Übung 05: Generics, Lambda

Abgabetermin: 21. 4. 2016, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ❑ G1 (Grimmer) ❑ G2 (Prähofer) ❑ G3 (Prähofer)

WIN: ❑ G1 (Khalil) ❑ G2 (Khalil) ❑ G3 (Hummel)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 4	24	Java-Programm, Ausgabe der Testläufe	Java-Programm	❑	

Stage 1: Double-linked list (12 Points)

We provide you with an interface *public interface List<T> extends Iterable<T>*. Implement this interface as a **double-linked list** (*LinkedList.java*) and provide all methods of this interface.

- *void add(int index, T value)* inserts an element at a given index (can throw an *IndexOutOfBoundsException*)
- *void add(T value)* prepends an element
- *T get(int index)* returns the element at a given index (can throw an *IndexOutOfBoundsException*)
- *T remove(int index)* removes the element at the given index (can throw an *IndexOutOfBoundsException*)
- *T removeLast()* removes the last element
- *int indexOf(T value)* returns the index of a given element, -1 if it does not exist
- *int size()* provides the number of elements in the list

Implement the nodes of this list in *ListNode.java*. Both classes need to have a generic type parameter, which specifies the type of the elements in your list.

Stage 2: Map, Filter, For Each (12 Points)

The interface *List* defines three further methods: *filter*, *map*, *forEach*. These methods allow processing all values of your list and apply operations (e.g. *map* or *filter*) on its values.

Implement the following methods:

- *<R> List<R> map(Function<? super T, ? extends R> mapper)*
  The '*map*' method is used to map each element to its corresponding result.

```
List<String> l = new LinkedList<>();
l.add("A");
l.add("AA");
l.add("AAA");
```

We can apply a function *mapper* to all elements in our list by using *map*. The result is a new list.

l.map(s -> s.length()) consumes all String values in your list, determines the length, and returns a new list of integer values *{1 2 3}*.

- *List<T> filter(Predicate<? super T> predicate)*
  The '*filter*' method is used to eliminate elements based on some criteria.

  ```
  List<String> l = new LinkedList<>();
  l.add(1);
  l.add(2);
  l.add(3);
  l.add(4);
  l.filter(i -> i % 2 == 0)  // {2 4}
  ```

  The filter function consumes all values of the list and creates a new list with all values that satisfy the predicate (i % 2 == 0).

- *T reduce(T identiy, BinaryOperator<T> accumulator)*
  The '*reduce*' method performs a reduction on the elements of the list, using the provided identity value and an accumulation function.

  ```
  List<String> l = new LinkedList<>();
  l.add(1);
  l.add(2);
  l.add(3);
  l.add(4);
  l.reduce(0, (a, b) -> a + b)  // 10
  ```

  The *reduce* function consumes all values of the list and uses the + to accumulate all values; the identity value is 0.

- void forEach(Consumer<? super T> action)
  The '*forEach*' method is used to iterate through the elements of the list. It applies the *Consumer* function to each element.

  l.forEach(s -> System.out.println(s)) would apply the lambda function to all elements, i.e., it prints all values of the list.