





SIT 333

Software Quality and Testing

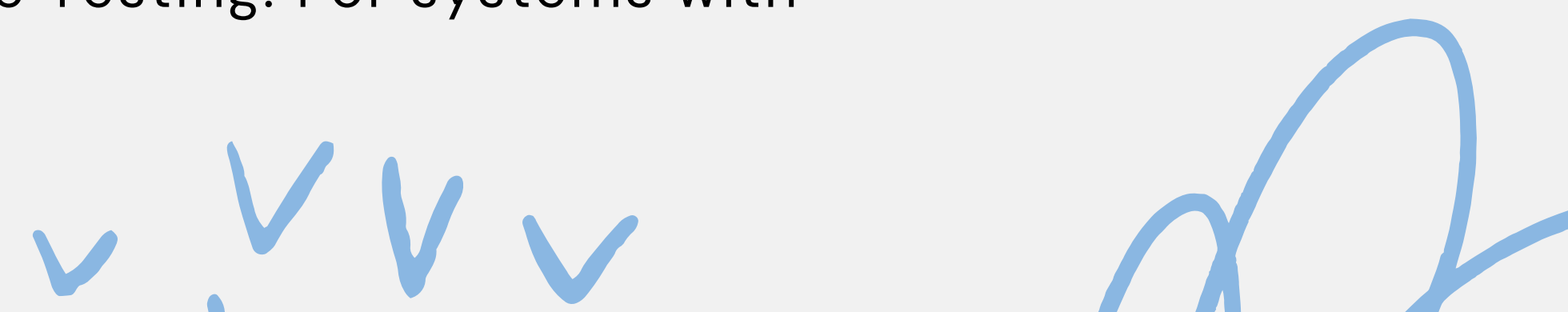


Name: Aditya
Student I'D: 224277942



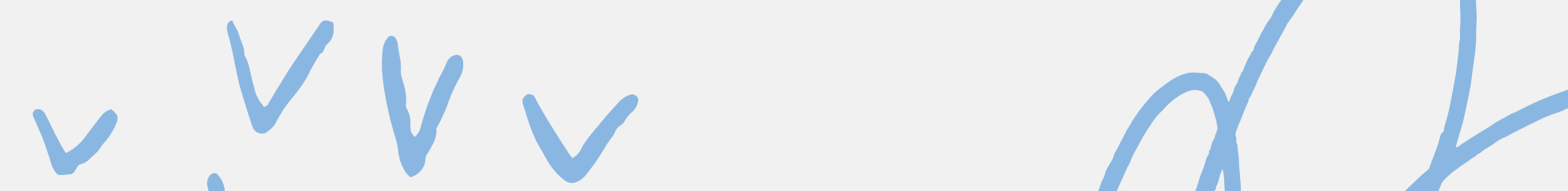


What is Functional (Black-Box) Testing?

- A software testing method that does not require internal knowledge of the system.
 - Focus is on input and output, not how the logic is implemented.
 - Main goal: verify that the system behaves as expected for a range of input scenarios.
 - Common Black Box techniques:
 - Boundary Value Analysis (BVA): Tests values at the edge of input domains.
 - Equivalence Class Testing (ECT): Groups inputs into classes to minimize test cases.
 - Decision Table Testing: Considers rules and conditions.
 - State Transition Testing and Use Case Testing: For systems with different states.
- 



Boundary Value Analysis(BVA)

- BVA tests the edges (boundaries) of input ranges where failures often occur.
 - Errors commonly happen at boundaries due to incorrect conditional logic.
 - Basic technique:
 - Choose input values just below, at, and just above the boundaries.
 - Typical values tested:
 - Minimum (min)
 - Just above minimum (min+1)
 - Nominal (mid-range value)
 - Just below maximum (max-1)
 - Maximum (max)
 - This is especially effective when input constraints are well defined.
- 

BVA Example – Triangle Classification

Function: `classifyTriangle(a, b, c)`

- Input: Three integers representing triangle side lengths.
- Output: One of four outcomes: Equilateral, Isosceles, Scalene, or Not a triangle

Boundary Value Test Cases:

- Case 1: $a = 100, b = 100, c = 100 \rightarrow$ Equilateral
- Case 2: $a = 100, b = 100, c = 200 \rightarrow$ Not a triangle (Fails triangle inequality)
- Case 3: $a = 1, b = 100, c = 100 \rightarrow$ Isosceles (a is at minimum)
- Case 4: $a = 200, b = 100, c = 100 \rightarrow$ Not a triangle (a is at max)

These test values target the boundaries of acceptable triangle side values.

BVA – Considerations and Variants

- Single Fault Assumption: Only one variable is at its boundary while others are nominal.
- Robustness Testing:
 - Includes invalid inputs slightly outside the boundaries (e.g., min - 1, max + 1).
 - Helps check exception handling.
- Worst-Case BVA:
 - All possible combinations of boundary values.
 - For n inputs $\rightarrow 5^n$ test cases.
- Limitations:
 - May miss bugs related to interactions between input variables.
 - Assumes variable independence which may not always hold.
 - Can create redundancy or miss corner cases if not used carefully.



Equivalence Class Testing (ECT) – Introduction

- Reduces the number of test cases by grouping inputs into equivalence classes.
- Each class represents a set of inputs expected to behave similarly.
- Types:
 - Valid Classes: Inputs that should be accepted by the system.
 - Invalid Classes: Inputs that should be rejected.

Example – Age Input:

- Valid class: $18 \leq \text{age} \leq 65$
- Invalid classes: $\text{age} < 18$, $\text{age} > 65$
- Instead of testing every age, test representative values like 18, 30, 65, 17, 66

Types of ECT and Example

- Weak Normal: One valid value from each class
- Strong Normal: All combinations of valid values from different classes
- Weak Robust: One invalid value from a class per test case
- Strong Robust: Combinations of invalid values across classes

Triangle Example:

- Valid class: $a = b = c$ (Equilateral)
- Invalid class: $a + b \leq c$ (Not a triangle)

Test	a	b	c	Expected Output
TN1	5	5	5	Equilateral
TN2	5	5	10	Not a Triangle
WR2	-1	5	5	invalid input ($a < 0$)
SR1	-1	-1	5	invalid input $a, b < 0$



ECT in Practice – Next Date Function

Function: getNextDate(day, month, year)

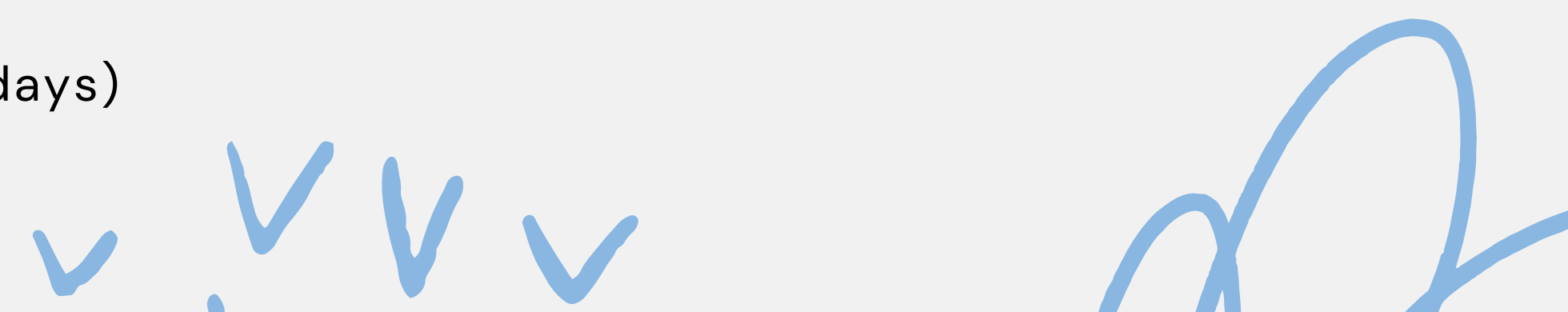
Valid Classes:

- Day: $1 \leq \text{day} \leq 31$
- Month: $1 \leq \text{month} \leq 12$
- Year: $1812 \leq \text{year} \leq 2012$

Invalid Classes:

- Day < 1 or > 31
- Month < 1 or > 12
- Year < 1812 or > 2012

Sample Test Cases:

- (15, 6, 1912) \rightarrow 16/6/1912 (valid)
 - (-1, 6, 1912) \rightarrow invalid day
 - (31, 6, 1912) \rightarrow invalid date (June has only 30 days)
 - (30, 6, 2000) \rightarrow 1/7/2000
- 

Comparing BVA and ECT + Real-World Use



Boundary Value Analysis (BVA) focuses on testing at the edges of input ranges to catch errors that often occur at the minimum and maximum thresholds. In contrast, Equivalence Class Testing (ECT) divides the input domain into partitions or logical classes to reduce the number of test cases while still maintaining effective coverage. BVA is simple and easy to automate but can grow quickly in test count when multiple variables are involved. ECT requires thoughtful classification but significantly reduces redundancy by selecting representative inputs

For example, BVA is ideal for form fields and numeric ranges like age or date inputs, while ECT works well for validating formats such as phone numbers or email addresses. Automating BVA is straightforward, whereas ECT may require additional logic to define valid and invalid classes. Despite their differences, using both BVA and ECT together improves testing efficiency and ensures thorough validation of both edge cases and broader input behavior.

Reflection and Conclusion

- BVA is a quick and effective technique for input edge validation.
- ECT reduces redundancy while ensuring good test coverage across input spaces.
- Both techniques are applicable in various real-world scenarios:
 - Web forms (date pickers, phone fields)
 - Business logic (age restrictions, item quantity checks)
- In unit testing (JUnit), both strategies can be implemented programmatically.
- A good tester often combines BVA and ECT for maximum fault detection.



ACTIVITY



Source Code:


```
1 package sit707_tasks;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 /**
7  * @author Ahsan Habib
8  */
9 public class DateUtilTest {
10
11     @Test
12     public void testStudentIdentity() {
13         String studentId = "224240108";
14         Assert.assertNotNull("Student ID is null", studentId);
15     }
16
17     @Test
18     public void testStudentName() {
19         String studentName = "jiya thakur";
20         Assert.assertNotNull("Student name is null", studentName);
21     }
22
23     @Test
24     public void testLeapYear() {
25         Assert.assertTrue(DateUtil.isLeapYear1(2020));
26         Assert.assertTrue(DateUtil.isLeapYear1(2024));
27         Assert.assertTrue(DateUtil.isLeapYear1(2000));
28         System.out.println("Test Passed: Valid leap years");
29     }
30
31     @Test
32     public void testNonLeapYear() {
33         Assert.assertFalse(DateUtil.isLeapYear1(2023));
34         Assert.assertFalse(DateUtil.isLeapYear1(1900));
35         Assert.assertFalse(DateUtil.isLeapYear1(2021));
36         System.out.println("Test Passed: Valid non-leap years");
37     }
38
39     @Test
40     public void testValidDay29InLeapYear() {
41         DateUtil date = new DateUtil(29, 2, 2024);
42         Assert.assertEquals(29, date.getDay());
43         System.out.println("Test Passed: Valid Feb 29 on leap year");
44     }
45
46     @Test(expected = RuntimeException.class)
47     public void testInvalidDay29InNonLeapYear() {
48         System.out.println("Expecting RuntimeException for Feb 29 in non-leap year");
49         new DateUtil(29, 2, 2023);
50     }
51
52     @Test(expected = RuntimeException.class)
53     public void testInvalidDay31InApril() {
54         System.out.println("Expecting RuntimeException for April 31");
55         new DateUtil(31, 4, 2024);
56     }
57 }
```


```
58     @Test
59     public void testValidDay30InApril() {
60         DateUtil date = new DateUtil(30, 4, 2024);
61         Assert.assertEquals(30, date.getDay());
62     }
63
64     @Test(expected = RuntimeException.class)
65     public void testInvalidDay32() {
66         System.out.println("Expecting RuntimeException for day 32");
67         new DateUtil(32, 1, 2024);
68     }
69
70     @Test(expected = RuntimeException.class)
71     public void testInvalidDayZero() {
72         System.out.println("Expecting RuntimeException for day 0");
73         new DateUtil(0, 1, 2024);
74     }
75
76     @Test(expected = RuntimeException.class)
77     public void testInvalidMonthZero() {
78         System.out.println("Expecting RuntimeException for month 0");
79         new DateUtil(1, 0, 2024);
80     }
81
82     @Test(expected = RuntimeException.class)
83     public void testInvalidMonthGreaterThanTwelve() {
84         new DateUtil(1, 13, 2024);
85     }
86
87     @Test(expected = RuntimeException.class)
88     public void testInvalidYearBelowRange() {
89         System.out.println("Expecting RuntimeException for year below 1700");
90         new DateUtil(1, 1, 1699);
91     }
92
93     @Test
94     public void testEndOfYearToNewYear() {
95         DateUtil date = new DateUtil(31, 12, 2024);
96         date.increment();
97         Assert.assertEquals(1, date.getDay());
98         Assert.assertEquals(1, date.getMonth());
99         Assert.assertEquals(2025, date.getYear());
100     }
101
102     @Test
103     public void testDecrementIntoLeapYearFebruary() {
104         DateUtil date = new DateUtil(1, 3, 2024); // Leap year
105         date.decrement();
106         Assert.assertEquals(29, date.getDay());
107         Assert.assertEquals(2, date.getMonth());
108     }
109 }
```

















Test case output:

Finished after 0.04 seconds

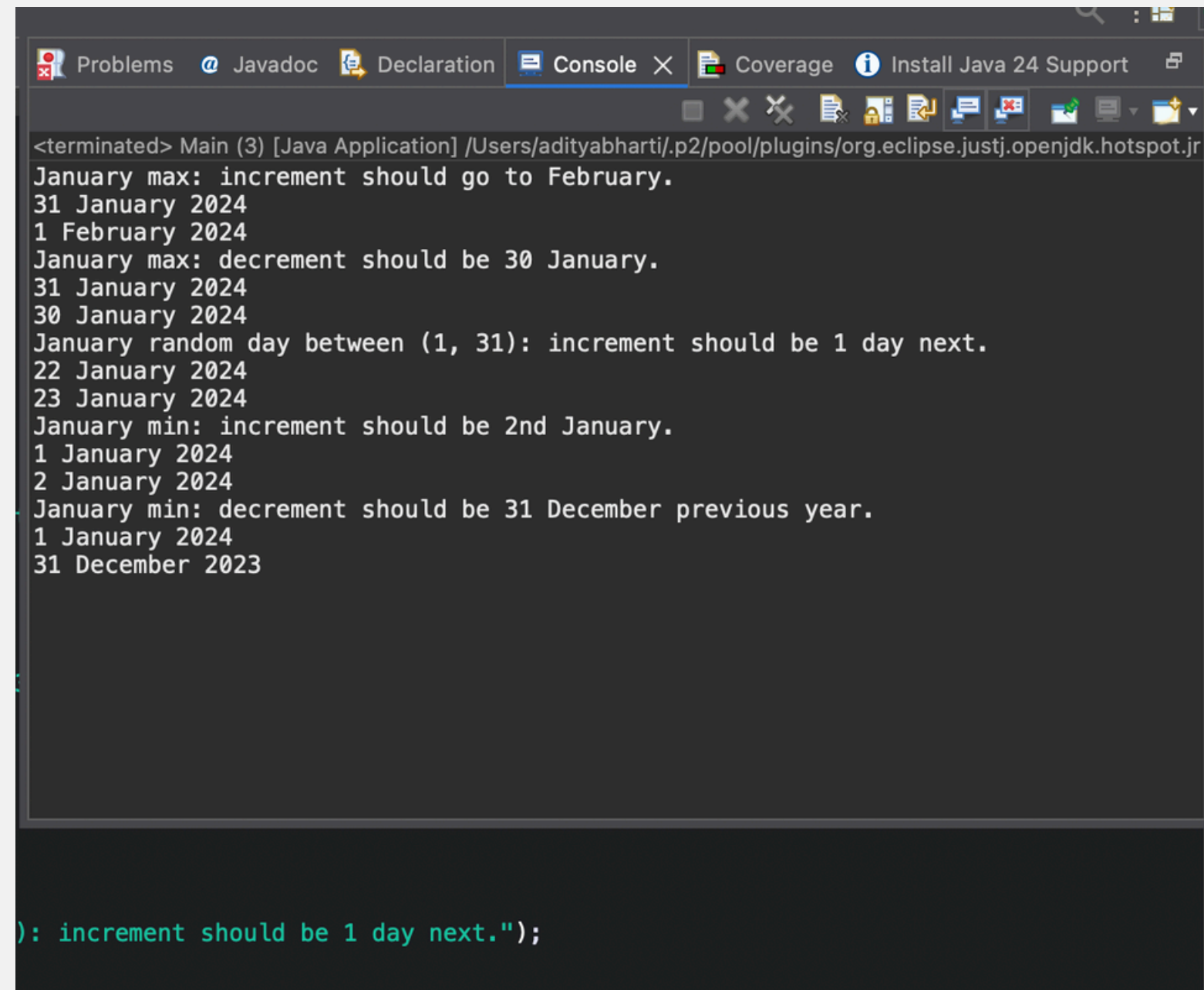
Runs: 15/15  Errors: 0  Failures: 0



✓  sit707_tasks.DateUtilTest [Runner: JUnit 4] (0.015 s)

- ✓  testLeapYear (0.007 s)
- ✓  testInvalidYearBelowRange (0.001 s)
- ✓  testStudentIdentity (0.000 s)
- ✓  testValidDay30InApril (0.000 s)
- ✓  testInvalidMonthGreaterThanTwelve (0.000 s)
- ✓  testNonLeapYear (0.001 s)
- ✓  testInvalidDay31InApril (0.000 s)
- ✓  testInvalidDayZero (0.001 s)
- ✓  testInvalidDay29InNonLeapYear (0.000 s)
- ✓  testValidDay29InLeapYear (0.001 s)
- ✓  testInvalidMonthZero (0.000 s)
- ✓  testEndOfYearToNewYear (0.001 s)
- ✓  testInvalidDay32 (0.000 s)
- ✓  testStudentName (0.000 s)
- ✓  testDecrementIntoLeapYearFebruary (0.001 s)

Console Output



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console (active), Coverage, and Install Java 24 Support. The console output is as follows:

```
<terminated> Main (3) [Java Application] /Users/adityabharti/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jr
January max: increment should go to February.
31 January 2024
1 February 2024
January max: decrement should be 30 January.
31 January 2024
30 January 2024
January random day between (1, 31): increment should be 1 day next.
22 January 2024
23 January 2024
January min: increment should be 2nd January.
1 January 2024
2 January 2024
January min: decrement should be 31 December previous year.
1 January 2024
31 December 2023
```

Below the console window, a snippet of Java code is visible:

```
): increment should be 1 day next.");
```

Github Link:

<https://github.com/Aditya2Verma/SIT333>

