

The screenshot shows an IDE interface with a dark theme. On the left is a file tree with a single file selected: `src > test > java > stack > ArrayListStackTest.java`. The code editor displays the following Java test code:

```
src > test > java > stack > ArrayListStackTest.java > ...
1 package stack;
2
3 import stack.ArrayListStack;
4 import org.junit.Test;
5 import static org.junit.Assert.*;
6
7 public class ArrayListStackTest {
8
9     @Test
10    public void testPushAndSize() {
11        ArrayListStack<Integer> stack = new ArrayListStack<>();
12
13        stack.push(item: 10);
14        stack.push(item: 20);
15
16        assertEquals(2, stack.size());
17    }
18
19    @Test
20    public void testPop() {
21        ArrayListStack<Integer> stack = new ArrayListStack<>();
22
23        stack.push(item: 5);
24        stack.push(item: 15);
25
26        int popped = stack.pop();
27
28        assertEquals(15, popped);
29    }
30
31    @Test
32    public void testPopOnEmptyThrowsException() {
33        ArrayListStack<Integer> stack = new ArrayListStack<>();
34
35        try {
36            stack.pop();
37        } catch (Exception e) {
38            assertEquals("Stack underflow", e.getMessage());
39        }
40    }
41
42    @Test
43    public void testPeek() {
44        ArrayListStack<Integer> stack = new ArrayListStack<>();
45
46        stack.push(item: 5);
47
48        assertEquals(5, stack.peek());
49    }
50
51    @Test
52    public void testPeekOnEmptyThrowsException() {
53        ArrayListStack<Integer> stack = new ArrayListStack<>();
54
55        try {
56            stack.peek();
57        } catch (Exception e) {
58            assertEquals("Stack underflow", e.getMessage());
59        }
60    }
61
62    @Test
63    public void testPushOnEmptyThrowsException() {
64        ArrayListStack<Integer> stack = new ArrayListStack<>();
65
66        try {
67            stack.push(item: 5);
68        } catch (Exception e) {
69            assertEquals("Stack overflow", e.getMessage());
70        }
71    }
72
73    @Test
74    public void testPushAndSize() {
75        ArrayListStack<Integer> stack = new ArrayListStack<>();
76
77        stack.push(item: 10);
78        stack.push(item: 20);
79
80        assertEquals(2, stack.size());
81    }
82
83    @Test
84    public void testPopOnEmptyThrowsException() {
85        ArrayListStack<Integer> stack = new ArrayListStack<>();
86
87        try {
88            stack.pop();
89        } catch (Exception e) {
90            assertEquals("Stack underflow", e.getMessage());
91        }
92    }
93
94    @Test
95    public void testPeek() {
96        ArrayListStack<Integer> stack = new ArrayListStack<>();
97
98        stack.push(item: 5);
99
100       assertEquals(5, stack.peek());
101   }
102 }
```

Below the code editor are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, TEST RESULTS, and PORTS. The TEST RESULTS tab is active, showing the following results:

- %TESTS 5, testPopOnEmptyThrowsException(stack.ArrayListStackTest)
- %TESTE 5, testPopOnEmptyThrowsException(stack.ArrayListStackTest)
- %RUNTIME42

To the right of the code editor is a sidebar titled "Test Runner for Java" which lists all the test methods defined in the code.