Software Documentation

# Level 3. Creating an object oriented computer program using Java (7540-037/7630-321)

## Author: Bogdan Pasterak

## Tutor: Angela Mercy

## Date: 30/01/2019

## Location: Letterkenny Training Centre

Attachments:

1. Listing newDate class code.
2. Listing testnewDate class code.
3. Test Data and expected results.
4. Test Documentation (Test Plan and Test Log).
5. Chosen Screen Shots from Test.

# Requirements

* Specification of assessment
* PC
* Eclipse ( Java environment )
* Word and Excel

# Design

According to the specification, the project consists of several parts.

* Class “newDate” consisting of five methods with a specific signature.
* Class “testnewDate” used to test the prevision class
* Designing and preforming the test using “testnewDate”.
* Test report.
* Attachments

# Technical

It should be noted that the class names are not compatible with the naming convention in Java. Valid names should be "NewDate" and "TestNewDate". However, the current names are acceptable and this was the customer's wish, I left them original.

In the project I tried not to use external libraries. The exception is "java.time.LocalDate" set to read the current date.

Class "newDate" in addition to the required methods, has one private auxiliary method "daysInMonth”. It serves to collect the number of days in a month, depending on the month and year.

The "testnewDate" class consists of the main method, four auxiliary methods responsible for options in the Menu and an auxiliary method that retrieves data from the user. In it, partial data validation is carried out. Because this is only a test class, in the case of bad data, the method asks you to enter the data again. The target date validation is carried out in the "newDate" class. This division results from the incompatibility of the data. Class A expects three variables int, while the user enters date as String.

In Class A, the "validDate" method checks the correctness of the date, the remaining public methods are covered by it. The exception is "leapYear" which is used by it, however this one works over the whole range of int.

In addition to the project code, a complete set of tests with data and results was attached. The test report is later in this document.

# End user

### Class newDate

It contains several useful methods for converting dates. Most of them use dates in the format (int day, int month, int year).

Methods:

* validDate - checks the correctness of the date, returns the boolean type (true if the date is correct)
* leapYear - calculates if the year is leap, returns the boolean type (true if year is leap). The input parameter (int year)
* julianDate - calculates the number of days from the beginning of the year. The first January is 1. Returns an int type.
* daysElapsed - calculates the number of days between two dates. Returns an int type.
* ageDate - calculates the age based on the date of birth and the current date. Returns an int type.

### Class testnewDate

Class A testing class. Runs an interface between the user and the tested class.

Methods:

* main - The program starts. Displays the Menu, waits for user selection and calls appropriate methods. Works in the loop until the user selects the "Quit" option.
* testLeapYear - the first of the Menu options. It tests the "leapYear" method.
* testJulianDate - Menu option 2. Calls the "julianDate" method by first checking the correctness of the date using the “validDate” method.
* testDaysElapsed - Menu option 3. Calls the "daysElapsed" method by first checking the correctness of the date using the “validDate” method.
* testAge – fourth Menu option. Returns age based on the date entered, calling method “ageDate”.
* getDate - auxiliary method. Takes from the user String, partially validates it to the format "dd-mm-yyyy". Converts date elements to type’s int. If it fails, it asks you to enter the date in a tentative way.

### Test Plan Java newDate (Excel)

Two sheets to test both classes.

* Test Data and Expected results - provides data for the test and expected results (program responses)
* Testing Documentation - includes a complete Test Plan and Test Log.

# Test Report

The completed test showed compliance with the assumptions.

Nevertheless, there are some details that require customer attention.

1. In test 32, the year is 0100. Theoretically, it is the correct value for the year. However, in "validDate" I have defined a range of acceptable years between 1,000 and 3,000. An acceptable range of years should be agreed with the customer. Attachment E. Fig. 1.
2. Test 53 and 54 reveal unforeseen in specification of the case. How to behave when the start date is more than the end date. In my implementation, the program returns a negative value. It seemed to me the most friendly and obvious solution. Attachment E. Fig. 2. However, if the solution does not receive approval, you can suggest one of the following:
   1. More radical, returning a positive value.
   2. Printing of the relevant message.
   3. More restrictive, return 0.
   4. The least less friendly throwing the exception.
3. In test 63 and 64 an additional message appears. The program detects the compatibility of the date of birth and the day of the year and congratulations on the birthday. This cinematography requires acceptance by the client. Attachment E. Fig. 3
4. Test 62, 67, 68 and 69 show similar situations as in case 2. This time the user has entered the date of birth which has not yet taken place. In this case, the program returns the value zero. As in case 2, you can also suggest other solutions like: b) or d) option. Attachment E. Fig. 4

**import** java.time.LocalDate;

/\*\*

\*

\* **@author** Bogdan Pasterak

\* Assessment Object Oriented Programing JAVA

\* Level 3 (7540-037/7630-321)

\* 28 Jan 2019

\*

\*/

// class newDate according to the convention, it should be called NewDate

**public** **class** newDate {

// Validate Date

**public** **boolean** validDate (**int** day, **int** month, **int** year) {

// range of valid year

**if** (year >= 1000 && year <= 3000)

// range of valid month

**if** (month >= 1 && month <= 12)

// range of valid day depending of month and year

**if**(day >= 1 && day <= daysInMonth(month, year))

// success !

**return** **true**;

**return** **false**;

}

// Checking if the year is leap

**public** **boolean** leapYear (**int** year) {

// initial assumption that no

**boolean** leap = **false**;

// Leap year can be evenly divided by 4

**if** (year % 4 == 0)

// and not by 100 , excluding evenly divided by 400

**if** (year % 100 != 0 || year % 400 == 0)

leap = **true**;

**return** leap;

}

**public** **int** julianDate (**int** day, **int** month, **int** year) {

// protect against bad date

**if** ( validDate(day, month, year) ) {

// the number of days is equal to the current day and ...

**int** number = day;

// total days in the preceding months

**for** (**int** m = 1; m < month ; m++) {

// adding days

number += daysInMonth(m, year);

}

**return** number;

// invalid date return 0

} **else** {

**return** 0;

}

}

**public** **int** daysElapsed (**int** day1, **int** month1, **int** year1,

**int** day2, **int** month2, **int** year2) {

// protect against bad date

**if** ( validDate(day1, month1, year1) && validDate(day2, month2, year2) ) {

// if date1 > date 2

**boolean** negative = **false**;

**int** days = 0;

// checking the order of dates

**if** (year1 > year2)

negative = **true**;

**else** **if** (year1 == year2) {

**if** (month1 > month2)

negative = **true**;

**else** **if** (month1 == month2 && day1 > day2)

negative = **true**;

}

// if bad order , swap date

**if** (negative) {

**int** temp = day1;

day1 = day2;

day2 = temp;

temp = month1;

month1 = month2;

month2 = temp;

temp = year1;

year1 = year2;

year2 = temp;

}

// dyfference between Julian date

days = julianDate(day2, month2, year2) - julianDate(day1, month1, year1);

// adding full years

**for** (**int** y = year1; y < year2; y++) {

days += (leapYear(y)) ? 366 : 365;

}

// for bad order negative number of days

**if** (negative)

**return** -days;

**else**

**return** days;

// invalid date return 0

} **else** {

**return** 0;

}

}

**public** **int** ageDate (**int** dayBirth, **int** monthBirth, **int** yearBirth) {

// protect against bad date

**if** ( validDate(dayBirth, monthBirth, yearBirth) ) {

**int** years = 0;

// get current date

LocalDate now = LocalDate.*now*();

**int** day = now.getDayOfMonth();

**int** month = now.getMonthValue();

**int** year = now.getYear();

years = year - yearBirth;

// if Birth Day don't pass

**if** ( monthBirth > month || ( monthBirth == month && dayBirth > day ) )

years--;

// Happy Birthday

**if** ( monthBirth == month && dayBirth == day )

System.***out***.println("Happy Birthday !");

// date later than today

**if** (yearBirth > year ||

(yearBirth == year && monthBirth > month) ||

(yearBirth == year && monthBirth == month && dayBirth > day))

years = 0;

**return** years;

// invalid date return 0

} **else** {

**return** 0;

}

}

// method that returns the number of days in a month

**private** **int** daysInMonth(**int** month, **int** year ) {

**switch** (month) {

// mounths with 31 days

**case** 1: **case** 3: **case** 5:**case** 7: **case** 8: **case** 10: **case** 12:

**return** 31;

// mounths with 30 days

**case** 4: **case** 6: **case** 9: **case** 11:

**return** 30;

// February depending on leap year

**case** 2:

**if** (leapYear(year))

**return** 29;

**else**

**return** 28;

// any wrong month

**default**:

**return** 0;

}

}

}

**import** java.util.Scanner;

/\*\*

\*

\* **@author** Bogdan Pasterak

\* Assessment Object Oriented Programing JAVA

\* Level 3 (7540-037/7630-321)

\* 28 Jan 2019

\*

\*/

// class testnewDate according to the convention, it should be called TestNewDate

**public** **class** testnewDate {

**static** Scanner *sc*;

**static** newDate *nDate*;

**public** **static** **void** main(String[] args) {

// new scanner

*sc* = **new** Scanner(System.***in***);

*nDate* = **new** newDate();

String choise;

// main loop

**do** {

// print menu

System.***out***.println("-----------------------");

System.***out***.println("\tMENU");

System.***out***.println(" 1. Leap year");

System.***out***.println(" 2. Julian date");

System.***out***.println(" 3. Days elapsed");

System.***out***.println(" 4. Age");

System.***out***.println(" 5. Quit");

System.***out***.println("-----------------------");

// user selection

System.***out***.print("Choose options : ");

choise = *sc*.nextLine();

System.***out***.println();

**switch** (choise) {

**case** "1":

*testLeapYear*();

**break**;

**case** "2":

*testJulianDate*();

**break**;

**case** "3":

*testDaysElapsed*();

**break**;

**case** "4":

*testAge*();

**break**;

**case** "5":

**break**;

**default**:

System.***out***.println("Incorrect option, try again");

}

} **while** ( ! choise.equals("5") );

System.***out***.println("Completion of the program !");

}

// Testing leapYear method

**private** **static** **void** testLeapYear() {

**int** year = 0;

// user type year

**do** {

System.***out***.print("Enter a year (yyyy) : ");

// try / catch for invalid Integer

**try** {

**int** number = Integer.*parseInt*(*sc*.nextLine());

// range of valid year

**if** (number < 1000 || number > 3000)

System.***out***.println("A year in the range of 1000 to 3000");

**else**

// success

year = number;

} **catch** (NumberFormatException e) {

System.***out***.println(e.getMessage());

System.***out***.println("Try again. The year must be a number");

}

} **while** (year == 0 );

System.***out***.print("\nYear " + year + " is ");

**if** ( *nDate*.leapYear(year) )

System.***out***.println("LEAP YEAR");

**else**

System.***out***.println("NOT A LEAP YEAR");

}

**private** **static** **void** testJulianDate() {

// user type date

**int**[] date = *getDate*();

// asaign value

**int** day = date[0];

**int** month = date[1];

**int** year = date[2];

// for valid date print Julian date, else message

**if** (*nDate*.validDate(day, month, year))

System.***out***.println("\nJulian date is : " + *nDate*.julianDate(day, month, year));

**else**

System.***out***.println("\nINVALID DATE");

}

**private** **static** **void** testDaysElapsed() {

// get two dates

System.***out***.println("START DATE");

**int** [] start = *getDate*();

System.***out***.println("END DATE");

**int** [] end = *getDate*();

// if dates are valid print number of days between

**if** ( *nDate*.validDate(start[0], start[1], start[2]) &&

*nDate*.validDate(end[0], end[1], end[2]))

System.***out***.println("\nDays elapsed : " +

*nDate*.daysElapsed(start[0], start[1], start[2],

end[0], end[1], end[2]));

**else**

System.***out***.println("\nINVALID DATE");

}

**private** **static** **void** testAge() {

// get birth date

System.***out***.println("Enter the date of birth");

**int** [] birth = *getDate*();

**if** ( *nDate*.validDate(birth[0], birth[1], birth[2]))

System.***out***.println("\nYour age is " +

*nDate*.ageDate(birth[0], birth[1], birth[2]));

**else**

System.***out***.println("\nINVALID DATE");

}

// get date from user helper metchod

**private** **static** **int**[] getDate() {

**boolean** ok = **false**;

**int** year = 0, month = 0, day = 0;

String date;

// user type date

**do** {

System.***out***.print("Enter a date (dd-mm-yyyy) : ");

// get String and remove all spaces

date = *sc*.nextLine().replaceAll(" ", "");

**if** (date.length() == 10 &&date.charAt(2) == '-' && date.charAt(2) == '-' ){

// try / catch for invalid Integer

**try** {

day = Integer.*parseInt*(date.substring(0, 2));

month = Integer.*parseInt*(date.substring(3, 5));

year = Integer.*parseInt*(date.substring(6));

// success

ok = **true**;

} **catch** (NumberFormatException e) {

System.***out***.println(e.getMessage());

System.***out***.println("Try again. Invalid number");

}

} **else** {

System.***out***.println("Invalid format. Try again.");

}

} **while** ( ! ok );

// return array

**return** **new** **int**[] {day, month, year};

}

}

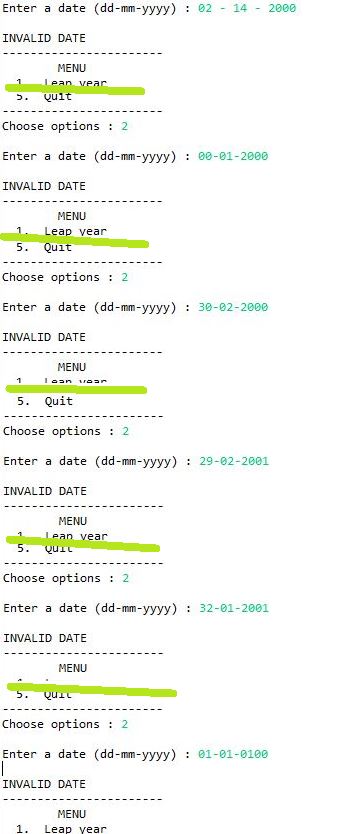
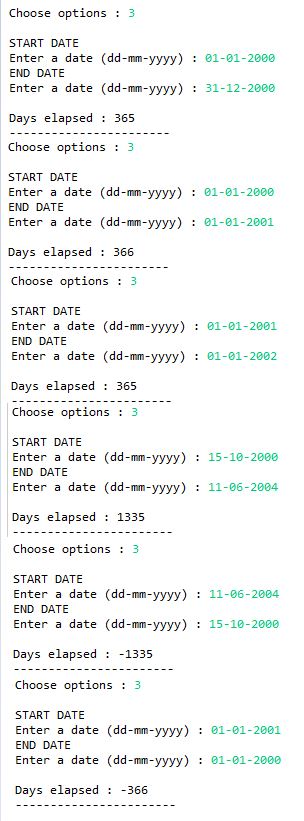


Fig:2 screen\_shot\Capture\_12.JPG

Test 53, 54: Test Report pos. 2.

Fig:1 screen\_shot\Capture\_07.JPG

Test 32: Test Report pos. 1.

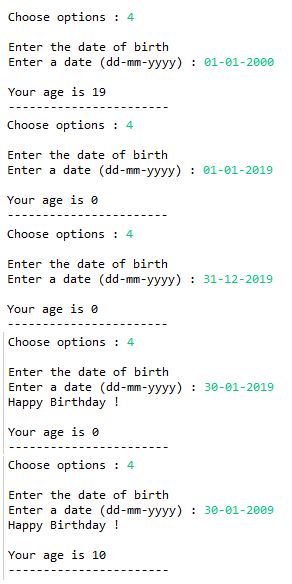
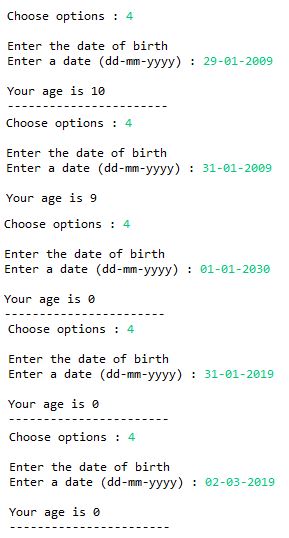


Fig:4 screen\_shot\Capture\_14.JPG

Test 67, 68, 69 : Test Report pos. 4.

Fig:3 screen\_shot\Capture\_14.JPG

Test 63, 64 : Test Report pos. 3.