Les collections

Exercice 1 : Prise en main

L'objectif de cet exercice est de manipuler très basiquement les collections. Pour cela, les objets stockés dans les collections utilisées seront du type Integer. Créer des collections de différents types qui stockent des instances de la classe Integer. Montrer les différences entre ces différentes classes de collections.

```
package fr.utt.sit.lo02.collections.pem;

import java.util.*;

public class TestCollections {

    public static void main (String[] args) {
      Integer un = new Integer(1);
      Integer deux = new Integer(2);
      Integer trois = new Integer(3);

      // Test Elements non dupliqués
      HashSet<Integer> hs = new HashSet<Integer>();

      hs.add(un);
      hs.add(deux);
      hs.add(trois);

      // Cet élément ne sera pas rajouté
      hs.add(un);

      for (Iterator it = hs.iterator(); it.hasNext(); ) {
          Integer itg = (Integer) it.next();
          System.out.println(itg);
      }

      ArrayList<Integer> al = new ArrayList<Integer>();

      al.add(un);
      al.add(deux);
      al.add(trois);
      al.add(un);

      for (Iterator it = al.iterator(); it.hasNext(); ) {
          Integer itg = (Integer) it.next();
          System.out.println(itg);
      }

      String unStr = "Un";
      String deuxStr = "Deux";
      String troisStr = "Trois";

      HashMap<String,Integer> hm = new HashMap<String,Integer>();
      hm.put(unStr, un);
      hm.put(deuxStr, deux);
      hm.put(troisStr, trois);

      for (Iterator<String> it = hm.keySet().iterator(); it.hasNext(); ) {
          String key = it.next();
          System.out.println(hm.get(key));
      }
```

```
    }
}
```

Exercice 2 : Enumérations

Modifier la classe Carte (disponible sur Moodle) en utilisant 2 énumérations : une pour les valeurs des cartes (appelée Valeur), et une pour les couleurs des cartes (appelée Couleur).
On considère que la carte la plus faible est le 7, et la plus forte l'as.
On considère que les couleurs des cartes peuvent être ordonnées par ordre croissant : Trêfle, Carreau, Cœur et Pique

```
package fr.utt.sit.lo02.td.collections.bataille;

public enum Couleur {
    TREFLE,
    CARREAU,
    COEUR,
    PIQUE
}
```

```
package fr.utt.sit.lo02.td.collections.bataille;

public enum Valeur {
    SEPT,
    HUIT,
    NEUF,
    DIX,
    VALET,
    DAME,
    ROI,
    AS
}
```

```
package fr.utt.sit.lo02.td.collections.bataille;

public class Carte {

    private Couleur couleur;
    private Valeur valeur;

    public Carte (Valeur valeur, Couleur couleur) {
      this.couleur = couleur;
      this.valeur = valeur;
    }

    public Couleur getCouleur() {
      return couleur;
    }

    public void setCouleur(Couleur couleur) {
      this.couleur = couleur;
    }

    public Valeur getValeur() {
      return valeur;
    }

    public void setValeur(Valeur valeur) {
/     this.valeur = valeur;
    }
```

```
    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append(this.valeur.toString());
      sb.append(" de ");
      sb.append(this.couleur.toString());
      return sb.toString();
    }
}
```

Exercice 3 : Jeu de bataille

L'objectif de cet exercice est de créer un jeu de bataille simplifié. Les règles du jeu sont simples :

- Le jeu est composé de 32 cartes ;
- Chaque joueur joue à chaque tour la carte du dessus de son tas ;
- La carte qui remporte un tour est la plus forte. En cas d'égalité c'est celle qui a été posée la première qui remporte le tour ;
- Les cartes remportées par un joueur sont placées à la fin de son tas ;
- Le joueur gagnant est celui qui possède toutes les cartes ;
- Un joueur perd s'il ne possède plus de carte en main.

On utilisera pour la suite des questions la classe Carte et les énumérations Valeur et Couleur implémentées dans l'exercice 2.

1. Créer une classe JeuDeCartes qui représente le des cartes qui seront jouées durant une partie. Le jeu doit être mélangeable et pour distribuer les cartes, on doit pouvoir prendre la carte située au dessus.

```
package fr.utt.sit.lo02.td.collections.bataille;

import java.util.LinkedList;

public class JeuDeCartes {

    public final static int NOMBRE_DE_CARTES = Valeur.values().length *
Couleur.values().length;

    private LinkedList<Carte> tasDeCartes;

    public JeuDeCartes() {

      tasDeCartes = new LinkedList<Carte>();

      for (Couleur c : Couleur.values()) {
          for (Valeur v : Valeur.values()) {
            Carte carte = new Carte(v, c);
            //System.out.println(carte);
            tasDeCartes.add(carte);
          }
      }
    }

    public void melanger() {
      for (int i = 0; i < JeuDeCartes.NOMBRE_DE_CARTES; i++) {
            int position = (int)
Math.round((JeuDeCartes.NOMBRE_DE_CARTES - 1) * Math.random());
            Carte carte = tasDeCartes.pop();
            tasDeCartes.add(position, carte);
```

```
    }
  }

  public Carte tirerCarteDuDessus() {
    return tasDeCartes.pop();
  }

  public Carte tirerCarte() {
    int position = (int) Math.round((JeuDeCartes.NOMBRE_DE_CARTES - 1)
* Math.random());
    return tasDeCartes.remove(position);
  }

  public boolean estVide() {
    return tasDeCartes.isEmpty();
  }

  public String toString() {
    return tasDeCartes.toString();
  }
}
```

2. Créer une classe Joueur. Ce dernier est identifié par son nom et possède dans sa main les cartes qu'il peut jouer.

```
package fr.utt.sit.lo02.collections.bataille;

import java.util.LinkedList;

public class Joueur {

  private String nom;
  private LinkedList<Carte> main;

  private Carte derniereCarteJouee;

  public Joueur(String nom) {
    this.nom = nom;
    main = new LinkedList<Carte>();
  }

  public void prendreCarte(Carte carte) {
    main.add(carte);
  }

  public Carte jouerCarte() {
    dernieCarteJouee = main.pop();
    return dernieCarteJouee;
  }

  public Carte dernieCarteJouee() {
    return dernieCarteJouee;
  }

  public boolean aGagne() {
    if (main.size() == JeuDeCartes.NOMBRE_DE_CARTES) {
        return true;
    } else {
        return false;
    }
```

```
    }

    public boolean aPerdu() {
      if (main.size() == 0) {
          return true;
      } else {
          return false;
      }
    }


    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String toString() {
      StringBuffer sb = new StringBuffer();
      sb.append(nom);
      sb.append(" : ");
      sb.append(main);
      sb.append("\n");
      return sb.toString();
    }
}
```

3. Créer une classe PartieDeCartes qui gère l'ensemble d'une partie (joueurs, tours de jeu, fin de partie, gestion des perdants, …).

```
package fr.utt.sit.lo02.td.collections.bataille;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class PartieDeCartes {

    private ArrayList<Joueur> joueurs;
    private JeuDeCartes cartes;
    private boolean partieEnCours;

    public PartieDeCartes() {
      joueurs = new ArrayList<Joueur>();
      cartes = new JeuDeCartes();
      cartes.melanger();
      partieEnCours = false;
    }

    public void ajouterJoueur(Joueur joueur) {
      if (partieEnCours == false) {
          joueurs.add(joueur);
      }
    }

    public void retirerJoueur(Joueur joueur) {
      joueurs.remove(joueur);
    }
```

```
    public void distribuerCartes() {

      this.partieEnCours = true;

      while (cartes.estVide() == false) {
          Iterator<Joueur> it = joueurs.iterator();
          while (it.hasNext()) {
            Joueur j = (Joueur) it.next();
            j.prendreCarte(cartes.tirerCarteDuDessus());
          }
      }
    }

    public void jouer() {

      ArrayList<Carte> cartesJouees = this.jouerCartes();
      System.out.println("Cartes jouées : " + cartesJouees);

      Carte cg = this.carteGagnante(cartesJouees);
      Joueur jg = this.joueurGagnant(cg);

      System.out.println(jg.getNom() + " remporte le tour avec " + cg);

      this.recupererCartesJouees(jg, cartesJouees);

      this.retirerPerdants();

    }

    private void recupererCartesJouees(Joueur j, ArrayList<Carte>
cartesJouees) {
      Iterator<Carte> it = cartesJouees.iterator();
      while (it.hasNext()) {
          Carte c = (Carte) it.next();
          j.prendreCarte(c);
      }
    }

    private ArrayList<Carte> jouerCartes() {
      ArrayList<Carte> cartesJouees = new ArrayList<Carte>();

      Iterator<Joueur> itDepot = joueurs.iterator();
      while (itDepot.hasNext()) {
          Joueur j = (Joueur) itDepot.next();
          cartesJouees.add(j.jouerCarte());
      }
      return cartesJouees;
    }

    private void retirerPerdants() {
      Iterator<Joueur> it = joueurs.iterator();
      while (it.hasNext()) {
          Joueur j = (Joueur) it.next();
          if (j.aPerdu()) {
            it.remove();
            System.out.println(j.getNom() + " a perdu !");
          }
      }
    }

    private Joueur joueurGagnant(Carte carte) {
      Iterator<Joueur> itGagnant = joueurs.iterator();
      boolean trouveGagnant = false;
```

```
      Joueur gagnant = null;
      while (itGagnant.hasNext() && trouveGagnant == false) {
          gagnant = (Joueur) itGagnant.next();
          if (carte.equals(gagnant.derniereCarteJouee())) {
            trouveGagnant = true;
          }
      }
      return gagnant;
    }

    private Carte carteGagnante(List<Carte> cartes) {
      Iterator<Carte> it = cartes.iterator();
      Carte meilleureCarte = null;
      while (it.hasNext()) {
          Carte c = (Carte) it.next();
          if (meilleureCarte == null) {
            meilleureCarte = c;
          } else {
            if (c.getValeur().ordinal() >
meilleureCarte.getValeur().ordinal()) {
                meilleureCarte = c;
            }
          }
      }
      return meilleureCarte;
    }

    public boolean estTerminee() {
      Iterator<Joueur> it = joueurs.iterator();
      boolean estTerminee = false;
      while (it.hasNext() && estTerminee == false) {
          Joueur j = (Joueur) it.next();
          estTerminee = j.aGagne();
      }

      return estTerminee;
    }

    public String toString() {
      return joueurs.toString();
    }

    public static void main(String[] args) {

      PartieDeCartes pdc = new PartieDeCartes();

      Joueur marcel = new Joueur("Marcel");
      Joueur raymond = new Joueur("Raymond");
      Joueur polo = new Joueur("Polo");
      Joueur simone = new Joueur("Simone");

      pdc.ajouterJoueur(marcel);
      pdc.ajouterJoueur(raymond);
      pdc.ajouterJoueur(polo);
      pdc.ajouterJoueur(simone);

      pdc.distribuerCartes();
      System.out.println(pdc);

      while (pdc.estTerminee() == false) {
          pdc.jouer();
          System.out.println(pdc);
          try {
```

```
            Thread.sleep(1000);
          } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
      }
    }
}
```

Exercices à faire en temps hors-encadrement (THE)

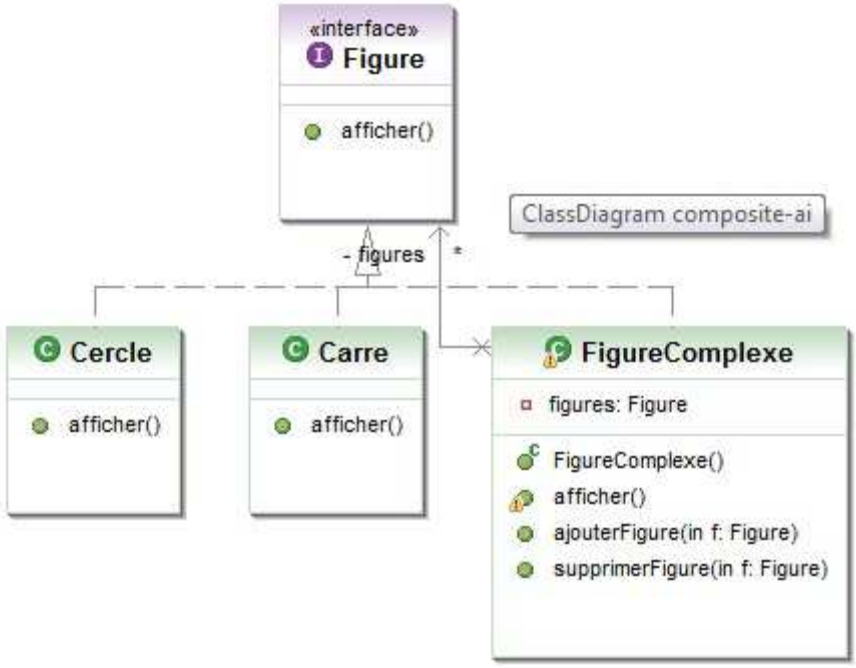
Exercice 3 : Le patron de conception composite (implémentation du patron vu en TD UML)

Le patron composite a pour objectif de permettre au client d'une classe de manipuler de manière identique un objet simple et une collection d'objet. Pour illustrer ce patron, nous prenons l'exemple d'un logiciel de dessin vectoriel qui manipule des figures. Une figure peut être ici une forme de base (carré, cercle, …) ou bien une composition de figures, qui peuvent elles mêmes être des formes de base ou des compositions de figures, etc.

Pour implanter ce patron, il suffit de voir les objets simples et les objets composites de la même manière. Pour cela on utilise une interface dédiée qui contient les opérations qui devront pouvoir s'appliquer sur un objet simple et sur un ensemble d'objets composés.

Dans la suite, on applique ce patron aux figures du logiciel de dessin.

1. Proposer un diagramme de classes qui respecte ce patron de conception. La seule opération spécifiée par l'interface sera l'affichage de la figure.



2. Implanter ce diagramme et tester l'affichage d'une figure.

```
package fr.utt.sit.lo02.collections.composite;

public interface Figure {

    public void afficher();

}
```

```
package fr.utt.sit.lo02.collections.composite;

public class Carre implements Figure {
```

```
    public void afficher() {
      System.out.println("Carré");
    }
}
```

```
package fr.utt.sit.lo02.collections.composite;

public class Cercle implements Figure {

    public void afficher() {
      System.out.println("Cercle");
    }
}
```

```
package fr.utt.sit.lo02.collections.composite;

import java.awt.Component;
import java.util.ArrayList;

public class FigureComplexe implements Figure {

    private ArrayList<Figure> figures;

    public FigureComplexe() {
      figures = new ArrayList<Figure>();
    }

    public void ajouterFigure(Figure f) {
      figures.add(f);
    }

    public void supprimerFigure(Figure f) {
      figures.remove(f);
    }

    public void afficher() {
      java.util.Iterator it = figures.iterator();

      while (it.hasNext()) {
          Figure f = (Figure) it.next();
          f.afficher();
      }
    }

    public static void main(String[] args) {

      Figure carre1 = new Carre();
      Figure cercle1 = new Cercle();
      FigureComplexe fc1 = new FigureComplexe();
      fc1.ajouterFigure(carre1);
      fc1.ajouterFigure(cercle1);

      Figure carre2 = new Carre();
      FigureComplexe fc2 = new FigureComplexe();
      fc2.ajouterFigure(carre2);
      fc2.ajouterFigure(fc1);

      fc2.afficher();
    }
}
```

3. Envisager d'autres cas où ce patron serait utile.

Toutes les situations où l'on a besoin de manipuler des ensembles de manière « atomique » : Par exemple, calculer le prix d'un produit.