LAB - 9: Black Box Testing

Group 5

Group Members

No.	Name	ID
1	Harshal Markana	201801143
2	Ishang Kumar	201801071
3	Raj Mahla	201801243
4	Darshan Prajapati	201801146
5	Jenil Khandhara	201801217
6	Sambhav Agrawal	201801063
7	Nipun Patel	201801234
8	Parthiv Patel	201801463
9	Mahi Patel	201801039
10	Sudiksha Thusu	201801469

Group Members Working on Question 1:
Mahi, Sudiksha, Jenil, Nipun, Darshan.

Test Cases: Jenil, Nipun, Sudiksha
Boundary Analysis: Jenil
Equivalence Classes: Darshan, Mahi
Code: Sudiksha, Mahi

1) **Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges 1 <= month <= 12, 1 <= day <= 31, 1900 <= year <= 2015.**
The possible output dates would be the previous date or invalid date. Design the equivalence class test cases? Write a set of test cases (i.e., test suite) – specific set of data – to properly test the programs. Your test suite should include both correct and incorrect inputs.

1. **Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.**
2. **Modify your programs such that it runs on eclipse IDE, 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.**

Answer:1

y: year
m: month
d: day

Ranges: 1900<y<2015 , 1<=m<=12, 1<=d<=31

Equivalence Classes:

1. Days
    a. d1: 1<=d<=28
    b. d2: d=29
    c. d3: d=30
    d. d4: d=31
    e. d5: d<1
    f. d6: d>31
2. Months
    a. m1: m<1
    b. m2: m>12
    c. m3: months with 30 days (m belongs to {4,6,9,11})
    d. m4: months with 31 day (m belongs to {1,3,5,7,8,10,12})
    e. m5: months with 28/29 (m=2)

3. Year
    a. y1: Leap years from 1900-2015
    b. y2: non-leap-years from 1900-2015
    c. y3: y>=2016
    d. y4: y<=1899

Test Cases:

Sr no.	**DAYS**	**MONTH**	**YEAR**	**OUTPUT**
1	**d1**	**m3,m4,m5**	**y1,y2**	**Prev Date**
2	**d2**	**m3,m4**	**y1,y2**	**Prev date**
3	**d2**	**m5**	**y1**	**Prev Date**
4	**d3**	**m3,m4**	**y1,y2**	**Prev Date**
5	**d4**	**m4**	**y1,y2**	**Prev Date**
6	**d5,d6**	**any**	**any**	**Invalid**
7	**any**	**m1,m2**	**any**	**Invalid**
8	**any**	**any**	**y3,y4**	**Invalid**

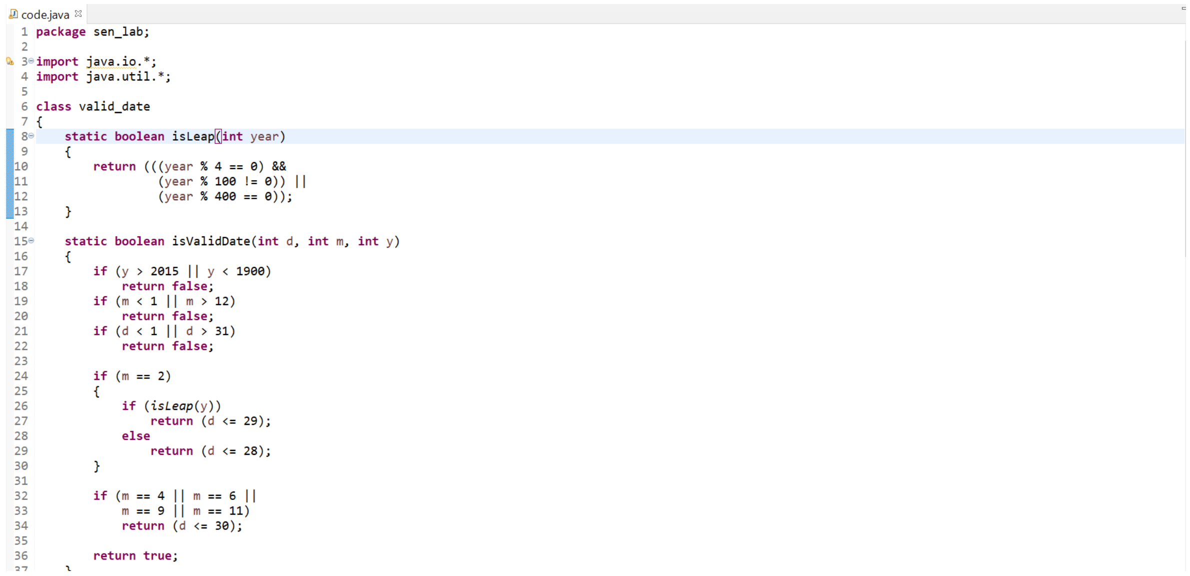
Boundary analysis

Sr no.	**DAYS**	**MONTH**	**YEAR**	**OUTPUT**
1	**any**	**any**	**1899**	**Invalid**
2	**any**	**any**	**2016**	**Invalid**
3	**any**	**0**	**any**	**Invalid**
4	**any**	**13**	**any**	**Invalid**
5	**0**	**any**	**any**	**Invalid**
6	**32**	**any**	**any**	**Invalid**
7	**29**	**2**	**Non leap**	**Invalid**
8	**30,31**	**2**	**any**	**Invalid**
9	**d1,d2,d3,d4**	**1**	**y1,y2**	**Prev Date**
10	**d1,d2,d3,d4**	**12**	**y1,y2**	**Prev Date**

11	**1**	**any**	**y1,y2**	**Prev Date**
12	**31**	**m1,m2,m5,m3**	**y1,y2**	**Invalid**
13	**31**	**m4**	**y1,y2**	**Prev Date**

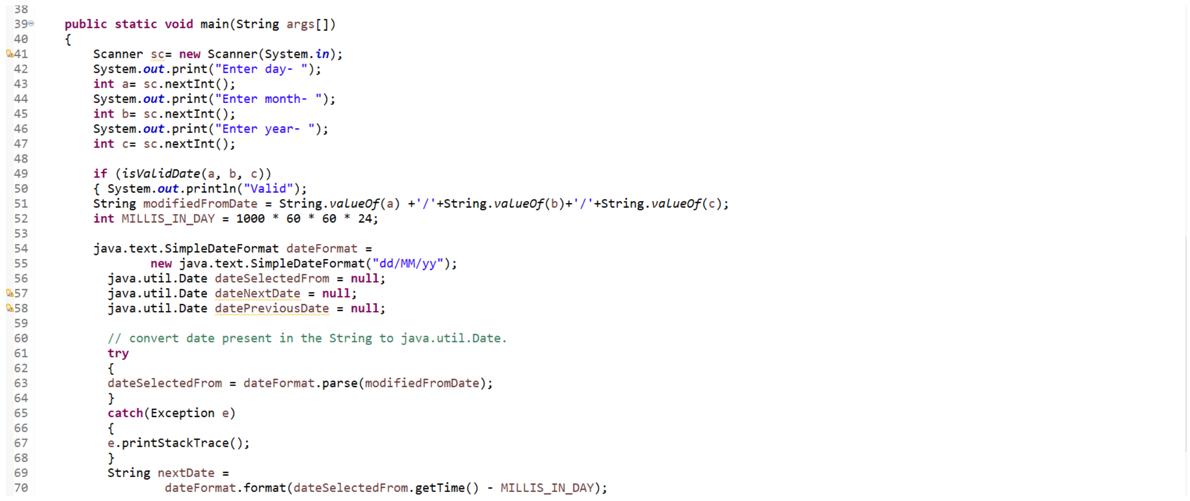
Answer 2:

Code:

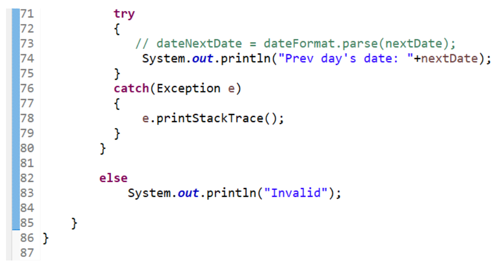


code.java

```
package sen_lab;

import java.io.*;
import java.util.*;

class valid_date
{
    static boolean isLeap(int year)
    {
        return (((year % 4 == 0) &&
                 (year % 100 != 0)) ||
                 (year % 400 == 0));
    }

    static boolean isValidDate(int d, int m, int y)
    {
        if (y > 2015 || y < 1900)
            return false;
        if (m < 1 || m > 12)
            return false;
        if (d < 1 || d > 31)
            return false;

        if (m == 2)
        {
            if (isLeap(y))
                return (d <= 29);
            else
                return (d <= 28);
        }

        if (m == 4 || m == 6 ||
            m == 9 || m == 11)
            return (d <= 30);

        return true;
    }
```



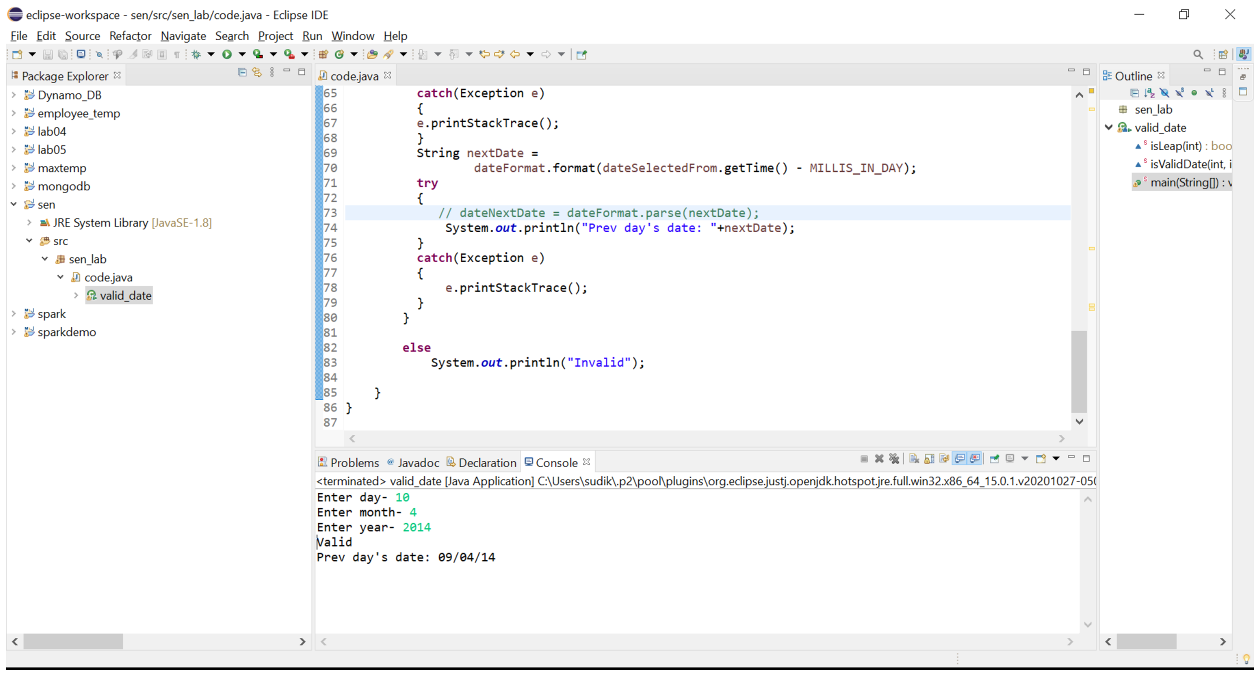
```

    public static void main(String args[])
    {
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter day- ");
        int a= sc.nextInt();
        System.out.print("Enter month- ");
        int b= sc.nextInt();
        System.out.print("Enter year- ");
        int c= sc.nextInt();

        if (isValidDate(a, b, c))
        { System.out.println("Valid");
        String modifiedFromDate = String.valueOf(a) +'/'+String.valueOf(b)+'/'+String.valueOf(c);
        int MILLIS_IN_DAY = 1000 * 60 * 60 * 24;

        java.text.SimpleDateFormat dateFormat =
                new java.text.SimpleDateFormat("dd/MM/yy");
          java.util.Date dateSelectedFrom = null;
          java.util.Date dateNextDate = null;
          java.util.Date datePreviousDate = null;

          // convert date present in the String to java.util.Date.
          try
          {
          dateSelectedFrom = dateFormat.parse(modifiedFromDate);
          }
          catch(Exception e)
          {
          e.printStackTrace();
          }
          String nextDate =
                  dateFormat.format(dateSelectedFrom.getTime() - MILLIS_IN_DAY);
```



```java
        try
        {
           // dateNextDate = dateFormat.parse(nextDate);
            System.out.println("Prev day's date: "+nextDate);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
      }

      else
          System.out.println("Invalid");

    }
}
```

Output:

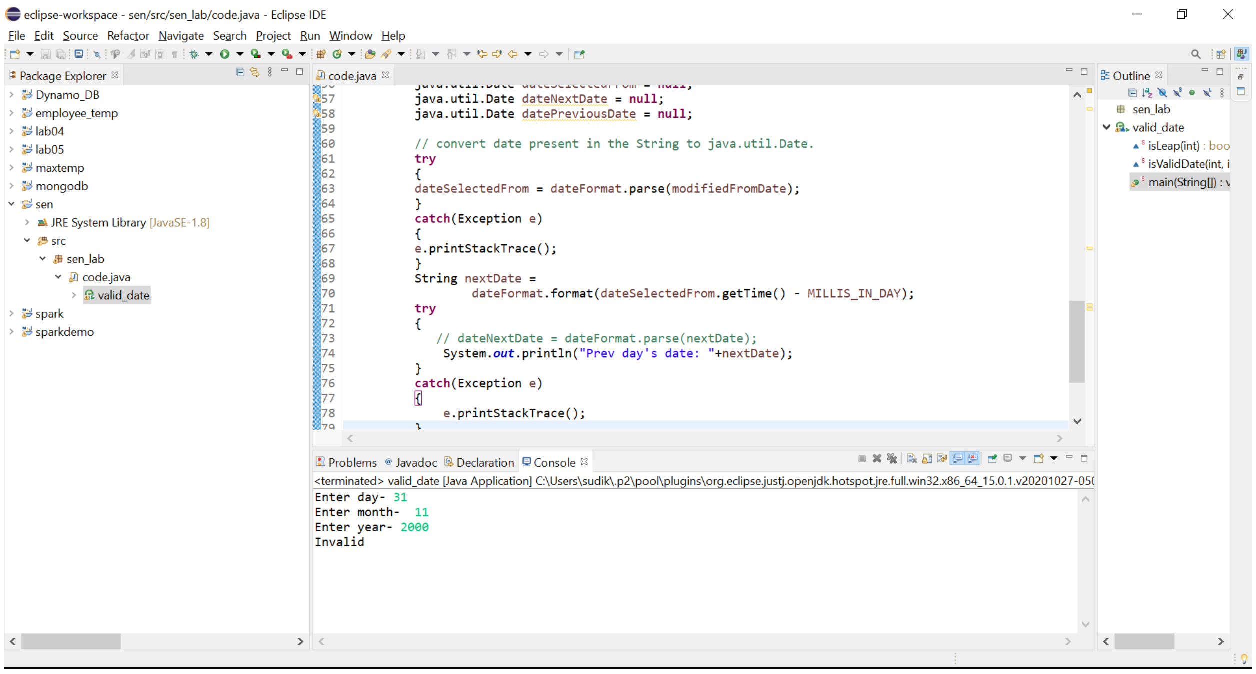
1. Valid Date



```java
        catch(Exception e)
        {
        e.printStackTrace();
        }
        String nextDate =
                dateFormat.format(dateSelectedFrom.getTime() - MILLIS_IN_DAY);
        try
        {
           // dateNextDate = dateFormat.parse(nextDate);
            System.out.println("Prev day's date: "+nextDate);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
      }

      else
          System.out.println("Invalid");

    }
}
```

```
<terminated> valid_date [Java Application] C:\Users\sudik\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-05
Enter day- 10
Enter month- 4
Enter year- 2014
Valid
Prev day's date: 09/04/14
```

2. Invalid Date



```
        java.util.Date dateNextDate = null;
        java.util.Date datePreviousDate = null;

        // convert date present in the String to java.util.Date.
        try
        {
        dateSelectedFrom = dateFormat.parse(modifiedFromDate);
        }
        catch(Exception e)
        {
        e.printStackTrace();
        }
        String nextDate =
                dateFormat.format(dateSelectedFrom.getTime() - MILLIS_IN_DAY);
        try
        {
           // dateNextDate = dateFormat.parse(nextDate);
            System.out.println("Prev day's date: "+nextDate);
        }
        catch(Exception e)
        {
            e.printStackTrace();
```

```
<terminated> valid_date [Java Application] C:\Users\sudik\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-05
Enter day- 31
Enter month-  11
Enter year- 2000
Invalid
```

Group Members Working on Question 2:
Ishang, Raj, Sambhav, Harshal, Parthiv

2. You are testing an e-commerce system that sells products like caps and jackets. The problem is to create functional tests using boundary-value analysis and equivalence class partitioning techniques for the webpage that accepts the orders. A screen prototype for the order-entry web page is shown below.

The system accepts a five-digit numeric item ID number from 00000 to 99999. The system accepts a quantity to be ordered, from 1 to 99. If the user enters a previously ordered item ID and a 0 quantity to be ordered, that item is removed from the shopping cart. Based on these inputs, the system retrieves the item price, calculates the item total (quantity times item price), and adds the item total to the cart total. Due to limits on credit card orders that can be processed, the maximum cart total is $999.99

Given constraints:

ID: 00000-99999

Quantity: 1-99

Max cart total: less than or equal to $999.99 i.e ≤ $999.99

- ## Equivalence classes:

1. **ID**
    - Between 00000-99999 (included) i.e. 00000 ≤ *ID* ≤ 99999
    - Less than 00000 (00000 excluded) i.e. ID < 00000
    - Greater than 99999 (99999 excluded) i.e. *ID* > 99999
2. **Quantity**
    - Between 0-99 (both inclusive) i.e. 0 < *quantity* ≤ 99
    - Less than 0 (0 excluded) i.e. *quantity* < 0
    - Greater than 99 (99 excluded) i.e. *quantity* > 99
3. **Cart total (in dollars)**

- Cart total between 0-999.99 (both inclusive) i.e. 0 ≤ *cart total* ≤ $ 999.99
- Cart total greater than 999.99 (999.99 excluded) i.e. *cart total* > $ 999.99

Assumptions:

1. Price of item ID=10001 is $100.

Test Cases	ItemID	Quantity	Outcome
ID<00000	-11111	-	Error
ID>99999	111111	-	Error
Quantity < 0	-	-1	Error
Quantity >99	-	100	Error
Valid ID	10001	1	$100
Valid Cart total	ID= 10001	1	Cart Total = $100
Invalid cart total	ID= 10001	10	Cart Total=$1000 Error (exceeding cart value)
Quantity=0 and item purchased previously	ID-10001	0	Item was removed
Quantity =0 , Item was not purchased previously	ID- 10001	0	Error (Item removed as quantity=0)