IUT de Paris Descartes – Base de la Programmation Objet

Travaux Dirigés – Sujet n°6

Objectifs

- Comprendre la visibilité des classes
- Généraliser une classe par l'emploi d'un paramètre générique

Listes génériques

Le but de l'exercice est de généraliser une classe conçue pour gérer des listes d'entiers à des listes d'éléments d'un type quelconque. La définition initiale de la classe `Liste` est la suivante :

```
// Liste.java
package listes;

// visiblité paquetage
class Maillon {
  public int valeur;
  public Maillon suivant;
  public Maillon(int valeur) {
    this.valeur = valeur;
    this.suivant = null;
  }
}

public class Liste {
  private Maillon premier;

  public Liste() {
    premier = null;
  }

  public Liste(Liste li) {
    this();
    for (Maillon m = li.premier;
         m != null; m = m.suivant)
      ajouter(m.valeur);
  }

  public void ajouter(int v) {
    Maillon m = new Maillon(v);
    if (premier == null)
      premier = m;
    else {
      Maillon tmp = premier;
      while (tmp.suivant != null)
        tmp = tmp.suivant;
      tmp.suivant = m;
    }
  }
}
```

Notez que cette classe repose sur une autre classe représentant les maillons de la liste chaînée. Les classes `Liste` et `Maillon` sont définies dans un unique fichier `Liste.java`. Cela est possible uniquement parce que la classe `Liste` est la seule classe publique. La classe `Maillon` a une visibilité *paquetage* et les deux classes sont les deux seules ayant été définies au sein du paquetage `listes`. Ainsi, la classe `Liste` est la seule à pouvoir employer la classe `Maillon`.

1. Ajoutez une méthode `toString` à la classe `Liste`.

Solution:

```
public class Liste {
  ...
  public String toString() {
    StringBuilder sb = new StringBuilder("[");
    boolean first = true;
    for (Maillon m = premier; m != null; m = m.suivant) {
      if (!first)
        sb.append(", ");
      else
        first = false;
      sb.append(Integer.toString(m.valeur));
    }
    return sb.toString() + "]";
  }
}
```

2. Écrivez un court programme qui ajoute les valeurs 7, 9, 11 et 6 à une liste puis affiche son contenu. Ce programme doit être défini en dehors du paquetage `listes`.

Solution:

```
import listes.Liste;

public class Appli {
  public static void main(String[] args) {
    Liste liste = new Liste();
    liste.ajouter(7); liste.ajouter(9);
    liste.ajouter(11); liste.ajouter(6);
    System.out.println(liste.toString());
  }
}
```

3. La méthode `ajouter` de la classe `Liste` doit parcourir tous les éléments avant de pouvoir ajouter la nouvelle valeur à la fin de la chaîne. Ajoutez un attribut référençant le dernier maillon de la chaîne et exploitez-le pour accélérer l'ajout d'une valeur.

Solution:

```
public class Liste {
  private Maillon premier, dernier;

  public Liste() {
    premier = dernier = null;
  }

  public void ajouter(int v) {
    Maillon m = new Maillon(v);
    if (premier == null)
      premier = m;
    else
      dernier.suivant = m;
    dernier = m;
```

```
  }
  ...
}
```

4. Pour permettre l'accès aux données, le paquetage `listes` propose la classe `Itérateur` ainsi définie :

```
package listes;

public class Itérateur {
  private Maillon courant;

  public Itérateur(Liste li) {
    courant = li.getPremier();
  }

  public boolean estFini() {
    return courant == null;
  }

  public int valeur() {
    return courant.valeur;
  }

  public void suivant() {
    courant = courant.suivant;
  }
}

public class Liste {
  ...
  // visiblité paquetage
  Maillon getPremier() {
    return premier;
  }
  ...
}
```

Notez qu'une méthode (`getPremier`) a été ajoutée à la classe `Liste`. Pour des raisons évidentes d'encapsulation, cette méthode est de visibilité *paquetage*. Réécrivez la méthode `toString` en vous reposant sur un itérateur.

Solution:

```
public class Liste {
  ...
  public String toString() {
    StringBuilder sb = new StringBuilder("[");
    boolean first = true;
    for (Itérateur it = new Itérateur(this); !it.estFini(); it.suivant()) {
      if (!first)
        sb.append(",␣");
      else
        first = false;
      sb.append(Integer.toString(it.valeur()));
    }
    return sb.toString() + "]";
  }
}
```

5. Généralisez la classe `Liste` à tout type d'élément. Il est suffisant d'indiquer les différences à introduire par rapport à la définition des classes `Maillon`, `Itérateur` et `Liste`.

Solution: Le type d'élément (que nous nommons `T`) va devenir un paramètre générique des trois classes. Les modifications à apporter aux classes `Maillon` et `Itérateur` sont limitées. Elles sont surlignées ci-dessous.

```java
public class Itérateur<T> {
  private Maillon<T> courant;

  public Itérateur(Liste<T> li) {
    courant = li.getPremier();
  }

  public boolean estFini() {
    return courant == null;
  }

  public T valeur() {
    return courant.valeur;
  }

  public void suivant() {
    courant = courant.suivant;
  }
}

class Maillon<T> {
  public T valeur;
  public Maillon<T> suivant;

  public Maillon(T valeur) {
    this.valeur = valeur;
    this.suivant = null;
  }
}
```

Celles apportées à la classe `Liste` sont de même nature. Le seul point à discuter concerne la méthode `toString` qui doit invoquer `toString` pour des objets de type `T`. Cela n'est possible que parce que les seuls types concrets pouvant instancier `T` sont des classes ou des tableaux et que `toString` fait partie des *protocoles standards* vus en cours.

```java
public class Liste<T> {
  private Maillon<T> premier, dernier;

  public Liste() {
    premier = dernier = null;
  }

  public Liste(Liste<T> li) {
    this();
    for (Maillon<T> m = li.premier;
                m != null; m = m.suivant)
      ajouter(m.valeur);
  }

  Maillon<T> getPremier() {
    return premier;
  }

  public void ajouter(T v) {
    Maillon<T> m = new Maillon<T>(v);
    if (premier == null)
      premier = dernier = m;
    else {
      dernier.suivant = m;
      dernier = m;
    }
  }

  public String toString() {
    StringBuilder sb;
    sb = new StringBuilder("[");
    boolean first = true;
    Itérateur<T> it;
    for (it = new Itérateur<T>(this);
          !it.estFini(); it.suivant()) {
      if (!first)
        sb.append(",␣");
      else
        first = false;
      sb.append(( it.valeur().toString() ));
    }
    return sb.toString() + "]";
  }
}
```

6. Adaptez en conséquence le programme de la question 2.

Solution:

```java
public class Appli {
  public static void main(String[] args) {
    Liste<Integer> liste = new Liste<Integer>();
    liste.ajouter(7);
    liste.ajouter(9);
    liste.ajouter(11);
    liste.ajouter(6);
```

```
    System.out.println(liste.toString());

    Liste<String> courses = new Liste<String>();
    courses.ajouter("sel");
    courses.ajouter("pain");
    courses.ajouter("nouilles");
    System.out.println(courses.toString());
  }
}
```