
Programmieren in Java
<http://proglang.informatik.uni-freiburg.de/teaching/java/2017/>

list-editor*Ein Editor für Listen*

Woche 09 Aufgabe 1/3

Herausgabe: 2017-06-26

Abgabe: 2017-07-07

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.Project `list-editor`Package `listeditor`

Klassen

<i>ListEditor</i>
<pre>protected ListEditor(PrintWriter out) public void run(Scanner in) public List<String> currentList() public void pushBack(String line) protected abstract void executeMissing(String cmd, Scanner restOfLine)</pre>
AbortEditor extends ListEditor
<pre>protected void executeMissing(String cmd, Scanner restOfLine)</pre>
RepeatEditor extends ListEditor
<pre>protected void executeMissing(String cmd, Scanner restOfLine)</pre>

In dem Skelett zu dieser Aufgabe finden Sie eine abstrakte Basisklasse `ListEditor` für einen Listeneditor. Der Listeneditor ist im Prinzip die Lösung von Aufgabe w04/list-operations: er erlaubt das interaktive Manipulieren von Wörterlisten.

Der Großteil der Funktionalität ist schon in der Klasse `ListEditor` implementiert. Wird die `run` Methode mit einem Scanner aufgerufen, der die Eingabe enthält, wird diese zeilenweise eingelesen. Die Zeilen werden dann als Kommandos für Listenoperationen interpretiert. Die Implementierung in `ListEditor` erkennt auch schon alle Kommandos aus w04/list-operations (siehe dazu auch die private `execute` Methode im Skelett).

Wird ein Kommando nicht erkannt, ruft `ListEditor` die abstrakte Methode `executeMissing` auf. Die Methode `executeMissing` erhält das „erste Wort“ des unbekannten Kommandos als `String cmd` und den Rest der Zeile als `Scanner restOfLine`.

Ihre Aufgabe ist es nun, zwei konkrete Listeneditoren durch Ableiten der `ListEditor` Klasse zu implementieren, `AbortEditor` und `RepeatEditor`. Diese sollen folgendes Verhalten zeigen, wenn ein unbekanntes Kommando eingegeben wird.

1. **AbortEditor**: Bei einem unbekannten Kommando soll eine `InputMismatchException` geworfen werden.
2. **RepeatEditor**: Hier soll zusätzlich das *Repeat*-Kommando `repeat <n> <input-line>` erkannt werden. Hierbei ist `<n>` die String Darstellung eines `int` und `<input-line>` ein String. Das Repeat-Kommando führt `<input-line>` n mal hintereinander als Kommando aus. Das heißt, steht in einer Zeile beispielsweise das Repeat-Kommando `repeat 3 print` hat das den selben Effekt wie drei aufeinanderfolgende Zeilen mit dem Kommando `print`. Ist $n \leq 0$, hat das Repeat-Kommando keinen Effekt. Andere unbekannte Kommandos sollen so behandelt werden wie bei **AbortEditor**.

Bei der Implementierung von **RepeatEditor** sollte die von **ListEditor** bereitgestellte Methode `pushBack` verwendet werden. Wird `pushBack` mit einem String `line` aufgerufen, wird dieser als nächste Zeile von `ListEditor.run` ausgeführt, vor allen anderen Zeilen der Eingabe. (Siehe hierzu auch den Code der Methode `ListEditor.run` im Skelett.)

Hinweise:

- Um ein besseres Testen mit JUnit zu ermöglichen kann dem Konstruktor zu **ListEditor** ein `PrintWriter`-Objekt übergeben werden, der für die Ausgabe der `print` Listenoperation verwendet wird. In der `main`-Methode im Skelett können Sie sehen, wie sich aus `System.out` (also `stdout`) ein `PrintWriter` erstellen lässt. In **ExampleTests** sehen Sie, wie man den einen `PrintWriter` erstellt, der den Output in einem String abspeichert, anstatt ihn auf `stdout` zu drucken.

Beispieltestfälle:

```
package listeditor;

import org.junit.Before;
import org.junit.Test;

import java.io.*;
import java.util.Collections;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

import static java.util.Arrays.asList;
import static org.junit.Assert.*;

/**
 * Created by fennell on 6/25/17.
 */
public class ExampleTests {

    private StringWriter outWriter;
    private PrintWriter out;
```

```

@Before
public void setUp() {
    // This line ensures that println works the same on Windows and MacOS/Linux.
    System.setProperty("line.separator", "\n");

    outWriter = new StringWriter();
    out = new PrintWriter(outWriter);
}

private void assertOutput(List<String> expectedOutputLines) {
    String prefix = "Welcome to the ListEditor. Enter a command.";
    StringBuilder expectedOutput = new StringBuilder();
    for (String line : expectedOutputLines) {
        expectedOutput.append(line + "\n");
    }
    assertEquals(prefix + "\n" + expectedOutput.toString(),
        outWriter.toString());
}

@Test
public void testAbort() {
    ListEditor editor = new AbortEditor(out);
    editor.run(new Scanner("append 5\nprint\nappend 6"));

    assertEquals(asList("5", "6"), editor.currentList());
    assertOutput(Collections.singletonList("5"));
}

@Test(expected = InputMismatchException.class)
public void testAbortFail() {
    ListEditor editor = new AbortEditor(out);
    editor.run(new Scanner("append 5\nblabla 6\nappend 6"));
}

@Test
public void testRepeat() {
    ListEditor editor = new RepeatEditor(out);
    editor.run(new Scanner("repeat 5 append 5\nprint\nappend 6"));
    assertEquals(asList("5", "5", "5", "5", "5", "6"),
        editor.currentList());
    assertOutput(Collections.singletonList("5 5 5 5 5"));
}
}

```