

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

NOM :

PRÉNOM :

► **Question 1.** Lequel des exemples ci-dessous définit de manière correcte une classe Java ? (1 réponse)

☐

```
1 #connect java compiler;
2 #connect java virtual machine;
3 class EvilCode {}
```

☐

```
1 import javavirtualmachine.*;
2 package javacompiler;
3 class EvilCode {
4     void method1() {}
5     int count;
6 }
```

☐

```
1 #
2 package javacompiler;
3 $import javavirtualmachine;
4 class EvilCode {
5     void method1() {}
6     int count;
7 }
```

☐

```
1 package javacompiler;
2 import javavirtualmachine;
3 Class EvilCode {
4     void method1() {}
5     int count;
6 }
```

☒

```
1 package javacompiler;
2 import javavirtualmachine.*;
3 class EvilCode {
4     void method1() {}
5     int count;
6 }
```

☐

```
1 package java compiler;
2 import java virtual machine;
3 class EvilCode {}
```

► **Question 2.** Indiquer les règles à respecter lors de la définition d'une classe afin de préserver le principe d'*encapsulation* (2 réponses).

- ☒ Définir les variables d'instance comme membres privés.
- ☒ Définir des méthodes publiques pour accéder et modifier les variables d'instances.
- ☐ Définir certaines variables d'instance comme membres publiques.
- ☐ Aucune des règles précédente.

► **Question 3.** En considérant la définition suivante de la classe *Person*, indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```
1 class Person {
2     public String name;
3     public int height;
4 }
```

```
1 class MyApp {
2     public static void main(String args[]) {
3         Person p = new Person();
4         p.name = "Jon";
5         anotherMethod(p);
6         System.out.println(p.name);
7         someMethod(p);
8         System.out.println(p.name);
9     }
10    static void someMethod(Person p) {
11        p.name = "someMethod";
12        System.out.println(p.name);
13    }
14    static void anotherMethod(Person p) {
15        p = new Person();
16        p.name = "anotherMethod";
17        System.out.println(p.name);
18    }
19 }
```

☐

```
anotherMethod
anotherMethod
someMethod
someMethod
```

☒

```
anotherMethod
Jon
someMethod
someMethod
```

☐

```
anotherMethod
Jon
someMethod
Jon
```

☐ Erreur à la compilation.

► **Question 4.** En considérant la signature de méthode `public String mm(int age, String name, double duration)`, indiquer les définitions qui permettent de *surcharger* correctement cette méthode (5 réponses).

- ☐ `private String mm(int val, String firstName, double dur)`
- ☐ `public void mm(int val1, String val2, double val3)`
- ☒ `String mm(String name, int age, double duration)`
- ☒ `float mm(double name, String age, byte duration)`
- ☒ `ArrayList<String> mm()`

- ☒ `char[] mm(double numbers)`
☒ `String mm()`

▷ **Question 5.** En considérant la définition suivante de la classe `Course`, indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```

1 class Course {
2     void enroll(long duration) {
3         System.out.println("long");
4     }
5     void enroll(int duration) {
6         System.out.println("int");
7     }
8     void enroll(String s) {
9         System.out.println("String");
10    }
11    void enroll(Object o) {
12        System.out.println("Object");
13    }
14 }
```

```

1 class MyApp {
2     public static void main(String args[]) {
3         Course course = new Course();
4         char c = 10;
5         course.enroll(c);
6         course.enroll("Object");
7     }
8 }
```

- ☐ Erreur à la compilation ☐ Erreur à l'exécution ☒ `int`
☐ `String` ☐ `long`
☐ `Object`

▷ **Question 6.** En considérant le code suivant, indiquer la/les affirmation(s) correcte(s).

```

1 class Flower {
2     public void fragrance() { System.out.println("Flower"); }
3 }
4 class Rose {
5     public void fragrance() { System.out.println("Rose"); }
6 }
7 class Lily {
8     public void fragrance() { System.out.println("Lily"); }
9 }
10 class Bouquet {
11     public void arrangeFlowers() {
12         Flower f1 = new Rose();
13         Flower f2 = new Lily();
14         f1.fragrance();
15     }
16 }
```

- ☐ L'affichage produit est : Flower
☐ L'affichage produit est : Rose
☐ L'affichage produit est : Lily
☒ Erreur à la compilation.

▷ **Question 7.** En considérant le code suivant, indiquer quelles sont les déclarations correctes qui peuvent être insérées à la place du marqueur `// INSERT CODE HERE` (3 réponses).

```

1 interface Movable {
2     void move();
3 }
4 class Person implements Movable {
5     public void move() { System.out.println("Person move"); }
6 }
7 class Vehicle implements Movable {
8     public void move() { System.out.println("Vehicle move"); }
9 }
10 class Test {
11     // INSERT CODE HERE
12     {
13         movable.move();
14     }
15 }
```

- ☒ `void walk(Movable movable)`
☒ `void walk(Person movable)`
☒ `void walk(Vehicle movable)`
☐ `void walk()`

▷ **Question 8.** Indiquer quelle(s) est/sont les affirmations correctes (2 réponses).

- ☒ L'héritage permet de faciliter la réutilisation de code.
☒ L'héritage permet au développeur de ne pas à avoir à modifier le même code dans différentes classes.
☐ Le polymorphisme passe des instructions spéciales au compilateur afin que le code produit puisse être exécuté sur plusieurs plateformes.
☐ Les méthodes polymorphes ne peuvent lever des exceptions.

▷ **Question 9.** En considérant les classes définies ci-dessous, indiquer quelle instruction de code peut être insérée individuellement à la place du marqueur `//INSERT CODE HERE` afin que l'exécution du code produise l'affichage suivant (2 réponses) :

Programmer - Mala Gupta
 Author - Mala Gupta

```

1 class Programmer {
2     void print() {
3         System.out.println("Programmer - Mala Gupta");
4     }
5 }
6 class Author extends Programmer {
7     void print() {
8         System.out.println("Author - Mala Gupta");
9     }
10 }
11 class MyApp {
12     Programmer a = new Programmer();
13     // INSERT CODE HERE
14     a.print();
15     b.print();
16 }

```

- ☐ Programmer b = new Programmer();
- ☒ Programmer b = new Author();
- ☒ Author b = new Author();
- ☐ Author b = new Programmer();
- ☐ Programmer b = ((Author) new Programmer());
- ☐ Author b = ((Author) new Programmer());

▷ **Question 10.** Quel est l'affichage produit par l'exécution du code suivant (1 réponse) :

```

1 class Course {
2     String courseName;
3     Course() {
4         Course c = new Course();
5         c.courseName = "POO";
6     }
7 }
8 class MyApp {
9     public static void main(String args[]) {
10         Course c = new Course();
11         c.courseName = "TOP";
12         System.out.println(c.courseName);
13     }
14 }

```

- ☐ L'affichage produit est : TOP.
- ☐ L'affichage produit est : POO.
- ☐ Erreur à la compilation.
- ☒ Erreur à l'exécution (une exception est levée).

▷ **Question 11.** Indiquer quelle(s) est/sont les affirmations correctes (3 réponses).

- ☐ Il n'est pas possible d'attraper des exceptions du type `RuntimeException`.
- ☒ On ne devrait pas attraper les exceptions du type `Error`.
- ☒ Si une méthode lève une exception vérifiée, cette exception doit être traitée par la méthode ou bien spécifiée dans sa clause `throws`.
- ☒ Si une méthode lève une exception de type `RuntimeException`, cette exception doit être incluse dans la clause `throws` de cette méthode.
- ☐ Les exceptions du type `RuntimeException` sont des exceptions vérifiées.

▷ **Question 12.** Indiquer quel est l'affichage produit par l'exécution du code suivant (1 réponse) :

```

1 class MyApp {
2     void method() {
3         try {
4             guru();
5             return;
6         } finally {
7             System.out.println("finally 1");
8         }
9     }
10     void guru() {
11         System.out.println("guru");
12         throw new StackOverflowError();
13     }
14     public static void main(String args[]) {
15         MyApp var = new MyApp();
16         var.method();
17     }
18 }

```

- ☐

guru
finally 1
- ☐

guru
- ☒

guru
finally 1
Exception in thread "main" java.lang.StackOverflowError
- ☐

guru
Exception in thread "main" java.lang.StackOverflowError
- ☐ Erreur à la compilation.

▷ **Question 13.** Indiquer quelles sont les affirmations correctes (3 réponses).

- ☒ Les exceptions permettent au développeur de séparer la logique de l'application de la manière dont les erreurs sont traitées.
- ☐ La gestion des exceptions accélère la vitesse d'exécution d'un programme.
- ☒ La gestion des exceptions permet de définir du code qui sera exécuté uniquement lorsqu'une portion de code déclenchera une exception.

☒ Un morceau de code qui attrape et traite des exceptions vérifiées peut également lever des exceptions non vérifiées.

▷ **Question 14.** En considérant les définitions de classes suivantes, indiquer quelles sont les affectations d'une variable possible (2 réponses).

```
1 interface Jumpable {}
2 class Animal {}
3 class Lion extends Animal implements Jumpable {}
```

- ☐ Jumpable var1 = new Jumpable();
- ☒ Animal var2 = new Animal();
- ☐ Lion var3 = new Animal();
- ☐ Jumpable var4 = new Animal();
- ☒ Jumpable var5 = new Lion();

▷ **Question 15.** Indiquer quelles sont les affirmations correctes (3 réponses).

- ☒ Une classe Java peut définir plusieurs méthodes.
- ☒ Une classe Java peut définir plusieurs variables.
- ☐ Une classe Java peut être définie dans plusieurs paquetages (*package*).
- ☒ Une classe Java peut importer plusieurs paquetages.
- ☐ Une classe Java ne peut définir plus de 108 constructeurs.

▷ **Question 16.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```
1 class Doctor {
2     protected int age;
3     protected void setAge(int val) { age = val; }
4     protected int getAge() { return age; }
5 }
6 class Surgeon extends Doctor {
7     Surgeon(String val) {
8         specialization = val;
9     }
10    String specialization;
11    String getSpecialization() { return specialization; }
12 }
13 class Hospital {
14     public static void main(String args[]) {
15         Surgeon s1 = new Surgeon("Liver");
16         Surgeon s2 = new Surgeon("Heart");
17         s1.age = 45;
18         System.out.println(s1.age + s2.getSpecialization());
19         System.out.println(s2.age + s1.getSpecialization());
20     }
21 }
```

- ☒ 45Heart
0Liver
- ☐ 45Liver
0Heart
- ☐ 45Liver
45Heart
- ☐ 45Heart
45Heart
- ☐ Erreur à la compilation.

▷ **Question 17.** En considérant le code suivant, indiquer quelle instruction insérée à la place du marqueur `/* INSERT CODE HERE */` permet d'afficher la valeur de la variable `pagesPerMin` (1 réponse).

```
1 class Printer {
2     int inkLevel;
3 }
4 class LaserPrinter extends Printer {
5     int pagesPerMin;
6     public static void main(String args[]) {
7         Printer myPrinter = new LaserPrinter();
8         System.out.println(/* INSERT CODE HERE */);
9     }
10 }
```

- ☐ `(LaserPrinter) myPrinter.pagesPerMin`
- ☒ `myPrinter.pagesPerMin`
- ☐ `LaserPrinter.myPrinter.pagesPerMin`
- ☒ `((LaserPrinter) myPrinter).pagesPerMin`

▷ **Question 18.** Indiquer quelles sont les affirmations correctes (2 réponses).

- ☒ Gérer les exceptions peut permettre d'éviter qu'une application ne *crash* ou qu'elle produise des résultats incorrects.
- ☐ Les gestionnaires d'exception ne peuvent pas définir un flot d'instructions alternatif lors d'une exception.
- ☒ Les gestionnaires d'exception permettent au développeur de définir des blocs de code différents pour traiter différentes cas d'exception.
- ☐ Les gestionnaires d'exception permettent de définir des classes préservant l'encapsulation.
- ☐ Les gestionnaires d'exceptions permettent d'améliorer l'efficacité du mécanisme d'héritage.

▷ **Question 19.** Indiquer quelles sont les affirmations correctes (2 réponses).

- ☒ Plusieurs variables peuvent référencer exactement le même objet en mémoire.
- ☐ Des valeurs stockées dans une variable primitive et une variable de référence peuvent être comparées en terme d'égalité aussi bien en utilisant l'opérateur `==` que la méthode `equals`.
- ☒ Une variable primitive ne peut référencer un objet.

▷ **Question 20.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```

1 public class MyCalendar {
2     public static void main(String arguments[]) {
3         Season season1 = new Season();
4         season1.name = "Spring";
5         Season season2 = new Season();
6         season2.name = "Autumn";
7         season1 = season2;
8         System.out.println(season1.name);
9         System.out.println(season2.name);
10    }
11 }
12 class Season {
13     String name;
14 }

```

- ☐ L'affichage produit est : Spring Autumn
- ☐ L'affichage produit est : Spring Spring
- ☒ L'affichage produit est : Autumn Autumn
- ☐ L'affichage produit est : Autumn Spring

► **Question 21.** Indiquer quelles sont les affirmations correctes à propos du code suivant (1 réponse).

```

1 class Shoe {}
2 class Boot extends Shoe {}
3 class ShoeFactory {
4     ShoeFactory(Boot val) {
5         System.out.println("boot");
6     }
7     ShoeFactory(Shoe val) {
8         System.out.println("shoe");
9     }
10 }

```

- ☒ La classe `ShoeFactory` possède au total 2 constructeurs.
- ☐ La classe `ShoeFactory` possède 3 constructeurs, 2 définis par l'utilisateur et 1 constructeur par défaut.
- ☐ La classe `ShoeFactory` ne compile pas.
- ☐ En ajoutant le constructeur suivant, la classe `ShoeFactory` possèdera 3 constructeurs : `private ShoeFactory (Shoe arg) {}`

► **Question 22.** Considérant les définitions de la classe `Person` et de l'interface `Movable`, la tâche est de déclarer une classe `Emp` qui hérite de la classe `Person` et qui réalise l'interface `Movable`. Indiquer la définition correcte (1 réponse).

```

1 class Person {}
2 interface Movable {}

```

- ☐ `class Emp implements Person extends Movable {}`
- ☐ `class Emp implements Person, Movable {}`
- ☒ `class Emp extends Person implements Movable {}`
- ☐ `class Emp extends Person, Movable {}`

► **Question 23.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```

1 class Phone {
2     static void call() {
3         System.out.println("Call-Phone");
4     }
5 }
6 class SmartPhone extends Phone{
7     static void call() {
8         System.out.println("Call-SmartPhone");
9     }
10 }
11 class TestPhones {
12     public static void main(String[] args) {
13         Phone phone = new Phone();
14         Phone smartPhone = new SmartPhone();
15         phone.call();
16         smartPhone.call();
17     }
18 }

```

- ☒ Call-Phone
- ☐ Call-Phone
- ☐ Call-SmartPhone
- ☐ Call-Phone
- ☐ null
- ☐ null
- ☐ Call-SmartPhone

► **Question 24.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse)

```

1 class ColorPack {
2     int shadeCount = 12;
3     static int getShadeCount() {
4         return shadeCount;
5     }
6 }
7 class Artist {
8     public static void main(String args[]) {
9         ColorPack pack1 = new ColorPack();
10        System.out.println(pack1.getShadeCount());
11    }
12 }

```

- ☐ L'affichage produit est : 10
- ☐ L'affichage produit est : 12
- ☐ Aucun affichage.
- ☒ Erreur à la compilation.

► **Question 25.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse)

```

1 class Book {
2     String ISBN;
3     Book(String val) {
4         ISBN = val;
5     }
6 }
7 class TestEquals {
8     public static void main(String[] args) {
9         Book b1 = new Book("1234-4657");
10        Book b2 = new Book("1234-4657");
11        System.out.print(b1.equals(b2) + ":");
12        System.out.print(b1 == b2);
13    }
14 }

```

- ☐ true:false
- ☐ true:true
- ☐ false:true
- ☒ false:false
- ☐ Erreur à la compilation. Il n'y a pas de méthode `equals` dans la classe `Book`.

► **Question 26.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse)

```

1 class Phone {
2     void call() {
3         System.out.println("Call-Phone");
4     }
5 }
6 class SmartPhone extends Phone {
7     void call() {
8         System.out.println("Call-SmartPhone");
9     }
10 }
11 class TestPhones {
12     public static void main(String[] args) {
13         Phone phone = new Phone();
14         Phone smartPhone = new SmartPhone();
15         phone.call();
16         smartPhone.call();
17     }
18 }

```

- ☐ Call-Phone
☐ Call-Phone
☒ Call-Phone
☐ Call-SmartPhone
☐ Call-Phone
☐ null
☐ null
☐ Call-SmartPhone

► **Question 27.** La tâche était de réaliser une classe `Pencil` respectant le principe d'encapsulation et possédant une variable d'instance `mode`. La valeur de la variable `mode` doit être accessible et modifiable depuis une autre classe. Indiquer la solution (1 réponse).

☐

```

1 class Pencil {
2     public String mode;
3 }

```

☒

```

1 class Pencil {
2     private String mode;
3     public String getMode() { return mode; }
4     public void setMode(String val) { mode = val; }
5 }

```

☐

```

1 class Pencil {
2     public String mode;
3     public String getMode() { return mode; }
4     public void setMode(String val) { mode = val; }
5 }

```

☐

```

1 class Pencil {
2     public String mode;
3     private String getMode() { return mode; }
4     private void setMode(String val) { mode = val; }
5 }

```

► **Question 28.** Quels sont les affirmations correctes concernant la méthode `main` utilisée pour démarrer une application Java ? (2 réponses)

- ☒ Une classe ne peut définir plusieurs méthodes `main`.
☒ Plusieurs classes dans une même application peuvent définir une méthode `main`.
☐ La méthode `main` peut prendre en paramètre une valeur de type `String` ou de type `String[]`.
☐ La méthode `main` ne peut créer d'objet de la classe dans laquelle la méthode est-elle même définie.

► **Question 29.** Indiquer l'affichage produit par l'exécution du code suivant (1 réponse).

```

1 class Paper {
2     Paper() {
3         this(10);
4         System.out.println("Paper:0");
5     }
6     Paper(int a) {
7         System.out.println("Paper:1");
8     }
9 }
10 class PostIt extends Paper {}
11 class TestPostIt {
12     public static void main(String[] args) {
13         Paper paper = new PostIt();
14     }
15 }

```

- ☐ Paper:1
☐ Paper:0
☐ Paper:0
☐ Paper:1
☒ Paper:1
☐ Paper:0
☐ Erreur à la compilation.
☐ Erreur à l'exécution.

► **Question 30.** En considérant le code suivant, indiquer quelle instruction insérée à la place du marqueur `/* INSERT CODE HERE */` permettrait à la classe `Jungle` de déterminer si la variable `animal` référence un objet de la classe `Lion` et d'afficher `1` dans ce cas (1 réponse).

```

1 class Animal{ float age; }
2 class Lion extends Animal { int claws;}
3 class Jungle {
4     public static void main(String args[]) {
5         Animal animal = new Lion();
6         /* INSERT CODE HERE */
7         System.out.println(1);
8     }
9 }

```

- ☒ if (animal instanceof Lion)
☐ if (animal instanceof Lion)
☐ if (animal == Lion)
☐ if (animal = Lion)

► **Question 31.** En considérant le fichier `Test.java` dont le contenu est le suivant et qui produit des erreurs à la compilation, indiquer les raisons de ces erreurs (2 réponses).


```

1 class Person {
2     Person(String value) {}
3 }
4 class Employee extends Person {}
5 class Test {
6     public static void main(String args[]) {
7         Employee e = new Employee();
8     }
9 }

```

- ☐ La classe **Person** ne compile pas.
- ☒ La classe **Employee** ne compile pas.
- ☒ Le constructeur par défaut ne peut appeler qu'un constructeur sans paramètre dans la classe mère.
- ☐ Le code qui crée un objet de type **Employee** dans la classe **Test** n'a pas passé une valeur de type **String** en paramètre au constructeur de la classe **Employee**.

▷ **Question 32.** En considérant le code suivant, indiquer les affirmations correctes (2 réponses).

```

1 class Bottle {
2     void Bottle() {}
3     void Bottle(WaterBottle w) {}
4 }
5 class WaterBottle extends Bottle {}

```

- ☐ Une classe mère ne peut prendre en paramètre d'un de ses constructeurs une valeur du type d'une de ses sous-classes.
- ☒ La classe compile correctement. Une classe mère peut utiliser des références dont le type est une de ses classes dérivées.
- ☐ La classe **Bottle** définit deux constructeurs.
- ☒ La classe **Bottle** ne donne accès qu'à un seul constructeur.

▷ **Question 33.** En considérant les définitions de classes suivantes, quel morceau de code inséré à la place du marqueur `// INSERT CODE HERE` définira une utilisation valide de la méthode `write` dans la méthode `article` (2 réponses)

```

1 class NoInkException extends Exception {}
2 class Pen {
3     void write(String val) throws NoInkException {
4         //.. some code
5     }
6     void article() {
7         // INSERT CODE HERE
8     }
9 }

```

- ☒

```

try {
    new Pen().write("story");
} catch (NoInkException e) {}

```
- ☐

```

try {
    (new Pen()).write("story");
} finally {}

```

- ☒

```

try {
    write("story");
} catch (Exception e) {}

```
- ☐

```

try {
    (new Pen()).write("story");
} catch (RuntimeException e) {}

```

▷ **Question 34.** L'exécution de la méthode `main` de la classe **TestJavaCourse** produit l'affichage 300. Indiquer quel serait l'affichage produit si la variable `enrollments` était définie en tant que variable de classe (1 réponse).

```

1 class Course {
2     int enrollments;
3 }
4 class TestJavaCourse {
5     public static void main(String args[]) {
6         Course c1 = new Course();
7         Course c2 = new Course();
8         c1.enrollments = 100;
9         c2.enrollments = 200;
10        System.out.println(c1.enrollments + c2.enrollments);
11    }
12 }

```

- ☐ L'exécution de **TestJavaCourse** affiche 300.
- ☐ L'exécution de **TestJavaCourse** affiche 200.
- ☒ L'exécution de **TestJavaCourse** affiche 400.
- ☐ La classe **TestJavaCourse** ne compile plus.