IT314: Software Engineering

LAB 08: Software Testing - Functional Testing (Black-Box)

Student Name: Gireesh Reddy Nimmala

Student ID: 202201122

Q1-

- **Equivalence Partitioning Test Cases**

Valid Date Equivalence Classes:

Test Case	Scenario Description	Input	Expected Outcome
1	Valid Date (Regular Day)	(19, 10, 2008)	(18, 10, 2008)
2	Valid Date (End of February, Leap Year)	(29, 2, 2012)	(28, 2, 2012)
3	Valid Date (End of February, Non Leap Year)	(28, 2, 2011)	(27, 2, 2011)
4	Valid Date (End of Month)	(31, 5, 2014)	(30, 5, 2014)
5	Valid Date (Month with 30 Days)	(30, 4, 2015)	(29, 4, 2015)

Invalid Date Equivalence Classes:

Test Case	Scenario Description	Input	Expected Outcome
1	Invalid Day (Zero Day)	(0, 11, 2010)	An Error message
2	Invalid Day (Negative Day)	(-1, 11, 2010)	An Error message
3	Invalid Month (Zero Month)	(1, 0, 2010)	An Error message
4	Invalid Month (Negative Month)	(1, -1, 2007)	An Error message
5	Invalid Year (Future Year)	(1, 11, 2035)	An Error message

Boundary Value Analysis Test Cases:

Test Case	Scenario Description	Input	Expected Outcome
1	Day Before First of the Month	(1, 1, 2015)	(31, 12, 2014)
2	Last Day of February, Non Leap Year	(28, 2, 2013)	(27, 2, 2013)
3	Last Day of February, Leap Year	(29, 2, 2016)	(28, 2, 2016)

4	**Day Boundary (31st Day)**	**(31, 12, 2015)**	**(30, 12, 2015)**
5	**Year Lower Boundary**	**(1, 1, 1900)**	**(31, 12, 1899)**
6	**Year Upper Boundary**	**(1, 1, 2015)**	**(31, 12, 2014)**
7	**Day Maximum for Months with 30 Days**	**(30, 4, 2015)**	**(29, 4, 2015)**
8	**Last Valid Input for Valid Year**	**(31, 12, 2015)**	**(30, 12, 2015)**

Equivalence Partitioning Test Cases:

Tester Action and Input Data	
(19, 10, 2008)	(18, 10, 2008)
(29, 2, 2012)	(28, 2, 2012)
(28, 2, 2011)	(27, 2, 2011)

(31, 5, 2014)	(30, 5, 2014)
(30, 4, 2015)	(29, 4, 2015)
(0, 11, 2010)	An Error message
(-1, 11, 2010)	An Error message
(1, 0, 2010)	An Error message

(1, -1, 2007)	An Error message
(1, 11, 2035)	An Error message

Boundary Value Analysis Test Cases:

Tester Action and Input Data	Expected Outcome
(1, 1, 2015)	(31, 12, 2014)

(28, 2, 2013)	(27, 2, 2013)
(29, 2, 2016)	(28, 2, 2016)
(31, 12, 2015)	(30, 12, 2015)
(1, 1, 1900)	(31, 12, 1899)
(1, 1, 2015)	(31, 12, 2014)
(30, 4, 2015)	(29, 4, 2015)
(31, 12, 2015)	(30, 12, 2015)

• Modify your programs such that it runs, and then execute your test suites on the program.
While executing your input data in a program, check whether the identified expected outcome (mentioned by you) is correct or not.
Java Program for Determining the Previous Date:

```
import java.util.Scanner;

public class PreviousDate {
 public static String previousDate(int day, int month, int year) {
// Check if the input year is valid
 if (year < 1900 || year > 2015) {
 return "Invalid Date";
 }

 // Check if the input month is valid
 if (month < 1 || month > 12) {
 return "Invalid Date";
 }

 // Check if the input day is valid
 if (day < 1 || day > daysInMonth(month, year)) {
 return "Invalid Date";
 }

 // Decrease the day to get the previous date
 day--;

 // If day becomes 0, go to the last day of the previous month  if
(day == 0) {
 month--; // Move to the previous month
 if (month == 0) { // If month becomes 0, go to December of the previous  year
 month = 12;
 year--;
 }
 day = daysInMonth(month, year); // Get the last day of the new month  }

 // Return the new date
 return day + "/" + month + "/" + year;
 }

 // Helper function to check the number of days in a given month and year
public static int daysInMonth(int month, int year) {
 switch (month) {
 case 4: case 6: case 9: case 11:
 return 30; // April, June, September, November have 30 days  case 2:
```

```
// Check for leap year
if (isLeapYear(year)) {
return 29; // February has 29 days in a leap year  } else {
return 28; // February has 28 days otherwise  }
default:
return 31; // All other months have 31 days
}
}

// Helper function to check if a year is a leap year
public static boolean isLeapYear(int year) {
if (year % 4 == 0) {
if (year % 100 == 0) {
return year % 400 == 0;
} else {
return true;
}
}
return false;
}

public static void main(String[] args) {
// Create a scanner object to get input from the user
Scanner sc = new Scanner(System.in);

// Prompt the user for input
System.out.print("Enter day: ");
int day = sc.nextInt();

System.out.print("Enter month: ");
int month = sc.nextInt();

System.out.print("Enter year: ");
int year = sc.nextInt();

// Call the function and display the previous date
String result = previousDate(day, month, year);
System.out.println("Previous date: " + result);

// Close the scanner
sc.close();
```

```
}
}
```

Explanation of Code:

1. Function Overview

The function previousDate(day, month, year) calculates the previous date given the input of a day, month, and year. It checks for the validity of the input and then returns either a valid previous date or "Invalid Date" if the input is not valid.

2. Input Validation

The code first ensures that the year, month, and day are valid:

- Year check: The year must be between 1900 and 2015 (inclusive). If it's outside this range, the function returns "Invalid Date".
- Month check: The month must be between 1 and 12. If the month is outside this range, the function returns "Invalid Date".
- Day check: The day is checked to ensure it is within the valid number of days for the given month and year (e.g., February can have 28 or 29 days depending on leap years).

3. Previous Day Calculation

Once the input is validated, the function proceeds to calculate the previous day:

- Decrement Day: It decreases the day by 1.
- If the day becomes 0, the function needs to move to the previous month:
    - The month is decremented by 1.
    - If the month becomes 0, it means that the date is now in the previous year, so:
        - The month is set to 12 (December).
        - The year is decreased by 1.
    - The last day of the previous month is calculated using the

daysInMonth() function.

4. Helper Functions

- daysInMonth(month, year): This function determines how many days are in a given month, considering whether the year is a leap year for February (using the isLeapYear() function).
- isLeapYear(year): This function checks if the year is a leap year. A year is a leap year if:
    - It is divisible by 4, and either:
        - It is not divisible by 100, or
        - It is divisible by 400.

5. Output

After adjusting the day, month, and year values, the function returns the previous date in the format day/month/year.

Testing the Program

Using the previously defined test cases, you can input the values manually. Here are some test cases you can use:

Test Case Input	Expected Output
(1, 1, 2013)	"31/12/2012"
(1, 3, 2011)	"28/2/2011"
(29, 2, 2012)	"28/2/2012"
(1, 5, 2009)	"30/4/2009"
(31, 1, 2007)	"30/1/2007"

(1, 13, 2009)	"Invalid date"
(32, 1, 1995)	"Invalid date"
(1, 1, 1809)	"Invalid date"

Checking Outcomes:

For each input, check if the output matches the expected outcome:

→ Run the program.

→ Input the day, month, and year as specified in the test cases.

→ Compare the output to the expected result. If they match, the test case passes; if not, it fails.

Q2-

a) Identify the equivalence classes for the system b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class. (Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence classes)

P1)

Equivalence Classes:

Test Case	Scenario Description	Input	Expected Outcome
Class 1	Empty array	[]	-1
Class 2	Value exists (first occurrence at index 0)	Array with value at index 0	0
Class 3	Value exists (first occurrence at index n, n > 0)	Array with value at index n	n
Class 4	Value does not exist in the array	Array without the value	-1
Class 5	Value exists with duplicates (return index of first value)	Array with duplicates of value	Index of the first occurrence

Test Cases:-

Here's a table summarizing the test cases for the linearSearch function, including the input, expected outcome, and the equivalence class each case covers:

Input (v, a)	Expected Output	Covers Equivalence Class
(5, [])	-1	1
(3, [3, 1, 2])	0	2
(4, [1, 4, 2])	1	3
(7, [1, 2, 3, 7])	3	4
(10, [1, 2, 3, 4])	-1	5
(2, [2, 3, 2, 1])	0	6
(1, [1, 1, 1, 1])	0	6
(5, [1, 2, 3, 5, 5])	3	6
(9, [1, 2, 9, 5, 9])	2	6
(0, [0, 1, 2, 3])	0	2

This table provides a clear overview of the test cases, the inputs provided, the expected outputs, and the corresponding equivalence classes each test case covers.

P2)

Equivalence Classes:

Test Case	Scenario Description	Input	Expected Outcome
Class 1	Empty array	[]	0
Class 2	Value exists once	Array with value appearing once	1
Class 3	Value exists multiple times	Array with value appearing n times	Count of occurrences (n)
Class 4	Value does not exist	Array without the value	0
Class 5	All elements are equal to value v	Array where all elements are v	Length of the array

Test Cases:

Input (v, a)	Expected Output	Covers Equivalence Class
(5, [])	0	1
(3, [1, 2, 3])	1	2
(4, [1, 2, 3])	0	4
(2, [2, 2, 2, 2])	4	5

(1, [1, 2, 1, 1])	3	3
(9, [1, 2, 3, 4])	0	4

(5, [5, 5, 5, 5, 5])	5	5
(0, [0, 0, 1])	2	3
(8, [2, 3, 5, 7])	0	4
(6, [1, 2, 3, 6, 6, 6])	3	3

P3)

Equivalence Classes:

Test Case	Scenario Description	Input	Expected Outcome
Class 1	Empty array	[]	-1
Class 2	Value exists at the first index	Array with value at index 0	0
Class 3	Value exists at a middle index	Array with value at a middle index	Index of v
Class 4	Value exists at the last index	Array with value at the last index	Index of last occurrence
Class 5	Value does not exist (less than smallest element)	Array where value < smallest element	-1

Class 6	Value does not exist (greater than largest element)	Array where value > largest element	-1
Class 7	Value does not exist (between two elements)	Array where value lies between two elements	-1
Class 8	Value exists with duplicates	Array with multiple occurrences of value v	Index of any occurrence

Test Cases:

Input (v, a)	Expected Output	Covers Equivalence Class
(5, [])	-1	1
(3, [1, 2, 3, 4])	2	2
(1, [1, 2, 3, 4])	0	2
(4, [1, 2, 3, 4])	3	4
(0, [1, 2, 3, 4])	-1	5
(5, [1, 2, 3, 4])	-1	6
(2, [1, 2, 2, 3, 4])	1	8
(6, [1, 2, 3, 4, 5])	-1	6
(3, [1, 2, 3, 3, 4])	2	8
(2, [1, 1, 1, 1])	1	2

P4)

Equivalence Classes:

Test Case Scenario Description Input Expected Outcome

Class 1 Invalid triangle (non-positive sides) Triangle with non-positive sides INVALID

Class 2 Invalid triangle (triangle inequality not satisfied) Triangle where triangle inequality fails INVALID

Class 3 Equilateral triangle (all sides equal) Triangle with all sides equal EQUILATERAL

Class 4 Isosceles triangle (two sides equal) Triangle with two sides equal ISOSCELES

Class 5 Scalene triangle (all sides different) Triangle with all sides different SCALENE

Test Cases:

Input (a, b, c)	Expected Outcome	Covers Equivalence Class
(0, 0, 0)	INVALID	1
(-1, 2, 3)	INVALID	1

(1, 1, 1)	EQUILATERAL	3
(2, 2, 3)	ISOSCELES	4
(2, 3, 4)	SCALENE	5
(5, 2, 2)	ISOSCELES	4
(1, 2, 3)	INVALID	2
(3, 3, 6)	INVALID	2
(2, 5, 3)	SCALENE	5
(7, 3, 10)	INVALID	2

P5)

Equivalence Classes:

Testcase	**Condition**	**Output**
Class 1	s1 is longer than s2	false
Class 2	s1 is an exact prefix of s2	true
Class 3	s1 is a partial prefix of s2	false
Class 4	s1 is empty	true
Class 5	s2 is empty and s1 is not	false
Class 6	s1 is equal to s2	true

Test Cases:

Input (s1, s2)	Expected Outcome	Covers Equivalence Class
("abc", "abcdef")	true	2
("abc", "ab")	false	3

Input (s1, s2)	Expected Outcome	Covers Equivalence Class
("abc", "xyzabc")	false	3
("", "abcdef")	true	4
("a", "")	false	5
("abc", "abc")	true	6
("longerPrefix", "short")	false	1
("abc", "abcde")	true	2
("prefix", "pre")	false	3
("xyz", "xyzxyz")	true	2

P6)

• Identifying the Equivalence Classes:

Valid Triangle Types:

- **Equilateral Triangle**: Side A = Side B = Side C
- **Isosceles Triangle**: Side A = Side B, or Side A = Side C, or Side B =

Side C

- **Scalene Triangle**: All sides unequal ($A \neq B \neq C$)
- **Right-Angled Triangle**: $A^2 + B^2 = C^2$ (Pythagorean theorem) or its permutations

Invalid Triangle Cases:

- **Not a Triangle**: $A + B \leq C$, $A + C \leq B$, or $B + C \leq A$
- **Non-positive Input**: Any side A, B, or C is less than or equal to zero

• Test Cases Covering the Identified Equivalence Classes:

Input (A, B, C)	Expected Output	Equivalence Classes Covered
(7 , 7, 7)	Equilateral Triangle	Equilateral Triangle

Input (A, B, C)	Expected Output	Equivalence Classes Covered
(8, 8, 9)	Isosceles Triangle	Isosceles Triangle
(3, 6, 8)	Scalene Triangle	Scalene Triangle
(3, 4, 5)	Right-Angled Triangle	Right-Angled Triangle
(5, 7, 14)	Not a Triangle	Not a Triangle
(0, 2, 4)	Invalid	Non-positive Input

- **Boundary Condition A + B > C (Scalene Triangle):**

Input (A, B, C)	Expected Output
(4, 5, 6)	Scalene Triangle
(6, 7, 12)	Scalene Triangle
(6, 7, 13)	Not a Triangle
(5, 7, 11)	Scalene Triangle

- **Boundary Condition A = C (Isosceles Triangle):**

Input (A, B, C)	Expected Output
(6, 7, 6)	Isosceles Triangle
(7, 10, 10)	Isosceles Triangle
(5, 9, 14)	Not a Triangle
(9, 9, 9)	Equilateral Triangle

- **Boundary Condition A = B = C (Equilateral Triangle):**

Input (A, B, C)	Expected Output
(6, 6, 6)	Equilateral Triangle
(8, 8, 8)	Equilateral Triangle
(7, 8, 14)	Not a Triangle
(7, 8, 13)	Scalene Triangle

- **Boundary Condition $A^2 + B^2 = C^2$ (Right-Angled Triangle):**

Input (A, B, C)	Expected Output
(6, 8, 10)	Right-Angled Triangle
(9, 12, 15)	Right-Angled Triangle
(6, 9, 14)	Not a Triangle
(7, 10, 12)	Scalene Triangle

• Non-Triangle Case:

Input (A, B, C)	Expected Output
(5, 6, 7)	Scalene Triangle
(7, 12, 20)	Not a Triangle
(5, 9, 14)	Not a Triangle
(6, 8, 14)	Scalene Triangle

• Non-Positive Input Case:

Input (A, B, C)	Expected Output
(4, 6, 0)	Invalid
(5, 7, -3)	Invalid
(0, 8, 10)	Invalid
(-4, 6, 9)	Invalid