Software Documentation

# Requirements

* PC
* Eclipse ( Java environment )
* Word and Excel

# Design

The entire software consists of two packages and related documentation and a set of screenshots. Package “differentCoat” contains classes required for the operation of the program.

* Class “PaintShopMenu” with main method.
* Class “CalculateMenu” contains the methods you call in the menu.
* Class “SurfaceCalculator” a set of methods for calculating the surface to be painted.
* Class “PaintRequiredCalculator” – “numberOfCans” method calculates the amount of paint cans needed
* Class “Customer” - class of Customer objects.
* Class “MyScanner” - methods for communication with the user.
* Class “IncorrectObjectTypeExeption” - exceptions when reading a file from a file.

Package “differentCoatTest”. It consists of a JUnit Test Case and a JUnit Test Suite grouping them.

* JUnit Test Suite “AllTests”.
* Junit Test Case “PaintRequiredCalculatorTest” – test “numberOfCans” method.
* Junit Test Case “SurfaceCalculatorTest” – test methods of “SurfaceCalculator”.
* Two Junit Test Case validate and format phone No in “MyScanner”.
* Three JUnit Test Case to test elements of “Customer” class.

# Technical

### Package “differentCoat”

1. Class “PaintShopMenu”. The program starts work here. The "main" method creates an ArrayList object containing client data stored on disk. Then it enters the loop in which Menu is displayed, the choice is read and the appropriate method is called. The choice of the method takes place using the switch structure. Loop repeats until we choose the "0" Quit option. After leaving the program, the work ends.
2. Class “CalculateMenu”. Contains methods called in “PaintShopMenu”.
   1. “newCustomer” - Joins a new customer list sent by reference letters. Customer data is collected from the user via the keyboard, using the methods contained in the “MyScanner” class.
   2. “searchCustomer” - Prints the data from the list about the selected customer on the screen. The customer's index in the list is retrieved from the auxiliary search method “getIndexCustomer” for this index based on the customer ID.
   3. “removeCustomer” - It allows you to remove a client from the list. Like “searchCustomer”, he uses “getIndexCustomer”.
   4. “displayCustomers” - Prints all clients currently on the list you have sent. It was possible to internalize the Enhanced type of loop. A single client is printed using the "toString" method.
   5. “editCustomer” - This method allows you to edit customer data. The exception is ID assigned to "rigid" to the client. The individual data can be edited or left unchanged by pressing Enter.
   6. “saveList” - Another method from the menu. It allows you to save a list with current customer data to a file.
   7. “loadList” - This method is called automatically when the program starts. In the event of a problem with the data being read to the level, it creates an entire list with three sample clients. If the problem occurred at the level of one client, it adds one example to already read.
   8. “getIndexCustomer” - This is a private auxiliary method. Searches the customer's index in the list with the given ID. If it does not exist, it asks you to enter the ID again.
3. Class “SurfaceCalculator”. It contains three methods for three types of rooms. Each of them calculates the surfaces of the walls to be painted. Returns a value of double type expressing surface in square meters.
   1. “squareRoom” - Calculates the surfaces of four walls with dimensions height and length.
   2. “rectangularRoom” - The rectangular room consists of two opposite wall pairs. Input data is height, length and width type of double.
   3. “cylindricalRoom” - This is actually one wall (rectangle) whose length is equal to the circumference of the wheel. To calculate the surface, we use the height, diameter, and value of Pi from the standard Math library.

### 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 “testnewDate” is 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.

# End user

An example of how to use the program is shown in Appendix E, Fig. 5 and 6. Launched in Eclipse. After launching, the program displays the Menu and accepts only 5 of the displayed options. After selecting the option according to your choice, you must enter the year, date or two dates.

The data format must be accurate, otherwise you will have to enter it again. For example, the number of the day, the month of lib must consist of two digits. For example "03-05-1995" and not "~~3-5-1995~~". The order and separator (-) are also important. First day then one month at the end year (four digit). The program allows spaces when entering the date.

After completing the calculation, the program returns to the entry point, displays the Menu and asks for a choice.

The end of the program operation follows after selecting option 5.

A little more effort requires running the program in a DOS environment. First of all, it is best to copy both classes to a separate directory. I run cmd.exe and go to this directory.

By using the command "javac - version" we check the correctness of the installed java environment. If this is the problem, it is best to seek help on the "https://www.java.com" website. If I do not receive the number of the installed version of the java compiler.

The next step is to compile a class with "main ()". We do it with the command "javac testnewDate.java" necessarily with the extension ".java". The compiler will create a file with the same name but with the extension ".class".

Now we can start the program. We will do this with the command "java testnewDate" this time without any extension. An example is shown in Appendix E, Fig. 7. The remaining flow of programs is the same as in the Eclipse environment.

# 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 {

**private** **static** Scanner *sc*;

**private** **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() {

**int**[] date;

**boolean** valid;

**do** {

// user type date

date = *getDate*();

// asaign value

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

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

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

// for valid date print Julian date, else message

valid = *nDate*.validDate(day, month, year);

**if** (valid)

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

**else**

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

} **while** (!valid);

}

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

**int**[] start, end;

**boolean** valid = **true**;

// get two dates

**do** {

**if** (!valid)

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

**else**

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

start = *getDate*();

valid = *nDate*.validDate(start[0], start[1], start[2]);

} **while** (!valid);

**do** {

**if** (!valid)

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

**else**

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

end = *getDate*();

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

} **while** (!valid);

// if dates are valid print number of days between

System.***out***.println("\nDays elapsed : " + *nDate*.daysElapsed(start[0], start[1], start[2],

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

}

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

**int**[] birth;

**boolean** valid;

**do** {

// get birth date

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

birth = *getDate*();

valid = *nDate*.validDate(birth[0], birth[1], birth[2]);

**if** (valid)

System.***out***.println("\nYour age is " + *nDate*.ageDate(birth[0], birth[1], birth[2]));

**else**

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

} **while** (!valid);

}

// 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 };

}

}

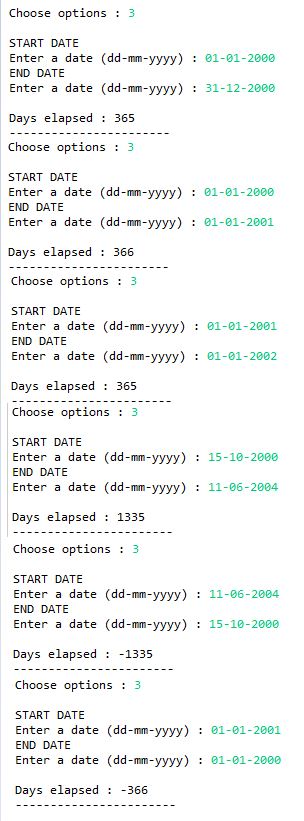


Fig:2 screen\_shot\Capture\_12.JPG

Test 53, 54: Test Report pos. 2.

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

Test 32: Test Report pos. 1.



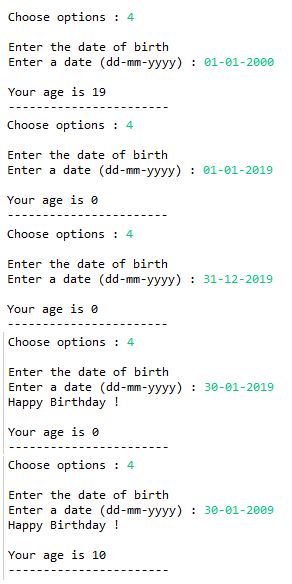
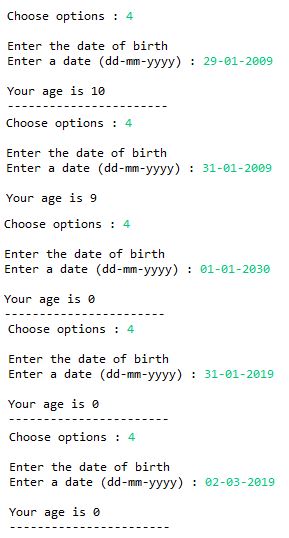
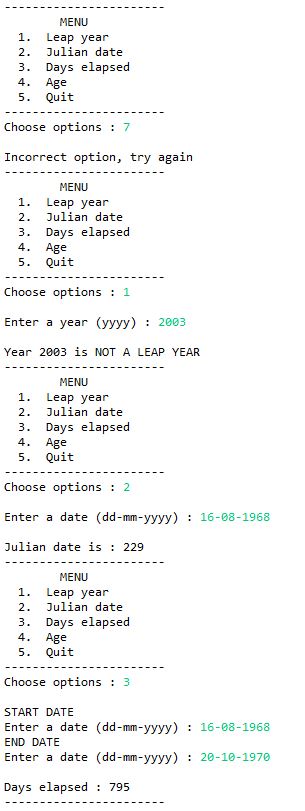


Fig:4 screen\_shot\Capture\_15.JPG

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

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

Test 63, 64 : Test Report pos. 3.



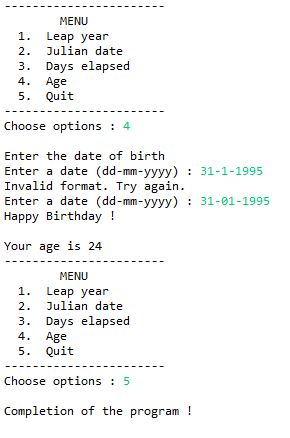


Fig: 5 and 6 Example of flow program

screen\_shot\Capture\_17/18 .JPG

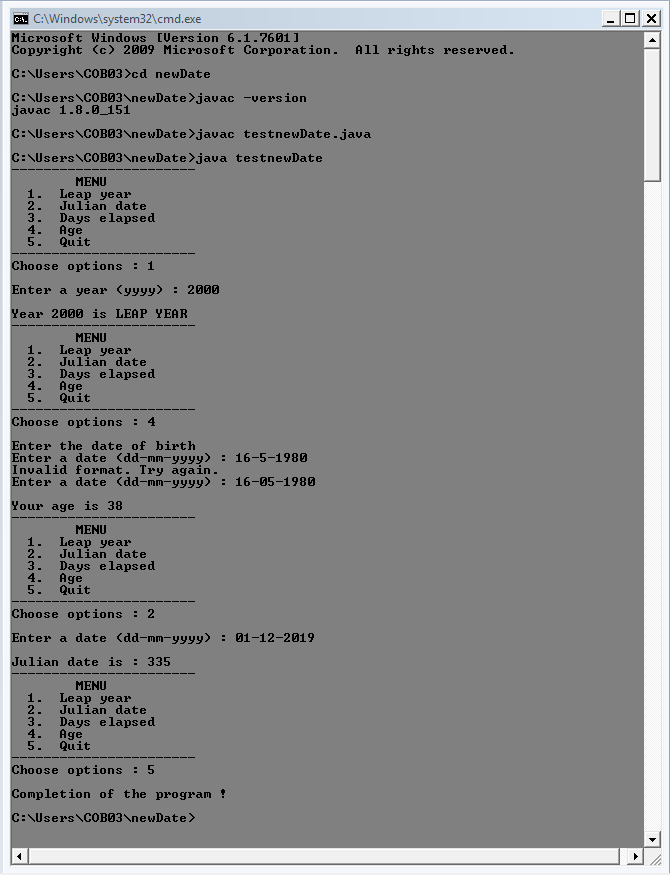


Fig: 7 Program launched in cmd.exe Windows, screen\_shot\Capture\_16.JPG

|  |
| --- |
| **newDate** |
|  |
| + valiDate ( day: int, month: int, year: int ) : boolean  + leapYear ( year: int ) : boolean  + julianDate (day: int, month: int, year: int ) : int  + daysElapsed ( day1: int, month1: int, year1: int, day2: int, month2: int, year2: int ) : int  + ageDate (dayBirth: int, monthBirth: int, yearBirth: int ) : int  - daysInMonth ( month: int, year: int ) : int |

|  |
| --- |
| **testnewData** |
| - sc : Scanner  - nDate : newDate |
| + main ( args: String[] ) : void  ­- testLeapYear ( ) : void  - testJulianData ( ) : void  - testDaysElapsed ( ) : void  - testAge ( ) : void  - getData ( ) : int[] |