Zadanie 7 25-28 listopada 2024 r.

kurs języka Java

jednokierunkowa lista uporządkowana

Instytut Informatyki
Uniwersytetu Wrocławskiego Paweł Rzechonek

Kolekcja jest obiektem, który gromadzi elementy jakiegoś określonego typu i pozwala traktować je jak zestaw danych, umożliwiając jednocześnie wykonywanie pewnych operacji na takim zestawie (na przykład dodawania, usuwania oraz wyszukiwania elementów). Ważną cechą każdej kolekcji jest możliwość sekwencyjnego udostępniania elementów z zestawu danych za pomocą iteratorów. Twoim zadaniem będzie zdefiniowanie własnej kolekcji generycznej w postaci jednokierunkowej listy uporządkowanej.

Część 1.

W pakiecie `structures` zdefiniuj interfejs generyczny `OrderedSequence<>`, reprezentujący kolekcję danych uporządkowaną według wartości.

```
public interface OrderedSequence<T extends Comparable<T>> {
    // ...
}
```

Interfejs ten powinien definiować podstawowe operacje, które można będzie wykonywać na uporządkowanym ciągu: dodanie elementu do ciągu `insert(T el)`, usunięcie elementu z ciągu `remove(T el)` o ile taki istnieje, wskazanie elementu najmniejszego `min()` i największego `max()`, sprawdzenie czy zadany element znajduje się w ciągu `search(T el)`, pobranie elementu z określonej pozycji `at(int pos)`, wskazanie pozycji na której znajduje się element `index(T el)`, ile jest wszystkich elementów w ciągu `size()`, itp.

Część 2.

W pakiecie `structures` zdefiniuj klasę `OrderedList<>`, która będzie jednokierunkową listą generyczną implementująca interfejs `OrderedSequence<>`. Na liście tej elementy mają być uporządkowane od najmniejszego do największego i nie mogą się na niej powtarzać takie same wartości. Klasa ta ma być opakowaniem dla homogenicznej struktury tworzonej wewnątrz na węzłach typu `Node<>`.

```
class OrderedList <T extends Comparable<T>>
implements OrderedSequence<T> {
    private Node<T> head;
    // ... implementacja OrderedSequence<T>
    @Override
    public String toString () { /*...*/ }
```

```
    private class Node <T extends Comparable<T>> {
        private Node<T> next;
        private T data;
        // ... implementacja wspomagająca OrderedSequence<T>
    }
}
```

Klasę wewnętrzną `Node<>` należy wyposażyć w podobne metody jak w interfejsie `OrderedSequence<>`, wtedy klasa `OrderedList<>` będzie tylko opakowaniem dla homogenicznej struktury listowej zbudowanej na węzłach i będzie tym samym dużo prostsza do zaimplementowania. Zgłaszaj wyjątki zawsze gdy to będzie konieczne, na przykład przy próbie włożenia do listy wartości `null` należy zgłosić wyjątek `NullPointerException`.

Część 3.

Kolekcję `OrderedList<>` uzupełnij o implementację interfejsu `Iterable<>`. Umożliwi to przeglądanie kolekcji za pomocą pętli *for-each*. Zdefiniuj własną klasę iteratora implementującą interfejs `Iterator<>` z metodami `hasNext()`, `next()` a także `remove()`. Metoda `remove()` może usunąć z kolekcji tylko ten element, który był udostępniony ostatnio wywołaną metodą `next()`. Zdefiniowany iterator niech będzie prywatną niestatyczną klasą wewnętrzną w kolekcji `OrderedList<>`.

Część 4.

Uzupełnij swoje zadanie o krótki program testowy napisany poza pakietem `structures`. Program ma rzetelnie sprawdzić działanie listy jednokierunkowej `OrderedList<>`. Przetestuj wszystkie metody interfejsu `OrderedSequence<>` w kolekcjach list jednokierunkowych dla różnych typów parametrycznych: `OrderedList<Integer>`, `OrderedList<String>` i `OrderedList<Date>`. Sprawdź też możliwość przeglądania tych kolekcji za pomocą iteratorów w pętli for-each.

Uwaga.

Implementując klasę listy jednokierunkowej nie korzystaj z żadnej kolekcji standardowej!