

# 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 capacity 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     @Test
60     //when 3 elemnts are inside(more than capacity so last one should get
61     // deleted and get replaced):
62     public void testPeek3(){
63         //testing when one elemnt is inside:
64         ring_buffer_1.push(16);
65         assertEquals(16, ring_buffer_1.peek());
66         //testing when two elemnt is inside(16, 10):
67         ring_buffer_1.push(-10);
68         assertEquals(16, ring_buffer_1.peek());
69         //testing when three elemnt is inside(16, 10):
70         ring_buffer_1.push(-12);
71         assertEquals(16, ring_buffer_1.peek());
72         //testing when three elemnt is inside(16, 10):
73         ring_buffer_1.push(98);
74         assertEquals(-10, ring_buffer_1.peek());
75     }
76 }
77
78     @Test
79     //when 1 elemnts is inside:
80     //it is oldest so it should get deleted and return 0 as
81     //only elemnt there
82     public void testPop1(){
83         ring_buffer_1.push(16);
84         ring_buffer_1.pop();
85         assertEquals(0, ring_buffer_1.peek());
86     }
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.

The screenshot shows two code snippets side-by-side, labeled (a) vorher and (b) nachher, and a test results window below them.

**(a) vorher:**

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries == buffer.length) {
41             pop();
42         }
43         buffer[(head + entries++) % buffer.length] = value;
44     }
45 }
```

**(b) nachher:**

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries > buffer.length) {
41             pop();
42         }
43         buffer[(head + entries++) % buffer.length] = value;
44     }
45 }
```

**BlueJ: Testergebnisse**

Test	Status
RingBufferTest.testPeek1()	✓
RingBufferTest.testPeek2()	✓
RingBufferTest.testPeek3()	✗
RingBufferTest.testPop1()	✓
RingBufferTest.testPop2()	✓
RingBufferTest.testPop3()	✗
RingBufferTest.testPush()	✗
RingBufferTest.testInitializer()	✓

Tests: 8    ✗Fehler:0    ✗Nicht bestanden:3    Gesamtzeit: 34ms

**(c) Testergebnisse**

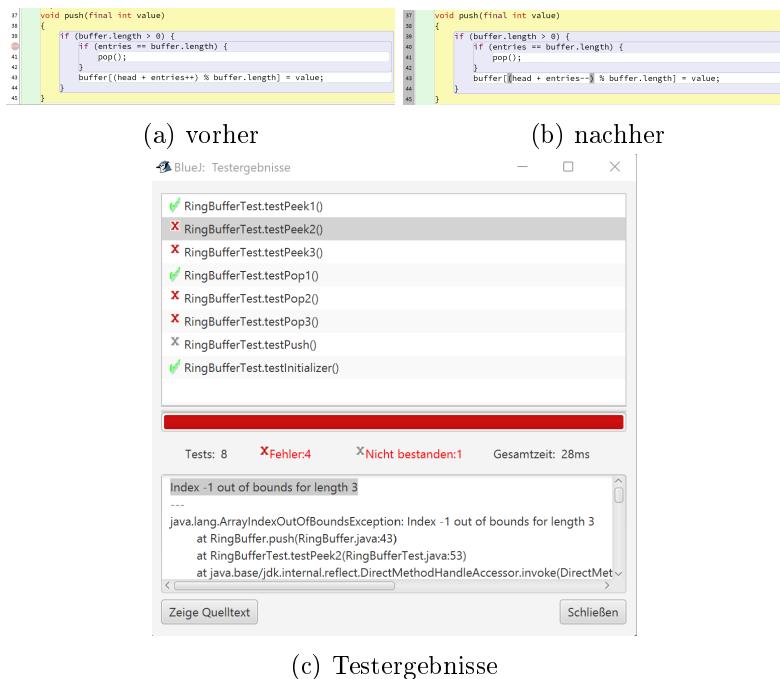
### Iteration 2

**Änderung:** In `push()` wird `entries-` statt `entries++` verwendet.

**Erwartete fehlschlagende Tests:**

1. `testPeek2(Error: Index -1 out of bounds for length 3)`
2. `testPeek3(Error: Index -1 out of bounds for length 3)`
3. `testPop2(Error: Index -1 out of bounds for length 3)`
4. `testPop3(Error: Index -1 out of bounds for length 3)`
5. `testPush(NichtBestanden: expected: <1> but was: <-1>)`

**Warum:** Nach einem Push wird die Größe reduziert statt erhöht; Testfälle, die Größe erwarten, werden fehlschlagen, oder `Index -1 out of bounds for length 3` passiert wird.



### Iteration 3

**Änderung:** In `push()` wird der eingefügte Wert durch eine Konstante ersetzt – `buffer[(head + entries++) % buffer.length] = 16;` – anstatt den tatsächlichen `value` zu speichern.

**Tatsächlich fehlschlagende Tests:** `testPeek3`, `testPop2`, `testPop3`

**Warum:** Jeder Push schreibt den konstanten Wert 16 in den Puffer, unabhängig vom `value`, so dass die Reihenfolge und die tatsächlichen Inhalte nicht den Erwartungen der Tests entsprechen. Beispiele:

- `testPeek3` erwartet nach `push(16)`, `push(-10)`, `push(-12)`, `push(98)` das älteste Element `-10` (`peek()`).  
Wegen der Konstante 16 sind alle Pufferinhalte statt `[-10,...]` 16, sodass `peek()` wieder 16 liefert.
- `testPop2` erwartet nach `push(16)`, `push(-10)`, `pop()` dass `peek()` den Wert `-10` liefert.  
Da beim Push `-10` mit 16 überschrieben wurde, wird statt `-10` 16 als ältestes Element vorgefunden.
- `testPop3` (Overflow + pop) erwartet als ältestes Element nach `pop() -3`; stattdessen läuft `pop()` auf 16.

```
// buggy
buffer[(head + entries++) % buffer.length] = 16;
```

```
// correct
buffer[(head + entries++) % buffer.length] = value;
```

(a) vorher

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries == buffer.length) {
41             pop();
42         }
43         buffer[(head + entries++) % buffer.length] = value;
44     }
45 }

```

(b) nachher

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries == buffer.length) {
41             pop();
42         }
43         buffer[(head + entries + 1) % buffer.length] = value;
44     }
45 }

```

(c) Testergebnisse

## Iteration 4

extbfÄnderung: In push() wird der Schreibindex mit `(head + entries + 1) % buffer.length` berechnet (anstelle von `(head + entries) % buffer.length`).

**Erwartete fehlschlagende Tests:** testPeek2, testPeek3, alle testPop\*

**Warum:** Die Reihenfolge in der Werte abgespeichert werden ist verschoben, Peek/Pop lesen falsche Werte.

(a) vorher

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries == buffer.length) {
41             pop();
42         }
43         buffer[(head + entries++) % buffer.length] = value;
44     }
45 }

```

(b) nachher

```

37 void push(final int value)
38 {
39     if (buffer.length > 0) {
40         if (entries == buffer.length) {
41             pop();
42         }
43         buffer[(head + entries + 1) % buffer.length] = value;
44     }
45 }

```

(c) Testergebnisse

## Iteration 5

Änderung: peek() gibt `buffer[(head + 1) % buffer.length]` anstatt `buffer[head]` zurück.  
**Erwartete fehlschlagende Tests:**, testPeek2, testPeek3, testPop2, testPop3,

**Warum:** Peek gibt nicht das älteste, sondern das zweitälteste Element zurück; erwartet werden jeweils bestimmte Werte (16 etc.).

```

47  /**
48   * Liefert das älteste Element aus dem Ringpuffer zurück, ohne es zu entnehmen.
49   * @return Das älteste Element im Ringpuffer.
50  */
51  int peek()
52  {
53      return buffer[head];
54  }

```

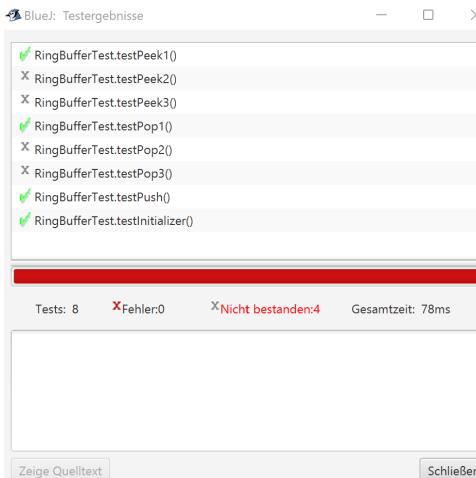
```

47  /**
48   * Liefert das älteste Element aus dem Ringpuffer zurück, ohne es zu entnehmen.
49   * @return Das älteste Element im Ringpuffer.
50  */
51  int peek()
52  {
53      return buffer[(head + 1) % buffer.length];
54  }

```

(a) vorher

(b) nachher



(c) Testergebnisse

## Iteration 6

extbfÄnderung: pop() setzt head = (head - 1 + buffer.length) % buffer.length; anstelle von exttthead = (head + 1) % buffer.length.

**Erwartete fehlschlagende Tests:** testPush, testPeek3, testPop1, testPop2, testPop3

**Warum:** Pop entfernt das falsche Element und die Reihenfolge der ältesten Elemente ändert sich, oder diese Fehler: Index -1 out of bounds for length 3

```

40  int pop()
41  {
42      final int value = peek();
43      head = (head + 1) % buffer.length;
44      --entries;
45      return value;
46  }

```

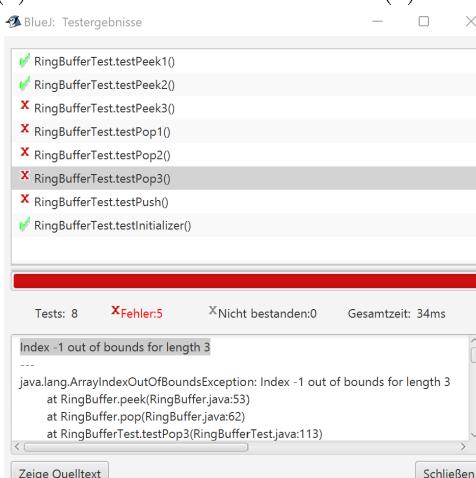
```

56  /**
57   * Entnimmt das älteste Element aus dem Ringpuffer.
58   * @return Das Element, das entnommen wurde.
59  */
60  int pop()
61  {
62      final int value = peek();
63      head = (head - 1) % buffer.length;
64      --entries;
65      return value;
66  }

```

(a) vorher

(b) nachher



(c) Testergebnisse

## Iteration 7

textbf{Änderung}: `pop()` reduziert `entries` nicht mehr

**Erwartete fehlschlagende Tests:** `testPush`, `testPop*`, `testInitializer`

**Warum:** Wenn `entries` nicht korrekt angepasst wird, stimmen Größen- und Leereindikatoren nicht mit den Annahmen der Tests überein.

(a) vorher
(b) nachher

BlueJ: Testergebnisse

Test	Status
RingBufferTest.testPeek1()	✓
RingBufferTest.testPeek2()	✓
RingBufferTest.testPeek3()	✗
RingBufferTest.testPop1()	✗
RingBufferTest.testPop2()	✗
RingBufferTest.testPop3()	✗
RingBufferTest.testPush()	✗
RingBufferTest.testInitializer()	✓

Tests: 8    ✗Fehler:0    ✗Nicht bestanden:5    Gesamtzeit: 48ms

Zeige Quelltext Schließen

(c) Testergebnisse

## Iteration 8

**Änderung:** Konstruktor initialisiert Buffer mit `new int[capacity - 1]` statt `new int[capacity]`.

**Erwartete fehlschlagende Tests:** `testPush`, `testPeek3`, `testPeek2`, `testPop2`, `testPop3`, `testInitializer`

**Warum:** Die effektive Speicherkapazität ist kleiner als erwartet; Überlaufverhalten ändert sich.

The screenshot shows two code snippets side-by-side and their corresponding test results.

**(a) vorher**

```

24 /**
25 * Erzeugt einen Ringpuffer.
26 * @param capacity Die maximale Anzahl von Einträgen, die gepuffert werden
27 */
28 RingBuffer(final int capacity)
29 {
30     buffer = new int[capacity];
31 }

```

**(b) nachher**

```

24 /**
25 * Erzeugt einen Ringpuffer.
26 * @param capacity Die maximale Anzahl von Einträgen, die gepuffert werden
27 */
28 RingBuffer(final int capacity)
29 {
30     buffer = new int[capacity - 1];
31 }

```

**(c) Testergebnisse**

BlueJ Testergebnisse window:

- Tests: 8
- X Fehler: 0
- X Nicht bestanden: 5
- Gesamtzeit: 26ms

The test results show 8 tests, 0 errors, and 5 failing tests. The failing tests are: testPush(), testPop(), testPush(), testPop(), testPush(), and testInitializer().

## Iteration 9

**Änderung:** size() gibt aus Versehen buffer.length statt entries zurück.

**Erwartete fehlschlagende Tests:** testPush, testInitializer **Warum:** Assertions, die die Anzahl belegter Einträge prüfen, vergleichen gegen die Kapazität statt die tatsächliche Anzahl.

The screenshot shows two code snippets side-by-side and their corresponding test results.

**(a) vorher**

```

6 /**
7 * Liefert die Anzahl der Elemente zurück, die sich im Puffer befinden, d.h.
8 * mit (gleich pop()) entnommen werden könnten.
9 * @return Die Anzahl der belegten Einträge im Puffer.
10 */
11 int size()
12 {
13     return entries;
14 }

```

**(b) nachher**

```

66 /**
67 * Liefert die Anzahl der Elemente zurück, die sich im Puffer befinden, d.h.
68 * mit (gleich pop()) entnommen werden könnten.
69 * @return Die Anzahl der belegten Einträge im Puffer.
70 */
71 int size()
72 {
73     return buffer.length;
74 }

```

**(c) Testergebnisse**

BlueJ Testergebnisse window:

- Tests: 8
- X Fehler: 0
- X Nicht bestanden: 2
- Gesamtzeit: 63ms

The test results show 8 tests, 0 errors, and 2 failing tests. The failing tests are: testPush() and testInitializer().

## Iteration 10

**Änderung:** peek() liefert statt 0 einen festen Sentinelwert (z. B. 42) wenn der Puffer leer ist.

**Erwartete fehlschlagende Tests:** testPeek1 ,testPop1

**Warum:** Dieser Test erwartet explizit 0 als Wert bei leerem Puffer; ein anderer Rückgabewert schlägt fehl.

```
51 |     int peek()
52 |     {
53 |         return buffer[head];
54 |     }
55 |
56 |     int peek()
57 |     {
58 |         if (entries > 0) {
59 |             return buffer[head];
60 |         }
61 |         else return 42;
62 |     }
63 | }
```

(a) vorher

Bluel: Testergebnisse

The screenshot shows a Java code editor with two code snippets. The first snippet is a single-line method definition: `int peek() { return buffer[head]; }`. The second snippet is a more complex implementation: `int peek() { if (entries > 0) { return buffer[head]; } else return 42; }`. Below the code editor is a test results window titled 'Bluel: Testergebnisse'. It lists several test methods: `RingBufferTest.testPeek1()`, `RingBufferTest.testPeek2()`, `RingBufferTest.testPeek3()`, `RingBufferTest.testPop1()`, `RingBufferTest.testPop2()`, `RingBufferTest.testPop3()`, `RingBufferTest.testPush()`, and `RingBufferTest.testInitializer()`. All tests except `testPop1()` are marked with a green checkmark. Below the list, it says 'Tests: 8' and 'Fehler: 0'. A red bar indicates 'Nicht bestanden: 2'. At the bottom, there is a stack trace for a failed assertion: 'expected: <0> but was: <42>' followed by a stack trace from org.opentest4j.AssertionFailedError.

(b) nachher

(c) Testergebnisse