LES COLLECTIONS

L'objectif de ce TP est la manipulation de collections.

Pascal le lapin

Pascal le lapin, doit déposer le week-end prochain un certain nombre d'oeufs et de poules en chocolat dans les jardins des maisons. Il a trop de travail et pour rester compétitif avec ses concurrentes les cloches, il a décidé d'investir dans des Lapins Pascals Automatiques avec Java Embarqué... (des LPA). Il a commencé lui-même leur programmation, vous allez l'aider à terminer cette tâche.

La nuit de Pâques, chaque LPA fera sa tournée qui consiste à visiter une liste de jardins. Pascal sait combien d'enfants habitent dans chacune de ces maisons et quel est leur poids. Les LPA devront déposer une confiserie pour chaque enfant, dont la taille sera adaptée à l'enfant.

Pascal initialise ses LPA en leur fournissant : une liste de jardins à visiter, une collection de chocolats à placer dans leur sac à dos (les LPA ont un grand sac à dos dans lequel ils transportent les chocolats à distribuer). Chaque LPA mémorisera le jardin où il se trouve pendant ses déplacements.

Voici le début du programme de Pascal (en Java :-)).

```
public class Chocolat {
    public static final String LAPIN = "lapin";
    public static final String CLOCHE = "cloche";
    public static final String OEUF = "oeuf";
    public static final String POULE = "poule";

    /** la forme du chocolat */
    private String nom;

    /** le poids du chocolat */
    private float poids;

    public Chocolat(String nom, float poids){
        this.nom=nom;
        this.poids = poids;
    }

    public boolean convient(Enfant e) {
        // A ECRIRE
    }
}
```

```
public class Enfant {
  /** son prnom */
  private String prnom;

  /** son poids */
  private float poids;

  public Enfant(String prnom, float poids){
    this.prnom = prnom;
    this.poids = poids;
  }

  public int getPoids(){
    return poids;
  }

  public void mangeChocolat(Chocolat c){
    System.out.println("Miam!");
  }
}
```

```java
public class Jardin{
    /** son adresse*/
    private String adresse;

    /** Les enfants qui habitent l*/
    private A_COMPLETER lesEnfants;

    public Jardin(String adresse){
        this.adresse = adresse;
        lesEnfants = A_COMPLETER;
    }

    public Iterator getLesEnfantsIterator(){
        return lesEnfants.iterator();
    }
    public String getAdresse() { return adresse; }
    public boolean ajouteEnfant(Enfant e){
        return lesEnfants.add(e);
    }
}
```

```java
public class LPA {
    /** les cadeaux...*/
    private A_COMPLETER lesChocolats;

    /** le parcours en termes de jardins */
    private A_COMPLETER lesJardins;

    /** le jardin o le LPA se trouve */
    private Jardin monJardin;

    /** l'iterateur qui permet de parcourir les jardins */
    private Iterator iterJardins;

    public LPA(A_COMPLETER lesChocolats, A_COMPLETER lesJardins) {
        this.lesJardins = lesJardins;
        this.lesChocolats = lesChocolats;
        // je dmarre du premier Jardin
        iterJardins = lesJardins.iterator();
        avancer();
    }

    public void dposerChocolatEnfant(Enfant e) {
        //A_ECRIRE
    }

    public void dposerChocolatJardin() {
        // A_ECRIRE
    }

    public boolean avancer() {
        // A_ECRIRE
    }
}
```

1. Complétez les déclarations et créations des collections à chaque occurrence du mot COMPLETER dans le programme précédent.
2. Écrivez la méthode `convient`. Un chocolat convient à un enfant si le poids du chocolat est égal à $\frac{1000}{poids\ de\ l'enfant}$, plus ou moins 1% .
3. Écrivez la méthode `avancer` qui met à jour les variables mémorisant le jardin où le LPA se trouve. La méthode ne fait rien si le LPA est stoppé. Elle retourne vrai si le LPA a pu avancer et faux sinon.

4. Écrivez la méthode `DéposerChocolatEnfant` qui déposera le premier chocolat trouvé dans le sac à dos convenant à l'enfant donné en paramètre. Aide : La méthode `remove` de la classe `Iterator` permet de retirer un élément d'une collection.
5. Écrivez la méthode `DéposerChocolatJardin` qui dépose un chocolat à chaque enfant habitant là où se trouve le LPA. Cette méthode ne doit rien faire si le LPA est stoppé. Vous pouvez utiliser les méthodes précédentes sans les avoir écrites.
6. Écrivez la partie de la méthode `main` de la classe `PascalLeLapin` qui fait avancer et distribuer les chocolats par les LPA à tour de rôle (chaque LPA sert un jardin, puis on recommence sur tous les LPA jusqu'à ce que tous les LPA aient fini).

```
public class  PascalLeLapin {
    public static void main(String[] a) {
        Set mesLPA = new HashSet();
        /* du code ici en quantité
           pour créer les enfants, les jardins, les chocolats
           les LPA, ... // afin de pouvoir tester votre programme
        */

        // Faire tourner mes LPA
        // ça, vous devez l'écrire!

    }
}
```

MaDeque

L'interface `Deque` définit le contrat à remplir pour une structure de données qui peut être utilisée aussi bien comme une pile (ajout et retrait à la fin de la structure), ou comme une file (ajout à la fin et retrait au début de la structure). L'interface `Deque` étend l'interface `Collection` et propose une vingtaine de méthodes.

Nous considérerons ici le cas de l'implémentation d'une structure de taille fixe arbitraire (qui ne peut pas être choisie par l'utilisateur). Vous devez proposer votre implémentation générique `MaDeque` de cette interface en donnant les attributs, le constructeur vide (i.e. sans paramètre) et le code pour les méthodes suivantes :

- `boolean offerFirst(E elt)` : ajoute un élément en début. Retourne vrai uniquement si l'élément a pu être ajouté.
- `E getFirst(E elt)` : retourne l'élément au début. Lève une exception `NoSuchElementException` si la deque est vide.
- `E removeFirst()` : retourne l'élément au début et le supprime de la structure. Lève une exception `NoSuchElementException` si la deque est vide.
- `boolean offerLast(E elt)` : ajoute un élément en fin. Retourne vrai uniquement si l'élément a pu être ajouté.
- `E removeLast()` : retourne l'élément à la fin et le supprime de la structure. Lève une exception `NoSuchElementException` si la deque est vide.
- `isEmpty()` : teste si la deque est vide.