Lab 08 - Software Testing

Name : Tushar S. Mori

SID : 202201502

Lab Session - Functional Testing(black-box)

Q1. Equivalence Class Test Cases for Previous Data Program :

1. Equivalence partitioning and boundary value analysis:

Equivalence Partitioning (EP)

Valid Inputs:

- Normal date: `15, 5, 2020` → Previous date: `14-05-2020`
- Last day of a month: `1, 3, 2021` → Previous date: `28-02-2021` (or `29-02-2020` for leap year)
- Leap year: `1, 1, 2020` → Previous date: `31-12-2019`

Invalid Inputs:

- Invalid month: `15, 5, 2020` → Error message
- Invalid day: `30, 2, 2020` → Error message (February only has 29 in leap years)
- Invalid year: `1, 1, 1800` → Error message

Boundary Value Analysis (BVA)

- **Boundary Cases:**
  - Day: `1, 1, 2021` → Previous date: `31-12-2020`
  - Last day of month: `1, 3, 2021` → Previous date: `28-02-2021`
  - First valid year: `1, 1, 1900` → Previous date: `31-12-1899` (invalid)
  - Last valid year: `1, 1, 2015` → Previous date: `31-12-2014`

Test Suite

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
(15, 6, 2010)	Previous valid date (14, 6, 2010)
(1, 1, 1900)	Previous valid date (31, 12, 1899)
(1, 5, 2010)	Previous valid date (30, 4, 2010)
(29, 2, 2012)	Previous valid date (28, 2, 2012)
(1, 3, 2000)	Previous valid date (29, 2, 2000)
(15, 6, 1899)	Error message : Invalid Year
(15, 6, 2016)	Error message : Invalid Year
(30, 2, 2010)	Error message: Invalid Date
(32,12, 2010)	Error message : Invalid Date
(0, 5, 2010)	Error message : Invalid Month
(15, 13, 2010)	Error message : Invalid Month
Boundary Value Analysis	
(1, 1, 1900)	(31, 12, 1899)
(31, 12, 2015)	(30, 12, 2015)
(1, 3, 2000)	(29, 2, 2000)
(28, 2, 2011)	(27, 2, 2011)

(29, 2, 2012)	(28, 2, 2012)
(30, 6, 2010)	(29, 6, 2010)
(31, 7, 2010)	(30, 7, 2010)
(1, 7, 2010)	(30, 6, 2010)
(1, 12, 2010)	(30, 11, 2010)
(30, 4, 2010)	(29, 4, 2010)
(31, 10, 2010)	(30, 10, 2010)
(1, 1, 2010)	(31, 12, 2009)
(31, 1, 2010)	(30, 1, 2010)

Q2. Programs :

Problem P1: Linear Search

Equivalence Partitioning (EP) for `linearSearch`:

1. **Valid Input Partition:**
    - **Case 1: The value `v` exists in the array (e.g., `v` is present in the array).**
    - **Case 2: The value `v` does not exist in the array (e.g., `v` is absent).**
    - **Case 3: The array is empty (edge case).**
2. **Invalid Input Partition:**
    - **Case 4: The array has invalid data types or is `null`.**

Boundary Value Analysis (BVA) for `linearSearch`

1. **Array Size Boundaries:**

- **Array with a single element.**
- **Small array (size 2-3).**
- **Large array (upper boundary size).**

2. **Search Value Boundaries:**
   - **Value is at the first position.**
   - **Value is at the last position.**
   - **Value is in the middle.**

Test Cases :

Tester Action and Input Data	Expected Outcome	Partition/Boundary
linearSearch(3, [1, 2, 3, 4, 5])	2	EP - Value exists
linearSearch(6, [1, 2, 3, 4, 5])	-1	EP - Value absent
linearSearch(3, [])	-1	EP - Empty array
linearSearch(3, null)	Error or Exception	EP - Null array
linearSearch(1, [1])	0	BVA - Single element array
linearSearch(1, [1, 2, 3])	0	BVA - Value at first index
linearSearch(3, [1, 2, 3])	2	BVA - Value at last index
linearSearch(2, [1, 2, 3])	1	BVA - Value in the middle
linearSearch(5, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])	4	BVA - Large array

Corrected Code :

```
int linearSearch(int v, int a[]) {
   for (int i = 0; i < a.length; i++) {
      if (a[i] == v) {
         return i; // Return the first index where value v is found
      }
```

```
    }
    return -1; // Return -1 if value v is not found
}
```

Problem P2:Count Item

Equivalence Partitioning (EP) for `countItem`

1. **Valid Input Partition:**
    - **Case 1: The value `v` appears multiple times in the array.**
    - **Case 2: The value `v` does not appear in the array.**
    - **Case 3: The array is empty (edge case).**
2. **Invalid Input Partition:**
    - **Case 4: The array is `null`.**

Boundary Value Analysis (BVA) for `countItem`

1. **Array Size Boundaries:**
    - **Array with a single element.**
    - **Array with multiple elements (small size).**
    - **Large array (upper boundary).**
2. **Search Value Boundaries:**
    - **Value appears exactly once in the array.**
    - **Value appears multiple times.**
    - **Value does not appear at all.**

Test Cases :

Tester Action and Input Data	Expected Outcome	Partition/Boundary
countItem(3, [1, 2, 3, 4, 5])	1	EP - Value appears once
countItem(3, [1, 3, 3, 3, 5])	3	EP - Value appears multiple times
countItem(6, [1, 2, 3, 4, 5])	0	EP - Value absent
countItem(3, [])	0	EP - Empty array

countItem(3, null)	Error or Exception	EP - Null array
countItem(1, [1])	1	BVA - Single element array
countItem(1, [1, 2, 3])	1	BVA - Value appears once
countItem(2, [1, 2, 2, 3])	2	BVA - Value appears multiple times
countItem(7, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])	1	BVA - Large array
countItem(0, [1, 2, 3])	0	BVA - Value does not appear

Corrected Code :

```
int countItem(int v, int a[]) {
   int count = 0;
   for (int i = 0; i < a.length; i++) {
      if (a[i] == v) {
         count++; // Increment count if value v is found
      }
   }
   return count; // Return the total count of value v
}
```

Problem P3:Binary Search

Equivalence Partitioning (EP) for `countItem`

1. **Valid Input Partition:**
   - **Case 1: The value `v` appears multiple times in the array.**
   - **Case 2: The value `v` does not appear in the array.**
   - **Case 3: The array is empty (edge case).**
2. **Invalid Input Partition:**
   - **Case 4: The array is `null`.**

Boundary Value Analysis (BVA) for `countItem`

1. **Array Size Boundaries:**
    - **Array with a single element.**
    - **Array with multiple elements (small size).**
    - **Large array (upper boundary).**
2. **Search Value Boundaries:**
    - **Value appears exactly once in the array.**
    - **Value appears multiple times.**
    - **Value does not appear at all.**

Test Cases :

Tester Action and Input Data	Expected Outcome	Partition/Boundary
binarySearch(3, [1, 2, 3, 4, 5])	2	EP - Value exists
binarySearch(6, [1, 2, 3, 4, 5])	-1	EP - Value does not exist
binarySearch(3, [])	-1	EP - Empty array
binarySearch(3, null)	Error or Exception	EP - Null array
binarySearch(1, [1])	0	BVA - Single-element array
binarySearch(1, [1, 2, 3])	0	BVA - Value at first index
binarySearch(3, [1, 2, 3])	2	BVA - Value at last index
binarySearch(2, [1, 2, 3])	1	BVA - Value in the middle
binarySearch(5, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])	4	BVA - Value in large array

Corrected Code :

```
int binarySearch(int v, int a[]) {
   int lo = 0;
   int hi = a.length - 1;

   while (lo <= hi) {
      int mid = (lo + hi) / 2;
```

```
        if (a[mid] == v) {
            return mid; // Return index where value v is found
        } else if (a[mid] < v) {
            lo = mid + 1; // Search right half
        } else {
            hi = mid - 1; // Search left half
        }
    }
    return -1; // Return -1 if value v is not found
}
```

Problem P4 : Triangle Classification

Equivalence Partitioning EP) for `triangle`

1. **Valid Input Partition:**
    - **Case 1: The sides form an equilateral triangle (all sides equal).**
    - **Case 2: The sides form an isosceles triangle (two sides equal).**
    - **Case 3: The sides form a scalene triangle (no sides equal).**
2. **Invalid Input Partition:**
    - **Case 4: The sides do not form a triangle (violating the triangle inequality or non-positive sides).**

Boundary Value Analysis (BVA) for `triangle`

1. **Side Length Boundaries:**
    - **Small values (e.g., side lengths of 1).**
    - **Large values (upper boundary).**
    - **Triangle inequality boundaries.**
2. **Triangle Type Boundaries:**
    - **Equilateral: All sides equal.**
    - **Isosceles: Two sides equal.**
    - **Scalene: No sides equal.**
    - **Invalid: Non-triangle or zero/negative sides.**

Test Cases :

Tester Action and Input Data	Expected Outcome	Partition/Boundary
triangle(3, 3, 3)	EQUILATERAL	EP - Equilateral triangle
triangle(3, 3, 2)	ISOSCELES	EP - Isosceles triangle
triangle(3, 4, 5)	SCALENE	EP - Scalene triangle
triangle(1, 1, 2)	INVALID	EP - Invalid triangle (violates inequality)
triangle(0, 4, 5)	INVALID	EP - Invalid triangle (zero side)
triangle(3, -1, 4)	INVALID	EP - Invalid triangle (negative side)
triangle(1, 1, 1)	EQUILATERAL	BVA - Minimum side lengths (valid equilateral)
triangle(1000, 1000, 1000)	EQUILATERAL	BVA - Large side lengths (equilateral)
triangle(5, 5, 10)	INVALID	BVA - Boundary condition for invalid triangle
triangle(2, 3, 4)	SCALENE	BVA - Valid scalene triangle
triangle(1, 1, 2)	INVALID	BVA - Triangle inequality boundary
triangle(10, 10, 5)	ISOSCELES	BVA - Valid isosceles triangle

Corrected Code :

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;

int triangle(int a, int b, int c) {
    if (a <= 0 || b <= 0 || c <= 0 || a >= b + c || b >= a + c || c >= a + b) {
        return INVALID; // Triangle inequality or non-positive lengths
    }
    if (a == b && b == c) {
        return EQUILATERAL; // All sides equal
    }
    if (a == b || a == c || b == c) {
        return ISOSCELES; // Two sides equal
    }
    return SCALENE; // No sides equal
}
```

Problem P5: Prefix Checking

Equivalence Partitioning (EP) for `prefix`

1. **Valid Input Partition:**
    - **Case 1: `s1` is a prefix of `s2`.**
    - **Case 2: `s1` is equal to `s2`.**
2. **Invalid Input Partition:**
    - **Case 3: `s1` is longer than `s2` (cannot be a prefix).**
    - **Case 4: `s1` is empty.**
    - **Case 5: `s2` is empty (only valid if `s1` is also empty).**

Boundary Value Analysis (BVA) for `prefix`

1. **String Length Boundaries:**
    - **One character in `s1`.**
    - **`s1` is empty.**
    - **`s1` and `s2` are of the same length.**
    - **`s1` is longer than `s2`.**
2. **Prefix Matching Boundaries:**
    - **`s1` is an exact match for `s2`.**
    - **`s1` is a partial match for `s2`.**

Test Cases :

Tester Action and Input Data	Expected Outcome	Partition/Boundary
prefix("pre", "prefix")	TRUE	EP - s1 is a prefix
prefix("test", "test")	TRUE	EP - s1 is equal to s2

prefix("test", "testing")	TRUE	EP - s1 is a prefix
prefix("longer", "short")	FALSE	EP - s1 is longer than s2
prefix("test", "")	FALSE	EP - s2 is empty
prefix("", "nonempty")	TRUE	EP - s1 is empty
prefix("", "")	TRUE	EP - both s1 and s2 are empty
prefix("a", "abc")	TRUE	BVA - s1 is one character prefix
prefix("abc", "abc")	TRUE	BVA - s1 is the same as s2
prefix("ab", "abc")	TRUE	BVA - s1 is a partial match
prefix("abc", "ab")	FALSE	BVA - s1 is longer than s2
prefix("abc", "a")	FALSE	BVA - s1 is longer than s2
prefix("abc", "abcd")	TRUE	BVA - prefix at upper boundary

Corrected Code :

```
public static boolean prefix(String s1, String s2) {
    if (s1.length() > s2.length()) {
        return false; // s1 cannot be a prefix of s2 if it's longer
    }
    for (int i = 0; i < s1.length(); i++) {
        if (s1.charAt(i) != s2.charAt(i)) {
            return false; // Mismatch found
        }
    }
    return true; // All characters matched
}
```

Problem P6: Triangle Classification with Floating Point Values

Equivalence Partitioning (EP) for `TriangleClassification`

1. **Valid Input Partition:**
    - **Case 1: The sides form an equilateral triangle (all sides equal).**
    - **Case 2: The sides form an isosceles triangle (two sides equal).**
    - **Case 3: The sides form a scalene triangle (no sides equal).**
    - **Case 4: The sides form a right-angled triangle (satisfies Pythagorean theorem).**
2. **Invalid Input Partition:**
    - **Case 5: The sides do not form a triangle (violating the triangle inequality or non-positive sides).**
    - **Case 6: Non-positive values for one or more sides.**

Boundary Value Analysis (BVA) for `TriangleClassification`

1. **Side Length Boundaries:**
    - **Very small positive values (e.g., 0.1).**
    - **Exact boundary values (e.g., values that satisfy equality conditions).**
    - **Very large positive values (upper boundary).**
2. **Triangle Type Boundaries:**
    - **Equilateral: All sides equal.**
    - **Isosceles: Two sides equal.**
    - **Scalene: No sides equal.**
    - **Right-angled: One side squared equals the sum of the squares of the other two sides.**
    - **Invalid: Non-triangle or zero/negative sides.**

Test Cases:

Tester Action and Input Data	Expected Outcome	Partition/Boundary
TriangleClassification(3.0, 3.0, 3.0)	Equilateral	EP - Equilateral triangle
TriangleClassification(3.0, 3.0, 2.0)	Isosceles	EP - Isosceles triangle
TriangleClassification(3.0, 4.0, 5.0)	Scalene	EP - Scalene triangle
TriangleClassification(1.0, 1.0, 2.0)	Invalid	EP - Invalid triangle
TriangleClassification(0.0, 4.0, 5.0)	Invalid	EP - Invalid triangle (zero side)
TriangleClassification(3.0, -1.0, 4.0)	Invalid	EP - Invalid triangle (negative side)
TriangleClassification(1.0, 1.0, 1.0)	Equilateral	BVA - Minimum side lengths
TriangleClassification(1000.0, 1000.0, 1000.0)	Equilateral	BVA - Large side lengths
TriangleClassification(5.0, 5.0, 10.0)	Invalid	BVA - Boundary condition for invalid triangle
TriangleClassification(2.0, 3.0, 4.0)	Scalene	BVA - Valid scalene triangle
TriangleClassification(3.0, 4.0, 5.0)	Right-angled	BVA - Valid right-angled triangle
TriangleClassification(3.0, 3.0, 3.0)	Equilateral	BVA - Exact match for equilateral
TriangleClassification(3.0, 4.0, 3.0)	Isosceles	BVA - Exact match for isosceles
TriangleClassification(6.0, 8.0, 10.0)	Right-angled	BVA - Right-angled triangle
TriangleClassification(1.0, 1.0, 2.0)	Invalid	BVA - Triangle inequality boundary

Corrected Code :

```
import java.util.Scanner;


public class TriangleClassification {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter length A: ");
        double A = scanner.nextDouble();
        System.out.print("Enter length B: ");
        double B = scanner.nextDouble();
        System.out.print("Enter length C: ");
        double C = scanner.nextDouble();

        if (A <= 0 || B <= 0 || C <= 0 || A + B <= C || A + C <= B || B + C <= A) {
            System.out.println("Invalid: Not a triangle");
        } else if (A == B && B == C) {
            System.out.println("Equilateral");
        } else if (A == B || A == C || B == C) {
            System.out.println("Isosceles");
        } else if (A * A + B * B == C * C || A * A + C * C == B * B || B * B + C * C == A * A)
{
            System.out.println("Right-angled");
        } else {
            System.out.println("Scalene");
        }
        scanner.close();
    }
}
```