TP n° 2 & 3 — Analyse de couverture de coe

Antoine de ROQUEMAUREL (Groupe 1.1)

1 Algorithme

L'algorithme à été développé en java. Afin de créer la liste, deux classes ont été créés : `List` et `Cell`.

1.1 `Cell`

`Cell` est une cellule de la liste, elle contient donc la cellule précédente, la suivante et la valeur concernée [1].

Cette classe possède la visibilité package afin que celle si ne soit pas utilisée à mauvais escient.

```
1  package list;
2
3  class Cell {
4    private int _data;
5    private Cell _previous;
6    private Cell _next;
7
8    public Cell(int data, Cell next) {
9      _data = data;
10     _previous = null;
11     _next = next;
12   }
13
14   public int getData() {
15     return _data;
16   }
17
18   public Cell getNext() {
19     return _next;
20   }
21
22   public void setNext(Cell _next) {
23     this._next = _next;
24   }
25 }
```

Listing 1 – Classe `Cell`

1.2 `List`

La classe `List` qui est publique, permet à l'utilisateur de créer une liste triée et d'y ajouter des éléments [2].

L'algorithme proposé à donc été implémenté dans la méthode `add`.

1. La valeur est de type `int`
2. Possibilité d'ajouter un élément depuis la méthode `add` ou depuis le constructeur

```java
package list;

public class List {
  private Cell _head;

  public List(int... data) {
    for(int d : data) {
      add(d);
    }
  }

  public void add(int data) {
    Cell current = _head;
    Cell previous = null;
    Cell newCell;
    boolean notEnd = current != null;

    while(notEnd && current != null) {
      if(current.getData() > data) {
        notEnd = false;
      } else {
        previous = current;
        current = current.getNext();
      }
    }
    newCell = new Cell(data, current);

    if(previous == null) {
      _head = newCell;
    } else {
      previous.setNext(newCell);
    }
  }

  @Override
  public String toString() {
    Cell current = _head;
    String ret = "";
    while(current != null) {
      ret += current.getData()+", ";
      current = current.getNext();
    }

    return ret;
  }
}
```

Listing 2 – Classe `List`

2 Instructions couvrant une insertion en position intermédiaire

2.1 Ajout des deux premiers entiers

Figure 2.1 est disponible la couverture pour l'insertion des deux premiers entiers, avant l'insertion de l'entier qui nous intéresse.

```
public void add(int data) {
    Cell current = _head;
    Cell previous = null;
    Cell newCell;
    boolean notEnd = current != null;

    while(notEnd && current != null) {
        if(current.getData() > data) {
            notEnd = false;
        } else {
            previous = current;
            current = current.getNext();
        }
    }
    newCell = new Cell(data, current);

    if(previous == null) {
        _head = newCell;
    } else {
        previous.setNext(newCell);
    }
}
```

2.2 Ajout de l'entier 4

Afin d'insérer l'entier 4, les entiers 1 et 5 devaient être déjà présents dans la liste, afin d'éviter que le rapport d'emma soit faussé par ces deux insertions, une nouvelle méthode à été créé insérant ces deux entiers, cette méthode est disponible ci-dessous.

```
public void add(int data) {
    Cell current = _head;
    Cell previous = null;
    Cell newCell;
    boolean notEnd = current != null;

    while(notEnd && current != null) {
        if(current.getData() > data) {
            notEnd = false;
        } else {
            previous = current;
            current = current.getNext();
        }
    }
    newCell = new Cell(data, current);

    if(previous == null) {
        _head = newCell;
    } else {
        previous.setNext(newCell);
    }
}
```

Comme le montre la figure 2.2, l'insertion du 4 ne passe que par une portion du code, en effet, les instructions d'insertion en tête ou en queue ne sont pas nécessaire ici, c'est ainsi que la condition `current != null` du while et le test `previous == null` ne sont pas nécessaire pour l'insertion intermédiaire. Nous pouvons observer avec la figure 2.1 que l'insertion en tête ou en queue à l'inverse passe dans ces deux instructions mais n'a pas besoin du test vérifiant si `current.getData() > data`.

On peut donc en déduire que c'est ce test qui permet à l'insertion intermédiaire de fonctionner : ça permet d'insérer notre élément à une place correct dans la liste triée.