Mini Projet de TP POO

République Algérienne Démocratique et populaire

Ministère de l'Enseignement Supérieur et de la Recherche

Scientifique Université des Sciences et de la Technologie Houari

Boumediene U.S.T.H.B

2 éme année ISIL

Section A / Groupe

04

Projet sur :

Oriented object programing

Réalisérigé par :

Lounas Idjourdikane

Guidoum Noeman

Projet POO

Groupe 4

Lounas Idjourdikane

Guidoum Noeman

Introduction

Les deux exercices suivants ont pour objectif le développement d'applications informatiques répondant à des besoins spécifiques. Dans le premier exercice, nommé "Gestion de la location de voitures", l'objectif est de concevoir un logiciel dédié à la gestion des locations de voitures au sein d'une agence spécialisée. Dans le deuxième exercice, intitulé "Application de calcul", le but est de créer une application Java capable d'effectuer différentes opérations mathématiques. Ces exercices intègrent des concepts de conception orientée objet avec des aspects pratiques d'implémentation Java, offrant ainsi une opportunité d'appliquer des concepts théoriques à des problèmes réels.

Exo 01 :

Pour gérer la location des voitures aux clients dans une agence de

location de voitures, on désire développer un logiciel.

Ce logiciel est principalement composé de deux classes « voiture et

client ».

1/ Donnez une déclaration de ces deux classes « la déclaration doit

contenir au moins trois attributs ».

2/ Choisissez deux attributs et donner leurs accesseurs et modificateurs,

Donner la méthode toString et equals de la classe client.

Introduction de l'exercice 1 :

L'exercice 1, intitulé "Gestion de la location des voitures", a pour but de concevoir un logiciel dédié à la gestion des locations de voitures au sein d'une agence de location. Ce logiciel repose sur la structure de deux classes principales, à savoir la classe "Voiture" et la classe "Client". Chacune de ces classes doit comporter au moins trois attributs.

Nous aborderons la déclaration de ces deux classes en fournissant des exemples d'attributs. De plus, nous sélectionnerons deux attributs de la classe "Client" et de la classe "Voiture", puis détaillerons la mise en place des accesseurs et modificateurs pour ces attributs. Enfin, nous discuterons de l'implémentation des méthodes toString et equals spécifiques à la classe "Client".

Classe Voiture :

1.1 Déclaration des attributs marque, couleur, année_fabrication :

La classe "Voiture" représente en Java une entité de voiture avec trois attributs privés : "marque" (une chaîne de caractères), "couleur" (une chaîne de caractères) et "année_fabrication" (un entier). L'utilisation du modificateur d'accès private pour ces attributs indique qu'ils ne peuvent être directement accédés ou modifiés depuis l'extérieur de la classe.

Pour interagir avec ces attributs, des méthodes d'accès publiques, également appelées "getters" et "setters", peuvent être implémentées dans la classe. Ces méthodes permettent respectivement d'obtenir et de définir les valeurs des attributs.

1.2 Déclaration du constructeur pour la classe "Voiture" en Java :

La ligne de code ci-dessus représente la définition du constructeur pour la classe "Voiture". Ce constructeur prend trois paramètres : "marque" (une chaîne de caractères), "couleur" (une chaîne de caractères), et "année_fabrication" (un entier). Lorsqu'un objet de la classe "Voiture" est instancié en utilisant ce constructeur, les valeurs fournies en paramètres sont utilisées pour initialiser les attributs correspondants de l'objet.

L'utilisation du mot-clé "this" dans le contexte du constructeur fait référence aux attributs spécifiques de l'objet en cours d'instanciation. Ainsi, "this.marque", "this.couleur", et "this.année_fabrication" désignent respectivement les attributs de l'objet en cours de création. Cette utilisation de "this" permet de faire la distinction entre les paramètres du constructeur et les attributs de la classe.

1.3 Définition des accesseurs et modificateurs pour la classe

"Voiture" :

Les paires de méthodes, getMarque & setMarque ainsi que getCouleur & setCouleur, agissent en tant qu'interfaces permettant d'accéder et de modifier les attributs privés "marque" et "couleur" de la classe "Voiture". Ces méthodes fournissent des moyens contrôlés pour obtenir et modifier les valeurs des attributs, garantissant ainsi l'encapsulation et favorisant une gestion sûre et modulaire des données.

En utilisant les accesseurs (getters), on peut récupérer la valeur actuelle des attributs, tandis que les modificateurs (setters) permettent de définir de nouvelles valeurs pour ces attributs. Cette approche assure un contrôle précis sur l'accès et la modification des données internes de la classe "Voiture", contribuant ainsi à maintenir la cohérence et l'intégrité du système.

Deuxième Partie :

Classe Client :

2.1 Déclaration des attributs nom, prenom, age :

La classe "Client" en Java définit une entité représentant un client, avec trois attributs privés : "nom" (une chaîne de caractères), "prenom" (une chaîne de caractères) et "age" (un entier). Ces attributs encapsulent les informations relatives au nom, prénom et âge du client, limitant leur accès direct depuis l'extérieur de la classe. La classe peut être utilisée comme un modèle pour créer des objets clients dans un programme Java.

2.2 Déclaration du constructeur Client :

Cette ligne de code représente le constructeur de la classe "Client" en Java. Le constructeur prend trois paramètres : "nom" (une chaîne de caractères), "prenom" (une chaîne de caractères) et "age" (un entier). Lorsqu'un objet de la classe "Client" est instancié en utilisant ce constructeur, les valeurs fournies en paramètres sont utilisées pour initialiser les attributs correspondants de l'objet. Ainsi, ce constructeur facilite la création d'objets "Client" en permettant l'initialisation rapide et cohérente de ses attributs.

2.3 Définition des accesseurs et modificateurs de la classe "Client" :

Les paires de méthodes, getNom & setNom et getAge & setAge, servent d'interfaces pour accéder et modifier respectivement les attributs privés "nom" et "age" de la classe "Client". Ces méthodes offrent un moyen contrôlé d'obtenir et de modifier la valeur de ces attributs, garantissant ainsi l'encapsulation et favorisant une gestion sûre et modulaire des données.

2.4 Définition de la méthode toString() :

La méthode toString est une fonction du langage de programmation Java utilisée pour convertir un objet en une représentation sous forme de chaîne de caractères. Elle retourne une chaîne qui combine les valeurs des propriétés de l'objet, incluant le nom, le prénom et l'âge.

2.5 Définition de la méthode equals() :

Cette méthode compare l'égalité entre deux objets de la classe "Client" en

se basant sur leurs propriétés spécifiques. Si toutes les propriétés sont égales, la méthode renvoie true ; sinon, elle renvoie false.

Conclusion de l’exercice 1 :

Cet exercice a permis de mettre en pratique des concepts fondamentaux de la programmation orientée objet, tels que la déclaration de classes, la gestion d'attributs avec accesseurs et modificateurs, ainsi que l'implémentation de méthodes utiles comme toString() et equals(). Ces compétences sont cruciales pour le développement de logiciels robustes et bien structurés, en particulier dans le domaine de la gestion de données complexes, telles que celles impliquées dans la location de voitures.

```java
package Exo1_project;

public class Main {

    public class MainClass {

        public static void main(String[] args) {
            Voiture voiture1 = new Voiture("Toyota",11923,2020);
            Client client1 = new Client("noeman","guidom",19);
            System.out.println("Informations sur la voiture :");
            System.out.println(voiture1.toString());

            System.out.println("\n Informations sur le client :");
            System.out.println(client1.toString());
            Client client2 = new Client("noeman","guidom",19);

            System.out.println("\nComparaison entre deux clients :");
            System.out.println("Les deux clients sont-ils égaux ? " + client1.equals(client2));
        }
    }

}
```

```java
package Exo1_project;

public class Voiture {
 private String marque ;
 private int matricule;
 private int annFabr ;
    public Voiture( String marque,int matricule,int annFabr) {
     this.marque=marque;
    this.matricule=matricule;
     this.annFabr=annFabr;
    }
    public String getMarque() {
        return marque;
    }
    public void setMarque(String marque) {
        this.marque = marque;
    }

    public int getAnnFabr() {
        return annFabr;
    }
    public void setAnnFabr(int annFabr) {
        this.annFabr = annFabr;
    }
    public int getMatricule() {
        return matricule;
    }
    public void setMatricule(int matricule) {
        this.matricule = matricule;
    }


}
```

```java
public class Client {
    private String nom ;
    private String prenom;
    private int age;
    public Client(String nom,String prenom,int age) {
    this.nom=nom;
    this.prenom=prenom;
        this.age=age;

    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
public String toString() {
    return "le nom est"+this.nom+"le prenom est:"+this.prenom+"l'age est:"+this.age;
}
public boolean equals(Client bnadem) {
    if (bnadem.nom==this.nom && bnadem.prenom==this.prenom && bnadem.age==this.age)
        return true;
    else return false ;
}
```

Exercice 02 : La Calculatrice

L’énoncé de l’Exercice :

Soit l’interface CalculMath qui possède une méthode Calcul () qui calcule et retourne le résultat d’une opération mathématique.

Soit la classe OperationBinaire qui représente les opérations mathématiques qui se basent sur deux valeurs et la classe OperationUnaire qui représente les opérations mathématiques qui se basent sur une seule valeur.

Soit les classes Addition, Soustraction, Multiplication et Division des classes qui assurent respectivement le calcul de l’addition, soustraction, multiplication et division de deux valeurs.

Soit les classes Sin, Cos, Log et Exp des classes qui assurent respectivement le calcul du sinus, cosinus, logarithme et exponentiel d’une valeur.

1/ Donnez la modélisation des classes décrites ci-dessus et liens de hiérarchie entre elles (Schéma UML pour une bonne présentation du modèle).

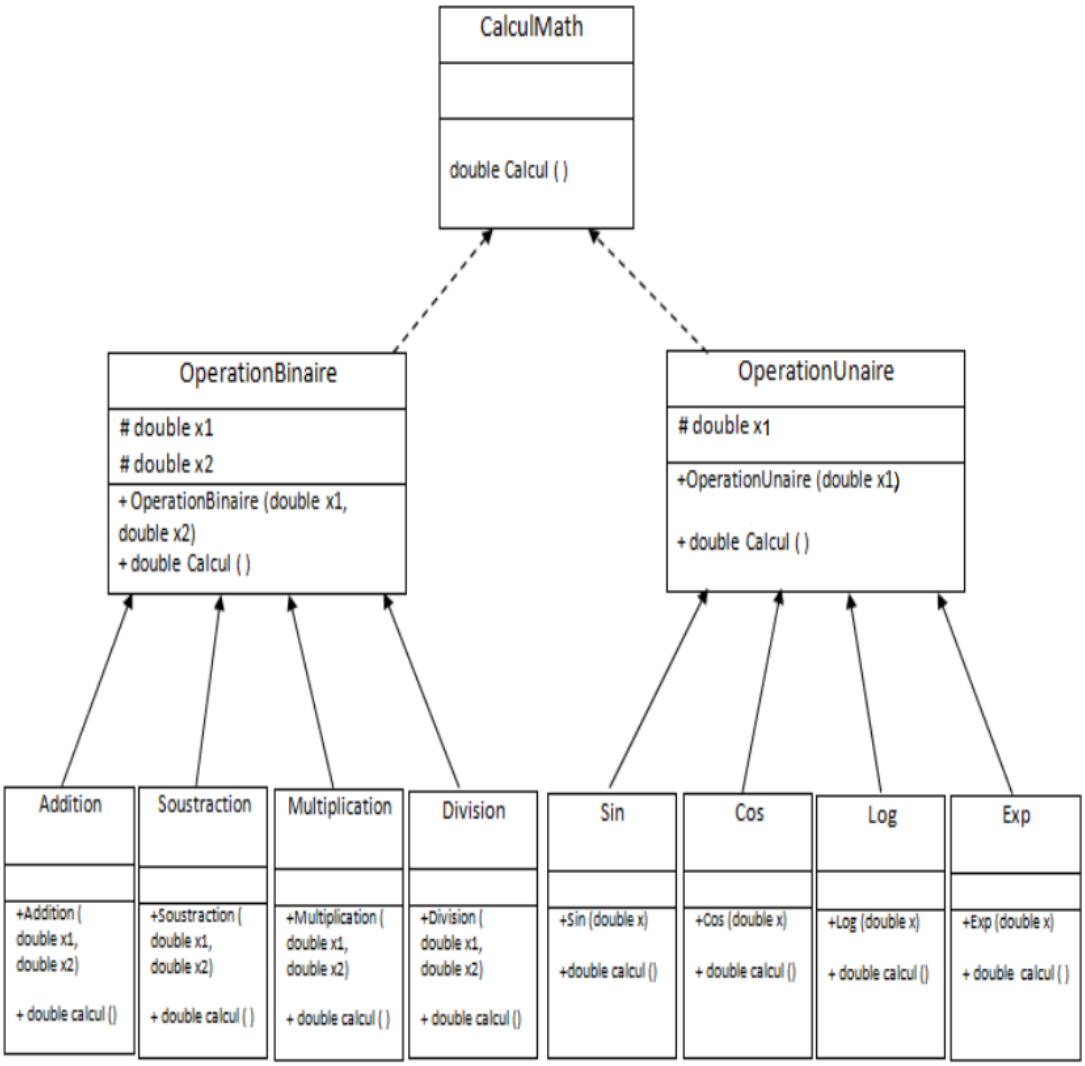
2/ Implémenter l’application Java correspondante à cette modélisation, Le programme doit être présenté en proposant un menu avec les différentes opérations au choix. Quand l’utilisateur choisit une opération, on lui demande de taper la valeur ou les valeurs nécessaires ensuite, on affiche le résultat de l’opération.

Introduction de l’exercice 2 :

L'exercice 2, intitulé "Calculatrice", propose la conception et l'implémentation d'une application Java permettant d'effectuer diverses opérations mathématiques. L'énoncé présente une interface CalculMath qui définit une méthode Calcul() destinée à calculer et renvoyer le résultat d'une opération mathématique. Deux classes, OperationBinaire et OperationUnaire, sont définies pour représenter respectivement les opérations basées sur deux valeurs et une seule valeur. En outre, plusieurs classes telles qu’Addition, Soustraction, Multiplication, Division, Sin, Cos, Log, et Exp sont créées pour réaliser des opérations spécifiques.

Première Partie :

Schéma UML :



Deuxième Partie :

1. **L‘interface CalculMath :**

- L'interface `CalculMath` sert de contrat pour les classes qui l'implémentent. Toute classe qui implémente cette interface doit fournir une implémentation de la méthode `Calcul`, qui effectue des opérations mathématiques et retourne un résultat de type double.

```java
package calculatrice;

public interface CalculMath {
 double calcul();
}
```

2. **La classe OperationBinaire :**

```java
package calculatrice;

public class OperationBinaire implements CalculMath {
    protected double x1,x2;
    public OperationBinaire(double x1,double x2) {
        this.x1=x1;
        this.x2=x2;
    }
    @Override
    public double calcul() {
        return 0;
    }
```

2.1 **Déclaration des attributs x1, x2 :**

- La classe `OperationBinaire` en Java implémente l'interface `CalculMath`. Elle contient deux champs protégés x1 et x2 de type double, utilisés pour stocker les deux opérandes d'une opération binaire.

2.2 **Déclaration du constructeur OperationBinaire **

- Le constructeur prend deux paramètres de type double x1 et x2 et initialise les champs correspondants de la classe avec ces valeurs.

2.3 **Redéfinition de la méthode Calcul() :**

- Cette méthode fournit une implémentation initiale qui retourne toujours zéro et peut être remplacée par des sous-classes pour effectuer des calculs spécifiques.

3. **La classe OperationUnaire :**

```
package calculatrice;

public class OperationUnaire implements CalculMath {
    protected double x1;
    public OperationUnaire (double x1) {
     this.x1=x1;}
    @Override
    public double calcul() {
        return 0;
    }

}
```

3.1 **Déclaration de l’attribut x :**

- La classe `OperationUnaire` en Java implémente l'interface `CalculMath` et contient un champ protégé x de type double, utilisé pour stocker l'opérande d'une opération unaire.

3.2 **Déclaration du constructeur OperationUnaire **

- Le constructeur prend un paramètre de type double (x) et initialise le champ correspondant de la classe avec cette valeur.

3.3 **Redéfinition de la méthode Calcul() :**

- Cette méthode fournit une implémentation initiale qui retourne toujours zéro et peut être remplacée par des sous-classes pour effectuer des calculs spécifiques.

4. **La classe Addition :**

- La classe `Addition` en Java hérite de la classe `OperationBinaire` et représente une opération d'addition. Son constructeur prend deux paramètres de type double x1 et x2 et les utilise pour initialiser les opérandes de l'opération. La méthode Calcul est implémentée pour effectuer l'addition des deux opérandes et renvoyer le résultat.

```
package calculatrice;

public class addition extends OperationBinaire{

    public addition(double x1,double x2) {
        super(x1,x2);
    }

    @Override
    public double calcul() {
        return x1+x2;
    }

}
```

5. **La classe Soustraction :**

- La classe `Soustraction` en Java hérite de la classe `OperationBinaire` et

représente une opération de soustraction. Son constructeur prend deux paramètres de type double (x1 et x2) et les utilise pour initialiser les opérandes de l'opération. La méthode Calcul est implémentée pour effectuer la soustraction des deux opérandes et renvoyer le résultat.

```java
package calculatrice;

public class Soustraction extends OperationBinaire {
    public Soustraction(double x1,double x2) {
        super(x1,x2);
    }

    @Override
    public double calcul() {
        return x1-x2;
    }
}
```

6. **La classe Multiplication :**

- La classe `Multiplication` en Java hérite de la classe `OperationBinaire` et représente une opération de multiplication. Son constructeur prend deux paramètres de type double (x1 et x2) et les utilise pour initialiser les opérandes de l'opération. La méthode Calcul est implémentée pour effectuer la multiplication des deux opérandes et renvoyer le résultat.

```java
package calculatrice;

public class Multiplication extends OperationBinaire {
        public Multiplication(double x1,double x2) {
            super(x1,x2);
        }

        @Override
        public double calcul() {
            return x1*x2;
        }
    }
```

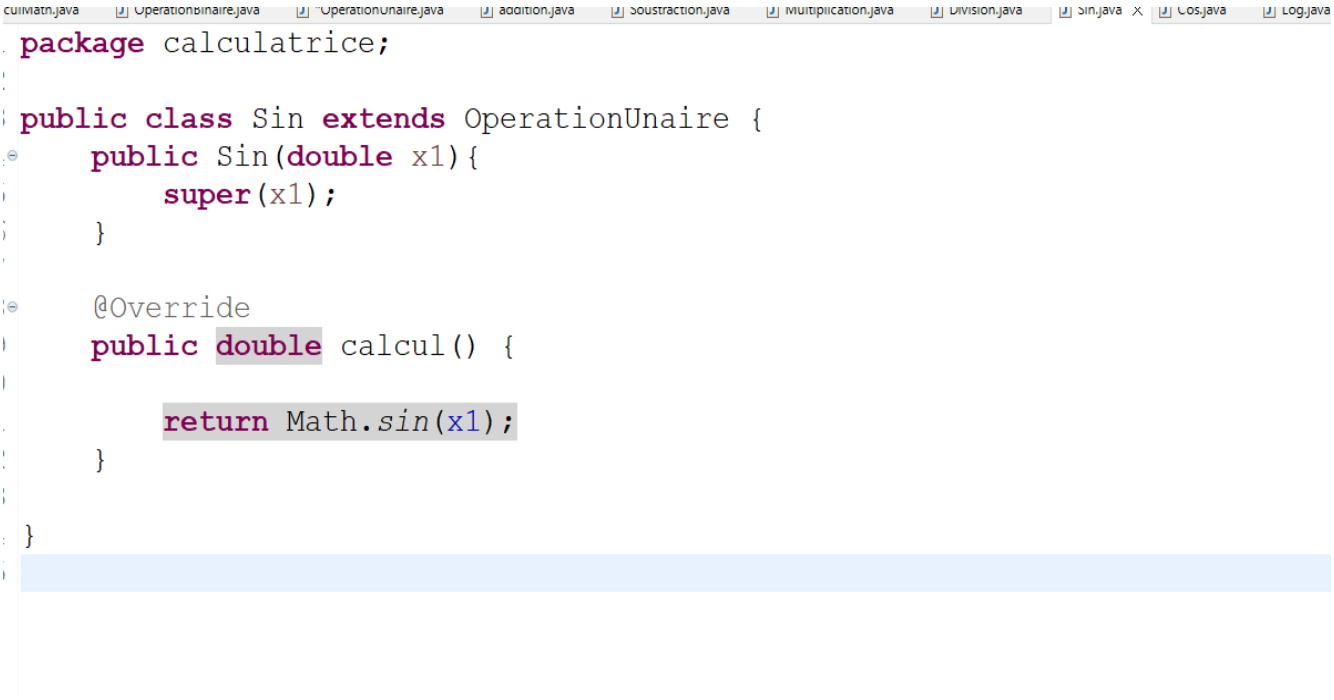
7. **La classe Division :**

- La classe `Division` en Java hérite de la classe `OperationBinaire` et représente une opération de division. Son constructeur prend deux paramètres de type double x1 et x2 et les utilise pour initialiser les opérandes de l'opération. La méthode Calcul est implémentée pour effectuer la division des deux opérandes et renvoyer le résultat. Elle gère également le cas où la division par zéro est tentée, en lançant une exception et renvoyant une valeur spéciale `Double.NaN`.

```java
package calculatrice;

public class Division extends OperationBinaire {
    public Division(double x1,double x2) {
        super(x1,x2);
    }

    @Override
    public double calcul() {
    try {
        if(x2 !=0)return x1/x2;
        else throw new ArithmeticException("Division par Zero");
    }
    catch (ArithmeticException  e) {
        System.out.println("Erreur"+e.getMessage());
        return Double.NaN;
    }
    }


}
```

8. **La classe Sin :**

- 1/La classe `Sin` en Java hérite de la classe `OperationUnaire` et représente une opération de calcul du sinus. Son constructeur prend un paramètre de type double x et l'utilise pour initialiser l'opérande de l'opération. La méthode Calcul est implémentée pour calculer le sinus de l'opérande et renvoyer le résultat à l'aide de la bibliothèque Math de Java.



9. **La classe Cos :**

- La classe `Cos` en Java hérite de la classe `OperationUnaire` et représente une opération de calcul du cosinus. Son constructeur prend un paramètre de type double (x) et l'utilise pour initialiser l'opérande de l'

opération. La méthode Calcul est implémentée pour calculer le cosinus de l'opérande et renvoyer le résultat à l'aide de la bibliothèque Math de Java.

```java
package calculatrice;

public class Cos extends OperationUnaire {
        public  Cos(double x1) {
        super(x1);
    }


        @Override
        public double calcul() {
            return Math.cos(x1);
        }

}
```

10. **La classe Log :**

- La classe `Log` en Java hérite de la classe `OperationUnaire` et représente une opération de calcul du logarithme naturel. Son constructeur prend un paramètre de type double (x) et l'utilise pour initialiser l'opérande de l'opération. La méthode Calcul est implémentée pour calculer le logarithme naturel de l'opérande, tout en gérant le cas où l'opérande est négatif ou nul en lançant une exception. En cas d'erreur, elle renvoie une valeur spéciale `Double.NaN`.

CalculMath.java | OperationBinaire.java | *OperationUnaire.java | addition.java | Soustraction.java | Multiplication.java | Division.java | Sin.java | Cos.java

```java
package calculatrice;

public class Log extends OperationUnaire {

    public Log(double x1) {
        super(x1);
    }

    @Override
    public double calcul() {
    try { if (x1>0) return Math.log(x1);
    else throw new ArithmeticException("Division par Zero");
}
catch (ArithmeticException  e) {
    System.out.println("Erreur"+e.getMessage());
    return Double.NaN;
}
    }
    }

```

11. **La classe Exp :**

- La classe `Exp` en Java hérite de la classe `OperationUnaire` et représente une opération de calcul de l'exponentielle. Son constructeur prend un paramètre de type double (x) et l'utilise pour initialiser l'opérande de l'opération. La méthode Calcul est implémentée pour calculer l'exponentielle de l'opérande et renvoyer le résultat à l'aide de la bibliothèque Math de Java.

```java
package calculatrice;

public class Exp extends OperationUnaire {

    public Exp(double x1) {
        super(x1);
    }

    @Override
    public double calcul() {
        return Math.exp(x1);
    }

}
```

12. **La classe Main :**

```java
package calculatrice;
import java.util.Scanner;

public class Calculator{

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nOpérations disponibles :");
            System.out.println("1. Addition");
            System.out.println("2. Soustraction");
            System.out.println("3. ");
            System.out.println("4. Division");
            System.out.println("5. Sinus");
            System.out.println("6. Cosinus");
            System.out.println("7. Logarithme");
            System.out.println("8. Exponentielle");
            System.out.println("0. Quitter");

            System.out.print("Choisissez une opération (0-8) : ");
            int choix = scanner.nextInt();

            if (choix == 0) {
                System.out.println("Au revoir !");
                break;
            }

            double x=0;
            double x1 = 0;
            double x2=0;

            if (choix >= 5) {
                System.out.print("Entrez la valeur x : ");
                x = scanner.nextDouble();
            } else {
                System.out.print("Entrez la valeur x1 : ");
                x1 = scanner.nextDouble();
                System.out.print("Entrez la valeur x2 : ");
                x2 = scanner.nextDouble();
            }
```

```
            switch (choix) {
            case 1:
                Addition addition = new Addition(x1, x2);
                System.out.println("Résultat : " + addition.calcul());
                break;
            case 2:
                Soustraction soustraction = new Soustraction(x1, x2);
                System.out.println("Résultat : " + soustraction.calcul());
                break;
            case 3:
                Multiplication multiplication = new Multiplication(x1, x2);
                System.out.println("Résultat : " + multiplication.calcul());
                break;
            case 4:
                Division division = new Division(x1, x2);
                System.out.println("Résultat : " + division.calcul());
                break;
            case 5:
                Sin sin = new Sin(x);
                System.out.println("Résultat : " + sin.calcul());
                break;
            case 6:
                Cos cos = new Cos(x);
                System.out.println("Résultat : " + cos.calcul());
                break;
            case 7:
                Log log = new Log(x);
                System.out.println("Résultat : " + log.calcul());
                break;
            case 8:
                Exp exp = new Exp(x);
                System.out.println("Résultat : " + exp.calcul());
                break;
            default:
                System.out.println("Opération invalide. Veuillez choisir une opération valide (0-8).");
        }
    }

    scanner.close();
}
```

12.1 **Importation de la classe Scanner et définition de l'objet Scanner :**

- Puisque nous utiliserons le concept de lecture, il nous est impératif d'importer la classe dédiée `Scanner`. Il est aussi nécessaire de créer un objet de classe `Scanner` qui aura comme but d'assurer les opérations de lecture.

12.2 **Affichage d'un menu des opérations

mathématiques :**

- Le code affiche un menu dans la console, présentant différentes opérations mathématiques numérotées de 1 à 8. Chaque numéro est associé à une opération spécifique : addition ‘1’, soustraction ‘2’, multiplication ‘3’, division ‘4’, sinus ‘5’, cosinus ‘6’, logarithme ‘7’, et exponentielle ‘8’. L'utilisateur pourrait utiliser ces numéros pour sélectionner une opération spécifique dans le cadre d'une calculatrice ou d'un programme similaire.

12.3 **Lecture du choix et Initialisation des variables :**

- Le code affiche un message demandant à l'utilisateur de choisir une opération à effectuer. Il utilise un objet `Scanner` pour lire l'entrée de l'utilisateur et stocke le choix dans une variable choix de type entier. En outre, il initialise trois variables de type double (x, x1, x2) avec des valeurs par défaut de zéro. Ces variables sont susceptibles d'être utilisées pour stocker des valeurs d'opérandes ou de résultats dans le programme ultérieur.

12.4 **Vérification du choix :**

- Le code vérifie si le choix de l'opération (stocké dans la variable choix) est supérieur ou égal à 5. Si c'est le cas, il demande à l'utilisateur de saisir une valeur x. Sinon, pour les choix inférieurs à 5, il demande à l'utilisateur de saisir deux valeurs x1 et x2. Les valeurs saisies par l'utilisateur sont stockées dans les variables correspondantes x, x1, ou x2 pour une utilisation ultérieure dans le programme.

12.5 **La Structure Switch Case :**

- Le code utilise une structure switch-case pour instancier un objet de type `CalculMath` en fonction du choix de l'opération (choix). Selon

la valeur de choix, il crée une instance de la classe correspondante, associée à une opération mathématique spécifique. Si le choix ne correspond à aucune opération valide, il affiche "Opération invalide" et sort du programme. L'objet `operation` est utilisé pour effectuer des calculs ultérieurs en utilisant l'opération sélectionnée.

12.6 **Exécution du Calcul et Affichage du Résultat:**

- Le code vérifie si l'objet `operation` a été initialisé (différent de null). Si c'est le cas, il appelle la méthode `Calcul` sur l'objet `operation` pour effectuer le calcul associé à l'opération sélectionnée. Le résultat du calcul est stocké dans la variable `resultat` et affiché dans la console avec un message indiquant le résultat.

Conclusion de l’exercice 2 :

Cet exercice a non seulement permis de mettre en pratique des concepts avancés de programmation orientée objet, y compris l'utilisation d'interfaces, l'héritage, et la création de classes spécialisées, mais il a également démontré l'importance de la modélisation UML pour planifier et visualiser la structure d'un système complexe. Il renforce les compétences nécessaires pour développer des applications orientées objet flexibles et extensibles.

Conclusion :

En combinant ces deux exercices, les participants ont eu l'occasion de renforcer leurs compétences en POO, de la conception à l'implémentation, tout en travaillant sur des problématiques pratiques de gestion de données et de calculs mathématiques. Ces exercices ont contribué à former une

perspective complète sur la manière d'appliquer les principes de la POO pour résoudre des problèmes réels de manière structurée et efficace.