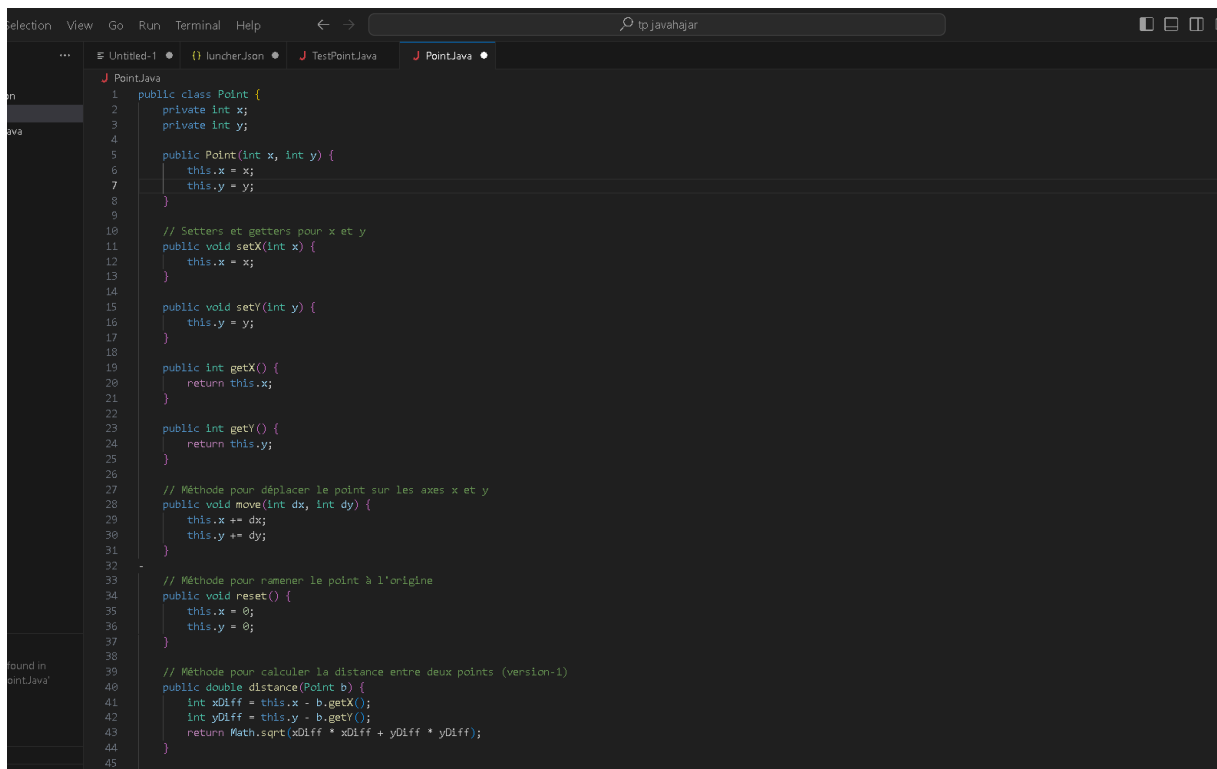


Travail à rendre : Poo Java

Réaliser par : IZIKKI Hajar

Encadrer par : M.OUKDACH

Class Point :



```
1 public class Point {
2     private int x;
3     private int y;
4
5     public Point(int x, int y) {
6         this.x = x;
7         this.y = y;
8     }
9
10    // Setters et getters pour x et y
11    public void setX(int x) {
12        this.x = x;
13    }
14
15    public void setY(int y) {
16        this.y = y;
17    }
18
19    public int getX() {
20        return this.x;
21    }
22
23    public int getY() {
24        return this.y;
25    }
26
27    // Méthode pour déplacer le point sur les axes x et y
28    public void move(int dx, int dy) {
29        this.x += dx;
30        this.y += dy;
31    }
32
33    // Méthode pour ramener le point à l'origine
34    public void reset() {
35        this.x = 0;
36        this.y = 0;
37    }
38
39    // Méthode pour calculer la distance entre deux points (version-1)
40    public double distance(Point b) {
41        int xDiff = this.x - b.getX();
42        int yDiff = this.y - b.getY();
43        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
44    }
45}
```

```

ion View Go Run Terminal Help ← → tp.javahajar

... [Untitled-1] [luncher.Json] [TestPointJava] [PointJava]
PointJava
25 }
26
27 // Méthode pour déplacer le point sur les axes x et y
28 public void move(int dx, int dy) {
29     this.x += dx;
30     this.y += dy;
31 }
32
33 // Méthode pour ramener le point à l'origine
34 public void reset() {
35     this.x = 0;
36     this.y = 0;
37 }
38
39 // Méthode pour calculer la distance entre deux points (version-1)
40 public double distance(Point b) {
41     int xDiff = this.x - b.getX();
42     int yDiff = this.y - b.getY();
43     return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
44 }
45
46 // Méthode pour calculer la distance entre deux points (version-2, méthode statique)
47 public static double distance(Point a, Point b) {
48     int xDiff = a.getX() - b.getX();
49     int yDiff = a.getY() - b.getY();
50     return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
51 }
52
53 // Redéfinition de la méthode equals pour comparer les coordonnées de deux points
54 @Override
55 public boolean equals(Object o) {
56     if (this == o) {
57         return true;
58     }
59     if (o == null || getClass() != o.getClass()) {
60         return false;
61     }
62
63     Point point = (Point) o;
64     return x == point.x && y == point.y;
65 }
66
67

```

Class TestPoint :

```

View Go Run Terminal Help ← → tp.javahajar

... [Untitled-1] [luncher.Json] [TestPointJava] [PointJava]
TestPointJava
1 public class TestPoint {
2     public static void main(String[] args) {
3         // Création de deux objets Point avec les mêmes coordonnées
4         Point p = new Point(2, 5);
5         Point q = new Point(2, 5);
6
7         // Test de la méthode equals sur les objets Point p et q
8         System.out.println("Résultat de p.equals(q) : " + p.equals(q)); // Attendu : true
9
10        // Affectation de q à la référence de p
11        q = p;
12
13        // Test de la méthode equals sur les objets Point p et q après affectation
14        System.out.println("Résultat de p.equals(q) après affectation : " + p.equals(q)); // Attendu : true
15
16        // Création d'objets Object p et q avec des instances de Point
17        Object objP = new Point(2, 5);
18        Object objQ = new Point(2, 5);
19
20        // Test de la méthode equals héritée de la classe Object
21        System.out.println("Résultat de objP.equals(objQ) : " + objP.equals(objQ)); // Attendu : false
22
23        // Redéfinition de la méthode equals héritée de la classe Object
24        // Comparaison basée sur les coordonnées des points
25        // Redéfinir equals() dans la classe Point comme indiqué ci-dessus
26
27        // Test de la méthode equals après redéfinition dans la classe Point
28        System.out.println("Résultat de objP.equals(objQ) après redéfinition : " + objP.equals(objQ)); // Attendu : true
29    }
30 }
31

```

Explication :

1. **Classe Point** : Elle définit un point en 2D avec des coordonnées x et y, des méthodes pour manipuler ces coordonnées, calculer la distance entre deux points, et redéfinit la méthode **equals** pour comparer les coordonnées de deux points pour l'égalité.
2. **Classe de test TestPoint** : Elle crée des objets **Point** et des objets **Object** avec les mêmes coordonnées, teste la méthode **equals** pour vérifier l'égalité des objets. Après redéfinition de **equals** dans la classe **Point**, elle montre comment la méthode **equals** est appelée pour les objets **Object** et montre que la comparaison se fait désormais sur les coordonnées des points.

Class rectangle :

```
C:\> Users > pc > Desktop > tp2 final > tp 22.java > J Rectangle.java > ...
1 public class Rectangle {
2     private Point topLeft; // Le coin haut gauche
3     private Point bottomRight; // Le coin bas droite
4
5     // Constructeur par défaut
6     public Rectangle() {
7         topLeft = new Point();
8         bottomRight = new Point();
9     }
10
11    // Constructeur avec les points haut gauche et bas droite
12    public Rectangle(Point h, Point b) {
13        topLeft = new Point(h);
14        bottomRight = new Point(b);
15    }
16
17    // Constructeur avec les points haut gauche et bas droite
18    /*public Rectangle(Point h, Point b) {
19        topLeft = new Point(h.getX(), h.getY());
20        bottomRight = new Point(b.getX(), b.getY());
21    }*/
22
23    // Méthode pour afficher les coordonnées des coins
24    public void afficher() {
25        System.out.println("Coin haut gauche : " + topLeft);
26        System.out.println("Coin bas droite : " + bottomRight);
27    }
28
29    // Méthode pour calculer la surface du rectangle
30    public double surface() {
31        double width = Math.abs(bottomRight.getX() - topLeft.getX());
32        double height = Math.abs(topLeft.getY() - bottomRight.getY());
33        return width * height;
34    }
35
36    // Méthode pour calculer le périmètre du rectangle
37    public double perimeter() {
38        double width = Math.abs(bottomRight.getX() - topLeft.getX());
39        double height = Math.abs(topLeft.getY() - bottomRight.getY());
40        return 2 * (width + height);
41    }
42
43    // Méthode pour effectuer un zoom
44    public void zoom(int deltax, int deltay) {
45        // Calculating the difference between x-coordinates and y-coordinates
46        double widthDiff = bottomRight.getX() - topLeft.getX();
47        double heightDiff = topLeft.getY() - bottomRight.getY();
```

```

// Méthode pour calculer le périmètre du rectangle
public double perimeter() {
    double width = Math.abs(bottomRight.getX() - topLeft.getX());
    double height = Math.abs(topLeft.getY() - bottomRight.getY());
    return 2 * (width + height);
}

// Méthode pour effectuer un zoom
public void zoom(int deltax, int deltay) {
    // Calculating the difference between x-coordinates and y-coordinates
    double widthDiff = bottomRight.getX() - topLeft.getX();
    double heightDiff = topLeft.getY() - bottomRight.getY();

    // Adjusting the width and height based on deltax and deltay
    double newWidth = widthDiff + deltax;
    double newHeight = heightDiff + deltay;

    // Updating the bottomRight coordinates to resize the rectangle
    bottomRight.setX(topLeft.getX() + newWidth);
    bottomRight.setY(topLeft.getY() - newHeight);

    topLeft.setX(topLeft.getX() - newWidth);
    topLeft.setY(topLeft.getY() + newHeight);
}

// Méthodes set et get pour modifier et lire les attributs d'un rectangle
public void setTopLeft(Point topLeft) {
    this.topLeft = new Point(topLeft.getX(), topLeft.getY());
}

public void setBottomRight(Point bottomRight) {
    this.bottomRight = new Point(bottomRight.getX(), bottomRight.getY());
}

public Point getTopLeft() {
    return new Point(topLeft.getX(), topLeft.getY());
}

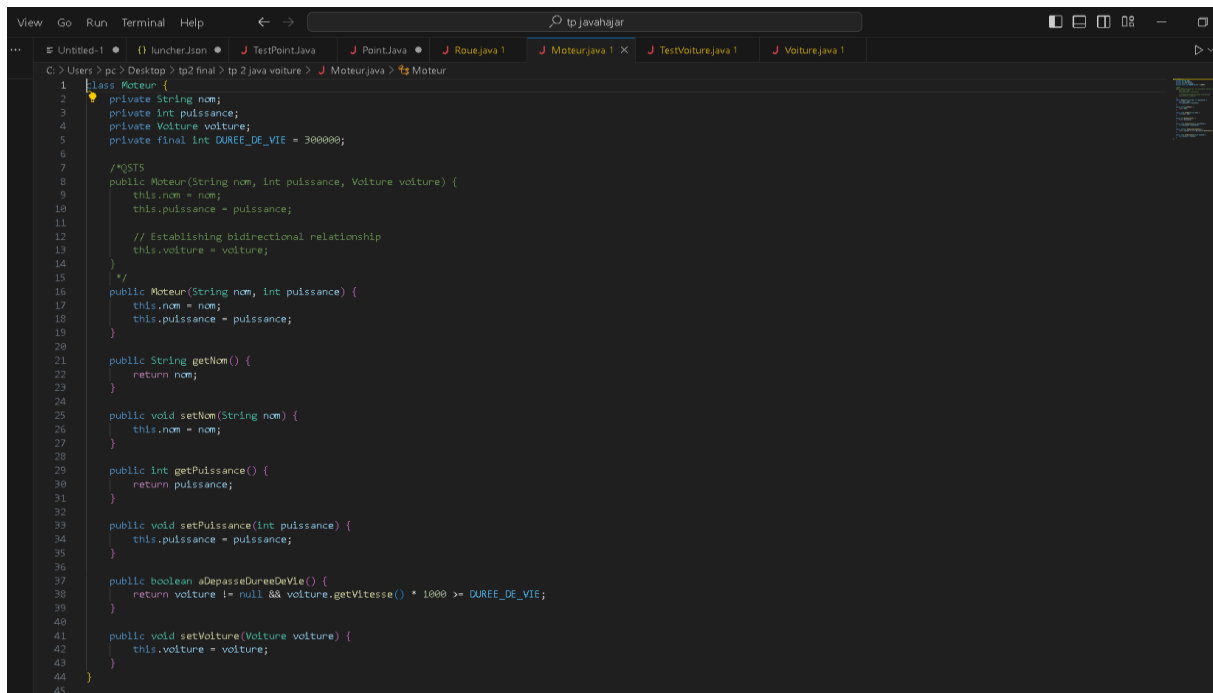
public Point getBottomRight() {
    return new Point(bottomRight.getX(), bottomRight.getY());
}

```

Explication :

La classe **Rectangle** permet de représenter un rectangle en utilisant deux points, le coin supérieur gauche et le coin inférieur droit. Les méthodes fournies permettent de calculer la surface, le périmètre et de manipuler les coins du rectangle.

Class Moteur :



```
1 class Moteur {
2     private String nom;
3     private int puissance;
4     private Voiture voiture;
5     private final int DUREE_DE_VIE = 300000;
6
7     /**
8      * public Moteur(String nom, int puissance, Voiture voiture) {
9          this.nom = nom;
10         this.puissance = puissance;
11
12         // Establishing bidirectional relationship
13         this.voiture = voiture;
14     }
15     */
16     public Moteur(String nom, int puissance) {
17         this.nom = nom;
18         this.puissance = puissance;
19     }
20
21     public String getNom() {
22         return nom;
23     }
24
25     public void setNom(String nom) {
26         this.nom = nom;
27     }
28
29     public int getPuissance() {
30         return puissance;
31     }
32
33     public void setPuissance(int puissance) {
34         this.puissance = puissance;
35     }
36
37     public boolean aDepasseDureeDeVie() {
38         return voiture != null && voiture.getVitesse() * 1000 >= DUREE_DE_VIE;
39     }
40
41     public void setVoiture(Voiture voiture) {
42         this.voiture = voiture;
43     }
44 }
45
```

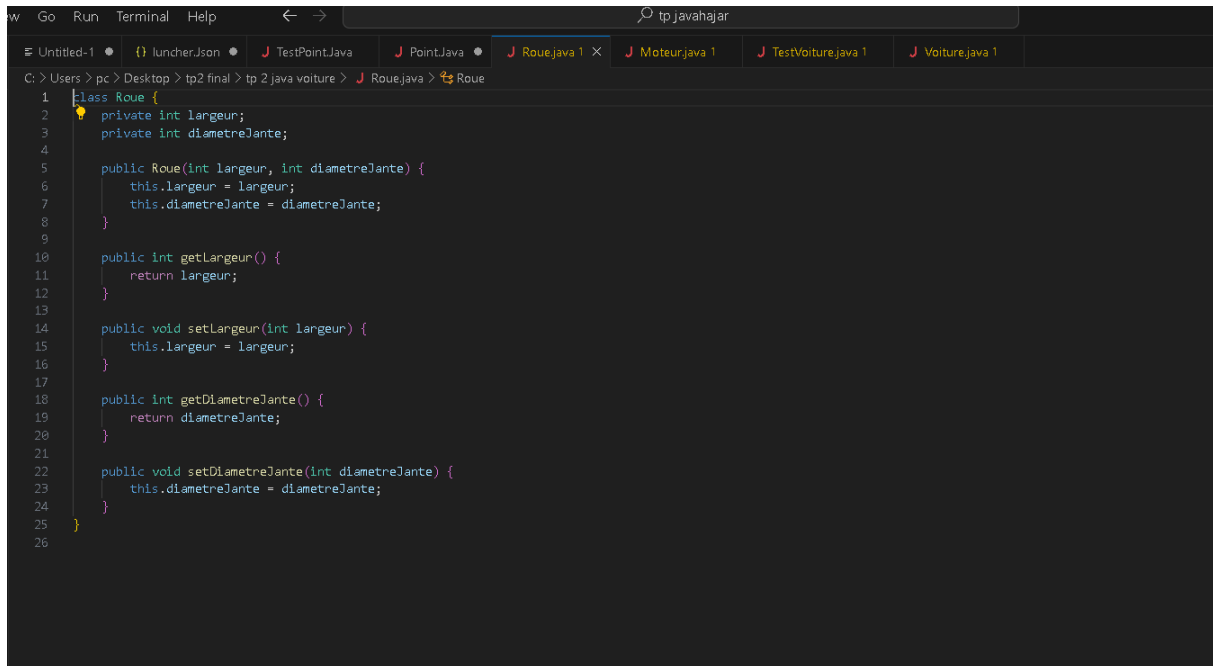
Explication des attributs et méthodes de la classe **Moteur** :

- **Attributs :**
 - **nom** : Représente le nom du moteur.
 - **puissance** : Représente la puissance du moteur.
 - **DUREE_DE_VIE** : Constante définissant la durée de vie du moteur en kilomètres. Cette valeur est définie comme constante.
- **Constructeur :**
 - **Moteur(String nom, double puissance)**: Initialise un moteur avec un nom et une puissance spécifiés.
- **Méthodes :**
 - **getNom()**: Retourne le nom du moteur.
 - **setNom(String nom)**: Modifie le nom du moteur.
 - **getPuissance()**: Retourne la puissance du moteur.
 - **setPuissance(double puissance)**: Modifie la puissance du moteur.
 - **getDureeDeVie()**: Retourne la durée de vie du moteur.

- **changerMoteur(Moteur nouveauMoteur)**: Permet de remplacer le moteur actuel par un nouveau moteur passé en paramètre. Cette méthode met à jour le nom et la puissance du moteur actuel avec ceux du nouveau moteur.

Cette classe **Moteur** est une représentation simple d'un moteur de voiture avec ses caractéristiques de base telles que le nom, la puissance et une durée de vie définie.

Class Roue :



```

1 class Roue {
2     private int largeur;
3     private int diametreJante;
4
5     public Roue(int largeur, int diametreJante) {
6         this.largeur = largeur;
7         this.diametreJante = diametreJante;
8     }
9
10    public int getLargeur() {
11        return largeur;
12    }
13
14    public void setLargeur(int largeur) {
15        this.largeur = largeur;
16    }
17
18    public int getDiametreJante() {
19        return diametreJante;
20    }
21
22    public void setDiametreJante(int diametreJante) {
23        this.diametreJante = diametreJante;
24    }
25 }
26

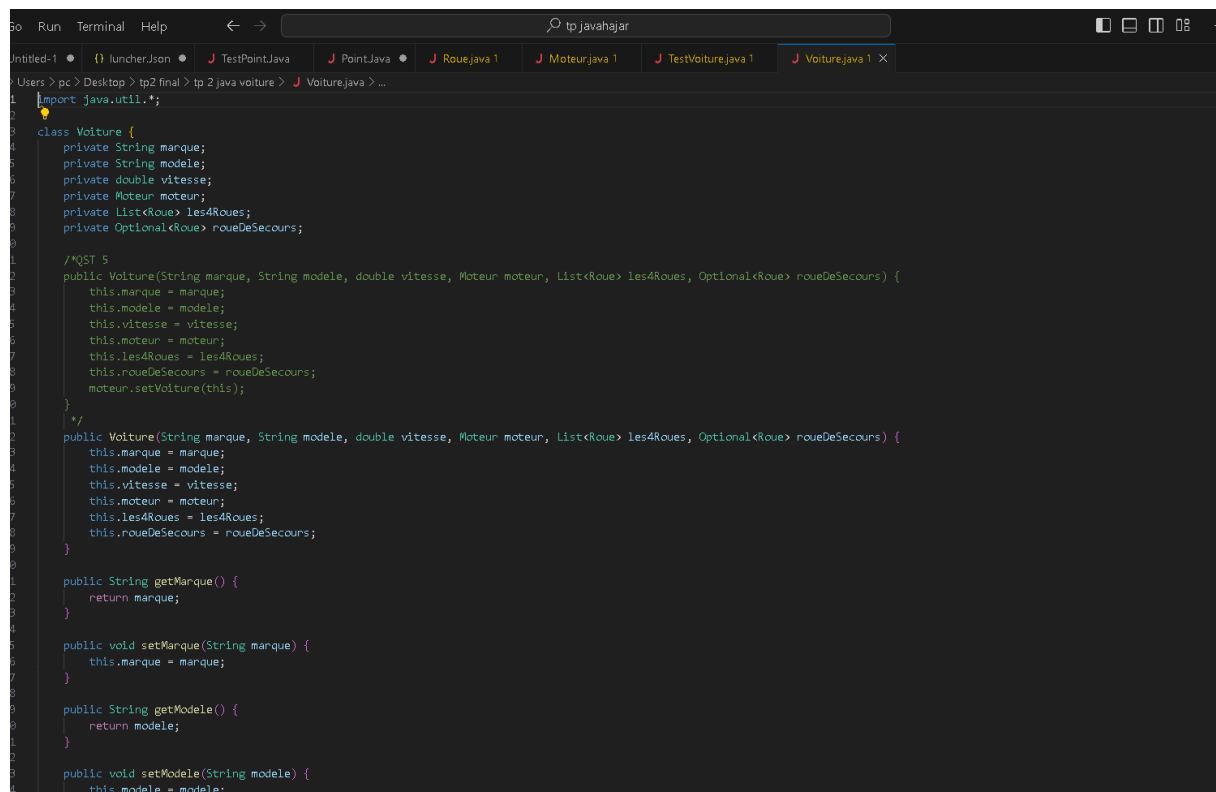
```

- **Attributs :**
 - **private int largeur** : Cet attribut représente la largeur de la roue.
 - **private int diametreJante** : Cet attribut représente le diamètre de la jante de la roue.
- **Constructeur :**
 - **public Roue(int largeur, int diametreJante)** : Ce constructeur initialise une instance de la classe **Roue** en spécifiant la largeur et le diamètre de la jante de la roue.
- **Méthodes :**
 - **public int getLargeur()** : Cette méthode permet de récupérer la valeur de la largeur de la roue.
 - **public void setLargeur(int largeur)** : Cette méthode permet de définir la valeur de la largeur de la roue.

- **public int getDiametreJante()** : Cette méthode permet de récupérer la valeur du diamètre de la jante de la roue.
- **public void setDiametreJante(int diametreJante)** : Cette méthode permet de définir la valeur du diamètre de la jante de la roue.

Ces attributs et méthodes encapsulent les caractéristiques fondamentales d'une roue, permettant ainsi de définir, d'accéder et de modifier la largeur et le diamètre de la jante de manière contrôlée à travers des méthodes spécifiques.

Class voiture :



```

1  import java.util.*;
2
3  class Voiture {
4      private String marque;
5      private String modele;
6      private double vitesse;
7      private Moteur moteur;
8      private List<Roue> les4Roues;
9      private Optional<Roue> roueDeSecours;
10
11      /*QST 5
12      public Voiture(String marque, String modele, double vitesse, Moteur moteur, List<Roue> les4Roues, Optional<Roue> roueDeSecours) {
13          this.marque = marque;
14          this.modele = modele;
15          this.vitesse = vitesse;
16          this.moteur = moteur;
17          this.les4Roues = les4Roues;
18          this.roueDeSecours = roueDeSecours;
19          moteur.setVoiture(this);
20      }
21      */
22      public Voiture(String marque, String modele, double vitesse, Moteur moteur, List<Roue> les4Roues, Optional<Roue> roueDeSecours) {
23          this.marque = marque;
24          this.modele = modele;
25          this.vitesse = vitesse;
26          this.moteur = moteur;
27          this.les4Roues = les4Roues;
28          this.roueDeSecours = roueDeSecours;
29      }
30
31      public String getMarque() {
32          return marque;
33      }
34
35      public void setMarque(String marque) {
36          this.marque = marque;
37      }
38
39      public String getModele() {
40          return modele;
41      }
42
43      public void setModele(String modele) {
44          this.modele = modele;
45      }

```

```

View Go Run Terminal Help
tp.javahajar

F Untitled-1 • {} luncher.json • J TestPoint.java J Point.java J Roue.java 1 J Moteur.java 1 J TestVoiture.java 1 J Voiture.java 1
C:\Users\pc\Desktop> tp2\final> tp2\java voiture > J Voiture.java > J Voiture.java

45
46
47 public double getVitesse() {
48     return vitesse;
49 }
50
51 public void setVitesse(double vitesse) {
52     this.vitesse = vitesse;
53 }
54
55 public Moteur getMoteur() {
56     return moteur;
57 }
58
59 public void setMoteur(Moteur moteur) {
60     this.moteur = moteur;
61 }
62
63 public List<Roue> getLes4Roues() {
64     return les4Roues;
65 }
66
67 public void setLes4Roues(List<Roue> les4Roues) {
68     this.les4Roues = les4Roues;
69 }
70
71 public Optional<Roue> getRoueDeSecours() {
72     return roueDeSecours;
73 }
74
75 public void setRoueDeSecours(Optional<Roue> roueDeSecours) {
76     this.roueDeSecours = roueDeSecours;
77 }
78
79 public void demarrer() {
80     System.out.println("La voiture démarre");
81 }
82
83 public void accelerer(double acceleration) {
84     System.out.println("Acceleration: " + acceleration + " km/h");
85 }
86
87 public int deQuellePuissance() {
88     return moteur.getPuissance();
89 }
90
91 public void changerLeMoteur(Moteur newMoteur) {
92     if (moteur != null && moteur.aDepasseDureeDeVie()) {
93         moteur = newMoteur;
94         System.out.println("Le moteur est changé.");
95     } else {
96         System.out.println("Le moteur ne peut pas être changé.");
97     }
98 }
99
100

```

Ce code représente la classe **Voiture**, qui est une modélisation d'un véhicule. Voici une explication détaillée de cette classe :

- **Attributs :**
 - **private String marque** : Cet attribut représente la marque de la voiture.
 - **private String modele** : Cet attribut représente le modèle de la voiture.
 - **private double vitesse** : Cet attribut représente la vitesse actuelle de la voiture.
 - **private Moteur moteur** : Cet attribut représente le moteur de la voiture (de type **Moteur**).
 - **private List<Roue> les4Roues** : Cet attribut représente une liste de quatre roues de la voiture (de type **List<Roue>**).
 - **private Optional<Roue> roueDeSecours** : Cet attribut représente une roue de secours optionnelle pour la voiture (de type **Optional<Roue>**).
- **Constructeur :**
 - **public Voiture(String marque, String modele, double vitesse, Moteur moteur, List<Roue> les4Roues, Optional<Roue> roueDeSecours)** : Ce constructeur initialise une instance de la classe **Voiture** en initialisant tous les attributs de la voiture avec les valeurs passées en paramètres.

- **Méthodes :**

- **getters et setters** : Ces méthodes permettent d'accéder et de modifier les attributs de la voiture de manière contrôlée.
- **demarrer()** : Cette méthode simule le démarrage de la voiture en affichant un message.
- **accellerer(double acceleration)** : Cette méthode simule l'accélération de la voiture en affichant un message avec la valeur de l'accélération.
- **deQuellePuissance()** : Cette méthode retourne la puissance du moteur associé à la voiture.
- **changerLeMoteur(Moteur newMoteur)** : Cette méthode permet de changer le moteur de la voiture par un nouveau moteur si le moteur actuel a dépassé sa durée de vie. Un message approprié est affiché selon la situation.

La classe **Voiture** est conçue pour représenter un véhicule avec des caractéristiques telles que sa marque, son modèle, sa vitesse, son moteur, ses roues et une éventuelle roue de secours, ainsi que des méthodes pour interagir avec ses composants.