Éléments de correction - Examen Java en POO

ali.zainoul.az@gmail.com

13 octobre 2023

Partie 1 : Questions de cours (2 points)

1. Quelle est la différence entre une classe abstraite et une interface en Java ? (0,5 point)

En Java, une classe abstraite peut contenir des méthodes concrètes (méthodes avec une implémentation) et des méthodes abstraites (méthodes déclarées mais sans implémentation). Une interface ne peut contenir que des méthodes abstraites (par défaut depuis Java 8) et des constantes publiques. Une classe peut hériter d'une seule classe abstraite, mais elle peut implémenter plusieurs interfaces.

Exemple de classe abstraite :

```
abstract class Forme {
    public abstract double calculerAire();
    public abstract double calculerPerimetre();
}
```

Exemple d'interface :

```
interface Dessinable {
    void dessiner();
    void effacer();
}
```

2. Qu'est-ce que l'héritage en Java et comment est-il utilisé dans la programmation orientée objet ? (0,5 point)

L'héritage en Java est un mécanisme qui permet à une classe (classe fille ou sous-classe) de hériter des attributs et des méthodes d'une autre classe (classe parent ou superclasse). Il favorise la réutilisation du code et la création de hiérarchies de classes. Les classes filles peuvent ajouter de nouvelles méthodes ou modifier le comportement des méthodes héritées.

Exemple d'héritage :

```
class Animal {
    String nom;

    public Animal(String nom) {
        this.nom = nom;
```

```
    }

    void manger() {
        System.out.println(nom + " mange de la nourriture.");
    }
}

class Chien extends Animal {
    public Chien(String nom) {
        super(nom);
    }

    void aboyer() {
        System.out.println(nom + " aboie.");
    }
}
```

3. Expliquez le concept de polymorphisme en Java et donnez un exemple d'utilisation. (0,5 point)

Le polymorphisme en Java permet à un objet d'adopter plusieurs formes en fonction du contexte. Cela signifie qu'un même nom de méthode peut être utilisé pour des objets de classes différentes, et la méthode appropriée sera appelée en fonction du type réel de l'objet.

Exemple de polymorphisme :

```
class Animal {
    void faireDuBruit() {
        System.out.println("Animal fait du bruit.");
    }
}

class Chat extends Animal {
    void faireDuBruit() {
        System.out.println("Chat fait du bruit.");
    }
}
```

Utilisation du polymorphisme :

```
Animal monAnimal = new Chat(); // Polymorphisme
monAnimal.faireDuBruit();
// Appelle la methode faireDuBruit() de la classe Chat
```

4. Qu'est-ce que la gestion des exceptions en Java ? Donnez un exemple d'utilisation de blocs try-catch. (0,5 point)

La gestion des exceptions en Java permet de gérer les erreurs et les exceptions pendant l'exécution d'un programme. Un bloc try-catch permet de tenter une opération potentiellement risquée dans le bloc try, et de capturer et gérer les exceptions qui pourraient être

générées.

Exemple de bloc try-catch :

```
try {
    int resultat = 10 / 0;
// Division par zero genere une exception
} catch (ArithmeticException e) {
    System.out.println("Erreur : Division par zero.");
}
```

---

1 Partie 2

Exercice 1 (3 points)

Énoncé de l'exercice :

Écrivez un programme Java qui calcule la somme des éléments d'un tableau d'entiers et affiche le résultat. Assurez-vous de compiler et d'exécuter le programme avec succès.

Correction :

Voici le code Java pour le programme "SumArray" qui effectue le calcul de la somme avec une seule boucle :

Listing 1 – Programme Java "SumArray" pour calculer la somme d'un tableau d'entiers avec entrées utilisateur et gestion des exceptions

```
import java.util.Scanner;

public class SumArray {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print("Enter the size of the array: "); // Prompt user for array size

            int size = 0;
            try {
                size = scanner.nextInt();
            } catch (java.util.InputMismatchException e) {
                System.out.println("Error: Please enter an integer for the array size.");
                return;
            }

            if (size <= 0) {
                System.out.println("Error: The array size must be a positive integer.");
                return;
            }

            int sum = 0;
            for (int i = 0; i < size; i++) {
                System.out.print("Enter element #" + (i + 1) + ": "); // Prompt user for array elements
                try {
                    int element = scanner.nextInt();
                    sum += element;
                } catch (java.util.InputMismatchException e) {
                    System.out.println("Error: Please enter an integer for array elements.");
                    return;
                }
            }

            System.out.println("Sum of array elements: " + sum);
        }
    }
}

/*
COMPILATION
# Compile the program
```

```
#   javac SumArray.java
# Execute the program
#   java SumArray
*/
```

Exercice 2 (4 points)

Énoncé de l'exercice :

Créez un package nommé "calculatrice" contenant quatre classes distinctes : "Addition," "Soustraction," "Multiplication," et "Division." Chaque classe doit avoir une méthode pour effectuer l'opération correspondante. Écrivez un programme Java nommé "Main" qui utilise ces classes pour effectuer des opérations mathématiques.

Correction :

You can find the Java solution to this exercise below. Please note that it was requested to code in French, but the solution is provided in English for pedagogical purposes.

First, let's create the package "calculatrice" and define the classes within separate files :

Addition.java :

```
package calculatrice;

public class Addition {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

Soustraction.java :

```
package calculatrice;

public class Soustraction {
    public static int subtract(int a, int b) {
        return a - b;
    }
}
```

Multiplication.java :

```
package calculatrice;

public class Multiplication {
    public static int multiply(int a, int b) {
        return a * b;
    }
}
```

Division.java :

```
package calculatrice;

public class Division {
    public static double divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero not allowed.");
        }
        return (double) a / b;
    }
}
```

Now, let's create the "Main" class that uses these classes to perform mathematical operations :

Main.java :

```
package calculatrice;

public class Main {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 5;

        int sum = Addition.add(num1, num2);
        int difference = Soustraction.subtract(num1, num2);
        int product = Multiplication.multiply(num1, num2);

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);

        try {
            double quotient = Division.divide(num1, num2);
            System.out.println("Quotient: " + quotient);
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

This code demonstrates a Java program with a package named "calculatrice" containing four separate classes : "Addition," "Soustraction," "Multiplication," and "Division." Each class has a method to perform the corresponding mathematical operation, and the "Main" class uses these classes to perform addition, subtraction, multiplication, and division operations.

Please note that the code is presented in English, although it was originally requested to be coded in French for pedagogical purposes. You can customize the values of ‘num1‘ and

‘num2‘ as needed.

Exercice 3 (3 points)

Énoncé de l’exercice :

Écrivez une classe Java "Rectangle" avec des attributs pour la longueur et la largeur. Ajoutez des méthodes pour calculer la surface et le périmètre du rectangle. Créez un objet "Rectangle" et affichez ses caractéristiques.

Correction :

Here’s a Java class "Rectangle" that calculates the area and perimeter of a rectangle, along with an example of creating a "Rectangle" object and displaying its characteristics :

Listing 2 – Java class "Rectangle" for area and perimeter calculation

```
public class Rectangle {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    public double calculateArea() {
        return length * width;
    }

    public double calculatePerimeter() {
        return 2 * (length + width);
    }

    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle(5.0, 3.0);
        System.out.println("Rectangle Characteristics:");
        System.out.println("Length: " + rectangle.length);
        System.out.println("Width: " + rectangle.width);
        System.out.println("Area: " + rectangle.calculateArea());
        System.out.println("Perimeter: " + rectangle.calculatePerimeter());
    }
}
```

This code defines a "Rectangle" class with attributes for length and width, and methods to calculate the area and perimeter. In the example, a "Rectangle" object is created with a length of 5.0 and a width of 3.0, and its characteristics are displayed.

Exercice 4 (4 points)

Énoncé de l'exercice :

Écrivez ce qui va afficher ce code Java et expliquez pourquoi :

```
class Animal {
    public void sound() {
        System.out.println("Animal making a sound");
    }
}

class Cat extends Animal {
    public void sound() {
        System.out.println("Cat making a sound");
        super.sound();
    }
}

class Dog extends Animal {
    public void sound() {
        System.out.println("Dog making a sound");
        super.sound();
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Cat cat = new Cat();
        Animal catAnimal = cat;
        Dog dog = new Dog();
        Animal dogAnimal = dog;
        animal.sound();
        cat.sound();
        dog.sound();
        catAnimal.sound();
        dogAnimal.sound();
    }
}
```

Correction :

Le code Java fourni définit une hiérarchie de classes - "Animal," "Cat" (Chat) et "Dog" (Chien). Chaque classe a une méthode "sound()" (son) et il y a une classe "Main" qui crée des instances de ces classes et appelle leurs méthodes "sound()".

Voici ce que le code affichera et pourquoi :

```
Animal making a sound
Cat making a sound
Animal making a sound
Dog making a sound
Animal making a sound
Cat making a sound
Animal making a sound
Dog making a sound
Animal making a sound
```

Explication : - ‘animal.sound() ;‘ : Appelle la méthode "sound()" de la classe "Animal," qui affiche "Animal making a sound" (l’animal fait un son).

- ‘cat.sound() ;‘ : Appelle la méthode "sound()" de la classe "Cat." À l’intérieur de la classe "Cat," elle affiche "Cat making a sound" (le chat fait un son), puis appelle ‘super.sound() ;‘, ce qui appelle la méthode "sound()" de la superclasse "Animal," affichant ainsi "Animal making a sound" (l’animal fait un son).

- ‘dog.sound() ;‘ : De manière similaire à la classe "Cat," la classe "Dog" affiche "Dog making a sound" (le chien fait un son), puis appelle ‘super.sound() ;‘, ce qui affiche "Animal making a sound" (l’animal fait un son).

- ‘catAnimal.sound() ;‘ : Étant donné que "catAnimal" est une référence à un objet "Cat," il se comporte de la même manière que ‘cat.sound() ;‘, affichant donc "Cat making a sound" suivi de "Animal making a sound."

- ‘dogAnimal.sound() ;‘ : De manière similaire à "catAnimal," "dogAnimal" est une référence à un objet "Dog," donc il affiche "Dog making a sound" suivi de "Animal making a sound."

Cette sortie démontre le concept de substitution de méthode et l’utilisation de ‘super‘ pour appeler une méthode de la superclasse.

Remarque : Le type de polymorphisme présent dans l’exemple donné est le "polymorphisme de substitution" ou "polymorphisme statique."

Le polymorphisme de substitution se produit lorsqu’une sous-classe fournit une implémentation spécifique pour une méthode déjà définie dans sa superclasse. Dans l’exemple donné, les classes "Cat" et "Dog" héritent de la classe "Animal" et fournissent des implémentations spécifiques de la méthode "sound()" héritée de la classe "Animal." Lorsque nous appelons la méthode "sound()" sur des instances de "Cat" ou "Dog," l’implémentation spécifique de la sous-classe est exécutée.

Résumé du Polymorphisme de Substitution : Dans cet exemple, nous avons exploré le concept de "polymorphisme de substitution" en Java. Le polymorphisme de substitution permet aux sous-classes de redéfinir les méthodes héritées de leur superclasse pour fournir une implémentation spécifique. Dans notre exemple, les classes "Cat" et "Dog" ont hérité de la classe "Animal" et ont redéfini la méthode "sound()" pour produire des sons spécifiques à chaque animal.

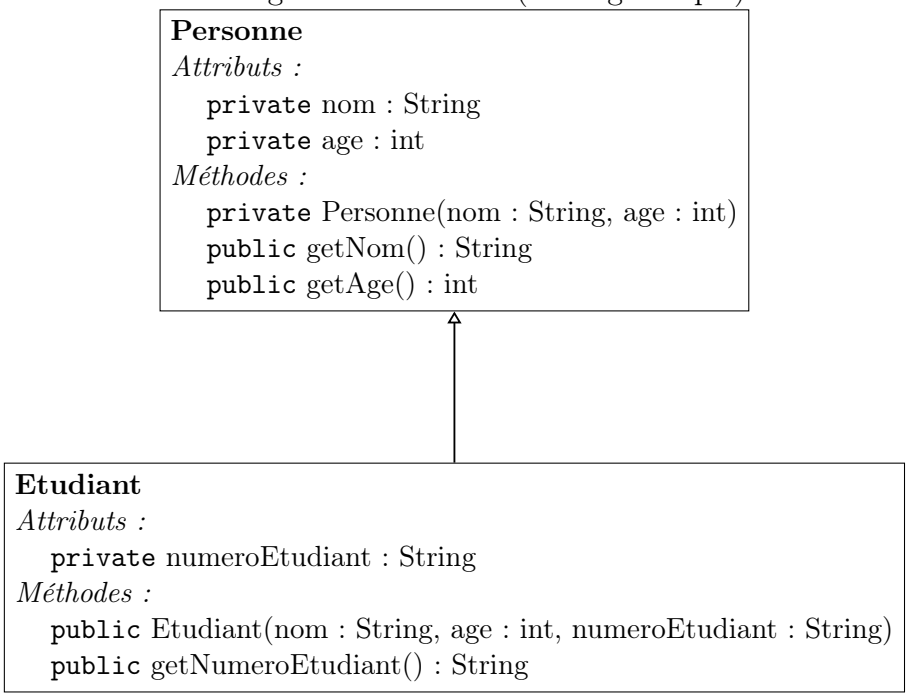
Ce mécanisme permet aux classes filles de personnaliser leur comportement tout en utilisant également le comportement de la classe parente. Une particularité de cet exemple est l'utilisation de l'appel 'super.sound() ;' à l'intérieur des méthodes redéfinies, ce qui permet d'appeler la méthode de la classe parente en plus de l'implémentation de la classe fille.

Le polymorphisme de substitution favorise la réutilisation du code, l'extensibilité et la flexibilité dans la programmation orientée objet, en permettant aux classes de fournir des comportements spécifiques tout en maintenant une structure de classe commune avec la superclasse.

Exercice 5 (4 points)

Énoncé de l'exercice :

En observant le diagramme de classe (héritage simple) suivant :



Codez les classes correspondantes et écrivez une classe Main testant votre programme.

Correction :

Voici le code Java correspondant à l'exercice 5, basé sur le diagramme de classe donné :

Listing 3 – Code Java pour les classes Personne et Etudiant

```
class Personne {
    private String nom;
    private int age;

    public Personne(String nom, int age) {
        this.nom = nom;
        this.age = age;
    }
```

```
    public String getNom() {
        return nom;
    }

    public int getAge() {
        return age;
    }
}

class Etudiant extends Personne {
    private String numeroEtudiant;

    public Etudiant(String nom, int age, String numeroEtudiant) {
        super(nom, age);
        this.numeroEtudiant = numeroEtudiant;
    }

    public String getNumeroEtudiant() {
        return numeroEtudiant;
    }
}

public class Main {
    public static void main(String[] args) {
        Personne personne = new Personne("Alice", 25);
        Etudiant etudiant = new Etudiant("Bob", 20, "E12345");

        System.out.println("Nom de la personne : " + personne.getNom());
        System.out.println("Age de la personne : " + personne.getAge());

        System.out.println("Nom de l'etudiant : " + etudiant.getNom());
        System.out.println("Age de l'etudiant : " + etudiant.getAge());
        System.out.println("Numero d'etudiant : " + etudiant.getNumeroEtudiant());
    }
}
```

Ce code Java implémente les classes "Personne" et "Etudiant" en respectant le diagramme de classe donné. La classe "Main" crée des instances de ces classes et affiche leurs attributs. Vous pouvez personnaliser les valeurs des attributs pour vos besoins.