La programmation orientée objet avancée

Exercice 1 : Héritage, classes abstraites et interfaces

Implanter les classes Vehicule, Voiture et Train telles qu'elles ont été partiellement vues en cours.

1. Manipuler toutes les formes possibles de types pour les références et de type pour l'instance. Conclure quant aux manières possibles de référencer un objet.
2. Surcharger la méthode rouler() en lui ajoutant en paramètre un nombre de kilomètres.
3. Redéfinir la méthode rouler() dans les classes Train et Voiture. Illustrer cette redéfinition par le biais de tests.

```
package fr.utt.sit.lo02.td3.exercice1;

public class Vehicule {

    public final int capaciteReservoir = 50;
    public final double consommation = 0.1;

    private int kilometres;
    private double essence;
    private boolean roule;

    public Vehicule (int kilometres) {
      this.essence = capaciteReservoir;
      this.kilometres = kilometres;
      this.roule = false;
    }

    public Vehicule () {
      this.essence = capaciteReservoir;
      this.kilometres = 0;
      this.roule = false;
    }

    public void rouler() {
      System.out.println("Le vehicule roule...");
      this.roule = true;
      while (this.roule && this.essence > 0) {
          this.kilometres++;
          this.essence -= this.consommation;
      }
    }

    public void rouler(int kilometres) {
      System.out.println("Le vehicule roule...");
      this.roule = true;
      int kilometresParcourus = 0;
      while (this.roule && this.essence > 0 && kilometresParcourus <=
kilometres) {
          this.kilometres++;
          this.essence -= this.consommation;
          kilometresParcourus++;
      }
    }

    public void stopper() {
      this.roule = false;
    }

    public static void main(String[] args) {
```

```
        Vehicule[] vehicules = new Vehicule[3];

        Vehicule vehicule = new Vehicule();
        Voiture voiture = new Voiture();
        Train train = new Train();

        vehicules[0] = vehicule;
        vehicules[1] = voiture;
        vehicules[2] = train;

        for (int i = 0; i < vehicules.length; i++) {
            vehicules[i].rouler();
        }
    }
}
```

```
package fr.utt.sit.lo02.td3.exercice1;

public class Voiture extends Vehicule {

    public Voiture(int kilometres) {
      super(kilometres);
    }

    public Voiture() {
      super();
    }

    public void rouler() {
      System.out.println("La voiture roule...");
      super.rouler();
    }

    public void rouler(int kilometres) {
      System.out.println("La voiture roule...");
      super.rouler(kilometres);
    }
}
```

```
package fr.utt.sit.lo02.td3.exercice1;

public class Train extends Vehicule {

    public Train (int kilometres) {
      super(kilometres);
      // TODO Auto-generated constructor stub
    }

    public Train () {
      super();
      // TODO Auto-generated constructor stub
    }

    public void rouler() {
      System.out.println("Le train roule...");
      super.rouler();
    }

    public void rouler(int kilometres) {
      System.out.println("Le train roule...");
```

```
        super.rouler(kilometres);
    }
}
```

4. Implanter la méthode rouler() sous forme de méthode abstraite dans la classe Vehicule. Conclure quant à la redéfinition précédente.

```
package fr.utt.sit.lo02.td3.exercice1;

public abstract class Vehicule {

    public final int capaciteReservoir = 50;
    public final double consommation = 0.1;

    private int kilometres;
    private double essence;
    private boolean roule;

    public Vehicule (int kilometres) {
      this.essence = capaciteReservoir;
      this.kilometres = kilometres;
      this.roule = false;
    }

    public Vehicule () {
      this.essence = capaciteReservoir;
      this.kilometres = 0;
      this.roule = false;
    }

    public abstract void rouler();

    public abstract void rouler(int kilometres);

    public void stopper() {
      this.roule = false;
    }
}
```

5. Définir une interface Pilotable qui permet de faire accélérer ou ralentir un Véhicule. Implanter cette interface.

```
package fr.utt.sit.lo02.td3.exercice1;

public interface Pilotable {

    public void accelerer();
    public void ralentir();

}
```

```
package fr.utt.sit.lo02.td3.exercice1;

public class Vehicule implements Pilotable {
//  public abstract class Vehicule implements Pilotable {

    public final int capaciteReservoir = 50;
    public final double consommation = 0.1;
```

```
    private int kilometres;
    private double essence;
    private boolean roule;
    private int vitesse;


    public Vehicule (int kilometres) {
      this.essence = capaciteReservoir;
      this.kilometres = kilometres;
      this.roule = false;
      this.vitesse = 0;
    }

    public Vehicule () {
      this.essence = capaciteReservoir;
      this.kilometres = 0;
      this.roule = false;
    }

    public void accelerer() {
      vitesse += 1;
      if (roule == false) {
          this.rouler();
      }
    }

    public void ralentir() {
      if (vitesse > 0) {
          vitesse-=1;
      }

      if (this.roule == true && vitesse == 0) {
          this.stopper();
      }
    }


    public void rouler() {
      System.out.println("Le vehicule roule...");
      this.roule = true;
      while (this.roule && this.essence > 0) {
          this.kilometres += this.vitesse;
          this.essence -= this.consommation * this.vitesse;
      }
    }

//  public abstract void rouler();

    public void rouler(int kilometres) {
      System.out.println("Le vehicule roule...");
      this.roule = true;
      int kilometresParcourus = 0;
      while (this.roule && this.essence > 0 && kilometresParcourus <=
kilometres) {
          this.kilometres += this.vitesse;
          this.essence -= this.consommation * this.vitesse;
          kilometresParcourus += this.vitesse;
      }
    }

    public void stopper() {
```

```
        this.roule = false;
    }

    public String toString() {
        StringBuffer sb = new StringBuffer ("Le vehicule ");
        if (this.roule == true) {
            sb.append("est à l'arret.");
        } else {
            sb.append("roule à la vitesse de ");
            sb.append(this.vitesse);
            sb.append(". ");
        }
        sb.append("Il a parcouru ");
        sb.append(this.kilometres);
        sb.append("kms.");
        return sb.toString();
    }
```

6. Définir une interface Orientable qui permet de faire tourner à gauche ou à droite une un Véhicule. Implanter cette interface. Est-ce qu'un Train est orientable ?

```
package fr.utt.sit.lo02.td3.exercice1;

public interface Orientable {

    public static final String[] Orientations = {"Nord", "Nord-Est", "Est",
"Sud-Est", "Sud", "Sud-Ouest", "Ouest", "Nord-Ouest"};

    public void tournerAGauche();
    public void tournerADroite();
}
```

```
package fr.utt.sit.lo02.td3.exercice1;

public class Voiture extends Vehicule implements Orientable {

    private int orientation;

    public Voiture(int kilometres) {
        super(kilometres);
    }

    public Voiture() {
        super();
        orientation = 0;
    }

    public void tournerAGauche() {
        orientation = (orientation + 1) % Orientations.length;
    }

    public void tournerADroite() {
        orientation = (orientation + Orientations.length - 1) %
Orientations.length;
    }

    public void rouler() {
        System.out.println("La voiture roule...");
//      super.rouler();
```

```
    }

    public void rouler(int kilometres) {
      System.out.println("La voiture roule...");
//    super.rouler(kilometres);
    }
}
```

7. Commander des instances de voitures et de train par leur interface Pilotable et Orientable.

Exercice 2 : Classes internes et anonymes

1. On désire maintenant prendre en compte la notion de moteur dans une voiture. Etant donné qu'un moteur n'a de sens que dans le cadre d'une voiture, on propose de l'implanter sous forme de classe interne (d'autres choix de modélisation seraient possibles). Implanter la classe interne Moteur.

```
public abstract class Vehicule implements Pilotable {

    public final int capaciteReservoir = 50;
    public final double consommation = 0.1;

    protected double essence;
    protected boolean roule;

    public Vehicule (int kilometres) {
      this.essence = capaciteReservoir;
      this.roule = false;
    }

    public Vehicule () {
      this.essence = capaciteReservoir;
      this.roule = false;
    }

    public abstract void rouler();

    public abstract void rouler(int kilometres);

    public void stopper() {
      this.roule = false;
    }

    public String toString() {
      StringBuffer sb = new StringBuffer ("Le vehicule a");
      sb.append(this.essence);
      sb.append("litres de carburant dans son réservoir. ");

      return sb.toString();
    }
}
```

```
public class VoitureAMoteur extends Vehicule implements Orientable {

    private int orientation;
    protected Moteur moteur;

    public class Moteur {
      private int kilometres;
```

```
      private int vitesse;

      public Moteur (int kilometres) {
          this.kilometres = kilometres;
          this.vitesse = 0;
      }

      public int getKilometres() {
          return kilometres;
      }

      public void ajouterKilometres(int kilometres) {
          this.kilometres += kilometres;
      }

      public int getVitesse() {
          return vitesse;
      }

      public void augmenterVitesse(int vitesse) {
          this.vitesse += vitesse;
      }

      public void diminuerVitesse(int vitesse) {
          this.vitesse -= vitesse;
      }
    }

    public VoitureAMoteur (int kilometres) {
      super();
      moteur = new Moteur(kilometres);
    }

    public VoitureAMoteur () {
      super();
      moteur = new Moteur(0);
    }

    @Override
    public void tournerADroite() {
      orientation = (orientation + 1) % Orientations.length;
    }

    @Override
    public void tournerAGauche() {
      orientation = (orientation + Orientations.length - 1) %
Orientations.length;
    }

    public void accelerer() {
      moteur.augmenterVitesse(1);
      if (super.roule == false) {
          this.rouler();
      }
    }

    public void ralentir() {
      if (moteur.getVitesse() > 0) {
          moteur.diminuerVitesse(1);
      }
```

```
      if (super.roule == true && moteur.getVitesse() == 0) {
          this.stopper();
      }
    }

    public void rouler() {
      System.out.println("La voiture roule...");
      this.roule = true;
      while (this.roule && this.essence > 0) {
          moteur.ajouterKilometres(moteur.getVitesse());
          this.essence -= this.consommation * moteur.getVitesse();
      }
    }

    public void rouler(int kilometres) {
      System.out.println("La voiture roule...");
      this.roule = true;
      int kilometresParcourus = 0;
      while (this.roule && this.essence > 0 && kilometresParcourus <=
kilometres) {
          moteur.ajouterKilometres(moteur.getVitesse());
          this.essence -= this.consommation * moteur.getVitesse();
          kilometresParcourus += moteur.getVitesse();
      }
    }

    public String toString() {
      StringBuffer sb = new StringBuffer ("Le vehicule ");
      if (super.roule == true) {
          sb.append("est à l'arret.");
      } else {
          sb.append("roule à la vitesse de ");
          sb.append(moteur.getVitesse());
          sb.append(". ");
      }

      sb.append("Le vehicule a parcouru ");
      sb.append(moteur.getKilometres());
      sb.append("kms. ");

      return sb.toString() + super.toString();
    }
}
```

2. Dans notre application, on désire créer une voiture unique dont l'implantation ne suit pas celle de la classe Voiture. Pour cela, on crée une classe anonyme, héritant de la classe Voiture.

```
    public static void main(String[] args) {

      VoitureAMoteur voitureAnonyme = new VoitureAMoteur () {
          public void rouler() {
            System.out.println("La voiture roule sans contrainte
d'essence...");
            this.roule = true;
            while (this.roule) {
                moteur.ajouterKilometres(moteur.getVitesse());
                this.essence -= this.consommation * moteur.getVitesse();
            }
          }
```

```
            public void rouler(int kilometres) {
              System.out.println("La voiture roule sans contraintes
d'essence...");
              this.roule = true;
              int kilometresParcourus = 0;
              while (this.roule && kilometresParcourus <= kilometres) {
                  moteur.ajouterKilometres(moteur.getVitesse());
                  this.essence -= this.consommation * moteur.getVitesse();
                  kilometresParcourus += moteur.getVitesse();
              }
            }

      };
}
```