

Seule une feuille recto-verso d'aide mémoire manuscrite est autorisée.

NOM :

PRÉNOM :

► **Question 1.** Lesquels des exemples ci-dessous définissent une classe Java qui compile sans erreur ?

☐

```
1 class EvilCode {
2   int[] tab;
3   static void f(int n) {
4     tab = new int[n];
5   }
6 }
```

☐

```
1 class EvilCode {
2   abstract void dizzy() ;
3 }
```

☒

```
1 package telecomnancy;
2 import java.util.List;
3 class EvilCode {
4   void call() {}
5   List<Integer> numbers;
6 }
```

☒

```
1 package telecomnancy;
2 class EvilCode {
3   void cry() {}
4   Object[] t = new String[10];
5 }
```

☐

```
1 package eu.telecomnancy.poo;
2 class EvilCode {
3   boolean cast() {
4     return 0;
5   }
6 }
```

☒

```
1 package telecomnancy;
2 class EvilCode {
3   Object count = "9";
4   void compute() {}
5 }
```

► **Question 2.** En considérant le code suivant, indiquer les affirmations correctes lors de l'exécution de la méthode `arrangeFlowers()`.

```
1 class Flower {
2   public static void fragrance() {
3     System.out.println("Flower");
4   }
5 }
6 class Camellia extends Flower {
7   public static void fragrance() {
8     System.out.println("Camellia");
9   }
10 }
11 class Bouquet {
12   public void arrangeFlowers() {
13     Flower f1 = new Camellia();
14     f1.fragrance();
15   }
16 }
```

- ☐ Le type dynamique de `f1` est `Flower`.
- ☒ Le type dynamique de `f1` est `Camellia`.
- ☒ Le type statique de `f1` est `Flower`.
- ☐ Le type statique de `f1` est `Camellia`.
- ☒ L'affichage produit est : `Flower`.
- ☐ L'affichage produit est : `Camellia`.

► **Question 3.** En considérant la signature de méthode `public int compute(int precision, List<Hypothese> h)`, indiquer les définitions qui permettent de *surcharger* correctement cette méthode.

- ☒ `public int compute(int precision, Hypothese[] h)`
- ☐ `public int compute(int digits, List<Hypothese> h)`
- ☐ `private int compute(int precision, List<Hypothese> h)`
- ☐ `public boolean compute(int precision, List<Hypothese> h)`
- ☒ `public int compute(int precision, List<Hypothese> h, int level)`
- ☒ `public int compute(List<Hypothese> h, int precision)`
- ☐ `public int calculate(int precision, List<Hypothese> h)`

► **Question 4.** En considérant le code suivant, indiquer quelle(s) est/sont la/les déclaration(s) correcte(s) qui peut/peuvent être insérée(s) à la place du marqueur `// INSERT CODE HERE`.

```
1 interface Hackable {
2   void hack();
3 }
4 class Laptop {
5   public void hack() { System.out.println("Laptop hacked"); }
6 }
7 class Smartphone implements Hackable {
8   public void hack() { System.out.println("Smartphone hacked"); }
9 }
10 class Test {
11   // INSERT CODE HERE
12   {
13     Hackable o = new SmartPhone();
14     o.hack();
15   }
16 }
```

- ☒ `void tryToHack()`
- ☐ `void tryToHack(Hackable o)`
- ☐ `void tryToHack(Laptop o)`
- ☐ `void tryToHack(Smartphone o)`

► **Question 5.** En considérant le code suivant, indiquer quelle(s) instruction(s) insérée(s) individuellement à la place du marqueur `/* INSERT CODE HERE */` permet(tent) d'afficher la valeur de la variable `power`.

```
1 class Robot {
2     public int power;
3 }
4 class SolarRobot extends Robot {
5     int solarCellCount;
6     public static void main(String args[]) {
7         SolarRobot myRobot = new SolarRobot();
8         System.out.println(/* INSERT CODE HERE */);
9     }
10 }
```

- ☒ `myRobot.power`
- ☐ `Robot.power`
- ☐ `(SolarRobot) myRobot.power`
- ☒ `((SolarRobot) myRobot).power`
- ☒ `((Robot) myRobot).power`

► **Question 6.** Indiquer l'affichage produit par l'exécution du code suivant.

```
1 class Plant {
2     void watering() {
3         System.out.println("Watering a plant");
4     }
5 }
6 class Cactus extends Plant {
7     static void watering() {
8         System.out.println("Watering a cactus");
9     }
10 }
11 class TestWatering {
12     public static void main(String[] args) {
13         Plant plant = new Plant();
14         Plant cactus = new Cactus();
15         plant.watering();
16         cactus.watering();
17     }
18 }
```

- ☐ Watering a plant
- ☐ Watering a plant
- ☐ Watering a plant
- ☐ Watering a cactus
- ☐ Watering a cactus
- ☐ Watering a plant
- ☒ Erreur à la compilation

► **Question 7.** Considérant les définitions de la classe `SuperHero` et de l'interface `CanFly`, la tâche est de déclarer une classe `Batman` qui hérite de la classe `SuperHero` et qui réalise l'interface `CanFly`. Indiquer la/les définition(s) correcte(s).

```
1 class SuperHero {}
2 interface CanFly {}
```

- ☐ `class Batman extends SuperHero, CanFly {}`
- ☐ `class Batman implements SuperHero, CanFly {}`
- ☐ `class Batman implements SuperHero extends CanFly {}`
- ☒ `class Batman extends SuperHero implements CanFly {}`

► **Question 8.** En considérant les classes définies ci-dessous, indiquer quelle(s) instruction(s) de code peut/peuvent être insérée(s) individuellement à la place du marqueur `//INSERT CODE HERE` afin que l'exécution du code produise l'affichage suivant :

```
Student says good lecture!
Student says good lecture!
```

```
1 class Student {
2     void print() {
3         System.out.println("Student says good lecture!");
4     }
5 }
6 class Gamer extends Student {
7     void print() {
8         System.out.println("Gamer says good game!");
9     }
10 }
11 class MyApp {
12     Student a = new Student();
13     // INSERT CODE HERE
14     a.print();
15     b.print();
16 }
```

- ☒ `Student b = new Student();`
- ☐ `Student b = new Gamer();`
- ☐ `Student b = ((Student) new Gamer());`
- ☐ `Student b = ((Gamer) new Student());`
- ☐ `Gamer b = new Gamer();`
- ☐ `Gamer b = new Student();`

► **Question 9.** Indiquer l'affichage produit par l'exécution du code suivant.

```

1 class DVD {
2     String title;
3
4     DVD(String t) {
5         title = t;
6     }
7 }
8 class BlueRay extends DVD {
9     BlueRay(String t) {
10        super(t);
11    }
12
13    public boolean equals(BlueRay br) {
14        return title.equals(br.title);
15    }
16 }
17 class TestEquals {
18     public static void main(String[] args) {
19         String name = "Game of Thrones";
20         DVD disc1 = new BlueRay(name);
21         BlueRay disc2 = new BlueRay(name);
22
23         System.out.print(disc1.equals(disc2) + ":");
24         System.out.print(disc2.equals(disc1) + ":");
25         System.out.print(disc1 == disc2);
26     }
27 }

```

- ☐ true:true:true
 - ☐ true:true:false
 - ☒ false:false:false
 - ☐ false:false:true
 - ☐ true:false:true
 - ☐ false:true:false
 - ☐ Erreur à la compilation.
- Aucune méthode equals n'est définie dans la classe DVD.

► **Question 10.** En considérant le code suivant, indiquer quelle instruction insérée à la place du marqueur `/* INSERT CODE HERE */` permettrait à la classe `NightsWatch` de déterminer si la variable `commander` référence un objet de la classe `JonSnow` et d'afficher "You know nothing" dans ce cas.

```

1 class Stark { }
2 class JonSnow extends Stark { }
3
4 class NightsWatch {
5     public static void main(String args[]) {
6         Stark commander = new JonSnow();
7         /* INSERT CODE HERE */
8         System.out.println("You know nothing");
9     }
10 }

```

- ☐ if (commander instanceof JonSnow))
- ☐ if (commander instanceof JonSnow)
- ☐ if (commander instanceof JonSnow))
- ☒ if (commander instanceof JonSnow)
- ☐ if (commander.getType() == JonSnow)

► **Question 11.** Indiquer l'affichage produit par l'exécution de la méthode `main` de la classe `TestSensor`.

```

1 class Sensor {
2     static double temperature;
3
4     void increase() {
5         temperature = temperature + 1;
6     }
7 }
8 class TestSensor {
9     public static void main(String args[]) {
10        Sensor s1 = new Sensor();
11        Sensor s2 = new Sensor();
12        s1.temperature = 15.;
13        s2.temperature = 21.;
14        s1.increase();
15        s2.increase();
16
17        System.out.println((s1.temperature + s2.temperature));
18    }
19 }

```

- ☐ L'exécution de `TestSensor` affiche 32.
- ☐ L'exécution de `TestSensor` affiche 34.
- ☐ L'exécution de `TestSensor` affiche 36.
- ☐ L'exécution de `TestSensor` affiche 38.
- ☐ L'exécution de `TestSensor` affiche 40.
- ☐ L'exécution de `TestSensor` affiche 42.
- ☒ L'exécution de `TestSensor` affiche 46.
- ☐ La classe `TestSensor` ne compile pas.

► **Question 12.** Considérer l'extrait de code Java suivant :

```
SpeedLimit.java
1 class BrokenEngineException extends Exception {
2     BrokenEngineException(String msg) {
3         super(msg);
4     }
5 }
6
7 class SpeedLimitException extends Exception {
8     SpeedLimitException(String msg) {
9         super(msg);
10    }
11 }
12
13 public class SpeedLimit {
14
15     public static void main(String args[]) throws Exception {
16
17         int speedLimit = 50;
18
19         try {
20             System.out.println("Entering the try block.");
21             if (speedLimit > 220) {
22                 throw new BrokenEngineException("Your engine is broken");
23             } if (speedLimit > 130) {
24                 throw new SpeedLimitException("Speed limit violation.");
25             }
26             System.out.println("Exiting the try block.");
27         } catch (SpeedLimitException e) {
28             System.out.println("Exception: " + e.getMessage());
29         } finally {
30             System.out.println("Inside finally block.");
31         }
32
33         System.out.println("After the catch block.");
34     }
35 }
36 }
```

Déterminer l'affichage de la méthode principale void main(String args[]).

Entering the try block.
Exiting the try block.
Inside the finally block.
After the catch block.

Quel serait l'affichage si la valeur de la variable speedLimit était 150 ?

Entering the try block.
Exception : Speed limit violation.
Inside the finally block.
After the catch block.

Quel serait l'affichage si la valeur de la variable speedLimit était 230 ?

Entering the try block.
Inside the finally block.
Exception in thread "main" BrokenEngineException :
Your engine is broken at SpeedLimit.main(SpeedLimit.java :24)

► **Question 13.** Considérer l'extrait de code Java suivant :

Constructors.java

```

1 class Animal {
2     Animal() {
3         System.out.println("cons de Animal");
4     }
5 }
6
7 class Bovide extends Animal {
8     Bovide() {
9         System.out.println("cons de Bovide");
10    }
11
12    Bovide(int x) {
13        this();
14        System.out.println("autre cons de Bovide");
15    }
16 }
17
18 class Vache extends Bovide {
19     Vache() {
20         super(3);
21         System.out.println("cons de Vache");
22     }
23
24     public static void main(String[] arg) {
25         new Vache();
26     }
27 }
```

Déterminer l'affichage de la méthode principale `void main(String args[])`.

cons de Animal
cons de Bovide
autre cons de Bovide
cons de Vache

Même question en supposant que l'on supprime l'instruction `this()` dans la classe `Bovide`

cons de Animal
autre cons de Bovide
cons de Vache

Même question en supposant que l'on supprime également l'instruction `super(3)` de la classe `Vache`

cons de Animal
cons de Bovide
cons de Vache

Que se passe-t-il si l'on supprime si l'on supprime également la définition du constructeur `Bovide()` définit aux lignes 8-10 de la classe `Bovide`.

erreur de compilation.

▷ **Question 14.** Expliquer le principe d'*encapsulation* en programmation orientée-objet. Illustrer votre propos par un exemple de code.

CORRECTION

▷ **Question 15.** Il n'est pas possible dans une même classe de définir deux méthodes dont les profiles seraient `void method(List<Integer> values)` et `void method(List<Object> values)`. Pourquoi? Donner une explication précise.

CORRECTION

▷ **Question 16.** Expliquer à quoi peuvent servir les interfaces.

CORRECTION

▷ **Question 17.** Expliquer la notion de *polymorphisme*. Illustrer votre propos par un exemple de code.

CORRECTION