

# Guidebook: Writing Java Code from Test Code

---

## **Table of Contents**

- 1. Introduction
- 2. Simple Example
- 3. Student Assignments
- 4. How to Run Java Tests in Visual Studio Code
- 5. Assignment Submission Guidelines

## 1. Introduction

This guidebook will help you learn how to write source code based on test code. You will practice analyzing test cases, writing the required implementation (source code), and verifying your solution using Visual Studio Code.

## 2. Simple Example

This section gives you a complete walk-through of how to read and understand a JUnit test, and how to write the matching source code.

We use a simple calculator class to demonstrate this process clearly.

When we build a program (like a sorting algorithm, searching algorithms, etc), we should make sure:

- It works correctly
- It doesn't break when something changes
- It handles errors properly

### 2.1 Problem Sample 1

That's where **test codes** come in—they help you automatically check whether our **main program behaves as expected**.

Test code for Calculator class	
CalculatorTest.java	
1	package dataStructure;
2	
3	import static org.junit.Assert.*;
4	import org.junit.Test;
5	
6	public class CalculatorTest {
7	
8	@Test
9	public void testAdd() {
10	Calculator calc = new Calculator();
11	int codeInput1 = 2;
12	int codeInput2 = 3;
13	int codeOutput = calc.add(codeInput1, codeInput2);
14	int expectedOutput = 5;
15	
16	try {
17	assertEquals("Addition test failed:", expectedOutput, codeOutput);
18	} catch (AssertionError ae) {
19	System.out.println(ae);
20	}
21	}
22	}

From the test case we create a class with the exact class and method names so that the test passes successfully.

The following is an explanation of each part of the test case:

1. `package dataStructure;`

The statement in line 1 places the file in a package named `dataStructure`.

Packages are used to organize Java classes into namespaces or modules.

Test cases and source code must be in the same package as illustrated below.

```
project-folder/
  └── src/
    └── dataStructure/
      ├── Calculator.java
      └── CalculatorTest.java
```

2. `import static org.junit.Assert.*;`

The import in line 3 imports all static methods from the `Assert` class in the JUnit framework — including:

- `assertEquals()`
- `assertTrue()`,
- `assertFalse()`

3. `import org.junit.Test;`

The import in line 4 needed when we use `@Test` annotation.

4. `public class CalculatorTest {`

Statement in line 6 defines a test class named `CalculatorTest`.

It will contain unit tests to verify the functionality of the `Calculator` class.

5. `@Test`

Annotation in line 8 tells JUnit that the method below is a test method, which should be executed when tests run.

6. `public void testAdd() {`

Statement in line 9 declares the test method called `testAdd`.

7. `Calculator calc = new Calculator();`

Statement in line 10 creates an object from the `Calculator` class so that you can call its methods.

```
8. int codeInput1 = 2;  
9. int codeInput2 = 3;
```

Statements in line 11 and 12 are the input values for the `add` method. Using separate variables for inputs is helpful for clarity and debugging.

```
10. int codeOutput = calc.add(codeInput1, codeInput2);
```

Statement in line 13 calls the `add` method with the given inputs and stores the result in `codeOutput`.

```
11. int expectedOutput = 5;
```

Statement in line 14 is the expected result of the `add(2, 3)` operation.

```
12. try {
```

Statement in line 16 starts a try-catch block — useful for **gracefully handling assertion failures** and printing messages.

```
13. assertEquals("Addition test failed:", expectedOutput, codeOutput);
```

Statement in line 17, compares `expectedOutput` with `codeOutput`.

If they are not equal, the assertion will fail and throw an `AssertionError` with the message "Addition test failed:".

```
14. } catch (AssertionError ae) {
```

Statement in line 18, catches the `AssertionError` if the test fails. This prevents the program from crashing and lets you log the error.

## 2.2 Answer Sample 1

Based on the test code `ArrayMinMaxTest.java`, we can create the source code with the following steps:

```
1. public class ArrayMinMax {
```

In the test case, an object is created using `new ArrayMinMax()`. Therefore, we need to define a class named `ArrayMinMax` to match the test case.

2. `public int[] findMinMax(int[] array) {`

The test calls a method named `findMinMax`, which accepts one parameter of type `int[]` (an integer array) and returns an `int[]` array.

Thus, we must create a method with the signature `int[] findMinMax(int[] array)` inside the class.

3. `return new int[] { min, max };`

After completing the search, return an array containing the found minimum value at index 0 and maximum value at index 1.

4. Provide comments in “**BAHASA INDONESIA**” in the method section, such as the code section **highlighted in yellow** below.

#### Source code which is made based on test code

Calculator.java

```
1 package dataStructure;
2
3 public class Calculator {
4
5     // Method untuk menjumlahkan dua bilangan bulat
6     public int add(int a, int b) {
7         // Mengembalikan hasil penjumlahan a dan b
8         return a + b;
9     }
10 }
```

## 2.3 Problem Sample 2

That’s where **test codes** come in—they help you automatically check whether our **main program behaves as expected**.

#### Test code for ArrayMinMax class

ArrayMinMaxTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class ArrayMinMaxTest {
7
8     @Test
9     public void testFindMinMax() {
10         int[] codeInput = { 80, 2, -3, 14, 95 };
11         int[] expectedOutput = { -3, 95 };
12
13         ArrayMinMax arraySum = new ArrayMinMax();
14         int[] codeOutput = arraySum.findMinMax(codeInput);
15
16         try {
17             assertEquals("Test Find MinMax:", expectedOutput, codeOutput);
18         } catch (AssertionError ae) {
```

```

19         System.out.println(ae);
20     }
21 }
22
23 @Test
24 public void testFindMinMaxEmpty() {
25     int[] codeInput = {};
26     // Empty Array, default value
27     int[] expectedOutput = { Integer.MAX_VALUE, Integer.MIN_VALUE };
28
29     ArrayMinMax arraySum = new ArrayMinMax();
30     int[] codeOutput = arraySum.findMinMax(codeInput);
31
32     // Memverifikasi output yang dihasilkan
33     try {
34         assertEquals("Test Find MinMax Empty:", expectedOutput,
35         codeOutput);
36     } catch (AssertionError ae) {
37         System.out.println(ae);
38     }
39 }

```

From the test case we create a class with the exact class and method names so that the test passes successfully.

## 2.4 Answer Sample 2

Based on the test code `ArrayMinMaxTest.java`, we can create the source code with the following steps:

1. `public class ArrayMinMax {`

In the test case, an object is created using `new ArrayMinMax()`. Therefore, we need to define a class named `Calculator` to match that.

2. `public int add(int a, int b) {`

The test calls a method named `add`, which takes **two integer parameters**. So, the class must include a method named `add` with this signature: `int add(int, int)`.

3. `return a + b;`

The test case expects the result of `add(2, 3)` to be 5. That means the method should return the sum of the two inputs `a` and `b`.

### Source code which is made based on test code

`Calculator.java`

1	<code>package dataStructure;</code>
2	
3	<code>public class ArrayMinMax {</code>
4	
5	<code>public int[] findMinMax(int[] array) {</code>
6	<code>if (array.length == 0) {</code>

```

7          // Jika array kosong, kembalikan nilai default (misalnya, min =
8          // Integer.MAX_VALUE dan max = Integer.MIN_VALUE)
9          return new int[] { Integer.MAX_VALUE, Integer.MIN_VALUE };
10         }
11
12         // Menganggap elemen pertama sebagai nilai min dan max
13         int min = array[0];
14         int max = array[0];
15
16         // Mengiterasi array untuk mencari nilai minimum dan maksimum
17         for (int num : array) {
18             if (num < min)
19                 min = num; // Menemukan nilai terkecil
20             if (num > max)
21                 max = num; // Menemukan nilai terbesar
22         }
23
24         // Mengembalikan nilai min dan max dalam array
25         return new int[] { min, max };
26     }

```

### 3. Problems

There are 12 problems to be solved with details explained in this section. All test cases are also available at the following link <https://github.com/mustmentari/test-cases-ASD->.

#### 3.1 Bubble Sort

Please create source code based on the test code below.

There are some additional explanations regarding the sorting algorithm as follows:

1. `import java.util.Arrays;`

This import is **necessary** because we are comparing two arrays.

2. `int codeInput2 = 2;`

We're testing **sorting logic with a limited number of iterations**. This allows the test case to check not just the final sorted array, but also **intermediate states** of the sorting process — for example, after 2 iterations only.

3. `int[] codeOutput = bSort.bubbleSort(codeInput1, codeInput2);`

Calls the sorting method and returns the result **after a limited number of passes(codeInput2)**. bSort is an instance of the BubbleSort class, which implements the method `bubbleSort(int[] arr, int maxIterations)`.

4. `assertEquals("Test 2:", Arrays.toString(expectedOutput),  
Arrays.toString(codeOutput));`

That compares **two arrays**, but converted to **Strings** first.

### Test code for BubbleSort class

#### BubbleSortTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class BubbleSortTest {
8     // Intermediate state testing
9     @Test
10    public void testBubbleSort1() {
11        BubbleSort bSort = new BubbleSort();
12        int[] codeInput1 = { 5, 2, 8, 1, 9 };
13        int codeInput2 = 2;
14        int[] codeOutput = bSort.bubbleSort(codeInput1, codeInput2);
15        // Expected state after 2 iterations
16        int[] expOutput = { 2, 1, 5, 8, 9 };
17
18        try {
19            assertEquals("Test 1:", Arrays.toString(expOutput), Arrays.toString(codeOutput));
20        } catch (AssertionError ae) {
21            System.out.println(ae);
22        }
23    }
24
25    @Test
26    public void testBubbleSort2() {
27        BubbleSort bSort = new BubbleSort();
28        int[] codeInput1 = { 121, -1, 9, 83, 52, -7, 3, -83, 75 };
29        int codeInput2 = 120;
30        int[] codeOutput = bSort.bubbleSort(codeInput1, codeInput2);
31        // Expected state after 120 iterations
32        int[] expOutput = { -83, -7, -1, 3, 9, 52, 75, 83, 121 };
33
34        try {
35            assertEquals("Test 2:", Arrays.toString(expOutput), Arrays.toString(codeOutput));
36        } catch (AssertionError ae) {
37            System.out.println(ae);
38        }
39    }
40
41    @Test
42    public void testBubbleSort3() {
43        BubbleSort bSort = new BubbleSort();
44        int[] codeInput1 = { -123, 53, 5, 23, -2, 48, 90, -83 };
45        int codeInput2 = -10;
46        int[] codeOutput = bSort.bubbleSort(codeInput1, codeInput2);
47        // Expected state after -10 iterations
48        int[] expOutput = { -123, 53, 5, 23, -2, 48, 90, -83 };
49
50        try {
51            assertEquals("Test 3:", Arrays.toString(expOutput), Arrays.toString(codeOutput));
52        } catch (AssertionError ae) {
53            System.out.println(ae);
54        }
55    }
56}
```

## 3.2 Selection Sort

Please create source code based on the test code below.

### Test code for SelectionSort class

#### SelectionSortTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class SelectionSortTest {
8     // Intermediate state testing
9     @Test
10    public void testSelectionSort1() {
11        SelectionSort sSort = new SelectionSort();
12        int[] codeInput1 = { 50, -23, 9, 33, -101, 89 };
13        int codeInput2 = 4;
14        int[] codeOutput = sSort.selectionSort(codeInput1, codeInput2);
15        int[] expectedOutput = { -101, -23, 9, 33, 50, 89 };
16
17        try {
18            assertEquals("Test 1:", Arrays.toString(expectedOutput),
19                Arrays.toString(codeOutput));
20            } catch (AssertionError ae) {
21                System.out.println(ae);
22            }
23
24        @Test
25        public void testSelectionSort2() {
26            SelectionSort sSort = new SelectionSort();
27            int[] codeInput1 = { 73, 3, -99, 47, 0, 890, -3, -56 };
28            int codeInput2 = 47;
29            int[] codeOutput = sSort.selectionSort(codeInput1, codeInput2);
30            int[] expectedOutput = { -99, -56, -3, 0, 3, 47, 73, 890 };
31
32            try {
33                assertEquals("Test 2:", Arrays.toString(expectedOutput),
34                    Arrays.toString(codeOutput));
35            } catch (AssertionError ae) {
36                System.out.println(ae);
37            }
38
39        @Test
40        public void testSelectionSort3() {
41            SelectionSort sSort = new SelectionSort();
42            int[] codeInput1 = { 65, -3, 0, 120, 23, 8, 90, -7, -22, 10 };
43            int codeInput2 = -50;
44            int[] codeOutput = sSort.selectionSort(codeInput1, codeInput2);
45            int[] expectedOutput = { 65, -3, 0, 120, 23, 8, 90, -7, -22, 10 };
46
47            try {
48                assertEquals("Test 3:", Arrays.toString(expectedOutput),
49                    Arrays.toString(codeOutput));
50            } catch (AssertionError ae) {
51                System.out.println(ae);
52            }
53        }
54    }
```

### 3.3 Insertion Sort

Please create source code based on the test code below.

### Test code for InsertionSort class

### InsertionSortTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class InsertionSortTest {
8     // Intermediate state testing
9     @Test
10    public void testInsertionSort1() {
11        InsertionSort iSort = new InsertionSort();
12        int[] codeInput1 = { -85, 44, 3, 77, -231 };
13        int codeInput2 = 3;
14        int[] codeOutput = iSort.insertionSort(codeInput1, codeInput2);
15        int[] expectedOutput = { -85, 3, 44, 77, -231 };
16
17        try {
18            assertEquals("Test 1:", Arrays.toString(expectedOutput),
19                Arrays.toString(codeOutput));
20        } catch (AssertionError ae) {
21            System.out.println(ae);
22        }
23
24        @Test
25        public void testInsertionSort2() {
26            InsertionSort iSort = new InsertionSort();
27            int[] codeInput1 = { 66, -1, 0, 104, -22, 8, 155, -7 };
28            int codeInput2 = 300;
29            int[] codeOutput = iSort.insertionSort(codeInput1, codeInput2);
30            int[] expectedOutput = { -22, -7, -1, 0, 8, 66, 104, 155 };
31
32            try {
33                assertEquals("Test 2:", Arrays.toString(expectedOutput),
34                    Arrays.toString(codeOutput));
35            } catch (AssertionError ae) {
36                System.out.println(ae);
37            }
38
39        @Test
40        public void testInsertionSort3() {
41            InsertionSort iSort = new InsertionSort();
42            int[] codeInput1 = { -7, 8, 99, -14, 89, 23, -123, 76, 33, -45 };
43            int codeInput2 = -8;
44            int[] codeOutput = iSort.insertionSort(codeInput1, codeInput2);
45            int[] expectedOutput = { -7, 8, 99, -14, 89, 23, -123, 76, 33, -45 };
46
47            try {
48                assertEquals("Test 3:", Arrays.toString(expectedOutput),
49                    Arrays.toString(codeOutput));
50            } catch (AssertionError ae) {
51                System.out.println(ae);
52            }
53        }
54    }
55}
```

## 3.4 Quick Sort

Please create source code based on the test code below.

There are some additional explanations regarding the quick sort algorithm as follows:

1. String codeInput3 = "middle";

This input parameter defines the pivot strategy used in the QuickSort algorithm. It can be "middle", "random", or "last".

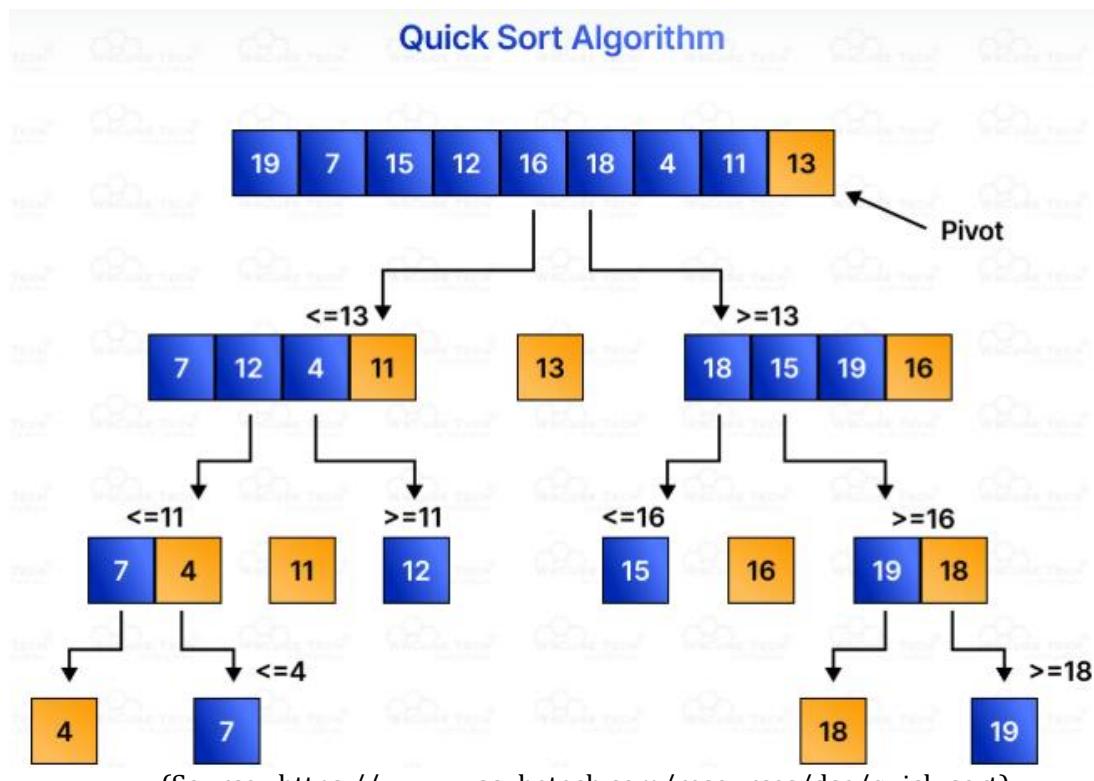
2. `int[] codeOutput = QuickSort.quickSort(codeInput1, codeInput2, codeInput3);`

Calls the updated QuickSort method that now accepts three parameters: the array to be sorted (`codeInput1`), the maximum number of iterations (`codeInput2`), and the pivot strategy (`codeInput3`).

3. `assertEquals(Arrays.toString(expectedOutput), Arrays.toString(codeOutput));`

Validates that the output of the sorting algorithm matches the expected result.

Quick Sort Algorithm illustration :



(Source : <https://www.wscubetech.com/resources/dsa/quick-sort>)

### Test code for QuickSort class

#### QuickSortTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import java.util.Arrays;
```

```

6  public class QuickSortTest {
7
8
9     @Test
10    // First Pivot - Intermediate state at iteration 2
11    public void testQuickSort1() {
12        int[] codeInput1 = { 89, 23, -90, 0, 117, 90, 2, 1, 8, -9 };
13        int codeInput2 = 2;
14        String codeInput3 = "first";
15
16        int[] codeOutput = QuickSort.quickSort(codeInput1, codeInput2, codeInput3);
17
18        // Simulated intermediate state after 2 iterations
19        int[] expectedOutput = { -90, -9, 0, 1, 23, 8, 2, 90, 117, 89 };
20
21        try {
22            assertEquals("Test 1 :", Arrays.toString(expectedOutput),
23                        Arrays.toString(codeOutput));
24        } catch (AssertionError ae) {
25            System.out.println(ae);
26        }
27
28     @Test
29     // Middle Pivot - Intermediate state at iteration 3
30     public void testQuickSort2() {
31         int[] codeInput1 = { -8, 0, 90, -12, 100, 23 };
32         int codeInput2 = 3;
33         String codeInput3 = "middle";
34
35         int[] codeOutput = QuickSort.quickSort(codeInput1, codeInput2, codeInput3);
36
37         // Simulated intermediate state after 3 iterations
38         int[] expectedOutput = { -12, -8, 0, 90, 23, 100 };
39
40         try {
41             assertEquals("Test 2 :", Arrays.toString(expectedOutput),
42                         Arrays.toString(codeOutput));
43         } catch (AssertionError ae) {
44             System.out.println(ae);
45         }
46
47     @Test
48     // Last Pivot - Intermediate state at iteration 4
49     public void testQuickSort3() {
50         int[] codeInput1 = { 90, -2, -11, 0, 23, 78, 9, 102, -4, 89, -9 };
51         int codeInput2 = 4;
52         String codeInput3 = "last";
53
54         int[] codeOutput = QuickSort.quickSort(codeInput1, codeInput2, codeInput3);
55
56         // Simulated intermediate state after 4 iterations
57         int[] expectedOutput = { -11, -9, -4, 0, -2, 23, 78, 9, 102, 89, 90 };
58
59         try {
60             assertEquals("Test 3 :", Arrays.toString(expectedOutput),
61                         Arrays.toString(codeOutput));
62         } catch (AssertionError ae) {
63             System.out.println(ae);
64         }
65     }
66 }

```

### 3.5 Shell Sort

Please create source code based on the test code below.

## Test code for ShellSort class

### ShellSortTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class ShellSortTest {
8     // Intermediate state testing
9     @Test
10    public void testShellSort1() {
11        ShellSort sSort = new ShellSort();
12        int[] codeInput1 = { -9, -99, 2, 78, 45, -105, 78, -33 };
13        int codeInput2 = 3;
14
15        int[] codeOutput = sSort.shellSort(codeInput1, codeInput2); // Sorting result
16        // Expected intermediate state
17        int[] expectedOutput = { -9, -33, -99, 2, 45, 78, -105, 78 };
18
19        try {
20            assertEquals("Test 1:", Arrays.toString(codeOutput),
21                Arrays.toString(expectedOutput));
21        } catch (AssertionError ae) {
22            System.out.println(ae);
23        }
24
25        @Test
26        public void testShellSort2() {
27            ShellSort sSort = new ShellSort();
28            int[] codeInput1 = { 67, -45, 0, -22, 0, 123, -3, -2, -34, 89, 2 };
29            int codeInput2 = 312;
30
31            int[] codeOutput = sSort.shellSort(codeInput1, codeInput2); // Sorting result
32            // Expected intermediate state
33            int[] expectedOutput = { -45, -34, -22, -3, -2, 0, 0, 2, 67, 89, 123 };
34
35            try {
36                assertEquals("Test 2:", Arrays.toString(codeOutput),
36                    Arrays.toString(expectedOutput));
37            } catch (AssertionError ae) {
38                System.out.println(ae);
39            }
40        }
41
42        @Test
43        public void testShellSort3() {
44            ShellSort sSort = new ShellSort();
45            int[] codeInput1 = { 8, -2, 89, -5, 47, 231, -9 };
46            int codeInput2 = -1;
47
48            int[] codeOutput = sSort.shellSort(codeInput1, codeInput2); // Sorting result
49            int[] expectedOutput = { 8, -2, 89, -5, 47, 231, -9 }; // Expected intermediate state
50
51            try {
52                assertEquals("Test 3:", Arrays.toString(codeOutput),
52                    Arrays.toString(expectedOutput));
53            } catch (AssertionError ae) {
54                System.out.println(ae);
55            }
56        }
57    }
```

## 3.6 Sequential Search

Please create source code based on the test code below.

There are some additional explanations regarding the searching algorithm as follows:

1. int codeInput2 = 317;

Statements in lines 14, 30 and 49 mention the variable `codeInput2` represent the value to find in the element array.

2. int[] expectedOutput = { 8, 9 }; // Found at index 8 after 9 iterations  
int[] expectedOutput = { -1, 10 }; // Not found after 10 checks

Statements in lines 16, 32, and 51 mention the unique statement to search tests. It not only checks whether the item was found, but also **how efficiently** (how many steps).

### Test code for SequentialSearch class

#### SequentialSearchTest.java

```
1 package datastructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class SequentialSearchTest {
8     @Test
9     public void testSequentialSearch1() {
10         SequentialSearch sSearch = new SequentialSearch();
11
12         int[] codeInput1 = { -1, 2, 15, 27, 90, 110, 113, 215, 317, 320 }; // Array to search
13         int codeInput2 = 317; // Target value to find
14         int[] codeOutput = sSearch.sequentialSearch(codeInput1, codeInput2);
15         int[] expectedOutput = { 8, 9 };
16         try {
17             assertEquals("Test 1: Sequential Search with target 317",
18                     Arrays.toString(expectedOutput),
19                     Arrays.toString(codeOutput));
20         } catch (AssertionError ae) {
21             System.out.println(ae);
22         }
23
24         @Test
25         public void testSequentialSearch2() {
26             SequentialSearch sSearch = new SequentialSearch();
27
28             int[] codeInput1 = { -1, 2, 15, 27, 90, 110, 113, 215, 317, 320 }; // Array to search
29             int codeInput2 = 15;
30             int[] codeOutput = sSearch.sequentialSearch(codeInput1, codeInput2);
31             int[] expectedOutput = { 2, 3 };
32
33             try {
34                 assertEquals("Test 1: Sequential Search with target 15",
35                         Arrays.toString(expectedOutput),
36                         Arrays.toString(codeOutput));
37             } catch (AssertionError ae) {
38                 System.out.println(ae);
39             }
40
41             // Test for target not found
42             @Test
43             public void testSequentialSearch3() {
44                 SequentialSearch sSearch = new SequentialSearch();
```

```

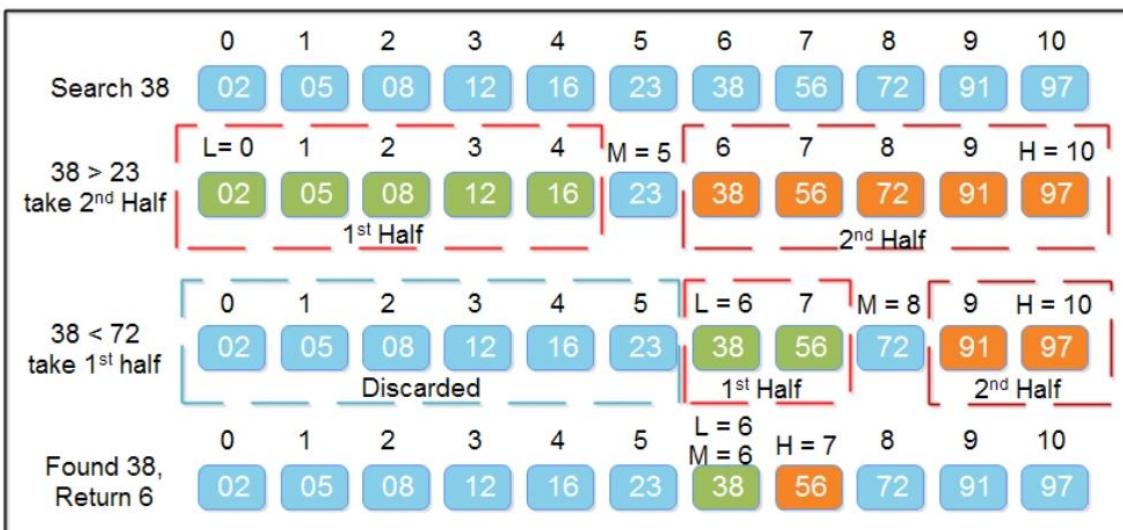
45     int[] codeInput1 = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 }; // Array to search
46     int codeInput2 = 321;
47     int[] codeOutput = sSearch.sequentialSearch(codeInput1, codeInput2);
48     int[] expectedOutput = { -1, 10 }; // Expected index (-1) and iterations (10)
49
50     try {
51         assertEquals("Test3: Sequential Search with target 321 (not found)",
52             Arrays.toString(expectedOutput),
53             Arrays.toString(codeOutput));
54     } catch (AssertionError ae) {
55         System.out.println(ae);
56     }
57 }

```

### 3.7 Binary Search

Please create source code based on the test code below.

Binary Search Illustration:



(Source : <https://ecomputernotes.com/what-is-c/binary-search-in-c>)

#### Test code for BinarySearch class

BinarySearchTest.java

```

1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import java.util.Arrays;
5 import org.junit.Test;
6
7 public class BinarySearchTest {
8     @Test
9     public void testBinarySearch1() {
10         // Initialize BinarySearch object
11         BinarySearch bSearch = new BinarySearch();
12
13         // Input for the test case: array and target
14         int[] codeInput1 = { -3, -1, 4, 7, 45, 51, 52, 70, 75, 90 }; // Array to search
15         int codeInput2 = 7; // Target value to find
16
17         // Call the binarySearch method
18         int[] codeOutput = bSearch.binarySearch(codeInput1, codeInput2);
}

```

```

19          // Expected output: index of the target and the number of iterations it took
20          int[] expectedResult = { 3, 4 };
21
22      try {
23          assertEquals("Test 1: Binary Search with target 7",
24          Arrays.toString(expectedOutput),
25          Arrays.toString(codeOutput));
26      } catch (AssertionError ae) {
27          System.out.println(ae);
28      }
29
30  @Test
31  public void testBinarySearch2() {
32      // Initialize BinarySearch object
33      BinarySearch bSearch = new BinarySearch();
34
35      // Input for the test case: array and target
36      int[] codeInput1 = { -3, -1, 4, 7, 45, 51, 52, 70, 75, 90 }; // Array to search
37      int codeInput2 = 75; // Target value to find
38
39      // Call the binarySearch method
40      int[] codeOutput = bSearch.binarySearch(codeInput1, codeInput2);
41
42      // Expected output: index of the target and the number of iterations it took
43      int[] expectedResult = { 8, 3 };
44
45      try {
46          assertEquals("Test 2: Binary Search with target 17",
47          Arrays.toString(expectedOutput),
48          Arrays.toString(codeOutput));
49      } catch (AssertionError ae) {
50          System.out.println(ae);
51      }
52
53  @Test
54  public void testBinarySearch3() {
55      // Initialize BinarySearch object
56      BinarySearch bSearch = new BinarySearch();
57
58      // Input for the test case: array and target
59      int[] codeInput1 = { -3, -1, 4, 7, 45, 51, 52, 70, 75, 90 }; // Array to search
60      int codeInput2 = 100; // Target value that is not in the array
61
62      // Call the binarySearch method
63      int[] codeOutput = bSearch.binarySearch(codeInput1, codeInput2);
64
65      // Expected output: -1 (not found) and the number of iterations it took
66      int[] expectedResult = { -1, 4 }; // Expected index (-1) and iterations (4)
67
68      try {
69          assertEquals("Test 3: Binary Search with target 100 (not found)",
70          Arrays.toString(expectedOutput),
71          Arrays.toString(codeOutput));
72      } catch (AssertionError ae) {
73          System.out.println(ae);
74      }
75  }

```

### 3.8 Stack

Please create source code based on the test code below.

There are some additional explanations regarding the stack algorithm as follows:

1. stack.isEmpty();

To verify whether the stack is empty after an operation

2. stack.isFull();

To test overflow conditions

### Test code for Stack class

#### StackTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class StackTest {
7
8     @Test
9     // push-pop-push-push
10    public void testStack1() {
11        Stack stack = new Stack(10);
12
13        // Input values
14        int push1 = 99;
15        int push2 = 50;
16        int push3 = -34;
17
18        // Expected and actual outputs
19        int expectedOutput1 = 99; // peek after 1st push
20        boolean expectedOutput2 = false; // isEmpty after 1st push
21        int codeOutput1 = 0;
22        boolean codeOutput2 = true;
23
24        int expectedOutput3 = 0; // peek after pop (stack empty)
25        boolean expectedOutput4 = true; // isEmpty after pop
26        int codeOutput3 = 0;
27        boolean codeOutput4 = false;
28
29        int expectedOutput5 = 50; // peek after 2nd push
30        boolean expectedOutput6 = false; // isEmpty after 2nd push
31        int codeOutput5 = 0;
32        boolean codeOutput6 = true;
33
34        int expectedOutput7 = -34; // peek after 3rd push
35        boolean expectedOutput8 = false; // isEmpty after 3rd push
36        int codeOutput7 = 0;
37        boolean codeOutput8 = true;
38
39        try {
40            // Step 1: Push(99)
41            stack.push(push1);
42            codeOutput1 = stack.peek();
43            codeOutput2 = stack.isEmpty();
44
45            assertEquals("Peek after first push", expectedOutput1, codeOutput1);
46            assertEquals("isEmpty after first push", expectedOutput2, codeOutput2);
47
48            // Step 2: Pop()
49            stack.pop();
50            codeOutput3 = stack.isEmpty() ? 0 : stack.peek(); // Peek if not empty
51            codeOutput4 = stack.isEmpty();
52
53            assertEquals("Peek after pop", expectedOutput3, codeOutput3);
54            assertEquals("isEmpty after pop", expectedOutput4, codeOutput4);
55
56            // Step 3: Push(50)
57        }
58    }
59}
```

```

54         stack.push(push2);
55         codeOutput5 = stack.peek();
56         codeOutput6 = stack.isEmpty();
57
58         assertEquals("Peek after second push", expectedOutput5, codeOutput5);
59         assertEquals("isEmpty after second push", expectedOutput6, codeOutput6);
60
61         // Step 4: Push(-34)
62         stack.push(push3);
63         codeOutput7 = stack.peek();
64         codeOutput8 = stack.isEmpty();
65
66         assertEquals("Peek after third push", expectedOutput7, codeOutput7);
67         assertEquals("isEmpty after third push", expectedOutput8, codeOutput8);
68
69     } catch (AssertionError ae) {
70         System.out.println(ae);
71     }
72 }
73
74 @Test
75 // pop-push-push-pop
76 public void testStack2() {
77     Stack stack = new Stack(10);
78
79     // Input values
80     int push1 = 77;
81     int push2 = -12;
82
83     // Expected and actual outputs
84     int expectedOutput1 = 0; // peek after pop on empty (assume 0 if empty)
85     boolean expectedOutput2 = true; // isEmpty should still be true
86     int codeOutput1 = 0;
87     boolean codeOutput2 = false;
88
89     int expectedOutput3 = 77; // peek after first push
90     boolean expectedOutput4 = false; // isEmpty should be false
91     int codeOutput3 = 0;
92     boolean codeOutput4 = true;
93
94     int expectedOutput5 = -12; // peek after second push
95     boolean expectedOutput6 = false;
96     int codeOutput5 = 0;
97     boolean codeOutput6 = true;
98
99     int expectedOutput7 = 77; // peek after final pop (should leave 77 on top)
100    boolean expectedOutput8 = false;
101    int codeOutput7 = 0;
102    boolean codeOutput8 = true;
103
104    try {
105        // Step 1: Pop (from empty stack)
106        stack.pop(); // should not crash if stack handles it gracefully
107        codeOutput1 = stack.isEmpty() ? 0 : stack.peek(); // avoid peek if empty
108        codeOutput2 = stack.isEmpty();
109
110        assertEquals("Peek after pop on empty stack", expectedOutput1, codeOutput1);
111        assertEquals("isEmpty after pop on empty stack", expectedOutput2, codeOutput2);
112
113        // Step 2: Push(77)
114        stack.push(push1);
115        codeOutput3 = stack.peek();
116        codeOutput4 = stack.isEmpty();
117
118        assertEquals("Peek after first push", expectedOutput3, codeOutput3);
119        assertEquals("isEmpty after first push", expectedOutput4, codeOutput4);
120
121        // Step 3: Push(-12)
122        stack.push(push2);
123        codeOutput5 = stack.peek();
124        codeOutput6 = stack.isEmpty();
125    }

```

```

126         assertEquals("Peek after second push", expectedOutput5, codeOutput5);
127         assertEquals("isEmpty after second push", expectedOutput6, codeOutput6);
128
129         // Step 4: Pop (should remove -12)
130         stack.pop();
131         codeOutput7 = stack.peek();
132         codeOutput8 = stack.isEmpty();
133
134
135         assertEquals("Peek after final pop", expectedOutput7, codeOutput7);
136         assertEquals("isEmpty after final pop", expectedOutput8, codeOutput8);
137
138     } catch (AssertionError ae) {
139         System.out.println(ae);
140     }
141 }
142
143 @Test
144 // push-pop-push until overflow
145 public void testStack3() {
146     Stack stack = new Stack(3);
147
148     // Input
149     int codeInput1 = 10;
150     int codeInput2 = 20;
151     int codeInput3 = 30;
152     int codeInput4 = 40;
153     int codeInput5 = 50; // overflow testing 3
154
155     // Expected and actual outputs
156     int expectedOutput1 = 10; // Peek after push(10)
157     boolean expectedOutput2 = false; // isEmpty after push(10)
158     int codeOutput1 = 0;
159     boolean codeOutput2 = true;
160
161     int expectedOutput3 = 20; // Peek after push(20)
162     boolean expectedOutput4 = false; // isEmpty after push(20)
163     int codeOutput3 = 0;
164     boolean codeOutput4 = true;
165
166     int expectedOutput5 = 10; // Peek after pop()
167     boolean expectedOutput6 = false;
168     int codeOutput5 = 0;
169     boolean codeOutput6 = true;
170
171     int expectedOutput7 = 30; // Peek after push(30)
172     boolean expectedOutput8 = false;
173     int codeOutput7 = 0;
174     boolean codeOutput8 = true;
175
176     int expectedOutput9 = 40; // Peek after push(40)
177     boolean expectedOutput10 = false;
178     int codeOutput9 = 0;
179     boolean codeOutput10 = true;
180
181     int expectedOutput11 = 50; // Peek after push(50)
182     boolean expectedOutput12 = true; // isFull after push(50), overflow
183     int codeOutput11 = 0;
184     boolean codeOutput12 = false;
185
186     try {
187         // Step 1: push(10)
188         stack.push(codeInput1);
189         codeOutput1 = stack.peek();
190         codeOutput2 = stack.isEmpty();
191         assertEquals("Peek after push(10)", expectedOutput1, codeOutput1);
192         assertEquals("isEmpty after push(10)", expectedOutput2, codeOutput2);
193
194         // Step 2: push(20)
195         stack.push(codeInput2);
196         codeOutput3 = stack.peek();

```

```

196     codeOutput4 = stack.isEmpty();
197     assertEquals("Peek after push(20)", expectedResult3, codeOutput3);
198     assertEquals("isEmpty after push(20)", expectedResult4, codeOutput4);
199
200     // Step 3: pop()
201     stack.pop();
202     codeOutput5 = stack.peek();
203     codeOutput6 = stack.isEmpty();
204     assertEquals("Peek after pop()", expectedResult5, codeOutput5);
205     assertEquals("isEmpty after pop()", expectedResult6, codeOutput6);
206
207     // Step 4: push(30)
208     stack.push(codeInput3);
209     codeOutput7 = stack.peek();
210     codeOutput8 = stack.isEmpty();
211     assertEquals("Peek after push(30)", expectedResult7, codeOutput7);
212     assertEquals("isEmpty after push(30)", expectedResult8, codeOutput8);
213
214     // Step 5: push(40)
215     stack.push(codeInput4);
216     codeOutput9 = stack.peek();
217     codeOutput10 = stack.isEmpty();
218     assertEquals("Peek after push(40)", expectedResult9, codeOutput9);
219     assertEquals("isEmpty after push(40)", expectedResult10, codeOutput10);
220
221     // Step 6: push(50) → overflow check
222     if (!stack.isFull()) {
223         stack.push(codeInput5);
224     }
225     codeOutput11 = stack.peek();
226     codeOutput12 = stack.isFull();
227
228     assertEquals("Peek after push(50)", expectedResult11, codeOutput11);
229     assertEquals("isFull after push(50)", expectedResult12, codeOutput12);
230
231 } catch (AssertionError ae) {
232     System.out.println(ae);
233 }
234 }
235 }
```

### 3.9 Queue

Please create source code based on the test code below.

#### Test code for Queue class

##### QueueTest.java

```

1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class QueueTest {
7
8     @Test
9     // enqueue-dequeue-enqueue-enqueue
10    public void testQueue1() {
11        Queue queue = new Queue(10);
12
13        // Input values
14        int enqueue1 = 99;
15        int enqueue2 = 50;
16        int enqueue3 = -34;
17
18        // Expected and actual outputs
19        int expectedResult1 = 99; // peek after 1st enqueue
```

```

20     boolean expectedOutput2 = false; // isEmpty after 1st enqueue
21     int codeOutput1 = 0;
22     boolean codeOutput2 = true;
23
24     int expectedOutput3 = 0; // peek after dequeue (queue empty)
25     boolean expectedOutput4 = true; // isEmpty after dequeue
26     int codeOutput3 = 0;
27     boolean codeOutput4 = false;
28
29     int expectedOutput5 = 50; // peek after 2nd enqueue
30     boolean expectedOutput6 = false; // isEmpty after 2nd enqueue
31     int codeOutput5 = 0;
32     boolean codeOutput6 = true;
33
34     int expectedOutput7 = -34; // peek after 3rd enqueue
35     boolean expectedOutput8 = false; // isEmpty after 3rd enqueue
36     int codeOutput7 = 0;
37     boolean codeOutput8 = true;
38
39     try {
40         // Step 1: Enqueue(99)
41         queue.enqueue(enqueue1);
42         codeOutput1 = queue.peek();
43         codeOutput2 = queue.isEmpty();
44
45         assertEquals("Peek after first enqueue", expectedOutput1, codeOutput1);
46         assertEquals("isEmpty after first enqueue", expectedOutput2, codeOutput2);
47
48         // Step 2: Dequeue()
49         queue.dequeue();
50         codeOutput3 = queue.isEmpty() ? 0 : queue.peek(); // Peek if not empty
51         codeOutput4 = queue.isEmpty();
52
53         assertEquals("Peek after dequeue", expectedOutput3, codeOutput3);
54         assertEquals("isEmpty after dequeue", expectedOutput4, codeOutput4);
55
56         // Step 3: Enqueue(50)
57         queue.enqueue(enqueue2);
58         codeOutput5 = queue.peek();
59         codeOutput6 = queue.isEmpty();
60
61         assertEquals("Peek after second enqueue", expectedOutput5, codeOutput5);
62         assertEquals("isEmpty after second enqueue", expectedOutput6, codeOutput6);
63
64         // Step 4: Enqueue(-34)
65         queue.enqueue(enqueue3);
66         codeOutput7 = queue.peek();
67         codeOutput8 = queue.isEmpty();
68
69         assertEquals("Peek after third enqueue", expectedOutput7, codeOutput7);
70         assertEquals("isEmpty after third enqueue", expectedOutput8, codeOutput8);
71
72     } catch (AssertionError ae) {
73         System.out.println(ae);
74     }
75
76     @Test
77     // dequeue-enqueue-enqueue-dequeue
78     public void testQueue2() {
79         Queue queue = new Queue(10);
80
81         // Input values
82         int enqueue1 = 77;
83         int enqueue2 = -12;
84
85         // Expected and actual outputs
86         int expectedOutput1 = 0; // peek after dequeue on empty (assume 0 if empty)
87         boolean expectedOutput2 = true; // isEmpty should still be true
88         int codeOutput1 = 0;
89         boolean codeOutput2 = false;

```

```

90
91     int expectedOutput3 = 77; // peek after first enqueue
92     boolean expectedOutput4 = false; // isEmpty should be false
93     int codeOutput3 = 0;
94     boolean codeOutput4 = true;
95
96     int expectedOutput5 = -12; // peek after second enqueue
97     boolean expectedOutput6 = false;
98     int codeOutput5 = 0;
99     boolean codeOutput6 = true;
100
102    int expectedOutput7 = 77; // peek after final dequeue (should leave 77 in front)
103    boolean expectedOutput8 = false;
104    int codeOutput7 = 0;
105    boolean codeOutput8 = true;
106
107   try {
108     // Step 1: Dequeue (from empty queue)
109     queue.dequeue(); // should not crash if queue handles it gracefully
110     codeOutput1 = queue.isEmpty() ? 0 : queue.peek(); // avoid peek if empty
111     codeOutput2 = queue.isEmpty();
112
113     assertEquals("Peek after dequeue on empty queue", expectedOutput1, codeOutput1);
114     assertEquals("isEmpty after dequeue on empty queue", expectedOutput2, codeOutput2);
115
116     // Step 2: Enqueue(77)
117     queue.enqueue(enqueue1);
118     codeOutput3 = queue.peek();
119     codeOutput4 = queue.isEmpty();
120
121     assertEquals("Peek after first enqueue", expectedOutput3, codeOutput3);
122     assertEquals("isEmpty after first enqueue", expectedOutput4, codeOutput4);
123
124     // Step 3: Enqueue(-12)
125     queue.enqueue(enqueue2);
126     codeOutput5 = queue.peek();
127     codeOutput6 = queue.isEmpty();
128
129     assertEquals("Peek after second enqueue", expectedOutput5, codeOutput5);
130     assertEquals("isEmpty after second enqueue", expectedOutput6, codeOutput6);
131
132     // Step 4: Dequeue (should remove 77)
133     queue.dequeue();
134     codeOutput7 = queue.peek();
135     codeOutput8 = queue.isEmpty();
136
137     assertEquals("Peek after final dequeue", expectedOutput7, codeOutput7);
138     assertEquals("isEmpty after final dequeue", expectedOutput8, codeOutput8);
139
140   } catch (AssertionError ae) {
141     System.out.println(ae);
142   }
143 }
144
145 @Test
146 // enqueue-dequeue-enqueue-enqueue (overflow)
147 public void testQueue3() {
148   Queue queue = new Queue(3);
149
150   // Input values
151   int enqueue1 = 10;
152   int enqueue2 = 20;
153   int enqueue3 = 30;
154   int enqueue4 = 40;
155
156   // Expected and actual outputs
157   int expectedOutput1 = 10; // Peek after enqueue(10)
158   boolean expectedOutput2 = false; // isEmpty after enqueue(10)
159   int codeOutput1 = 0;
160   boolean codeOutput2 = true;
161 }
```

```

162     int expectedOutput3 = 20; // Peek after enqueue(20)
163     boolean expectedOutput4 = false; // isEmpty after enqueue(20)
164     int codeOutput3 = 0;
165     boolean codeOutput4 = true;
166
167     int expectedOutput5 = 10; // Peek after dequeue
168     boolean expectedOutput6 = false;
169     int codeOutput5 = 0;
170     boolean codeOutput6 = true;
171
172     int expectedOutput7 = 30; // Peek after enqueue(30)
173     boolean expectedOutput8 = false;
174     int codeOutput7 = 0;
175     boolean codeOutput8 = true;
176
177     int expectedOutput9 = 40; // Peek after enqueue(40)
178     boolean expectedOutput10 = false;
179     int codeOutput9 = 0;
180     boolean codeOutput10 = true;
181
182     try {
183         // Step 1: Enqueue(10)
184         queue.enqueue(enqueue1);
185         codeOutput1 = queue.peek();
186         codeOutput2 = queue.isEmpty();
187         assertEquals("Peek after enqueue(10)", expectedOutput1, codeOutput1);
188         assertEquals("isEmpty after enqueue(10)", expectedOutput2, codeOutput2);
189
190         // Step 2: Enqueue(20)
191         queue.enqueue(enqueue2);
192         codeOutput3 = queue.peek();
193         codeOutput4 = queue.isEmpty();
194         assertEquals("Peek after enqueue(20)", expectedOutput3, codeOutput3);
195         assertEquals("isEmpty after enqueue(20)", expectedOutput4, codeOutput4);
196
197         // Step 3: Dequeue
198         queue.dequeue();
199         codeOutput5 = queue.peek();
200         codeOutput6 = queue.isEmpty();
201         assertEquals("Peek after dequeue", expectedOutput5, codeOutput5);
202         assertEquals("isEmpty after dequeue", expectedOutput6, codeOutput6);
203
204         // Step 4: Enqueue(30)
205         queue.enqueue(enqueue3);
206         codeOutput7 = queue.peek();
207         codeOutput8 = queue.isEmpty();
208         assertEquals("Peek after enqueue(30)", expectedOutput7, codeOutput7);
209         assertEquals("isEmpty after enqueue(30)", expectedOutput8, codeOutput8);
210
211         // Step 5: Enqueue(40)
212         queue.enqueue(enqueue4);
213         codeOutput9 = queue.peek();
214         codeOutput10 = queue.isEmpty();
215         assertEquals("Peek after enqueue(40)", expectedOutput9, codeOutput9);
216         assertEquals("isEmpty after enqueue(40)", expectedOutput10, codeOutput10);
217
218     } catch (AssertionError ae) {
219         System.out.println(ae);
220     }
221 }
222 }
```

### 3.10 Single Linked List

Please create source code based on the test code below.

## Test code for Single Linked List class

### SingleLinkedListTest.java

```
1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class SingleLinkedListTest {
7
8     @Test
9     // addFirst-RemoveFirst-addFirst-addFirst-removeFirst
10    public void testSLL1() {
11        SingleLinkedList list = new SingleLinkedList();
12
13        // Input values
14        int codeInput1 = 10;
15        int codeInput2 = 20;
16        int codeInput3 = 30;
17
18        // Expected and actual outputs
19        int expectedOutput1 = 10; // peek after first insert (head element)
20        boolean expectedOutput2 = false; // isEmpty after first insert
21        int codeOutput1 = 0;
22        boolean codeOutput2 = true;
23
24        int expectedOutput3 = 0; // peek after remove (list empty)
25        boolean expectedOutput4 = true; // isEmpty after remove
26        int codeOutput3 = 0;
27        boolean codeOutput4 = false;
28
29        int expectedOutput5 = 20; // peek after second insert (new head)
30        boolean expectedOutput6 = false; // isEmpty after second insert
31        int codeOutput5 = 0;
32        boolean codeOutput6 = true;
33
34        int expectedOutput7 = 30; // peek after third insert
35        boolean expectedOutput8 = false; // isEmpty after third insert
36        int codeOutput7 = 0;
37        boolean codeOutput8 = true;
38
39        try {
40            // Step 1: AddFirst(10)
41            list.addFirst(codeInput1);
42            codeOutput1 = list.peekFirst();
43            codeOutput2 = list.isEmpty();
44
45            assertEquals("Peek after first insert", expectedOutput1, codeOutput1);
46            assertEquals("isEmpty after first insert", expectedOutput2, codeOutput2);
47
48            // Step 2: RemoveFirst() (before remove, check if list is empty or not)
49            list.removeFirst();
50            codeOutput3 = list.isEmpty() ? 0 : list.peekFirst(); // Peek if not empty
51            codeOutput4 = list.isEmpty();
52
53            assertEquals("Peek after remove", expectedOutput3, codeOutput3);
54            assertEquals("isEmpty after remove", expectedOutput4, codeOutput4);
55
56            // Step 3: AddFirst(20)
57            list.addFirst(codeInput2);
58            codeOutput5 = list.peekFirst();
59            codeOutput6 = list.isEmpty();
60
61            assertEquals("Peek after second insert", expectedOutput5, codeOutput5);
62            assertEquals("isEmpty after second insert", expectedOutput6, codeOutput6);
63
64            // Step 4: AddFirst(30)
65            list.addFirst(codeInput3);
66            codeOutput7 = list.peekFirst();
67            codeOutput8 = list.isEmpty();
```

```

68     assertEquals("Peek after third insert", expectedOutput7, codeOutput7);
69     assertEquals("isEmpty after third insert", expectedOutput8, codeOutput8);
70
71     // Step 5: RemoveFirst() again
72     list.removeFirst();
73     int expectedOutput9 = 20; // peek should show 20 after removal of 30
74     boolean expectedOutput10 = false; // isEmpty should be false
75     codeOutput7 = list.peekFirst();
76     codeOutput8 = list.isEmpty();
77
78     assertEquals("Peek after second remove", expectedOutput9, codeOutput7);
79     assertEquals("isEmpty after second remove", expectedOutput10, codeOutput8);
80
81 } catch (AssertionError ae) {
82     System.out.println(ae);
83 }
84
85
86 @Test
87 // addLast-removeLast-addFirst-addFirst-removeLast
88 public void testSLL2() {
89     SingleLinkedList list = new SingleLinkedList();
90
91     // Input values
92     int codeInput1 = 10;
93     int codeInput2 = 20;
94     int codeInput3 = 30;
95
96     // Expected and actual outputs
97     int expectedOutput1 = 10; // peek after first insert (head element)
98     boolean expectedOutput2 = false; // isEmpty after first insert
99     int codeOutput1 = 0;
100    boolean codeOutput2 = true;
101
102    int expectedOutput3 = 0; // peek after remove (list empty)
103    boolean expectedOutput4 = true; // isEmpty after remove
104    int codeOutput3 = 0;
105    boolean codeOutput4 = false;
106
107    int expectedOutput5 = 20; // peek after second insert (new head)
108    boolean expectedOutput6 = false; // isEmpty after second insert
109    int codeOutput5 = 0;
110    boolean codeOutput6 = true;
111
112    int expectedOutput7 = 30; // peek after third insert
113    boolean expectedOutput8 = false; // isEmpty after third insert
114    int codeOutput7 = 0;
115    boolean codeOutput8 = true;
116
117    try {
118        // Step 1: AddLast(10)
119        list.addLast(codeInput1);
120        codeOutput1 = list.peekFirst();
121        codeOutput2 = list.isEmpty();
122
123        assertEquals("Peek after first insert", expectedOutput1, codeOutput1);
124        assertEquals("isEmpty after first insert", expectedOutput2, codeOutput2);
125
126        // Step 2: RemoveLast() (before remove, check if list is empty or not)
127        list.removeLast();
128        codeOutput3 = list.isEmpty() ? 0 : list.peekFirst(); // Peek if not empty
129        codeOutput4 = list.isEmpty();
130
131        assertEquals("Peek after remove", expectedOutput3, codeOutput3);
132        assertEquals("isEmpty after remove", expectedOutput4, codeOutput4);
133
134        // Step 3: AddFirst(20)
135        list.addFirst(codeInput2);
136        codeOutput5 = list.peekFirst();
137        codeOutput6 = list.isEmpty();
138
139    }

```

```

140         assertEquals("Peek after second insert", expectedOutput5, codeOutput5);
141         assertEquals("isEmpty after second insert", expectedOutput6, codeOutput6);
142
143         // Step 4: AddFirst(30)
144         list.addFirst(codeInput3);
145         codeOutput7 = list.peekFirst();
146         codeOutput8 = list.isEmpty();
147
148         assertEquals("Peek after third insert", expectedOutput7, codeOutput7);
149         assertEquals("isEmpty after third insert", expectedOutput8, codeOutput8);
150
151         // Step 5: RemoveLast() again
152         list.removeLast();
153         int expectedOutput9 = 20; // peek should show 20 after removal of 30
154         boolean expectedOutput10 = false; // isEmpty should be false
155         codeOutput7 = list.peekFirst();
156         codeOutput8 = list.isEmpty();
157
158         assertEquals("Peek after second remove", expectedOutput9, codeOutput7);
159         assertEquals("isEmpty after second remove", expectedOutput10, codeOutput8);
160
161     } catch (AssertionError ae) {
162         System.out.println(ae);
163     }
164 }
165
166 @Test
167 // removeFirst-addFirst-addFirst-removeFirst-addLast-addLast-removeLast
168 public void testSLL3() {
169     SingleLinkedList list = new SingleLinkedList();
170
171     // Input values
172     int codeInput1 = 10;
173     int codeInput2 = 20;
174     int codeInput3 = 30;
175     int codeInput4 = 40;
176
177     // Expected and actual outputs
178     boolean expectedOutput1 = true; // isEmpty after removeFirst on empty list
179     boolean codeOutput1 = false;
180
181     int expectedOutput2 = 10; // peek after addFirst(10)
182     boolean expectedOutput3 = false; // isEmpty after addFirst(10)
183     int codeOutput2 = 0;
184     boolean codeOutput3 = true;
185
186     int expectedOutput4 = 20; // peek after addFirst(20)
187     boolean expectedOutput5 = false; // isEmpty after addFirst(20)
188     int codeOutput4 = 0;
189     boolean codeOutput5 = true;
190
191     int expectedOutput6 = 10; // peek after removeFirst()
192     boolean expectedOutput7 = false; // still not empty
193     int codeOutput6 = 0;
194     boolean codeOutput7 = true;
195
196     int expectedOutput8 = 10; // peekFirst still 10 after addLast(30)
197     boolean expectedOutput9 = false;
198     int codeOutput8 = 0;
199     boolean codeOutput9 = true;
200
201     int expectedOutput10 = 10; // peekFirst still 10 after addLast(40)
202     boolean expectedOutput11 = false;
203     int codeOutput10 = 0;
204     boolean codeOutput11 = true;
205
206     int expectedOutput12 = 10; // peekFirst still 10 after removeLast()
207     boolean expectedOutput13 = false;
208     int codeOutput12 = 0;
209     boolean codeOutput13 = true;
210 }
```

```

211     try {
212         // Step 1: removeFirst() on empty list
213         list.removeFirst();
214         codeOutput1 = list.isEmpty();
215         assertEquals("isEmpty after removeFirst on empty", expectedOutput1, codeOutput1);
216
217         // Step 2: addFirst(10)
218         list.addFirst(codeInput1);
219         codeOutput2 = list.peekFirst();
220         codeOutput3 = list.isEmpty();
221         assertEquals("Peek after addFirst(10)", expectedOutput2, codeOutput2);
222         assertEquals("isEmpty after addFirst(10)", expectedOutput3, codeOutput3);
223
224         // Step 3: addFirst(20)
225         list.addFirst(codeInput2);
226         codeOutput4 = list.peekFirst();
227         codeOutput5 = list.isEmpty();
228         assertEquals("Peek after addFirst(20)", expectedOutput4, codeOutput4);
229         assertEquals("isEmpty after addFirst(20)", expectedOutput5, codeOutput5);
230
231         // Step 4: removeFirst() → front should now be 10
232         list.removeFirst();
233         codeOutput6 = list.peekFirst();
234         codeOutput7 = list.isEmpty();
235         assertEquals("Peek after removeFirst()", expectedOutput6, codeOutput6);
236         assertEquals("isEmpty after removeFirst()", expectedOutput7, codeOutput7);
237
238         // Step 5: addLast(30)
239         list.addLast(codeInput3);
240         codeOutput8 = list.peekFirst();
241         codeOutput9 = list.isEmpty();
242         assertEquals("Peek after addLast(30)", expectedOutput8, codeOutput8);
243         assertEquals("isEmpty after addLast(30)", expectedOutput9, codeOutput9);
244
245         // Step 6: addLast(40)
246         list.addLast(codeInput4);
247         codeOutput10 = list.peekFirst();
248         codeOutput11 = list.isEmpty();
249         assertEquals("Peek after addLast(40)", expectedOutput10, codeOutput10);
250         assertEquals("isEmpty after addLast(40)", expectedOutput11, codeOutput11);
251
252         // Step 7: removeLast()
253         list.removeLast();
254         codeOutput12 = list.peekFirst();
255         codeOutput13 = list.isEmpty();
256         assertEquals("Peek after removeLast()", expectedOutput12, codeOutput12);
257         assertEquals("isEmpty after removeLast()", expectedOutput13, codeOutput13);
258
259     } catch (AssertionError ae) {
260         System.out.println("Test failed: " + ae.getMessage());
261     }
262 }
263 }
```

### 3.11 Double Linked List

Please create source code based on the test code below.

<b>Test code for Double Linked List class</b>
DoubleLinkedListTest.java

```

1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class DoubleLinkedListTest {
```

```

7
8     @Test
9     // addFirst-removeFirst-addFirst-addFirst-removeFirst
10    public void testDLL1() {
11        DoubleLinkedList list = new DoubleLinkedList();
12
13        // Input values
14        int codeInput1 = -50;
15        int codeInput2 = 200;
16        int codeInput3 = 15;
17
18        // Expected outputs
19        int expectedOutput1 = -50; // peek after first insert
20        boolean expectedOutput2 = false;
21        int expectedOutput3 = 0; // peek after remove (empty)
22        boolean expectedOutput4 = true;
23
24        int expectedOutput5 = 200;
25        boolean expectedOutput6 = false;
26
27        int expectedOutput7 = 15;
28        boolean expectedOutput8 = false;
29
30        int expectedOutput9 = 200; // peek after removeFirst again
31        boolean expectedOutput10 = false;
32
33        try {
34            list.addFirst(codeInput1);
35            assertEquals(expectedOutput1, list.peekFirst());
36            assertEquals(expectedOutput2, list.isEmpty());
37
38            list.removeFirst();
39            assertEquals(expectedOutput3, list.isEmpty() ? 0 : list.peekFirst());
40            assertEquals(expectedOutput4, list.isEmpty());
41
42            list.addFirst(codeInput2);
43            assertEquals(expectedOutput5, list.peekFirst());
44            assertEquals(expectedOutput6, list.isEmpty());
45
46            list.addFirst(codeInput3);
47            assertEquals(expectedOutput7, list.peekFirst());
48            assertEquals(expectedOutput8, list.isEmpty());
49
50            list.removeFirst();
51            assertEquals(expectedOutput9, list.peekFirst());
52            assertEquals(expectedOutput10, list.isEmpty());
53
54        } catch (AssertionError ae) {
55            System.out.println(ae);
56        }
57    }
58
59    @Test
60    // addLast-removeLast-addFirst-addLast-removeLast
61    public void testDLL2() {
62        DoubleLinkedList list = new DoubleLinkedList();
63
64        // Input values
65        int codeInput1 = 120;
66        int codeInput2 = -30;
67        int codeInput3 = 75;
68
69        // Expected outputs
70        int expectedOutput1 = 120; // peek after addLast(120)
71        boolean expectedOutput2 = false; // isEmpty after addLast(120)
72
73        int expectedOutput3 = 0; // peek after removeLast (empty)
74        boolean expectedOutput4 = true; // isEmpty after removeLast
75
76        int expectedOutput5 = -30; // peek after addFirst(-30)
77        boolean expectedOutput6 = false; // isEmpty after addFirst(-30)

```

```

77
78     int expectedOutput7 = -30; // peek after addLast(75)
79     int expectedOutput8 = -30; // peek after removeLast
80
81     boolean expectedOutput9 = false; // isEmpty after removeLast
82
83     try {
84         // Step 1: addLast(120)
85         list.addLast(codeInput1);
86         assertEquals(expectedOutput1, list.peekFirst());
87         assertEquals(expectedOutput2, list.isEmpty());
88
89         // Step 2: removeLast()
90         list.removeLast();
91         assertEquals(expectedOutput3, list.isEmpty() ? 0 : list.peekFirst());
92         assertEquals(expectedOutput4, list.isEmpty());
93
94         // Step 3: addFirst(-30)
95         list.addFirst(codeInput2);
96         assertEquals(expectedOutput5, list.peekFirst());
97         assertEquals(expectedOutput6, list.isEmpty());
98
99         // Step 4: addLast(75)
100        list.addLast(codeInput3);
101        assertEquals(expectedOutput7, list.peekFirst());
102
103        // Step 5: removeLast()
104        list.removeLast();
105        assertEquals(expectedOutput8, list.peekFirst());
106        assertEquals(expectedOutput9, list.isEmpty());
107
108    } catch (AssertionError ae) {
109        System.out.println(ae);
110    }
111 }
112
113
114 @Test
115 // removeFirst, addFirst, addLast, addFirst, removeFirst, removeLast,
116 // removeFirst.
117 public void testDLL3() {
118     DoubleLinkedList list = new DoubleLinkedList();
119
120     // Input values
121     int codeInput1 = 0;
122     int codeInput2 = 999;
123     int codeInput3 = -100;
124
125     // Expected outputs
126     boolean expectedOutput1 = true; // isEmpty after removeFirst (should not crash)
127     int expectedOutput2 = 0; // peek after addFirst(0)
128     boolean expectedOutput3 = false; // isEmpty after addFirst(0)
129     int expectedOutput4 = 0; // peek after addLast(999)
130     boolean expectedOutput5 = false; // isEmpty after addLast(999)
131     int expectedOutput6 = -100; // peek after addFirst(-100)
132     boolean expectedOutput7 = false; // isEmpty after addFirst(-100)
133     int expectedOutput8 = 0; // peek after removeFirst()
134     boolean expectedOutput9 = false; // isEmpty after removeFirst()
135     int expectedOutput10 = 0; // peek after removeLast()
136     boolean expectedOutput11 = true; // isEmpty after removeLast()
137
138     try {
139         // Step 1: removeFirst() (should not crash)
140         list.removeFirst();
141         assertEquals(expectedOutput1, list.isEmpty());
142
143         // Step 2: addFirst(0)
144         list.addFirst(codeInput1);
145         assertEquals(expectedOutput2, list.peekFirst());
146         assertEquals(expectedOutput3, list.isEmpty());
147
148         // Step 3: addLast(999)

```

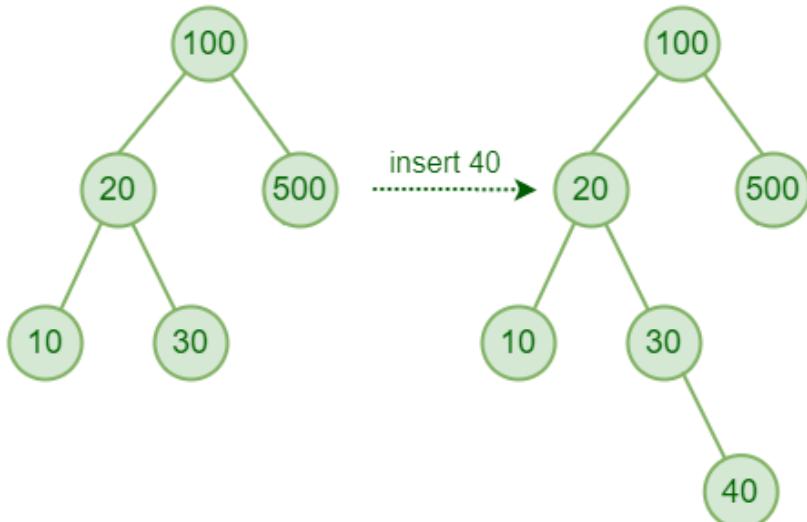
```

149     list.addLast(codeInput2);
150     assertEquals(expectedOutput4, list.peekFirst());
151     assertEquals(expectedOutput5, list.isEmpty());
152
153     // Step 4: addFirst(-100)
154     list.addFirst(codeInput3);
155     assertEquals(expectedOutput6, list.peekFirst());
156     assertEquals(expectedOutput7, list.isEmpty());
157
158     // Step 5: removeFirst()
159     list.removeFirst();
160     assertEquals(expectedOutput8, list.peekFirst());
161     assertEquals(expectedOutput9, list.isEmpty());
162
163     // Step 6: removeLast()
164     list.removeLast();
165     assertEquals(expectedOutput10, list.peekFirst());
166     assertEquals(expectedOutput11, list.isEmpty());
167
168     // Step 7: removeFirst()
169     list.removeFirst();
170     assertEquals(expectedOutput11, list.isEmpty());
171
172 } catch (AssertionError ae) {
173     System.out.println(ae);
174 }
175
176 }
177 }
```

### 3.12 Binary Tree

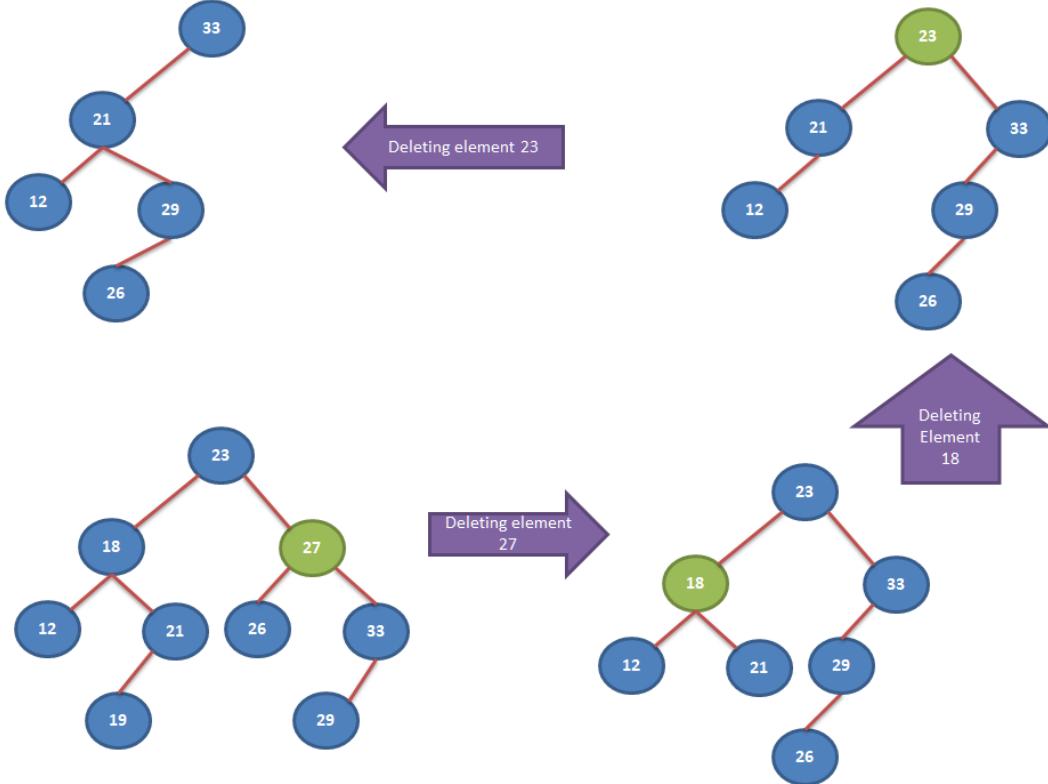
Please create source code based on the test code below.

Binary Tree inserts process illustration:



(Source : <https://venkys.io/articles/details/binary-search-tree-insertion>)

Binary Tree delete process illustration:



(Source : <https://simpletechtalks.com/binary-search-tree-deletion-of-node-explained-with-simple-example/>)

### Test code for Binary Tree class

#### BinaryTreeTest.java

```

1 package dataStructure;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class BinaryTreeTest {
7
8     @Test
9     public void testBST1() {
10         BinaryTree bst = new BinaryTree();
11
12         int codeInput1 = 50;
13         int codeInput2 = 30;
14         int codeInput3 = 70;
15         int codeInput4 = 20;
16         int codeInput5 = 40;
17         int codeInput6 = 60;
18         int codeInput7 = 80;
19
20         try {
21             // Insert elements
22             bst.insert(codeInput1);
23             bst.insert(codeInput2);
24             bst.insert(codeInput3);
25             bst.insert(codeInput4);
26             bst.insert(codeInput5);
27             bst.insert(codeInput6);
28             bst.insert(codeInput7);
29
30             assertEquals(50, bst.root.data);
31             assertEquals(30, bst.root.left.data);
32             assertEquals(70, bst.root.right.data);
33             assertEquals(20, bst.root.left.left.data);
34             assertEquals(40, bst.root.left.right.data);
35             assertEquals(60, bst.root.right.left.data);
36             assertEquals(80, bst.root.right.right.data);
37
38             bst.delete(50);
39
40             assertEquals(40, bst.root.data);
41             assertEquals(30, bst.root.left.data);
42             assertEquals(60, bst.root.right.data);
43             assertEquals(20, bst.root.left.left.data);
44             assertEquals(80, bst.root.left.right.data);
45             assertEquals(null, bst.root.right.left);
46             assertEquals(null, bst.root.right.right);
47
48             bst.delete(40);
49
50             assertEquals(30, bst.root.data);
51             assertEquals(null, bst.root.left);
52             assertEquals(60, bst.root.right.data);
53             assertEquals(null, bst.root.right.left);
54             assertEquals(null, bst.root.right.right);
55
56             bst.delete(30);
57
58             assertEquals(null, bst.root);
59         } catch (Exception e) {
60             e.printStackTrace();
61         }
62     }
63 }
```

```

28
29         // Searches
30         assertTrue(bst.search(codeInput1));
31         assertFalse(bst.search(100));
32         assertTrue(bst.search(codeInput2));
33         assertTrue(bst.search(codeInput3));
34
35         // Delete and re-check
36         bst.delete(codeInput2); // Remove 30
37         assertFalse(bst.search(codeInput2));
38         assertTrue(bst.search(codeInput5)); // 40 still exists
39
40     } catch (AssertionError ae) {
41         System.out.println(ae);
42     }
43 }
44
45 @Test
46 public void testBST2() {
47     BinaryTree bst = new BinaryTree();
48
49     int codeInput1 = 50;
50     int codeInput2 = 30;
51     int codeInput3 = 70;
52     int codeInput4 = 20;
53     int codeInput5 = 40;
54
55     try {
56         // Delete from empty
57         bst.delete(50);
58         assertTrue(bst.isEmpty());
59
60         // Search in empty
61         assertFalse(bst.search(50));
62         assertFalse(bst.search(30));
63
64         // Insert elements
65         bst.insert(codeInput1);
66         bst.insert(codeInput2);
67         bst.insert(codeInput3);
68         bst.insert(codeInput4);
69         bst.insert(codeInput5);
70
71         assertTrue(bst.search(codeInput5)); // 40
72         assertTrue(bst.search(codeInput4)); // 20
73
74         bst.delete(codeInput5); // Remove 40
75         bst.delete(codeInput4); // Remove 20
76
77         assertFalse(bst.search(codeInput5));
78         assertFalse(bst.search(codeInput4));
79         assertFalse(bst.isEmpty()); // Tree still has nodes
80
81     } catch (AssertionError ae) {
82         System.out.println(ae);
83     }
84 }
85
86 @Test
87 public void testBST3() {
88     BinaryTree bst = new BinaryTree();
89
90     int codeInput1 = 25;
91     int codeInput2 = 15;
92     int codeInput3 = 35;
93     int codeInput4 = 20;
94
95     try {
96         bst.insert(codeInput1);
97         assertTrue(bst.search(codeInput1));
98

```

```

99         assertFalse(bst.search(100));
100
101         bst.insert(codeInput2);
102         bst.insert(codeInput3);
103         assertTrue(bst.search(codeInput2));
104
105         bst.delete(codeInput2);
106         assertFalse(bst.search(codeInput2));
107
108         bst.insert(codeInput4);
109         assertTrue(bst.search(codeInput4));
110
111         bst.delete(codeInput3);
112         assertFalse(bst.search(codeInput3));
113
114     } catch (AssertionError ae) {
115         System.out.println(ae);
116     }
117 }
118 }
119 }
```

#### 4. How to Run Java Tests in Visual Studio Code

1. Install Visual Studio Code from <https://code.visualstudio.com>
2. Install the Java Extension Pack from the Extensions Marketplace
3. Create a new folder for your Java project and open it in VS Code
4. Add your `java` files inside the folder `dataStructure` (e.g., `Calculator.java` and `CalculatorTest.java`)
5. Use the integrated terminal:

`javac -cp .:junit-4.13.2.jar:hamcrest-core-1.3.jar *.java`

`java -cp .:junit-4.13.2.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest`

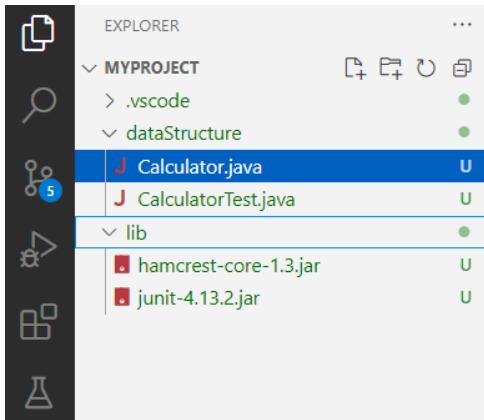
**or**

put .jar files inside the folder “lib” like the pictures below. You can directly download .jar files from this link <https://github.com/mustmentari/test-cases-ASD-/tree/main>

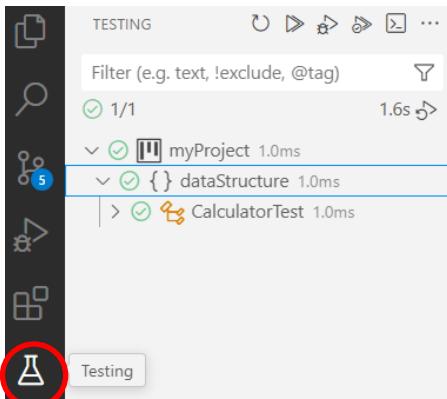
6. Create a java testing file in the package you have created. Copy and paste the test code provided in this guidebook.
7. Create another new java file to store your source. Then, create source code according to the test code in each question.
8. Analyze the output to debug and refine your implementation

 Example Screenshot:

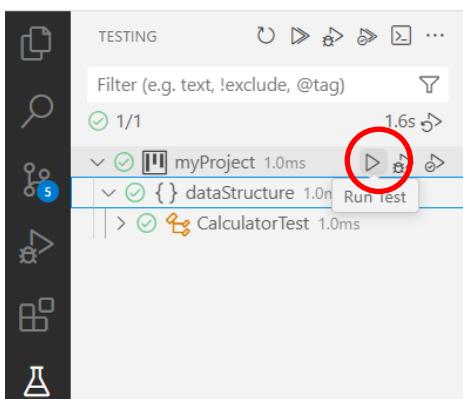
The image below shows the file structure within the project.



Then, click the testing icon according to the part circled in red in the image below. If this icon does not appear, make sure you have run the project.



Then, click run test according to the section circled in red in the image below to ensure that your source code matches the existing test code. If there is a green checklist mark shown according to the image, then your source code is correct, if a red cross appears, it means that the source code you created is not in accordance with the existing test code.



## 5. Assignment Submission Guidelines

### Submission Deadline:

#### Deadline for problems 1-6

- **Date:** April 28, 2025
- **Time:** 11:59 PM (WIB)

#### Deadline for problems 7-12

- **Date:** Mei 8, 2025
- **Time:** 11:59 PM (WIB)

### Submission Steps:

#### 1. Prepare Your Assignment File

Ensure your assignment file is in the correct format and is complete. The file name should follow this format:

Assignment1\_FullName\_Date

Example: Assignment1\_MustikaMentari\_28Apr2025

#### 2. Create a Google Drive Folder

Create a new folder in Google Drive with the following name:

Assignment [Course/Topic] - [Full Name]

Example: Assignment Programming 1 - Mustika Mentari

#### 3. Upload Your File

After creating the folder (Your project folder), upload your assignment file into that folder.

Make sure the uploaded file is the correct one and follow the instructions given.

#### 4. Share the Folder

Once the file is uploaded, make sure that the folder/file can be accessed by the instructor or teaching assistants. To do so:

- Right-click on the folder or file, select "**Get link**".
- Set the access to "**Anyone with the link**" and make sure the permission is set to "**Viewer**".

#### 5. Send the Google Drive Link

Send the link to your Google Drive folder to google formulir on this link

<https://forms.gle/yvs2M1MbExb4a9B57>.