

SE 3170: Lab 3a

Testing Boundary Conditions Using Exceptions Handling

1. TODO:

Part 1

- 1- First, use **throws** method to fix the code. When you finish, take the screenshot of the passed result with your code.

```
1 package part_a;
2 import org.junit.*;[]
5
6 public class BearingTest {
7
8     @Test
9     public void answersValidBearing() throws Exception {
10         assertEquals(new Bearing(Bearing.MAX).value(), Bearing.MAX);
11     }
12
13     @Test
14     public void answersAngleBetweenItAndAnotherBearing() throws Exception {
15         assertEquals(new Bearing(15).angleBetween(new Bearing(12)), 3);
16     }
17
18     @Test
19     public void angleBetweenIsNegativeWhenThisBearingSmaller() throws Exception {
20         assertEquals(new Bearing(12).angleBetween(new Bearing(15)), -3);
21     }
22
23 }
24
```

- 2- Next, Replace the throw exception with the “try/catch” method, run the code again and submit the screenshot of the **passed** result with your code

```
1 package part_a;
2 import org.junit.*;[]
5
6 public class BearingTest {
7
8     @Test
9     public void answersValidBearing() {
10         try {
11             assertEquals(new Bearing(Bearing.MAX).value(), Bearing.MAX);
12         } catch (Exception e) {
13             e.printStackTrace();
14         }
15     }
16
17     @Test
18     public void answersAngleBetweenItAndAnotherBearing() {
19         try {
20             assertEquals(new Bearing(15).angleBetween(new Bearing(12)), 3);
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
25
26     @Test
27     public void angleBetweenIsNegativeWhenThisBearingSmaller() {
28         try {
29             assertEquals(new Bearing(12).angleBetween(new Bearing(15)), -3);
30         } catch (Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35 }
36
```

2. TODO:

Write 8 test cases

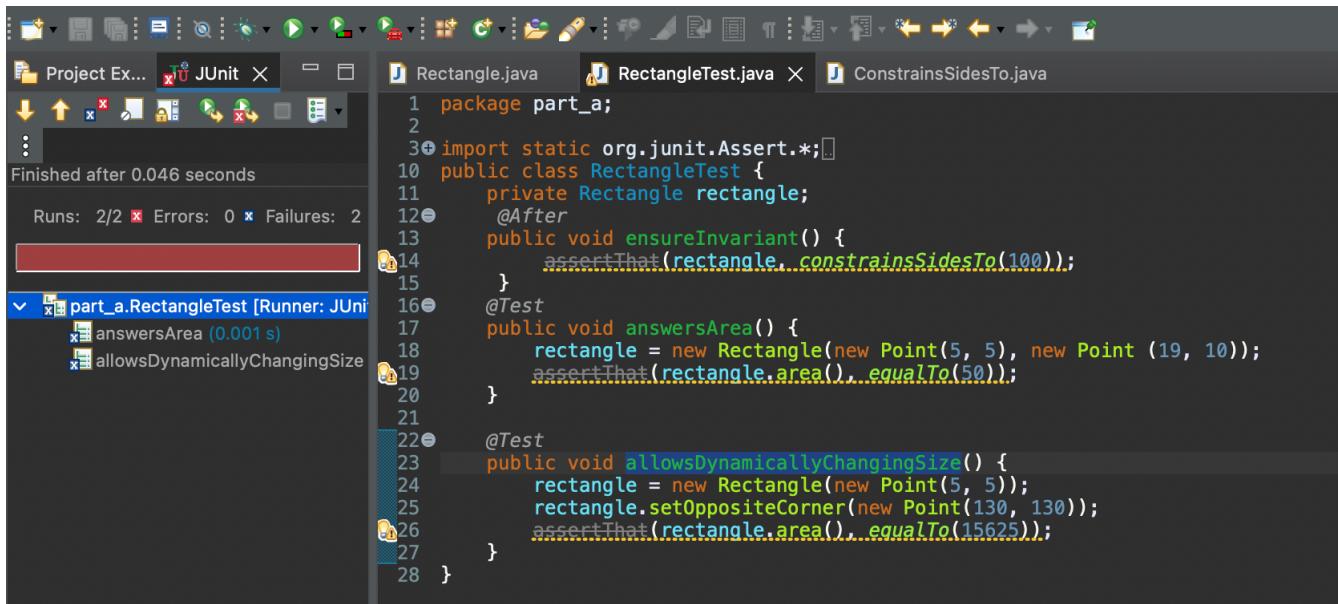
The screenshot shows an IDE interface with several windows. At the top, there's a toolbar with various icons. Below it, a tab bar with three tabs: 'Bearing.java', 'BearingTest.java X', and 'BearingOutOfRangeException.java'. The 'BearingTest.java' tab is currently active.

In the center, the code editor displays the 'BearingTest.java' file. The code contains eight test methods, each using the `@Test` annotation and `assertThat` to check the `angleBetween` method of the `Bearing` class. The test cases cover angles between 0 and 355 degrees, 90 and 180 degrees, 100 and 55 degrees, 12 and 123 degrees, 360 and 0 degrees, and 200 and 300 degrees. Each test includes a try-catch block to handle exceptions and print stack traces if assertions fail.

```
54     //part 2
55
56● @Test
57     public void angleBetweenZeroAnd355() {
58         try {
59             assertThat(new Bearing(0).angleBetween(new Bearing(355)), equalTo(-355));
60         } catch (Exception e) {
61             e.printStackTrace();
62         }
63     }
64
65● @Test
66     public void angleBetween90And180() {
67         try {
68             assertThat(new Bearing(90).angleBetween(new Bearing(180)), equalTo(-90));
69         } catch (Exception e) {
70             e.printStackTrace();
71         }
72     }
73
74● @Test
75     public void angleBetween100And55() {
76         try {
77             assertThat(new Bearing(100).angleBetween(new Bearing(55)), equalTo(45));
78         } catch (Exception e) {
79             e.printStackTrace();
80         }
81     }
82
83● @Test
84     public void angleBetween12And123() {
85         try {
86             assertThat(new Bearing(12).angleBetween(new Bearing(123)), equalTo(-111));
87         } catch (Exception e) {
88             e.printStackTrace();
89         }
90     }
91
92● @Test
93     public void angleBetween360And0() {
94         try {
95             assertThat(new Bearing(360).angleBetween(new Bearing(0)), equalTo(360));
96         } catch (Exception e) {
97             e.printStackTrace();
98         }
99     }
100
101● @Test
102     public void angleBetween200And300() {
103         try {
104             assertThat(new Bearing(200).angleBetween(new Bearing(300)), equalTo(-100));
105         } catch (Exception e) {
106             e.printStackTrace();
107         }
108     }
109
110
111
112
113
114
115
116
117
```

On the left side of the code editor, there's a vertical bar with some icons. At the bottom left, there's a 'Failure Trace' button. On the right side of the code editor, there are several small icons.

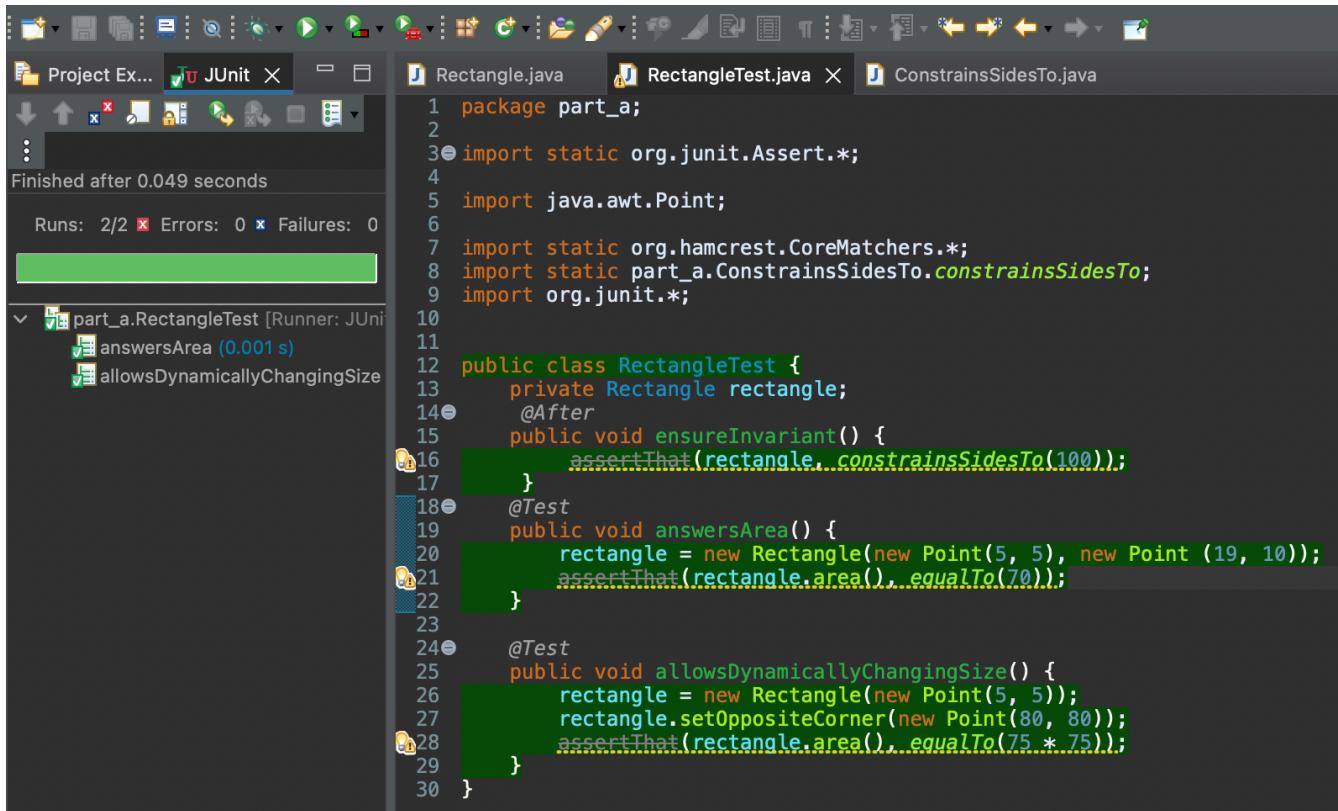
TODO:2



```
1 package part_a;
2
3+ import static org.junit.Assert.*;
10 public class RectangleTest {
11     private Rectangle rectangle;
12     @After
13     public void ensureInvariant() {
14         assertThat(rectangle, constraintsSidesTo(100));
15     }
16     @Test
17     public void answersArea() {
18         rectangle = new Rectangle(new Point(5, 5), new Point(19, 10));
19         assertThat(rectangle.area(), equalTo(50));
20     }
21     @Test
22     public void allowsDynamicallyChangingSize() {
23         rectangle = new Rectangle(new Point(5, 5));
24         rectangle.setOppositeCorner(new Point(130, 130));
25         assertThat(rectangle.area(), equalTo(15625));
26     }
27 }
28 }
```

Yes, there are errors!

3. TODO: Fix the error(s) of the code and run the code again.



```
1 package part_a;
2
3+ import static org.junit.Assert.*;
4
5 import java.awt.Point;
6
7 import static org.hamcrest.CoreMatchers.*;
8 import static part_a.ConstraintsSidesTo.constraintsSidesTo;
9 import org.junit.*;
10
11
12 public class RectangleTest {
13     private Rectangle rectangle;
14     @After
15     public void ensureInvariant() {
16         assertThat(rectangle, constraintsSidesTo(100));
17     }
18     @Test
19     public void answersArea() {
20         rectangle = new Rectangle(new Point(5, 5), new Point(19, 10));
21         assertThat(rectangle.area(), equalTo(70));
22     }
23     @Test
24     public void allowsDynamicallyChangingSize() {
25         rectangle = new Rectangle(new Point(5, 5));
26         rectangle.setOppositeCorner(new Point(80, 80));
27         assertThat(rectangle.area(), equalTo(75 * .75));
28     }
29 }
30 }
```

4. TODO:

Answer the following questions:

1. What is throw exception and how does it fix the code?

-The throw keyword is used to manually generate an exception when an error condition occurs.
-It stops the execution of the program unless handled by try-catch.
-In the lab, it helps ensure that invalid values (e.g., out-of-range bearings) do not proceed, preventing incorrect calculations.

2. What is try-catch method and how does it fix the code

-try-catch is used to handle exceptions gracefully instead of stopping execution.
-It allows the program to continue running by catching errors and executing alternative logic.
-In the lab, it ensures that even if an exception occurs, the program can recover instead of crashing.

3. Is there any difference between throw exception and try-catch method? If yes, explain.

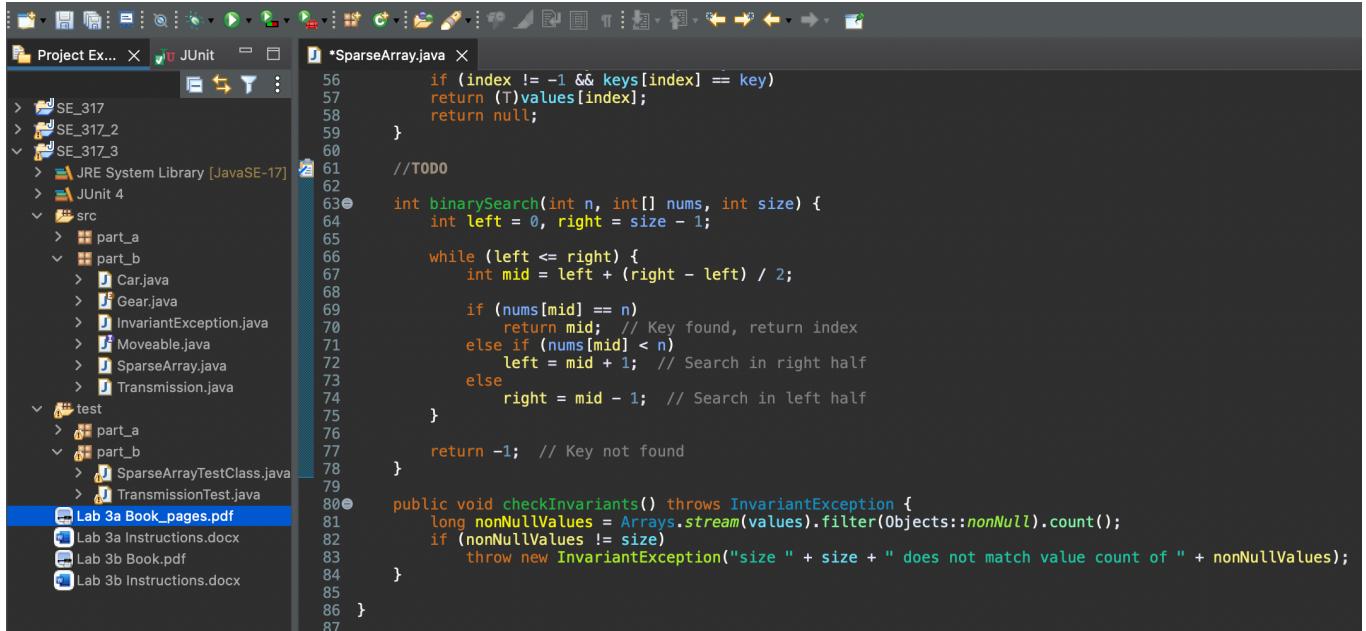
Yes, there is a difference:

-throw is used to generate an exception when something goes wrong.
-try-catch is used to handle an exception and prevent a program crash.
-throw is used inside a method to reject invalid input, while try-catch is used outside the method to handle errors and continue execution.

SE 3170: Lab 3b

Testing Ranges by Embedding Invariant Methods

2- TODO 1: Implement the iterative Binary search function in SparseArray.java and take screenshot of the code



```
56         if (index != -1 && keys[index] == key)
57             return (T)values[index];
58         return null;
59     }
60
61     //TODO
62
63     int binarySearch(int n, int[] nums, int size) {
64         int left = 0, right = size - 1;
65
66         while (left <= right) {
67             int mid = left + (right - left) / 2;
68
69             if (nums[mid] == n)
70                 return mid; // Key found, return index
71             else if (nums[mid] < n)
72                 left = mid + 1; // Search in right half
73             else
74                 right = mid - 1; // Search in left half
75
76         }
77         return -1; // Key not found
78     }
79
80     public void checkInvariants() throws InvariantException {
81         long nonNullValues = Arrays.stream(values).filter(Objects::nonNull).count();
82         if (nonNullValues != size)
83             throw new InvariantException("size " + size + " does not match value count of " + nonNullValues);
84     }
85
86 }
87
```

4- TODO 2: Write the following new test class in SparseArrayTestClass.java. Run the code, and take screenshots of the code and test case output

```
1 package part_b;
2
3 import org.junit.*;
4 import static org.hamcrest.CoreMatchers.*;
5 import static org.junit.Assert.*;
6
7 public class SparseArrayTestClass {
8
9     private SparseArray<Object> array;
10
11    @Before
12    public void create() {
13        array = new SparseArray<>();
14    }
15
16    @Test
17    public void handlesInsertionInDescendingOrder() {
18        array.put(7, "seven");
19        array.checkInvariants();
20
21        array.put(6, "six");
22        array.checkInvariants();
23
24        assertThat(array.get(6), equalTo("six"));
25        assertThat(array.get(7), equalTo("seven"));
26    }
27
28 }
```

What is the problem? Answer this in your lab report and fix the code in order to make the test pass.

The image shows two side-by-side Java IDE windows. The top window displays the JUnit test results for 'SparseArrayTestClass'. It shows 'Runs: 1/1' and 'Errors: 0' with a green bar indicating success. The bottom window shows the source code for 'SparseArray.java' and its test class 'SparseArrayTestClass'. The test class contains a test method 'handlesInsertionInDescendingOrder' that inserts 'seven' at index 7 and 'six' at index 6, then checks the invariants. The code for 'SparseArray.java' includes a 'put' method that updates the size only if the value is not null, and a 'checkInvariants' method that counts non-null values.

```
1 package part_b;
2
3 import org.junit.*;
4 import static org.hamcrest.CoreMatchers.*;
5 import static org.junit.Assert.*;
6
7 public class SparseArray<Object> array;
8
9     @Before
10    public void create() {
11        array = new SparseArray<Object>();
12    }
13
14    @Test
15    public void handlesInsertionInDescendingOrder() {
16        array.put(7, "seven");
17        array.checkInvariants();
18
19        array.put(6, "six");
20        array.checkInvariants();
21
22        assertThat(array.get(6), equalTo("six"));
23        assertThat(array.get(7), equalTo("seven"));
24    }
25
26}
27
28}
29
```



```
1 package part_b;
2
3 import java.util.*;
4
5 public class SparseArray<T> {
6
7     public static final int INITIAL_SIZE = 1000;
8     private int[] keys = new int[INITIAL_SIZE];
9     private Object[] values = new Object[INITIAL_SIZE];
10    private int size = 0;
11    public void put(int key, T value) {
12        if (value == null) return;
13
14        int index = binarySearch(key, keys, size);
15        if (index != -1 && keys[index] == key)
16            values[index] = value;
17        else {
18            insertAfter(key, value, index);
19            size++;
20        }
21    }
22}
```

What Was the Issue?

- The size was not updating correctly when inserting new elements.
- `checkInvariants()` counted more non-null values than size, causing `InvariantException`.

How Was It Fixed?

- We increment `size++` only when a new key-value pair is inserted.

5- TODO 3: Write two test cases for the following conditions. Run the 2 test codes, and take screenshots of the code and test cases outputs

The screenshot shows an IDE interface with two tabs open: `SparseArray.java` and `*SparseArrayTestClass.java`. The `*SparseArrayTestClass.java` tab contains the following Java code:

```
22     array.checkInvariants();
23
24     assertThat(array.get(6), equalTo("six"));
25     assertThat(array.get(7), equalTo("seven"));
26 }
27 */
28 @Test
29 public void insertNullValue() {
30     array.put(0, null); // Insert null
31     array.checkInvariants(); // Check consistency
32     assertThat(array.size(), equalTo(0)); // Expected size should be 0
33 }
34
35 @Test
36 public void insertReplaceValue() {
37     array.put(6, "seis"); // Insert "seis"
38     array.put(6, "six"); // Replace with "six"
39     assertThat(array.get(6), equalTo("six")); // Should return "six"
40 }
```

The left panel shows the project structure and a JUnit runner window indicating the test was finished after 0.036 seconds with 2 runs, 0 errors, and 0 failures.

Part 2

Correct Reference (Correct Initial State)

6- TODO 4: Inspect and run the following TransmissionTest code. Take a screenshot of the code and output

The screenshot shows an IDE interface with two tabs: 'Transmission.java' and 'TransmissionTest.java'. The 'TransmissionTest.java' tab is active, displaying the following Java code:

```
package part_b;

import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;

public class TransmissionTest {
    private Transmission transmission;
    private Car car;

    @Test
    public void remainsInDriveAfterAcceleration() {
        transmission.shift(Gear.DRIVE);
        car.accelerateTo(35);
        assertThat(transmission.getGear(), equalTo(Gear.DRIVE));
    }

    @Test
    public void ignoresShiftToParkWhileInDrive() {
        transmission.shift(Gear.DRIVE);
        car.accelerateTo(30);
        transmission.shift(Gear.PARK);
        assertThat(transmission.getGear(), equalTo(Gear.DRIVE));
    }

    @Test
    public void allowsShiftToParkWhenNotMoving() {
        transmission.shift(Gear.DRIVE);
        car.accelerateTo(30);
        car.brakeToStop();
        transmission.shift(Gear.PARK);
        assertThat(transmission.getGear(), equalTo(Gear.PARK));
    }
}
```

The code editor has syntax highlighting and line numbers. A tooltip 'assertThat(transmission.getGear(), equalTo(Gear.DRIVE));' is visible over the assertion in the first test method. The bottom left corner shows a 'Failure Trace' panel with a single entry: 'java.lang.NullPointerException: Cannot ... at part_b.TransmissionTest.ignoresSh...'. The top status bar indicates 'Finished after 0.044 seconds'.

7- - TODO 5: The code above will not run. You will need to fix the test code as follows and take a screenshot:

The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows "TransmissionTest [Runner: JUnit]".
- JUnit View:** Shows "Finished after 0.021 seconds" with "Runs: 3/3" and "Errors: 0" and "Failures: 0".
- Code Editor:** Displays the `TransmissionTest.java` file with the following content:

```
1 package part_b;
2
3 import org.junit.*;
4 import static org.junit.Assert.*;
5 import static org.hamcrest.CoreMatchers.*;
6
7 public class TransmissionTest {
8     private Transmission transmission;
9     private Car car;
10    @Before
11    public void create() {
12        car = new Car();
13        transmission = new Transmission(car);
14    }
15
16    @Test
17    public void remainsInDriveAfterAcceleration() {
18        transmission.shift(Gear.DRIVE);
19        car.accelerateTo(35);
20        assertThat(transmission.getGear(), equalTo(Gear.DRIVE));
21    }
22
23    @Test
24    public void ignoresShiftToParkWhileInDrive() {
25        transmission.shift(Gear.DRIVE);
26        car.accelerateTo(30);
27        transmission.shift(Gear.PARK);
28        assertThat(transmission.getGear(), equalTo(Gear.DRIVE));
29    }
30
31    @Test
32    public void allowsShiftToParkWhenNotMoving() {
33        transmission.shift(Gear.DRIVE);
34        car.accelerateTo(30);
35        car.brakeToStop();
36        transmission.shift(Gear.PARK);
37        assertThat(transmission.getGear(), equalTo(Gear.PARK));
38    }
}
```

The code editor has syntax highlighting and several assertions are underlined with dotted lines, indicating they are failing or have errors.

Part 3 **Submission**

1. Answer to “what does checkInvariants () method do?”

-The checkInvariants() method ensures data consistency in the SparseArray class.
-It counts the non-null values stored in the array and compares them to the size variable.
-If there is a mismatch, it throws an InvariantException to prevent incorrect data tracking.
-This helps detect errors where the internal size does not correctly reflect the actual number of stored elements.

2. Answer to “what Transmission.java class does?”

-The Transmission class controls a car's gear shifting behavior.
-It ensures that dangerous or invalid gear shifts are prevented:

- If the car is moving, it ignores a shift to Park.
- If the car is stopped, it allows shifting to Park.

-It maintains the correct state of the transmission based on the car's speed.
-It mimics real-world transmission behavior by enforcing gear shift constraints.