Opgave 2: Studie van de List implementatie

```
public static void PrintList() {
    LList<Integer> exp = RandomGenerator.GenerateLList( size: 10);

    String print = exp.toString();

    exp.moveToStart();


    for (int i = 0; i < exp.length(); i++) {

        System.out.println(exp.getValue());
        exp.next();
    }

}
```

Voor het creëren van de lijst gebruikt kom ik subiet nog op terug want dit gebeurt in de klasse RandomGenerator. Voor het afdrukken wil ik alle waarden apart laten printen, hiervoor zorg ik eerst dat ik bij het begin van de lijst ben met exp.moveToStart( ). Daarna ga ik met een for lus beginnende van 0 tot het einde van de lijst (exp.length( )). Ik druk de waarde af en ga naar de volgende met exp.next ( ).

```
package com.company;

import java.util.Random;

public class RandomGenerator {
    public static LList<Integer> GenerateLList(int size) {
        LList<Integer> out = new LList<~>();

        Random random = new Random();

        for (int i = 0; i < size; i++)
        {
            out.append(random.nextInt( bound: 10)+1);
        }
        return out;
    }
}
```

We zien hier dat we een for lus hebben die elke keer met de method append een nieuw random getal (tussen 1 en 10) achteraan de lijst zet.

```
public static void RemoveElement(int location){

    LList<Integer> randomList = RandomGenerator.GenerateLList( size: 10);
    String stringListStart = randomList.toString();
    System.out.println(stringListStart);

    location--;
    randomList.moveToPos(location);
    randomList.remove();

    randomList.moveToStart();
    String stringListEnd = randomList.toString();
    System.out.println(stringListEnd);
}
```

Voor het tweede deel van de oefening waarbij er gevraagd werd om het zesde element te verwijderen hiervoor vraag ik de locatie op en trek er daarna 1 van af omdat de list bij nul begint te tellen. Dan gaan we met de method moveToPos naar deze locatie en verwijderen deze met de method remove. Om na te kijken maak ik voor en na deze bewerking een string van de list met toString en druk deze af. Het resultaat geeft:

```
< | 5 2 6 10 6 4 9 6 2 6 >
< | 5 2 6 10 6 9 6 2 6 >
```

Bij het derde deel van de opdracht kijken we hoe de method insert werkt.

```
public static void InsertElementOld(int location,int number){

    LList<Integer> randomList = RandomGenerator.GenerateLList( size: 10);
    String stringListStart = randomList.toString();
    System.out.println(stringListStart);

    randomList.moveToPos(location);
    randomList.insert(number);

    randomList.moveToStart();
    String stringListEnd = randomList.toString();
    System.out.println(stringListEnd);
}
```

Bij het oproepen moet je hier zowel de locatie als het getal meegeven. We gaan hier ook naar deze locatie met de method moveToPos en daarna voegen we op die plaats het meegegeven getal daarin. Hieronder kunnen we ziet war er gebeurt als ik locatie 3 en nummer 50 meegeef.

```
< | 2 10 2 8 2 2 1 5 8 5 >
< | 2 10 2 50 8 2 2 1 5 8 5 >
```

We zien dat deze het getal 50 tussen het derde en vierde element wordt geplaats. Nu is getal 50 het vierde element geworden.

Ik heb een nieuwe method newInsert gemaakt die zorgt dat het getal wel op derde plaats wordt gezet.

```
public void newInsert(E it) {
  this.prev();
  curr.setNext(new Link<E>(it, curr.next()));
  if (tail == curr) tail = curr.next();  // New tail
  cnt++;
}
```

Deze method is dezelfde als Insert, maar wordt voorafgegaan door this.prev. Dit zorgt dat we 1 element naar voorschuiven. Als we nu weer dezelfde parameters meegegeven krijgen we:

```
< | 1 7 8 9 1 1 9 6 10 3 >
< | 1 7 50 8 9 1 1 9 6 10 3 >
```

We zien dat het getal 50 nu het derde element is in onze lijst.

Bij het eerste deel van de opdracht riepen we de remove method aan om het zesde element te verwijderen. Daarbij moesten we eerst locatie -1 doen om het juiste element te laten verwijderen. We kunnen nu ook op dezelfde manier een nieuwe method (newRemove) aanmaken die daar rekening met houdt.

```
public E newRemove() {
  this.prev();
  if (curr.next() == null) return null; // Nothing to remove
  E it = curr.next().element();         // Remember value
  if (tail == curr.next()) tail = curr; // Removed last
  curr.setNext(curr.next().next());     // Remove from list
  cnt--;                // Decrement count
  return it;                            // Return value
}
```

We zien dat ook hier de method wordt voorafgegaan met prev( ). Wanneer we terug het zesde element willen verwijderen krijgen we:

```
< | 3 4 5 1 3 9 4 2 2 3 >
< | 3 4 5 1 3 4 2 2 3 >
```

Besluit:

Werken met linked lists is niet al te moeilijk omdat er al veel methods bestaan die je gewoonweg kan gebruiken. Het enige waaraan je moet denken is dat de telling van de elementen in de list bij 0 begint. Wanneer je dus bijvoorbeeld het vierde element wilt toevoegen of verwijderen, moet je het getal 3 meegeven als parameter of zorgen dat je programma op voorhand één van het meegegeven getal aftrekt.