Praktische Informatik I

Tutor:in: Praktikum 13: Mi, 17-20, MZH 0230, Marina Koksharova/Laura Eiermanns

WiSe 2025/26

Bearbeiter:in: Ramtin Behehst Aeen

Übungsblatt 3

Lösungsvorschlag

Aufgabe 1 Testfälle definieren

8 tests are defined as follows:

1. **testPush**

   Zweck: Überprüft, dass Pushen von Elementen die Größe erhöht bis zur Kapazität und bei Überlauf die ältesten Werte überschreibt.

   Testfolge: push(10), push(-10), push(-12) (voll), push(0), push(4) (Overflow)

   Erwartung: Die Größe wächst bis auf 3 (Kapazität) und bleibt danach unverändert; ältere Elemente werden überschrieben.

2. **testInitializer**

   Zweck: Überprüft, dass ein frisch initialisierter RingBuffer leer ist.

   Testfolge: keine Operationen nach Konstruktion

   Erwartung: size() == 0

3. **testPeek1**

   Zweck: Prüft peek() auf leerem Puffer.

   Testfolge: keine Operationen

   Erwartung: peek() liefert als Sentinelwert 0 für leeren Puffer.

4. **testPeek2**

   Zweck: Prüft peek() mit 1..3 Elementen — es soll das älteste Element zurückgeben.

   Testfolge: push(16), push(-10), push(-12)

   Erwartung: peek() = 16 (ältestes eingefügtes Element)

5. **testPeek3**

   Zweck: Prüft peek(), wenn der Puffer überläuft und das älteste Element überschrieben wird.

   Testfolge: push(16), push(-10), push(-12), push(98)

   Erwartung: Nach Überlauf ist das älteste Element -10 (16 wurde überschrieben), daher peek() = -10.

6. **testPop1**

   Zweck: Prüft pop() bei einem einzigen Element: Entfernen des ältesten Elements und leerer Puffer.

Testfolge: push(16), pop()

Erwartung: Nach pop() ist peek() = 0 (leer)

7. **testPop2**

   Zweck: Prüft pop() bei 2 Elementen: Das zweite Element soll nach pop() zum ältesten Element werden.

   Testfolge: push(16), push(-10), pop()

   Erwartung: peek() = -10

8. **testPop3**

   Zweck: Prüft pop() kombiniert mit einem vorherigen Overflow: Nach mehreren Pushes und einer Pop-Operation soll das erwartete älteste Element zurückgegeben werden.

   Testfolge: push(16), push(-10), push(-3), push(8) (Overflow), pop()

   Erwartung: Nach Overflow enthält der Puffer (-10, -3, 8); pop() entfernt -10; peek() = -3.

Die vollständigen Testimplementierungen sind im folgenden Listing enthalten:

```
1  // Importiert assertEquals usw. sowie Test-Annotationen
2  import static org.junit.jupiter.api.Assertions.*;
3  import org.junit.jupiter.api.AfterEach;
4  import org.junit.jupiter.api.BeforeEach;
5  import org.junit.jupiter.api.Test;
6
7  /**
8   * Diese Klasse definiert die Tests für die Klasse <Klasse ergänzen>.
9   *
10  * @author Surface
11  */
12 public class RingBufferTest
13 {
14     private RingBuffer ring_buffer_1;
15     @BeforeEach
16     public void setUp()
17     {
18         // Hier Anweisungen einfügen, die vor jedem Test ausgeführt werden
19         ring_buffer_1 = new RingBuffer(3);
20     }
21     @Test
22     public void testPush(){
23         ring_buffer_1.push(10);
24         assertEquals(1, ring_buffer_1.size());
25         ring_buffer_1.push(-10);
26         assertEquals(2, ring_buffer_1.size());
27         ring_buffer_1.push(-12);
28         assertEquals(3, ring_buffer_1.size());
29         //testing when we go over the capicity size(it should overwrite oldest)
30         ring_buffer_1.push(0);
31         assertEquals(3, ring_buffer_1.size());
32         ring_buffer_1.push(4);
33         assertEquals(3, ring_buffer_1.size());
34     }
35     @Test
36     public void testInitializer(){
37         assertEquals(0, ring_buffer_1.size());
38     }
39
40     @Test
41     //when no element is inside:
42     public void testPeek1(){
43         //testing when no element is inside:
44         assertEquals(0, ring_buffer_1.peek());
45     }
```

```
46      @Test
47      //when 2 or 3 elemnts are inside:
48      public void testPeek2(){
49          //testing when one elemnt is inside:
50          ring_buffer_1.push(16);
51          assertEquals(16, ring_buffer_1.peek());
52          //testing when two elemnt is inside(16, 10):
53          ring_buffer_1.push(-10);
54          assertEquals(16, ring_buffer_1.peek());
55          //testing when three elemnt is inside(16, 10):
56          ring_buffer_1.push(-12);
57          assertEquals(16, ring_buffer_1.peek());
58
59      }
60      @Test
61      //when 3 elemnts are inside(more than capicity so last one should get
62      // deleted and get replaced):
63      public void testPeek3(){
64          //testing when one elemnt is inside:
65          ring_buffer_1.push(16);
66          assertEquals(16, ring_buffer_1.peek());
67          //testing when two elemnt is inside(16, 10):
68          ring_buffer_1.push(-10);
69          assertEquals(16, ring_buffer_1.peek());
70          //testing when three elemnt is inside(16, 10):
71          ring_buffer_1.push(-12);
72          assertEquals(16, ring_buffer_1.peek());
73          //testing when three elemnt is inside(16, 10):
74          ring_buffer_1.push(98);
75          assertEquals(-10, ring_buffer_1.peek());
76
77      }
78
79      @Test
80      //when 1 elemnts is inside:
81      //it is oldest so it should get deleted and return 0 as
82      //only elemnt there
83      public void testPop1(){
84          ring_buffer_1.push(16);
85          ring_buffer_1.pop();
86          assertEquals(0, ring_buffer_1.peek());
87
88      }
89
90      @Test
91      //when 2 elemnts is inside:
92      //first one is oldest one, so it should get deleted and return second as
93      //oldest element there
94      public void testPop2(){
95          ring_buffer_1.push(16);
96          ring_buffer_1.push(-10);
97          ring_buffer_1.pop();
98          assertEquals(-10, ring_buffer_1.peek());
99
100     }
101
102     @Test
103     //when more than 3 elemnts is inside:
104     //first one is oldest one, so it should get deleted and return second one as
105     //oldest element there
106     public void testPop3(){
107         ring_buffer_1.push(16);
108         ring_buffer_1.push(-10);
109         ring_buffer_1.push(-3);
110         // here the 16 will be replace with -10
111         ring_buffer_1.push(8);
112         //-10 will be deleted
113         ring_buffer_1.pop();
114         //-3 will be returned as oldest
115         assertEquals(-3, ring_buffer_1.peek());
116
117     }
118 }
```

Aufgabe 2 Iterative Änderungstests

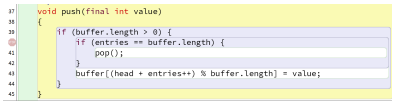
In den folgenden 10 Iterationen ist jeweils genau eine kleine Änderung an `RingBuffer.java` angenommen. Unter jeder Änderung steht, welcher Test aus `RingBufferTest.java` vermutlich fehlschlagen würde und weshalb. Diese Aufstellung hilft beim gezielten Testen und Zurückrollen von Fehlern.

Iteration 1

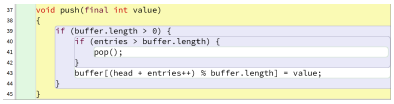
Änderung: In `push()` wird `if (entries > buffer.length)` benutzt statt `if (entries == buffer.length)` (oder `>=`).

Erwartete fehlschlagende Tests: `testPush`, `testPeek3`, `testPop3`

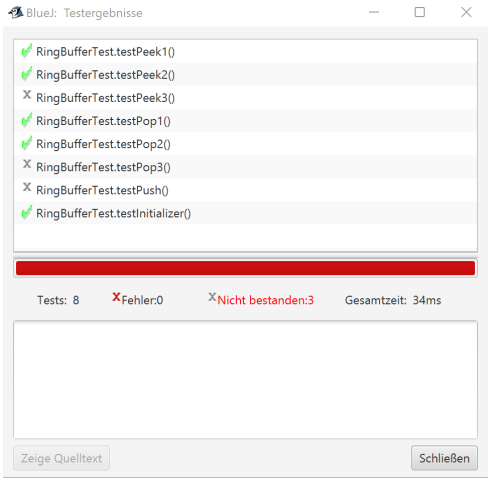
Warum: Bei genau voller Kapazität wird kein Overwrite ausgelöst, die Logik zum Überschreiben des ältesten Elements tritt nicht in Kraft.



(a) vorher



(b) nachher



(c) Testergebnisse