Programmation et Conception Orientées Objet : POO en Java

Jean-Luc BOURDON

Département des Sciences Informatiques
Université de Cergy-Pontoise

2017-2018

Composition et délégation

► Sommaire

Utilisation

Une méthode peut utiliser les propriétés et méthodes d'une autre instance :

```
public class Point {
 public int x, y;

 public Point(int x, int y) {
   this.x = x; this.y = y;
 }

 public double distanceTo(Point p) {
   int dx = x - p.x;
   int dy = y - p.y;
   return Math.sqrt(dx*dx + dy*dy);
 }
}
```


Composition

Afin d'implémenter ses services, une instance peut créer des instances et conserver leurs références dans des champs :

```
public class Circle {
 private Point center;
 private int radius;

 public Circle(int x, int y, int radius) {
   center = new Point(x,y);
   this.radius = radius;
 }

 public int getX() { return center.x; }
 public int getY() { return center.y; }
 public int getRadius() { return radius; }
}
```

Agrégation

Une instance peut simplement posséder des références vers des instances :

```
public class Circle {
 private Point center, point;

 public Circle(Point center, Point point) {
   this.center = center; this.point = point;
 }

 public Point getCenter() { return center; }

 public int getRadius() {
   int dx = x - p.x, dy = y - p.y;
   return Math.sqrt(dx*dx + dy*dy);
 }
}
```


Agrégation et délégation

Délégation du calcul de la distance à l'instance `center` de la classe `Point` :

```
public class Circle {
 private Point center, point;

 public Circle(Point center, Point point) {
   this.center = center; this.point = point;
 }

 public Point getCenter() { return center; }

 public int getRadius() {
   return center.distanceTo(point);
 }
}
```


Agrégation récursive

Il est possible de créer des structures récursives :

```
public class Node {
  private Node[] nodes;
  private String name;

  public Node(Node[] nodes, String name) {
    this.nodes = nodes; this.name = name;
  }

  public Node(String name) {
    this.nodes = new Node[0]; this.name = name;
  }
}
```


Agrégation récursive

Il est ensuite possible d'implémenter des méthodes de façon récursive :

```
public class Node {
  private Node[] nodes;
  private String name;

  /* Constructeurs du slide précédent. */

  public void print() {
    System.out.print("["+name+"]");
    for (int i = 0; i < nodes.length; i++)
      nodes[i].print();
  }
}
```

Agrégation récursive

Exemple d'utilisation de la classe précédente :

```
Node a = new Node("a");
Node b = new Node("b");
Node c = new Node("c");
Node d = new Node("d");
Node ab = new Node(new Node[] {a,b}, "ab");
Node cd = new Node(new Node[] {c,d}, "cd");
Node abcd = new Node(new Node[] {ab, cd}, "abcd");
abcd.print();
```

Cet exemple produit la sortie suivante :

```
[abcd] [ab] [a] [b] [cd] [c] [d]
```


Classe interne statique

Il est possible de définir une classe à l'intérieur d'une autre :

```
public class LinkedList {
  public static class Node {
    private String data;
    private Node next;

    public Node(String data, Node next) {
      this.data = data; this.next = next;
    }
  }
}
```

Il est possible d'instancier la classe interne sans qu'une instance de `LinkedList` existe car elle est statique :

```
LinkedList.Node node = new LinkedList.Node("a", null);
```


Classe interne statique

Il est également possible de la rendre privée à la classe `LinkedList` :

```
public class LinkedList {
  private static class Node {
    private String data;
    private /*LinkedList.*/Node next;

    public Node(String data, Node next) {
      this.data = data; this.next = next;
    }
  }
}
```

Dans ce cas, seules les méthodes de `LinkedList` pourront l'utiliser. Notez que des méthodes statiques définies dans `LinkedList` peuvent également utiliser cette classe interne du fait qu'elle soit statique.

Classe interne statique

Exemple d'implémentation de méthodes dans la classe `LinkedList` :

```
public class LinkedList {
  /* Code de la classe interne statique Node. */

  private Node first = null;

  public void add(String data) {
    first = new Node(data, first);
  }

  public void print() {
    Node node = first;
    while (node!=null) {
      System.out.print("["+node.data+"]");
      node = node.next;
    }
  }
}
```

Classe interne statique

Exemple d'utilisation de la classe précédente :

```
LinkedList list = new LinkedList();
list.add("a");
list.add("b");
list.add("c");
list.print();
```

Cet exemple produit la sortie suivante :

```
[c] [b] [a]
```


Classe interne statique

Une classe interne statique ne peut accéder qu'aux champs et méthodes statiques de la classe qui la contient :

```
public class LinkedList {

  private static class Node {
    /* Champs et méthodes de Node. */
    boolean isFirst() {
      return this==first;  // interdit !
    }
  }

  private Node first;
  /* Autres champs et méthodes de LinkedList. */
}
```


Classe interne

En revanche, si la classe interne n'est pas statique, elle peut accéder aux champs de classe qui la contient :

```
public class LinkedList {

  private class Node {
    /* Champs et méthodes de Node. */
    private boolean isFirst() {
      return this==first;  // autorisé !
    }
  }

  private Node first;
  /* Autres champs et méthodes de LinkedList. */
}
```


Classe interne

Java insère dans l'instance de `Node` une référence vers l'instance de `LinkedList` qui a permis de la créer :

```
public class LinkedList {
  private class Node {
    /* Champs et méthodes de Node. */
    private boolean isFirst() {
      return this==/*référenceVersLinkedList.*/first;
    }
  }

  public void add(String data) {
    first = new Node(data, first);
  }

  /* Autres champs et méthodes de la classe LinkedList. */
}
```

Classe interne

Il est possible d'utiliser la méthode `isFirst` dans `LinkedList` :

```
public class LinkedList {
  /* Code de la classe interne statique Node
     et champs et méthodes de la classe LinkedList. */

  public void print() {
    Node node = first;
    while (node!=null) {
      System.out.print("["+node.data
                         +","+node.isFirst()+"]");
      node = node.next;
    }
  }
}
```


Classe interne

Exemple d'utilisation de la classe précédente :

```
LinkedList list = new LinkedList();
list.add("a");
list.add("b");
list.add("c");
list.print();
```

Cet exemple produit la sortie suivante :

```
[c,true][b,false][a,false]
```


Exemple : itérateurs

Nous ajoutons une classe interne afin de pouvoir parcourir la liste :

```
public class LinkedList {
  /* Classe interne Node, autres champs et méthodes. */
  public Iterator iterator() { return new Iterator(); }

  public class Iterator {
    private Node node;
    public Iterator() { node = first; }
    public boolean hasNext() { return node!=null; }
    public String next() {
      String data = node.data; node = node.next;
      return data;
    }
  }
}
```


Exemple : itérateurs

Exemple d'utilisation de l'itérateur :

```
LinkedList list = new LinkedList();
list.add("a");
list.add("b");
list.add("c");
LinkedList.Iterator iterator = list.iterator();
while (iterator.hasNext())
  System.out.print("["+iterator.next()+"]");
```

Cet exemple produit la sortie suivante :

```
[c][b][a]
```

Notez que la classe interne `Node` n'est accessible que par les méthodes de `LinkedList` et de ses classes internes car elle est privée.