LinkedList

LinkedList в Java — это реализация связного списка (традиционной структуры данных). По сути, элементы в связном списке сортируются по их позиции вставки относительно других элементов. Элементы всегда добавляются в конец связного списка.

Уникальность связного списка заключается в том, что элементы хранят ограниченные знания о других элементах списка. В нашем случае класс LinkedList в Java известен как *двусвязный список*. Он считается двусвязным, потому что каждый элемент имеет указатель на следующий и предыдущий элементы списка.

Мы создадим LinkedList для хранения данных о городах, которые встречаются нашим героям в их путешествиях. Чтобы получить полную функциональность доступных методов, нам нужно будет указать LinkedList в качестве базового и конкретного классов:

```
LinkedList<String> cityList = new LinkedList<>();
cityList.add("Elddim");
cityList.add("Crystwind");
cityList.add("Fallraen");
cityList.add("Meren");
cityList.add("Lang");

printCollection(cityList);
```

Запуск кода показывает эти значения для cityList:

```
Elddim, Crystwind, Fallraen, Meren, Lang,
```

Как наглядно видно на рис. 4.3, они действительно перечислены в том порядке, в котором были добавлены.

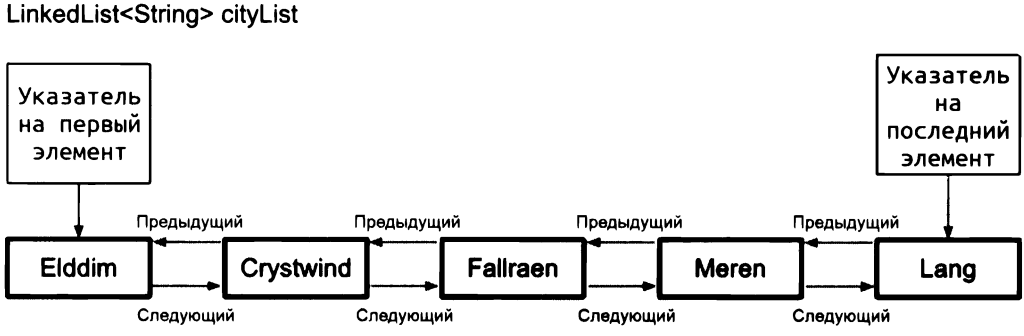

Рис. 4.3. Визуальное представление связного списка cityList

Как и в ArrayList, к элементам LinkedList можно обращаться по числовому индексу:

```
System.out.println(cityList.get(3));
```

Выполнение этого кода добавляет следующую строку в наш вывод:

```
Meren
```

Как уже было показано на примере `ArrayList`, также можно выполнять удаление по индексу или по элементу:

```
cityList.remove("Meren");
printCollection(cityList);
```

Разница в том, что за кулисами должны были произойти следующие шаги:

- элемент `Meren` найден;
- элемент `Meren` удален;
- элемент `Lang` установил свой указатель предыдущего элемента на `Fallraen`;
- элемент `Fallraen` установил свой указатель следующего элемента на `Lang`.

Запуск кода теперь возвращает следующее:

```
Elddim, Crystwind, Fallraen, Meren, Lang,
Meren
Elddim, Crystwind, Fallraen, Lang,
```

`LinkedList` также предоставляет другие уникальные и полезные методы. Если бы нам нужно было заглянуть в начало списка и вывести располагающийся там элемент, можно было бы использовать этот вариант:

```
System.out.println(cityList.peek());
printCollection(cityList);
```

Этот код добавляет следующие строки в вывод:

```
Elddim
Elddim, Crystwind, Fallraen, Lang,
```

То же самое мы можем сделать с последним элементом:

```
System.out.println(cityList.peekLast());
printCollection(cityList);
```

Это дает следующий результат:

```
Lang
Elddim, Crystwind, Fallraen, Lang,
```

Кроме того, мы можем "вытащить" элемент из списка при помощи `poll`:

```
System.out.println(cityList.poll());
printCollection(cityList);
```

Последние две строки вывода выглядят следующим образом:

```
Elddim
Crystwind, Fallraen, Lang,
```

Существует также метод `pollLast()`, демонстрирующий аналогичное поведение для последнего элемента в списке. Разница между методами `peek()` и `poll()` заключается в следующем:

- `peek()` возвращает значение первого элемента в списке;
- `poll()` возвращает значение первого элемента в списке и удаляет этот элемент из списка.

Мы обсудим метод pollLast() и его аналог pollFirst() в разделе, посвященном упорядоченным коллекциям.

> **Примечание.** Помимо того, что каждый элемент имеет указатели на предыдущий и следующий элементы, LinkedList содержит указатели на первый и последний элементы в списке.

Важно понимать, когда использовать ArrayList, а когда LinkedList. Все сводится к количеству элементов и типу операции. Технически, существует понятие, известное как *нотация Big O*. По сути, это способ представления операционной сложности операции или алгоритма. Добавление элемента в ArrayList — это операция O(n). Время, необходимое для выполнения операции (O), пропорционально количеству (n) элементов в списке. Это связано с тем, что при добавлении в ArrayList необходимо выполнить поиск по индексам, прежде чем понять, куда вставить новый элемент.

С другой стороны, добавление элемента в LinkedList — это (если использовать нотацию Big O) операция O(1). Это означает, что время, необходимое для выполнения операции (O), неизменно, независимо от количества элементов в списке. Это происходит потому, что по умолчанию все новые элементы в LinkedList попадают в конец. Java не нужно тратить дополнительное время на то, чтобы понять, куда их поместить.

В результате, если мы планируем использовать большие списки и больше добавлять в список, чем читать из него, то имеет смысл использовать LinkedList. С другой стороны, если мы собираемся тратить большую часть времени на чтение элементов списка (и не добавлять их слишком много), то ArrayList или Vector подойдут как нельзя лучше.

Более подробно мы обсудим связные списки в одной из следующих глав.

Словари

Одним из наиболее уникальных типов коллекций являются *словари*[1]. Словари полезны для хранения небольших объемов данных, основанных на ключах/значениях. Некоторые базы данных, такие как Apache Cassandra, используют словари для обеспечения небольших уровней денормализации (хранения дополнительных свойств в строке данных) во избежание необходимости дополнительного запроса.

Прежде чем мы начнем работать со словарями, создадим небольшой статический метод для вывода содержимого нашего словаря в консоль. Нельзя использовать метод printCollection(), так как класс Map не наследуется от базового класса Collection.

---

[1] Иногда их еще называют "карты" или "мапы". — *Прим. перев.*