TP N°3

Héritage, Polymorphisme et Abstraction

Préparée par : RGUIBI Farah

Objectif :

Ce travail pratique approfondit les concepts fondamentaux de la POO en Java à travers l'implémentation de plusieurs hiérarchies de classes. Le TP couvre l'encapsulation, l'héritage, le polymorphisme et la modélisation d'entités du monde réel.

FIGURE 1 – Java

1 Les exercices :

Exercice 1 :

1.1 Partie 1 : Hiérarchie Rectangle/Carré :

1.1.1 Idée de l'exercice :

Créer une hiérarchie de classes modélisant des formes géométriques avec héritage. Cet exercice introduit les concepts d'héritage simple, d'attributs protégés, et de réutilisation de code via la relation IS-A. Le carré hérite du rectangle car un carré est un cas particulier de rectangle où longueur égale largeur.

1.1.2 Approche adoptée :

J'ai développé une classe parent `Rectangle` avec trois attributs protégés : `longueur`, `largeur` et `nom`. La classe propose deux constructeurs (par défaut et paramétré) et des méthodes pour calculer la surface et afficher les informations. La classe `Carre` hérite de Rectangle et redéfinit le constructeur pour garantir l'égalité des côtés en forçant largeur = longueur.

1.1.3 Points clés de l'implémentation :

- Utilisation du modificateur **protected** permettant l'accès direct aux attributs dans la sous-classe
- Constructeur par défaut dans Rectangle : longueur=5.0, largeur=13.0, nom="rectangle"
- Constructeur paramétré acceptant les trois valeurs
- Méthode `surface()` calculant l'aire : longueur × largeur
- Méthode `afficher()` présentant forme, dimensions et surface
- Dans Carre : appel de `super()` puis modification de `nom` et `largeur`
- Héritage automatique de la méthode `surface()` sans redéfinition

1.1.4 Diagramme de la hiérarchie Rectangle/Carré :

Voilà le diagramme UML de la hiérarchie Rectangle/Carré :

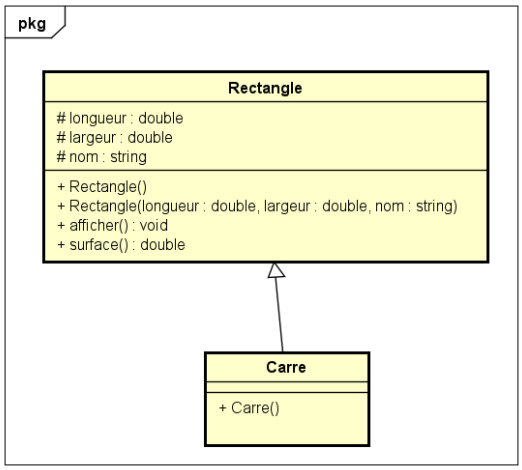

FIGURE 2 – Diagramme de la hiérarchie Rectangle/Carré

1.1.5 Résultat :

Le programme `MainRectangle` demande à l'utilisateur les dimensions d'un rectangle personnalisé, l'affiche avec sa surface calculée, puis crée et affiche un carré utilisant les valeurs par défaut (côté de 5.0). L'héritage permet au carré de bénéficier automatiquement du calcul de surface sans duplication de code.

```java
package TP3;

class Rectangle {
    protected double longueur;
    protected double largeur;
    protected String nom;

    public Rectangle() {
        this.longueur = 5.0;
        this.largeur = 13.0;
        this.nom="rectangle";}
    public Rectangle(double longueur, double largeur,String nom) {
        this.longueur = longueur;
        this.largeur = largeur;
        this.nom=nom;}
    public void afficher() {
        System.out.println("Forme: " + nom);
        System.out.println("Longueur: " + longueur);
        System.out.println("Largeur: " + largeur);
        System.out.println("Surface: " + surface());}
    public double surface() {
        return longueur * largeur;}}
```

FIGURE 3 – La classe rectangle

```java
package TP3;


class Carre extends Rectangle {
    public Carre() {
        super();
        this.nom = "carré";
        this.largeur = this.longueur;
    }
}
```

FIGURE 4 – La classe carre

```java
package TP3;

import java.util.Scanner;

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

        System.out.println(" Rectangle ");
        System.out.print("Entrez la longueur du rectangle: ");
        double longueur = scanner.nextDouble();
        System.out.print("Entrez la largeur du rectangle: ");
        double largeur = scanner.nextDouble();
        System.out.println("entrer le nom du rectangle");
        String nom=scanner.next();
        Rectangle rect = new Rectangle(longueur, largeur,nom);
        rect.afficher();

        System.out.println(" Carré ");
        Carre carre = new Carre();
        carre.afficher();}}
```

FIGURE 5 – Main de la hiérarchie Rectangle/-Carré

```
 Rectangle
Entrez la longueur du rectangle: 2
Entrez la largeur du rectangle: 3
entrer le nom du rectangle
rectangle
Forme: rectangle
Longueur: 2.0
Largeur: 3.0
Surface: 6.0
 Carré
Forme: carré
Longueur: 5.0
Largeur: 5.0
Surface: 25.0
```

FIGURE 6 – resultat de la hiérarchie Rectangle/-Carré

1.2 Partie 2 : Composition Point/Segment :

1.2.1 Idée de l'exercice :

Créer des classes liées par composition plutôt que par héritage. Cet exercice illustre la relation HAS-A : un segment possède deux points (origine et extrémité). Il introduit également la redéfinition de `toString()` et l'implémentation de calculs géométriques (distance euclidienne).

1.2.2 Approche adoptée :

J'ai développé une classe `Point` encapsulant les coordonnées (x, y) avec constructeurs et accesseurs. La classe `Segment` contient deux attributs de type Point et propose un constructeur acceptant quatre coordonnées pour créer automatiquement les deux points. La méthode `longueur()` calcule la distance entre origine et extrémité en utilisant le théorème de Pythagore.

1.2.3 Points clés de l'implémentation :

- Classe Point avec attributs privés `x` et `y` de type double
- Constructeur par défaut de Point : origine (0.0, 0.0)
- Redéfinition de `toString()` retournant "(x, y)" pour affichage formaté
- Classe Segment avec **composition** : attributs `orig` et `extrem` de type Point
- Constructeur de Segment acceptant 4 doubles et créant deux objets Point
- Méthode `afficher()` utilisant `toString()` des points pour affichage

- Méthode `longueur()` calculant : ((x2-x1)² + (y2-y1)²)
- Utilisation de `Math.sqrt()` pour la racine carrée

1.2.4 Diagramme de la composition Point/Segment :

Voilà le diagramme UML montrant la composition Point/Segment :

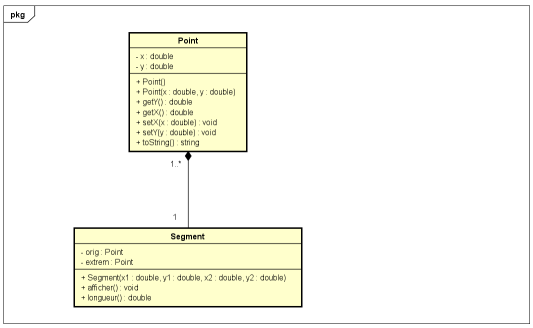

FIGURE 7 – Diagramme de la composition Point/Segment

1.2.5 Résultat :

Le programme `MainPoint` demande à l'utilisateur les coordonnées de l'origine et de l'extrémité du segment. Il affiche ensuite les deux points avec leur représentation formatée grâce à `toString()`, puis calcule et affiche la longueur du segment. La composition permet de réutiliser la classe Point sans héritage.

```java
package TP3;

class Point {
    private double x;
    private double y;

    public Point() {
        this.x = 0.0;
        this.y = 0.0;}
    public Point(double x, double y) {
        this.x = x;
        this.y = y;}
    public double getX() {return x; }
    public double getY() {return y;}
    public void setX(double x) {this.x=x;}
    public void setY(double y) {this.y=y;}
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

FIGURE 8 – La classe point

```java
package TP3;

class Segment {
    private Point orig;
    private Point extrem;

    public Segment(double x1, double y1, double x2, double y2) {
        this.orig = new Point(x1, y1);
        this.extrem = new Point(x2, y2);
    }
    public void afficher() {
        System.out.println("Segment:");
        System.out.println("  Origine: " + orig);
        System.out.println("  Extrémité: " + extrem);
    }

    public double longueur() {
        double dx = extrem.getX() - orig.getX();
        double dy = extrem.getY() - orig.getY();
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

FIGURE 9 – la classe segment

```java
package TP3;

import java.util.Scanner;

public class MainPoint {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println(" Segment 1");
        System.out.print("Entrez x1 (origine): ");
        double x1 = scanner.nextDouble();
        System.out.print("Entrez y1 (origine): ");
        double y1 = scanner.nextDouble();
        System.out.print("Entrez x2 (extrémité): ");
        double x2 = scanner.nextDouble();
        System.out.print("Entrez y2 (extrémité): ");
        double y2 = scanner.nextDouble();
        Segment seg = new Segment(x1, y1, x2, y2);
        seg.afficher();
        System.out.println("  Longueur: " + seg.longueur());

    }
}
```

FIGURE 10 – Main de la composition Point/-Segment

```
 Segment 1
Entrez x1 (origine): 2
Entrez y1 (origine): 3
Entrez x2 (extrémité): 5
Entrez y2 (extrémité): 4
Segment:
  Origine: (2.0, 3.0)
  Extrémité: (5.0, 4.0)
  Longueur: 3.1622776601683795
```

FIGURE 11 – Resultat de la Composition Point/-Segment

Exercice 2 : Hiérarchie Bâtiment/Maison

1.2.6 Idée de l'exercice :

Créer une hiérarchie modélisant des bâtiments avec spécialisation par héritage. Cet exercice illustre l'encapsulation stricte (attribut privé dans le parent), la redéfinition de méthodes avec `@Override`, et l'utilisation de `super()` pour transmettre des paramètres au constructeur parent.

1.2.7 Approche adoptée :

J'ai développé une classe `Batiment` avec un attribut privé `adresse` et des accesseurs. La classe `Maison` hérite de Batiment et ajoute l'attribut `nbPieces`. Les deux classes proposent constructeurs par défaut et paramétrés. La méthode `afficher()` est redéfinie dans Maison pour inclure le nombre de pièces tout en réutilisant l'adresse via le getter du parent.

1.2.8 Points clés de l'implémentation :

- Encapsulation stricte : attribut `adresse` est **private** dans Batiment
- Constructeur par défaut de Batiment : adresse = "Rue eljezzar"
- Accesseurs `getAdresse()` et `setAdresse()` pour l'encapsulation
- Maison utilise `super(adresse)` pour transmettre l'adresse au parent
- Ajout de l'attribut `nbPieces` (int) avec ses accesseurs
- **Redéfinition** de `afficher()` avec annotation `@Override`
- Utilisation de `getAdresse()` du parent pour accéder à l'attribut privé

1.2.9 Diagramme de la hiérarchie Bâtiment/Maison :

Voilà le diagramme UML de la hiérarchie Bâtiment/Maison :

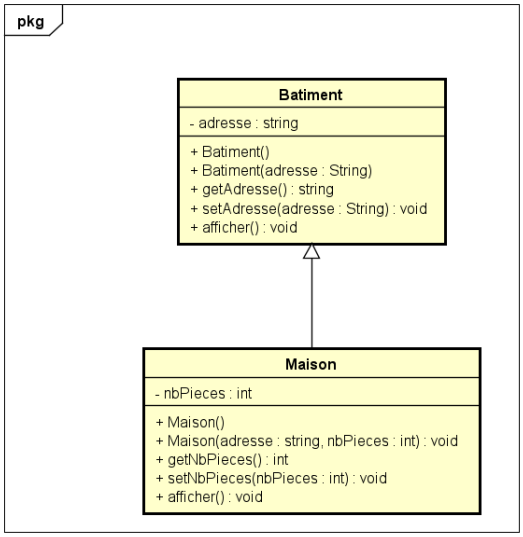

FIGURE 12 – Diagramme de la Hiérarchie Bâtiment/Maison

1.2.10 Résultat :

Le programme `MainBatiment` crée d'abord un bâtiment avec une adresse saisie par l'utilisateur et l'affiche. Ensuite, il crée une maison avec adresse et nombre de pièces personnalisés. La redéfinition de `afficher()` permet d'afficher les informations complètes de la maison incluant les données héritées et ajoutées.

```java
package TP3;

class Batiment {
    private String adresse;

    public Batiment() {
        this.adresse = "Rue eljezzar";}
    public Batiment(String adresse) {
        this.adresse = adresse;}
    public String getAdresse() {
        return adresse;
    }
    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }

    public void afficher() {
        System.out.println("Bâtiment:");
        System.out.println("  Adresse: " + adresse);
    }
}
```

FIGURE 13 – La classe batiment

```java
package TP3;

class Maison extends Batiment {
    private int nbPieces;

    public Maison() {
        super();
        this.nbPieces = 0;}
    public Maison(String adresse, int nbPieces) {
        super(adresse);
        this.nbPieces = nbPieces;}
    public int getNbPieces() {
        return nbPieces;}
    public void setNbPieces(int nbPieces) {
        this.nbPieces = nbPieces;}
    @Override
    public void afficher() {
        System.out.println("Maison:");
        System.out.println("  Adresse: " + getAdresse());
        System.out.println("  Nombre de pièces: " + nbPieces);
    }
}
```

FIGURE 14 – la classe maison

```java
package TP3;

import java.util.Scanner;
public class MainBatiment {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Bâtiment");
        System.out.print("Entrez l'adresse du bâtiment: ");
        String adresseBat = scanner.nextLine();
        Batiment bat = new Batiment(adresseBat);
        bat.afficher();

        System.out.println("Maison ");
        System.out.print("Entrez l'adresse de la maison: ");
        String adresseMaison = scanner.nextLine();
        System.out.print("Entrez le nombre de pièces: ");
        int nbPieces = scanner.nextInt();
        Maison maison = new Maison(adresseMaison, nbPieces);
        maison.afficher();
        }}
```

FIGURE 15 – Main de la Hiérarchie Bâtiment/-Maison

```
Bâtiment
Entrez l'adresse du bâtiment: rue eljazzar 10
Bâtiment:
  Adresse: rue eljazzar 10
Maison
Entrez l'adresse de la maison: quartier jrayfat SAFI
Entrez le nombre de pièces: 10
Maison:
  Adresse: quartier jrayfat SAFI
  Nombre de pièces: 10
```

FIGURE 16 – Resultat de la Hiérarchie Bâtiment/Maison

Exercice 3 : Hiérarchie Employé (Ouvrier, Cadre, Patron)

1.2.11 Idée de l'exercice :

Créer une hiérarchie modélisant différents types d'employés avec calcul polymorphe des salaires. Cet exercice illustre l'utilisation de classes abstraites, le polymorphisme, les méthodes abstraites, et les attributs statiques partagés entre instances.

1.2.12 Approche adoptée :

J'ai développé une classe abstraite Employe avec les attributs communs (matricule, nom, prénom, date de naissance) et une méthode abstraite getSalaire(). Trois classes concrètes héritent de Employe :

Ouvrier : salaire basé sur le SMIG + ancienneté Cadre : salaire selon un indice (1 à 4) Patron : salaire calculé comme pourcentage du chiffre d'affaires

1.2.13 Points clés de l'implémentation :

- Classe abstraite Employe avec méthode abstraite getSalaire()
- Attributs protégés (protected) pour permettre l'héritage
- Ouvrier : constante statique SMIG = 2500, calcul avec ancienneté (100 DH/an), plafond à 2×SMIG
- Cadre : indice (1-4) détermine le salaire via une structure switch
- Patron : attribut statique CA (chiffre d'affaires) partagé, salaire = CA × pourcentage/100
- Polymorphisme : ArrayList<Employe> contient tous les types d'employés
- Méthode afficher() dans la classe parent pour l'affichage des informations

1.2.14 Diagramme de la hiérarchie Employé :

Voilà le diagramme UML montrant Employe (classe abstraite) et ses trois classes filles :

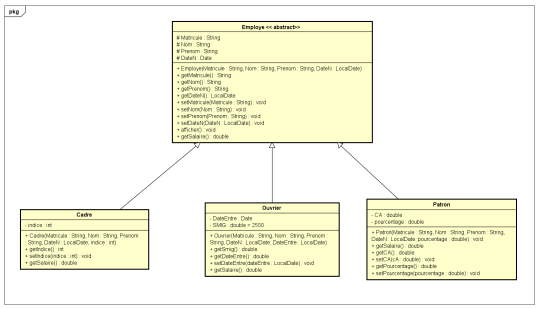

FIGURE 17 – Diagramme de la hiérarchie Employe (classe abstraite) et ses trois classes filles

1.2.15 Résultat :

Le programme MainEmploye permet la saisie interactive des informations pour un ouvrier, un cadre et un patron. Il stocke tous les employés dans une liste polymorphe et affiche leurs informations complètes avec leurs salaires respectifs calculés selon leur type.

```java
package TP3;
import java.time.LocalDate;

public abstract class Employe {
    protected String Matricule,Nom,Prenom;
    protected LocalDate  DateN;

    public Employe(String Matricule,String Nom,String Prenom,LocalDate DateN) {
    this.Matricule=Matricule;
    this.Nom=Nom;
    this.Prenom=Prenom;
    this.DateN=DateN;
    }
    public String getMatricule() {return Matricule;}
    public String getNom() {return Nom;}
    public String getPrenom() {return Prenom;}
    public LocalDate getDateN() {return DateN;}
    public void setMatricule(String Matricule) {this.Matricule=Matricule;}
    public void setNom(String Nom) {this.Nom=Nom;}
    public void setPrenom(String Prenom) {this.Prenom=Prenom;}
    public void setDateN(LocalDate DateN) {this.DateN=DateN;}
    public void afficher() {
        System.out.println("le nom de l'employe est + " + Nom  );
        System.out.println("le Prenom de l'employe est + " + Prenom  );
        System.out.println("la date de naissance  de l'employe est + " + DateN  );
        System.out.println("le Matricule de l'employe est + " + Matricule  );
    }
    public abstract double getSalaire();
    }
```

FIGURE 18 – La classe employe

```java
package TP3;

import java.time.LocalDate;

public class Patron extends Employe {
    private static double CA;
    private double pourcentage;
    public Patron(String Matricule,String Nom,String Prenom,LocalDate DateN,double pourcentage) {
        super(Matricule,Nom,Prenom,DateN);
        this.pourcentage=pourcentage;
    }
    @Override
    public double getSalaire() {
        return CA*pourcentage /100;
    }
    public static double getCA() {
        return CA;
    }
    public static void setCA(double cA) {
        CA = cA;
    }
    public double getPourcentage() {
        return pourcentage;
    }
    public void setPourcentage(double pourcentage) {
        this.pourcentage = pourcentage;
    }
}
```

FIGURE 19 – la classe patron

```java
package TP3;
import java.time.LocalDate;

public class Ouvrier extends Employe {

private  LocalDate DateEntre;
private static final double SMIG = 2500;

    public Ouvrier(String Matricule,String Nom,String Prenom,LocalDate DateN,LocalDate DateEntre) {
        super(Matricule,Nom,Prenom,DateN);
        this.DateEntre=DateEntre;
    }

    public static double getSmig() {
        return SMIG;
    }

    public LocalDate getDateEntre() {
        return DateEntre;
    }

    public void setDateEntre(LocalDate dateEntre) {
        DateEntre = dateEntre;
    }

    @Override
    public double getSalaire() {
        int ancienneté = LocalDate.now().getYear() - DateEntre.getYear();
        double salaire = SMIG + ancienneté * 100;
        if (salaire > SMIG * 2) {
            salaire = SMIG * 2;
        }
        return salaire;
    }}
```

FIGURE 20 – La classe ouvrier

```java
package TP3;

import java.time.LocalDate;

public class Cadre extends Employe {

private int indice;

    public Cadre(String Matricule,String Nom,String Prenom,LocalDate DateN,int indice) {
        super(Matricule,Nom,Prenom,DateN);
        this.indice=indice;
    }
    public int getIndice() {return indice;}
    public void setIndice(int indice) {this.indice=indice;}
    @Override
    public double getSalaire() {
        switch(indice) {
        case 1:return 13000;
        case 2:return 15000;
        case 3 :return 17000;
        case 4 :return 20000;
        default : return 0;
        }
    }
```

FIGURE 21 – La classe cadre

```java
package TP3;
import java.time.LocalDate;

public class MainEmploye {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ArrayList<Employe> listeEmployes = new ArrayList<>();

        System.out.print("Entrez le chiffre d'affaire commun aux patrons : ");
        double chiffreAffaire = sc.nextDouble();
        Patron.setCA(chiffreAffaire);

        for (int i = 1; i <= 1; i++) {
            sc.nextLine();
            System.out.println("\n--- Ouvrier " + i + " ---");
            System.out.print("Matricule : "); String matricule = sc.nextLine();
            System.out.print("Nom : "); String nom = sc.nextLine();
            System.out.print("Prénom : "); String prenom = sc.nextLine();
            System.out.print("Année de naissance (yyyy) : "); int anneeN = sc.nextInt();
            System.out.print("Mois de naissance (1-12) : "); int moisN = sc.nextInt();
            System.out.print("Jour de naissance : "); int jourN = sc.nextInt();
            LocalDate dateN = LocalDate.of(anneeN, moisN, jourN);

            System.out.print("Année d'entrée : "); int anneeEntree = sc.nextInt();
            System.out.print("Mois d'entrée (1-12) : "); int moisEntree = sc.nextInt();
            System.out.print("Jour d'entrée : "); int jourEntree = sc.nextInt();
            LocalDate dateEntree = LocalDate.of(anneeEntree, moisEntree, jourEntree);

            listeEmployes.add(new Ouvrier(matricule, nom, prenom, dateN, dateEntree));
        }

        for (int i = 1; i <= 1; i++) {
            sc.nextLine();
            System.out.println("\n--- Cadre " + i + " ---");
            System.out.print("Matricule : "); String matricule = sc.nextLine();
            System.out.print("Nom : "); String nom = sc.nextLine();
            System.out.print("Prénom : "); String prenom = sc.nextLine();
            System.out.print("Année de naissance (yyyy) : "); int anneeN = sc.nextInt();
            System.out.print("Mois de naissance (1-12) : "); int moisN = sc.nextInt();
            System.out.print("Jour de naissance : "); int jourN = sc.nextInt();
            LocalDate dateN = LocalDate.of(anneeN, moisN, jourN);

            System.out.print("Indice du cadre (1-4) : "); int indice = sc.nextInt();

            listeEmployes.add(new Cadre(matricule, nom, prenom, dateN, indice));
        }

        for (int i = 1; i <= 1; i++) {
            sc.nextLine();
            System.out.println("\n--- Patron " + i + " ---");
            System.out.print("Matricule : "); String matricule = sc.nextLine();
            System.out.print("Nom : "); String nom = sc.nextLine();
            System.out.print("Prénom : "); String prenom = sc.nextLine();
            System.out.print("Année de naissance (yyyy) : "); int anneeN = sc.nextInt();
            System.out.print("Mois de naissance (1-12) : "); int moisN = sc.nextInt();
            System.out.print("Jour de naissance : "); int jourN = sc.nextInt();
            LocalDate dateN = LocalDate.of(anneeN, moisN, jourN);

            System.out.print("Pourcentage du patron : "); double pourcentage = sc.nextDouble();

            listeEmployes.add(new Patron(matricule, nom, prenom, dateN, pourcentage));
        }

        sc.close();

        System.out.println("\n\n--- Liste des employés ---");
        for (Employe e : listeEmployes) {
            e.afficher();
            System.out.println("Salaire : " + e.getSalaire() + " DH\n");
        }
    }
}
```

FIGURE 22 – Main de la Hiérarchie employe

```
Bâtiment
Entrez l'adresse du bâtiment: rue eljazzar 10
Bâtiment:
  Adresse: rue eljazzar 10
Maison
Entrez l'adresse de la maison: quartier jrayfat SAFI
Entrez le nombre de pièces: 10
Maison:
  Adresse: quartier jrayfat SAFI
  Nombre de pièces: 10
```

FIGURE 23 – Resultat de la Hiérarchie employe

Exercice 4 : Hiérarchie Véhicule (Voiture, Camion)

1.2.16 Idée de l'exercice :

Créer une hiérarchie modélisant différents types de véhicules avec comportements spécifiques. Cet exercice illustre l'utilisation de classes abstraites avec méthodes abstraites multiples, le polymorphisme comportemental, et la redéfinition de toString().

1.2.17 Approche adoptée :

J'ai développé une classe abstraite Vehicule avec les attributs communs (matricule, prix, année) et deux méthodes abstraites demarre() et accelerer(). Un attribut statique compteur permet de suivre le nombre de véhicules créés. Deux classes concrètes héritent de Vehicule :

Voiture : démarre doucement, accélère rapidement Camion : démarre lentement, accélère progressivement

1.2.18 Points clés de l'implémentation :

- Classe abstraite Vehicule avec deux méthodes abstraites pour le comportement
- Attributs privés avec encapsulation complète (getters/setters)
- Attribut statique compteur incrémenté dans le constructeur
- Redéfinition de toString() avec annotation @Override pour l'affichage
- Voiture et Camion : implémentation différente des méthodes demarre() et accelerer()
- Polymorphisme : tableau Vehicule[] contient différents types de véhicules
- Utilisation de LocalDate pour gérer l'année du modèle

1.2.19 Diagramme de la hiérarchie Véhicule :

Voilàle diagramme UML montrant Vehicule (classe abstraite) et ses deux classes filles :

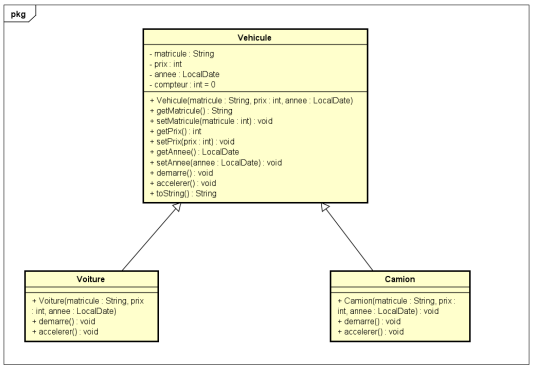

FIGURE 24 – Diagramme de la Hiérarchie Vehicule (classe abstraite) et ses deux classes filles

1.2.20 Résultat :

Le programme MainVoiture crée un parc automobile de 2 véhicules (une voiture et un camion) avec saisie interactive. Il affiche ensuite les informations de chaque véhicule via toString() et démontre le polymorphisme en appelant les méthodes demarre() et accelerer() qui se comportent différemment selon le type de véhicule.

```java
package TP3;

import java.time.LocalDate;

public abstract class Vehicule {
private String matricule;
private int prix;
private LocalDate annee;
private static int compteur = 0;
    public Vehicule(String matricule,int prix,LocalDate annee) {
        this.matricule=matricule;
        this.prix=prix;
        this.annee=annee;
        compteur ++;

    }
    public String getMatricule() {
        return matricule;
    }
    public void setMatricule(String matricule) {
        this.matricule = matricule;
    }
    public int getPrix() {
        return prix;
    }
    public void setPrix(int prix) {
        this.prix = prix;
    }
    public LocalDate getAnnee() {
        return annee;
    }
    public void setAnnee(LocalDate annee) {
        this.annee = annee;
    }
public abstract void demarre() ;
public abstract void accelerer() ;
@Override
public String toString() {
    return "Matricule : " + matricule + ", Prix : " + prix + ", Année : " + annee;
}
}
```

FIGURE 25 – La classe vehicule

```java
package TP3;

import java.time.LocalDate;

public class Voiture extends Vehicule {

    public Voiture(String matricule,int prix,LocalDate annee) {
        super(matricule,prix,annee);
    }
    @Override
    public void demarre() {
        System.out.println("La voiture " + getMatricule() + " démarre doucement.");
    }

    @Override
    public void accelerer() {
        System.out.println("La voiture " + getMatricule() + " accélère rapidement.");
    }
}
```

FIGURE 26 – la classe voiture

```java
package TP3;

import java.time.LocalDate;

public class Camion extends Vehicule {

    public Camion(String matricule,int prix,LocalDate annee) {
        super(matricule,prix,annee);
    }
    @Override
        public void demarre() {
            System.out.println("Le camion " + getMatricule() + " démarre lentement.");
        }

        @Override
        public void accelerer() {
            System.out.println("Le camion " + getMatricule() + " accélère progressivement.");
        }
}
```

FIGURE 27 – La classe camion

```java
package TP3;

import java.time.LocalDate;

public class MainVoiture {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Vehicule[] parc = new Vehicule[2];

        System.out.println("--- Création d'une Voiture ---");
        System.out.print("Matricule: ");
        String matriculeVoiture = sc.next();

        System.out.println("Date du modèle : ");
        System.out.print("Année : ");
        int anneeV = sc.nextInt();
        System.out.print("Mois : ");
        int moisV = sc.nextInt();
        System.out.print("Jour : ");
        int jourV = sc.nextInt();
        LocalDate dateVoiture = LocalDate.of(anneeV, moisV, jourV);
        System.out.print("Prix de la voiture : ");
        int prixVoiture = sc.nextInt();
        parc[0] = new Voiture(matriculeVoiture, prixVoiture,dateVoiture);
        System.out.println("\n--- Création d'un Camion ---");
        System.out.print("Matricule: ");
        String matriculeCamion = sc.next();
        System.out.println("Date du modèle : ");
        System.out.print("Année : ");
        int anneeC = sc.nextInt();
        System.out.print("Mois : ");
        int moisC = sc.nextInt();
        System.out.print("Jour : ");
        int jourC = sc.nextInt();
        LocalDate dateCamion = LocalDate.of(anneeC, moisC, jourC);
        System.out.print("Prix du camion : ");
        int prixCamion = sc.nextInt();
        parc[1] = new Camion(matriculeCamion, prixCamion,dateCamion);
        sc.close();
        System.out.println("\n--- Parc Auto ---");
        for (Vehicule v : parc) {
            System.out.println(v);
            v.demarre();
            v.accelerer();
            System.out.println("-------------------------");
        }
    }
}
```

FIGURE 28 – Main de la Hiérarchie vehicule

```
--- Création d'une Voiture ---
Matricule: audi
Date du modèle :
Année : 2005
Mois : 05
Jour : 29
Prix de la voiture : 500000

--- Création d'un Camion ---
Matricule: bmw
Date du modèle :
Année : 2005
Mois : 05
Jour : 29
Prix du camion : 400000

--- Parc Auto ---
Matricule : audi, Prix : 500000, Année : 2005-05-29
La voiture audi démarre doucement.
La voiture audi accélère rapidement.
-------------------------
Matricule : bmw, Prix : 400000, Année : 2005-05-29
Le camion bmw démarre lentement.
Le camion bmw accélère progressivement.
-------------------------
```


FIGURE 29 – Resultat de la Hiérarchie vehicule

2 Conclusion

Ce travail pratique m'a permis de maîtriser les concepts fondamentaux de la programmation orientée objet en Java. À travers quatre exercices progressifs, j'ai approfondi l'héritage simple et l'encapsulation (Rectangle/Carré, Bâtiment/Maison), la composition d'objets (Point/Segment), ainsi que l'utilisation de classes abstraites et du polymorphisme (Employé, Véhicule). Ces concepts constituent les piliers essentiels pour concevoir des applications Java robustes et maintenables, et leur application pratique sur des cas concrets a consolidé ma compréhension de l'architecture logicielle orientée objet.