Les flux

Exercice 1 : Interaction avec l'utilisateur : la classe Console

L'objectif de cet exercice est d'écrire une classe *Console* qui permette d'encapsuler l'ensemble des mécanismes de lecture et d'écriture liés à l'entrée et la sortie standard tout en encapsulant les différentes exceptions qui peuvent être levées. La spécification de cette classe vous est donnée :

```
public class Console {

    private final static String COMMANDE_QUITTER = "quitter";

    private InputStream input;
    private PrintStream output;

    public Console();
    public Console (InputStream in, OutputStream out);

    public void afficher (String message);
    public int lireInt();
    public String lireChaine();

    public void echo();
}
```

1. Ecrire les méthodes de lecture de types simples et chaines de caractères.
2. Ecrire une méthode d'affichage de chaine de caractères.
3. Illustrer le fonctionnement de cette console avec une méthode *echo()* qui, de manière infinie, affiche à l'écran une chaine saisie par l'utilisateur. Une chaine prédéfinie permettra de terminer la méthode *echo()*.

```
package fr.utt.sit.lo02.td.flux.console;

import java.io.*;

public class Console {

    private final static String COMMANDE_QUITTER = "quitter";

    private InputStream input;
    private PrintStream output;

    public Console() {
      this.input = System.in;
      this.output = System.out;
    }

    public Console (InputStream in, OutputStream out) {
      this.input = in;
      this.output = new PrintStream(out);
    }

    public void afficher (String message) {
      output.println(message);
    }

    public int lireInt() {

      BufferedReader br = new BufferedReader(new InputStreamReader(input));
      String chaine = null;
```

```
      try {
          chaine = br.readLine();
      } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
      }

      int resultat = Integer.parseInt(chaine);

/* ALTERNATIVE A FAIRE FAIRE AUX ETUDIANTS EN PREMIER CHOIX POUR EXPLIQUER
 * POURQUOI CA NE FONCTIONNE PAS
 * Explication : On lit certes en mode octet, mais on lit des données qui
 * sont des caractères encodés. Si on utilise DataInputStream, les 4
 * premiers octets lus (soit l'encodage Unicode des 2 premiers caractères)
 * seront traduits en un entier, ce qui n'a pas de sens. Au delà, on voit
 * que même en cas de mauvaise utilisation d'une combinaison flux/filtres
 * tel que c'est le cas ici, Java ne lève aucune exeption. C'est donc de
 * la responsabilité du développeur de s'assurer de la signification et de
 * la cohérence des données lues ou écrites.
      DataInputStream dis = new DataInputStream (input);
      int resultat = 0;

      try {
          resultat = dis.readInt();
      } catch (IOException e) {
          this.afficher(e.getMessage());
      }*/
      return resultat;
    }


    public String lireChaine() {
      BufferedReader br = new BufferedReader (new
InputStreamReader(input));
      String resultat = null;
      try {
          resultat = br.readLine();
      } catch (IOException e) {
          this.afficher(e.getMessage());
      }
      return resultat;
    }

    public void echo() {
      this.afficher("Bienvenue dans la console !");

      String saisie = null;

      do {
          this.afficher("Veuillez saisir une chaine de caractËres (taper "
+ Console.COMMANDE_QUITTER + " pour quitter) :");
          saisie = this.lireChaine();

          if (saisie != null) {
            this.afficher("Vous avez saisi : " + saisie);
          } else {
            this.afficher("Vous n'avez rien saisi ou une erreur s'est
produite...");
          }
      } while (saisie.equals(Console.COMMANDE_QUITTER) == false);

      this.afficher("Au revoir !");
```

```
    }

    public static void main (String[] args) {

      Console console = new Console();
      int i = console.lireInt();

      System.out.println(i);

      console.echo();
    }
}
```

Exercice 2 : Rendre les voitures persistantes

On désire pouvoir sauvegarder et charger des voitures, telles qu'elles ont été définies lors des précédents TD. Pour cela trois formats de sauvegarde sont possibles : une sauvegarde textuelle, une sauvegarde binaire et la sérialisation.

1. Proposer deux méthodes statiques de la classe *Voiture* qui permettent d'effectuer ces opérations de sauvegarde et chargement pour les trois formats données ci-dessus.
2. Quel choix de format vous semble le plus judicieux ? Pour quelle utilisation ?

```
package fr.utt.sit.lo02.flux.voiturepersistante;

import java.io.*;

public class Vehicule implements Pilotable, Serializable {

    private static final long serialVersionUID = 1L;

    public final static int TEXT_OUTPUT = 0;
    public final static int BINARY_OUTPUT = 1;
    public final static int SERIALIZED_OUTPUT = 2;

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

    private double essence;
    private boolean roule;
    private Moteur moteur;

    public class Moteur implements Serializable {

      private static final long serialVersionUID = 1L;
      private int kilometres;
      private int vitesse;

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

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

      public void setVitesse(int vitesse) {
          this.vitesse = vitesse;
      }
```

```
    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 Vehicule (int kilometres) {
    this.essence = capaciteReservoir;
    this.roule = false;
    moteur = new Moteur(kilometres);
}

public Vehicule () {
    this.essence = capaciteReservoir;
    this.roule = false;
    moteur = new Moteur(0);
}

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

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

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


public void rouler() {
    System.out.println("Le vehicule 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("Le vehicule 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 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(moteur.getVitesse());
          sb.append(". ");
      }
      sb.append("Il a parcouru ");
      sb.append(moteur.getKilometres());
      sb.append("kms.");
      return sb.toString();
    }

    public double getEssence() {
      return essence;
    }

    public void setEssence(double essence) {
      this.essence = essence;
    }

    public Moteur getMoteur() {
      return moteur;
    }

    public void setMoteur(Moteur moteur) {
      this.moteur = moteur;
    }

    public static void saveObject (Vehicule vehicule, String fileName, int
output) {

      try {
          switch(output) {
          case Vehicule.TEXT_OUTPUT:
            FileWriter fw = new FileWriter(fileName);
            PrintWriter pw = new PrintWriter(fw);

            pw.println (vehicule.getEssence());
            pw.println(vehicule.getMoteur().getKilometres());
            pw.println(vehicule.getMoteur().getVitesse());

            fw.close();
```

```
            break;
          case Vehicule.BINARY_OUTPUT:
            FileOutputStream fos = new FileOutputStream(fileName);

            DataOutputStream dos = new DataOutputStream (fos);

            dos.writeDouble(vehicule.getEssence());
            dos.writeInt(vehicule.getMoteur().getKilometres());
            dos.writeInt(vehicule.getMoteur().getVitesse());
            fos.close();
            break;
          case Vehicule.SERIALIZED_OUTPUT:
            fos = new FileOutputStream(fileName);
            ObjectOutputStream oos = new ObjectOutputStream (fos);

            oos.writeObject(vehicule);
            fos.close();
            break;
          default:
            System.err.println("Cannot save object. Unknown output format
(" + output + ").");
          }

      } catch (IOException e) {
          e.printStackTrace();
      }
    }

    public static Vehicule loadObject (String fileName, int output) {

      Vehicule vehicule = new Vehicule();

      try {
          switch(output) {
          case Vehicule.TEXT_OUTPUT:
            FileReader fr = new FileReader(fileName);
            BufferedReader br = new BufferedReader(fr);

            double essence = Double.parseDouble(br.readLine());
            int kilometres = Integer.parseInt(br.readLine());
            int vitesse = Integer.parseInt(br.readLine());

            vehicule.setEssence(essence);
            vehicule.getMoteur().setKilometres(kilometres);
            vehicule.getMoteur().setVitesse(vitesse);

            fr.close();
            break;
          case Vehicule.BINARY_OUTPUT:
            FileInputStream fis = new FileInputStream(fileName);
            DataInputStream dis = new DataInputStream (fis);

            vehicule.setEssence(dis.readDouble());
            vehicule.getMoteur().setKilometres(dis.readInt());
            vehicule.getMoteur().setVitesse(dis.readInt());

            fis.close();
            break;
          case Vehicule.SERIALIZED_OUTPUT:
            fis = new FileInputStream(fileName);
            ObjectInputStream ois = new ObjectInputStream (fis);
```

```
            try {
                vehicule = (Vehicule) ois.readObject();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            fis.close();
            break;
          default:
            System.err.println("Cannot load object. Unknown output format
(" + output + ").");
          }

      } catch (IOException e) {
          e.printStackTrace();
      }

      return vehicule;
    }


    public static void main(String[] args) {

      Vehicule vEcrit = new Vehicule(1000);
      Vehicule vLu;

      Vehicule.saveObject(vEcrit, "v.txt", Vehicule.TEXT_OUTPUT);
      vLu = Vehicule.loadObject("v.txt", Vehicule.TEXT_OUTPUT);
      System.out.println(vLu);

      Vehicule.saveObject(vEcrit, "v.bin", Vehicule.BINARY_OUTPUT);
      vLu = Vehicule.loadObject("v.bin", Vehicule.BINARY_OUTPUT);
      System.out.println(vLu);

      Vehicule.saveObject(vEcrit, "v.ser", Vehicule.SERIALIZED_OUTPUT);
      vLu = Vehicule.loadObject("v.ser", Vehicule.SERIALIZED_OUTPUT);
      System.out.println(vLu);

    }
}
```