Grundlagen der Programmkonstruktion
Übungsblatt 7

1 Rekursive Datenstrukturen - IntList

Die folgenden Aufgaben beziehen sich auf die Beispielprogramme `IntList.java` und `IntListNode.java` aus der Vorlesung vom 26. November. Sie finden die Programme auf der Homepage unter `http://www.complang.tuwien.ac.at/franz/programmkonstruktion/pk12w12p/`

1.1 Nachvollziehen der Datenstruktur (0.5)

Im folgenden Codefragment wird eine `IntList` erstellt und bearbeitet. Versuchen Sie, die Methodenaufrufe und ihre Auswirkungen auf die Datenstruktur nachzuvollziehen. Zeichnen Sie dazu bei jedem Schritt die Knoten der Datenstruktur mit ihren Verbindungen und markieren Sie die Änderungen. Verwenden Sie die rekursive Implementierung in `IntListNode`.

```
IntList list = new IntList();

list.add(5);
list.add(2);
list.add(7);
list.sort();
list.remove(2);
```

1.2 boolean isEmpty() (0.3)

Implementieren Sie in `IntList` eine Methode `boolean isEmpty()`. Diese soll `true` zurückliefern, wenn die Liste leer ist, ansonsten `false`. Geben Sie eine Abschätzung der Laufzeit dieser Methode ab.

1.3 void clear() (0.3)

Implementieren Sie eine Methode `void clear()`, die die Liste leert, d.h. nach einem Aufruf von `void clear()` soll die Liste leer sein.

1.4 int size() (0.5)

Implementieren Sie eine Methode `int size()`. Diese soll die Anzahl der Elemente zurückliefern. Beachten Sie, dass für eine leere Liste diese Methode den Wert 0 zurückliefern soll. Geben Sie eine Abschätzung der Laufzeit dieser Methode ab.

Sie dürfen zum Implementieren dieser Methode beliebige weitere Methoden in `IntListNode` hinzufügen.

1.5 int removeFirst() (0.5)

Implementieren Sie eine Methode `int removeFirst()`. Diese soll das erste Element aus der Liste entfernen und den Inhalt des ersten Elements zurückliefern. Ignorieren Sie den Sonderfall, dass `int removeFirst()` auf eine leere Liste angewandt wird.

1.6 int get(int index) (0.6)

Implementieren Sie eine Methode `int get(int index)`. Diese soll das Element an der `index`-ten Stelle zurückliefern.

Sie dürfen zum Implementieren dieser Methode beliebige weitere Methoden in `IntListNode` hinzufügen.

1.7 void addLast(int elem) (0.7)

Implementieren Sie eine Methode `void addLast(int elem)`. Diese soll im Gegensatz zur Methode `void add(int elem)` ein Element an das Ende der Liste hinzufügen. Beachten Sie, dass diese Methode auch auf eine leere Liste angewandt werden kann.

Sie dürfen zum Implementieren dieser Methode beliebige weitere Methoden in `IntListNode` hinzufügen.

1.8 void removeAll(int elem) (0.7)

In `IntList` ist bereits die Methode `void remove(int elem)` implementiert. Diese entfernt *ein* Vorkommen von `elem` aus der Liste:

```
// entferne erstes Vorkommen von elem aus der Liste
// keine Vernderung wenn elem nicht in der Liste ist
public void remove(int elem) { /* ...*/
```

Erstellen Sie eine Methode `void removeAll(int elem)`, die *alle* Vorkommen von `elem` aus der Liste entfernt.

Sie dürfen zum Implementieren dieser Methode beliebige weitere Methoden in `IntListNode` hinzufügen.

1.9 Bonusaufgabe: boolean remove(int elem)

Ersetzen Sie die Methode `void remove(int elem)` aus der aktuellen Implementierung von `IntList` (und auch die entsprechenden Methode aus `IntListNode`) durch eine neue Methode `boolean remove(int elem)`. Diese soll das erste Vorkommen von `elem` aus der Liste entfernen, und zusätzlich `true` zurückliefern, wenn ein Element entfernt wurde, ansonsten `false`.