() {
8         return counter;
9     }
10 }
11
12 class TestCounter {
13     public static void main(String args[]) {
14         Counter c1 = new Counter();
15         c1.increment();
16         Counter c2 = new Counter();
17         c2.increment();
18
19         System.out.println(c2.getValue());
20     }
21 }

```

- ☐ Ce code ne compile pas (erreur ligne 2).
- ☐ Ce code ne compile pas (erreur ligne 5).
- ☐ Ce code ne compile pas (erreur lignes 15 et 17).
- ☐ Ce code compile et le programme affiche : 0
- ☐ Ce code compile et le programme affiche : 1
- ☒ Ce code compile et le programme affiche : 2

Question 3. (2 pt)

Considérer les classes suivantes et indiquer les réponses correctes :

```

1 class Except extends Exception {}
2
3 class ErrorMaze {
4     public static void main(String args[]) {
5         int n=0;
6         try {
7             for (n=1; n<4; n++)
8                 f(n);
9         } catch (Except e) {
10            System.out.println("main.catch("+n+")");
11        } finally {
12            System.out.println("main.finally("+n+")");
13        }
14    }
15
16    public static void f(int n) throws Except {
17        try {
18            if (n!=1) throw new Except();
19        } catch (Except e) {
20            System.out.println("f.catch("+n+")");
21            throw e;
22        } finally {
23            System.out.println("f.finally("+n+")");
24        }
25    }
26 }

```

- À la première itération :
 - ☐ il n'y a pas de première itération
 - ☒ le code affiche **f.finally(1)**
 - ☐ le code affiche **f.catch(1)** suivi de **main.finally(1)**
- À la seconde itération :
 - ☐ il n'y a pas de seconde itération
 - ☐ le code affiche **f.finally(2)**
 - ☐ le code affiche **f.catch(2)** suivi de **main.finally(2)** suivi de **f.finally(2)**
 - ☒ le code affiche **f.catch(2)** suivi de **f.finally(2)** suivi de **main.catch(2)** suivi de **main.finally(2)**
 - ☐ le code affiche **f.catch(2)** suivi de **main.catch(2)** suivi de **main.finally(2)** suivi de **f.finally(2)**
 - ☐ le code affiche **f.catch(2)** suivi de **main.catch(2)** suivi de **f.finally(2)** suivi de **main.finally(2)**
- À la troisième itération :
 - ☒ il n'y a pas de troisième itération
 - ☐ le code affiche la même chose que pour la seconde itération mais avec la valeur 3 (au lieu de 2);

▷ **Question 4.** (1 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 Animal {
2     protected long uniqueId;
3
4     public Animal(long myid) {
5         this.uniqueId = myid;
6     }
7     public boolean equals(Animal a) {
8         return (a.uniqueId == this.uniqueId);
9     }
10 }
11
12 class Duck extends Animal {
13     private String name;
14     public Duck(long myid, String name) {
15         super(myid);
16         this.name = name;
17     }
18     public boolean equals(Duck d) {
19         return (d.uniqueId == this.uniqueId
20             && this.name.equals(d.name));
21     }
22 }
23
24 class Main {
25     public static void main(String args[]) {
26         boolean flag = false;
27         Duck donald = new Duck(1L, "Donald");
28         Duck cloneA = new Duck(1L, "CloneA");
29         Animal cloneB = new Duck(1L, "CloneB");
30         // instruction à insérer
31         System.out.println(flag);
32     }
33 }

```

	Erreur	Ok	Affichage
flag = donald.equals(cloneA);	<input type="checkbox"/>	<input checked="" type="checkbox"/>	false
flag = cloneA.equals(donald);	<input type="checkbox"/>	<input checked="" type="checkbox"/>	false
flag = donald.equals(cloneB);	<input type="checkbox"/>	<input checked="" type="checkbox"/>	true
flag = cloneB.equals(donald);	<input type="checkbox"/>	<input checked="" type="checkbox"/>	true

▷ **Question 5.** (1 pt)

Indiquez les concepts mis en œuvre dans le sujet de TP portant sur la course (Race, Team, Racer, etc.)

Vrai	Faux	
<input checked="" type="checkbox"/>	<input type="checkbox"/>	l'encapsulation.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	l'héritage.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	le polymorphisme.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	la délégation.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	la généricité.

▷ **Question 6.** (1 pt)

Considérer les classes suivantes et indiquer **la réponse correcte. Justifier votre réponse.**

```

1 import java.util.ArrayList;
2
3 class TestArray {
4     public static void main(String args[]) {
5         ArrayList<Object> names = new ArrayList<Object>();
6         names.add("coin");
7         names.add("coin");
8         printValues(names);
9     }
10
11     public static void printValues(ArrayList<String> tab) {
12         for (int i=0; i<tab.size(); i++)
13             System.out.print(tab.get(i));
14     }
15 }

```

- ☒ Ce code ne compile pas (erreur à la ligne 8).
☐ Ce code compile et le programme affiche : coincoin
☐ Ce code compile et le programme affiche : coin

Justification :

Appel impossible, car la classe `ArrayList<String>` n'est pas une sous-classe de `ArrayList<Object>`

▷ **Question 7.** Tableaux à taille fixe (2 pt)

En Java, il n'existe pas à proprement parlé de tableaux à n dimensions. Un tableau à 2 dimensions est modélisé par un tableau dont chaque élément est un autre tableau. Cependant, rien ne garanti que chaque "tableau élément" a la même dimension. Complétez la méthode suivante qui indiquera si un tableau à deux dimensions est "régulier", autrement dit si tous les tableaux éléments ont la même longueur. Vous traiterez avec soin le cas où les références sont `null` et vous retournerez le résultat le plus tôt possible.

```

1 public static boolean isRegular(Object[] [] t) {
2     // à compléter
3 }

```

Réponse

```

1 public static boolean isRegular(Object[] t) {
2     if (t[0] == null)
3         return false;
4     int n = t[0].length;
5     for (int i=0; i<t.length; i++)
6         if (t[i] != null && t[i].length != n)
7             return false;
8     return true;
9 }

```

Fin réponse

▷ Question 8. (2 pt)

Le programme suivant est constitué d'une seule classe. Récrivez ce programme en utilisant l'héritage, le polymorphisme et la liaison dynamique.

Mess.java

```

1 import java.util.List;
2 import java.util.ArrayList;
3
4 class FormeGeometrique {
5     public static final int NON_DEFINI = 0;
6     public static final int RECTANGLE = 1;
7     public static final int CERCLE = 2;
8     public static final int TRIANGLE = 3;
9     public static final int COMPOSEE = 4;
10
11     private int type = NON_DEFINI;
12
13     private double a;
14     private double b;
15     private double c;
16
17     private List<FormeGeometrique> formes;
18
19     public FormeGeometrique() {
20         this.type = COMPOSEE;
21         this.formes = new ArrayList<FormeGeometrique>();
22     }
23
24     public FormeGeometrique(double r) {
25         this.type = CERCLE;
26         this.a = r;
27     }
28
29     public FormeGeometrique(double l, double ll) {
30         this.type = RECTANGLE;
31         this.a = l;
32         this.b = ll;
33     }
34
35     public FormeGeometrique(double c1, double c2, double c3) {
36         this.type = TRIANGLE;
37         this.a = c1;
38         this.b = c2;
39         this.c = c3;
40     }
41
42     public void add(FormeGeometrique f) {
43         this.formes.add(f);
44     }
45
46     public double calculerPerimetre() {
47         switch (this.type) {
48             case RECTANGLE:
49                 return 2.*a+2.*b;
50             case CERCLE:
51                 return 2.*Math.PI*a;
52             case TRIANGLE:
53                 return a+b+c;
54             case COMPOSEE:
55                 double s = 0.;
56                 for (FormeGeometrique f : formes)
57                     s += f.calculerPerimetre();
58                 return s;
59             default:
60                 return 0.;
61         }
62     }
63
64     public double calculerAire() {
65         switch (this.type) {
66             case RECTANGLE:
67                 return a*b;
68             case CERCLE:
69                 return Math.PI*a*a;
70             case TRIANGLE:
71                 return 1./4.*Math.sqrt((a+b+c)*(-a+b+c)*(a-b+c)*(a+b-c));
72             case COMPOSEE:
73                 double s = 0.;
74                 for (FormeGeometrique f : formes)
75                     s += f.calculerAire();
76                 return s;
77             default:
78                 return 0.;
79         }
80     }
81 }
82

```

```
83 class DirtyProgram {  
84     public static void main(String args[]) {  
85         FormeGeometrique f1 = new FormeGeometrique(10,10);  
86         FormeGeometrique f2 = new FormeGeometrique(10,15,17);  
87         FormeGeometrique f3 = new FormeGeometrique();  
88         f3.add(f1);  
89         f3.add(f2);  
90         System.out.println("perimetre = "+f3.calculerPerimetre());  
91     }  
92 }
```

▷ **Question 9.** (1 pt)

Quel est l'intérêt de réaliser une telle transformation. Justifiez votre réponse.

Il est plus facile de maintenir le programme (rajouter des types d'objets, corriger des erreurs, etc.). On pourrait factoriser des comportements

▷ **Question 10.** (1 pt)

Quelle est la différence conceptuelle entre une interface et une classe abstraite ?

L'interface définit un contrat (permet d'avoir différente implémentation). La classe abstraite sert à factoriser du code tout en interdisant l'instanciation (usage typique dans le cadre d'un patron *template method*).

▷ **Question 11.** (1 pt)

Pourquoi Java autorise-t-il la réalisation multiple d'interface et non pas l'héritage multiple ?

Il n'y a pas de problème de conflit dans les interfaces alors que ce n'est pas le cas avec des classes (quelles méthodes appelées en cas de conflit ?)

▷ **Question 12.** (2 pt)

Considérer la classe suivante et indiquer **la réponse correcte** si l'instructions proposée est insérée en ligne 11.

Chose.java

```

1 class Machin {}
2
3 class Bidule {}
4
5 class Truc extends Machin {}
6
7 class Chose {
8     public static void main(String args[]) {
9         Object o = new Truc();
10        Machin a = new Machin();
11        // instruction a insérer
12    }
13 }
```

```

Truc t = (Truc) o;
Machin m = (Machin) o;
Bidule b = (Bidule) o;
Truc tr = (Truc) a;
Machin m2 = (Machin) a;
Bidule b2 = (Bidule) a;
```

	Erreur (compilation)	Erreur (exécution)	Ok
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

▷ **Question 13.** Schéma mémoire (3 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 `PokeWorld` aux points identifiés dans le code source (indiqués par le marqueur `// POINT_? memory schema at this point`).

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

PokeWorld.java

```

1 class PokeWorld {
2     public static void main(String args[]) {
3         PokeBall ball = new PokeBall();
4
5         Pokemon pikachu = new Mouse();
6         Pokemon pikachu2 = pikachu;
7         ball.catchPokemon(pikachu);
8         ball.catchPokemon(pikachu2);
9         System.out.println(ball);
10        System.out.println("pikachu == pikachu2 : "+(pikachu == pikachu2));
11        System.out.println("pikachu.equals(pikachu2) : "+(pikachu.equals(pikachu2)));
12        // POINT_1 -- draw memory schema at this point
13
14        Pokemon raichu = pikachu2.evolve();
15        ball.catchPokemon(raichu);
16        System.out.println(ball);
17        System.out.println("pikachu == pikachu2 : "+(pikachu == pikachu2));
18        System.out.println("pikachu == raichu : "+(pikachu == raichu));
19        System.out.println("pikachu2 == raichu : "+(pikachu2 == raichu));
20        System.out.println("pikachu.equals(pikachu2) : "+(pikachu.equals(pikachu2)));
21        System.out.println("pikachu.equals(raichu) : "+(pikachu.equals(raichu)));
22        System.out.println("pikachu2.equals(raichu) : "+(pikachu2.equals(raichu)));
23        // POINT_2 -- draw memory schema at this point
24
25        pikachu2 = raichu.regress();
26        System.out.println(ball);
27        System.out.println("pikachu == pikachu2 : "+(pikachu == pikachu2));
28        System.out.println("pikachu == raichu : "+(pikachu == raichu));
29        System.out.println("pikachu2 == raichu : "+(pikachu2 == raichu));
30        System.out.println("pikachu.equals(pikachu2) : "+(pikachu.equals(pikachu2)));
31        System.out.println("pikachu.equals(raichu) : "+(pikachu.equals(raichu)));
32        System.out.println("pikachu2.equals(raichu) : "+(pikachu2.equals(raichu)));
33        // POINT_3 -- draw memory schema at this point
34    }
35 }

```

Pokemon.java

```

1 import java.util.List;
2 import java.util.ArrayList;
3
4 abstract class Pokemon {
5     protected double size;
6     protected double weight;
7     protected String type;
8
9     public abstract Pokemon evolve() ;
10    public abstract Pokemon regress() ;
11
12    public String toString() {
13        return "[s:"+this.size+" w:"+this.weight+"]";
14    }
15 }
16
17 class PokeBall {
18     private List<Pokemon> pokemons;
19
20     public PokeBall() {
21         this.pokemons = new ArrayList<Pokemon>();
22     }
23
24     public void catchPokemon(Pokemon p) {
25         this.pokemons.add(p);
26     }
27
28     public String toString() {
29         String s = "";
30         for (Pokemon p : this.pokemons) {
31             s = s + p.toString() + ";";
32         }
33         return s;
34     }
35 }

```

Mouse.java

```

1 class Mouse extends Pokemon {
2     public Mouse() {
3         this.size = 0.4;
4         this.weight = 6.0;
5         this.type = "Mouse";
6     }
7
8     public Pokemon evolve() {
9         return new SuperMouse();
10    }
11
12    public Pokemon regress() {
13        return null;
14    }
15
16    public boolean equals(Object o) {
17        if (!(o instanceof Mouse))
18            return false;
19        Mouse m = (Mouse) o;
20        return (m.size == this.size && m.weight == this.weight
21            && m.type.equals(this.type));
22    }
23 }
24
25 class SuperMouse extends Mouse {
26     public SuperMouse() {
27         this.size = 0.8;
28         this.weight = 30.0;
29         this.type = "SuperMouse";
30     }
31
32    public Pokemon evolve() {
33        return this;
34    }
35
36    public Pokemon regress() {
37        this.size -= 0.4;
38        this.weight -= 24.0;
39        this.type = "Mouse";
40        return this;
41    }
42 }

```

Réponse

```

// POINT 1
[s:0.4 w:6.0];[s:0.4 w:6.0];
pikachu == pikachu2 : true
pikachu.equals(pikachu2) : true

```

```
// POINT 2
[s:0.4 w:6.0];[s:0.4 w:6.0];[s:0.8 w:30.0];
pikachu == pikachu2 : true
pikachu == raichu : false
pikachu2 == raichu : false
pikachu.equals(pikachu2) : true
pikachu.equals(raichu) : false
pikachu2.equals(raichu) : false

// POINT 3
[s:0.4 w:6.0];[s:0.4 w:6.0];[s:0.4 w:6.0];
pikachu == pikachu2 : false
pikachu == raichu : false
pikachu2 == raichu : true
pikachu.equals(pikachu2) : true
pikachu.equals(raichu) : true
pikachu2.equals(raichu) : true
```

Fin réponse
