# Assignment brief and mapping

**Submission Dates: 15 - 02 -2019**

|  |  |
| --- | --- |
| All Tasks |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task** |  | **Evidence** |  | **Unit coverage**  **(LO & AC**  **references)** |  | **Grading ref** |
| 1. Implement a software design using procedural programming |  | Written identification and code sample as per candidate brief |  | 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 17 |  | 0/1 |
| 2. Refine a procedural program to improve quality |  | Code developed as per student brief |  | 2.1, 2.2, 2.3, 2.4 |  | 0/1 |
| 3. Test the operation of a procedural program |  | Code developed as per student brief |  | 3.1, 3.2, 3.3, 3.4m 3.5, 3.6 |  | 0/1 |
| 4. Document a computer program |  | Written identification and code developed as per student brief |  |  |  | 0/1 |

**Assignment mark sheet**

### Candidate name Candidate number

**Bogdan Pasterak ZVO 0269**

Centre name: Donegal ETB Centre number 079 065

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | **Evidence** | | | **Achieved** |
| 1 | Written identification and code sample as per candidate brief | | |  |
| 2 | Code developed as per student brief | | |  |
| 3 | Code developed as per student brief | | |  |
| 4 | Written identification and code developed as per student brief | | |  |
|  | |  | Overall grade  (Pass/Fail) |  |

All parts of the tasks must be passed to allow the unit to be claimed.

|  |  |  |
| --- | --- | --- |
| **I can confirm that the evidence listed for this unit is my own work and was carried out**  **under the conditions and context specified in the assessment specification.** | | |
| Candidate signature |  | Date |
| I confirm that the candidate has achieved all the requirements of this unit with the evidence listed and the assessment was conducted under the specified conditions and context, and is valid, authentic, reliable, current and sufficient. | | |
| Tutor/assessor signature |  | Date |
| Quality assurance co-ordinator’s signature (where applicable) |  | Date |
| Qualifications consultant signature (where applicable) |  | Date |

### Where tasks involve inclusion of evidence, for example written descriptions or screenshots of code, these should be referred to and explained as appropriate in assessment reports.

**Evidence to be handed in:**

* Written reports for all tasks
* Reports should include screenshot evidence throughout

|  |  |  |  |
| --- | --- | --- | --- |
| **Learning Objective 1 : Implement a software design using procedural programming**  Assessment criteria | | | |
| The learner can: | Achieved 1 Yes  O No | Evidence Ref  PO Portfilio  PL Program Listing  A Appendix | Comment |
| 1.1 Identify the program modules and data and file structures required to implement a given design |  |  |  |
| 1.2 Select, declare and initialize variable and data structure types and sizes to implement design requirements |  |  |  |
| 1.3 Select and implement control structures to meet the design algorithms |  |  |  |
| 1.4 Select and declare file structures to meet design file storage requirements |  |  |  |
| 1.5 Select and use standard input/output commands to implement design requirements |  |  |  |
| 1.6 Make effective use of operators and predefined functions |  |  |  |
| 1.7 Correctly use parameter passing **mechanisms**. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Learning Objective 2: Refine a procedural program to improve quality** | | | | |
|  |  |  | |  |
| Assessment criteria |  |  |  | |
| The learner can: | Achieved 1 Yes  O No | Evidence Ref  PO Portfilio  PL Program Listing  A Appendix | Comment | |
| 2.1 Use an agreed standard for naming, comments and code layout |  |  |  | |
| 2.2 Define user functions to replace repeating code sequences |  |  |  | |
| 2.3 Implement data validation for inputs |  |  |  | |
| 2.4 Identify and implement opportunities for error handling and reporting. |  |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Learning Objective 3: Test the operation of a procedural program** | | | |
| Assessment criteria:  The learner can: | Achieved 1 Yes  O No | Evidence Ref  PO Portfilio  PL Program Listing  A Appendix | Comment |
| 3.1 Make effective use of available debugging tools |  |  |  |
| 3.2 Prepare a test strategy |  |  |  |
| 3.3 Select suitable test data and determine expected test results |  |  |  |
| 3.4 Record actual test results to enable comparison with expected results |  |  |  |
| 3.5 Analyze actual test results against expected results to identify discrepancies |  |  |  |
| 3.6 Investigate test discrepancies to identify and rectify their causes. |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Learing Outcome 4: Document a Computer Program**  Assessment criteria | | | |
| The learner can | Achieved 1 Yes  O No | Evidence Ref  PO Portfilio  PL Program Listing  A Appendix | Comment |
| 4.1 Create documentation to assist the users of a computer program |  |  |  |
| 4.2 Create documentation for the support and maintenance of a computer program. |  |  |  |

**Grade Summary**

|  |  |  |
| --- | --- | --- |
|  | Target Score for pass | Actual Score |
| Learning Objective 1: Implement a software design using procedural programming | 7/7 |  |
| Learning Objective 2: Refine a procedural program to improve quality | 4/4 |  |
| Learning Objective 3: Test the operation of a procedural program | 6/6 |  |
| Learning Outcome 4: Document a Computer Program | 2/2 |  |
| Total | Out of 19 |  |
| Grade (Pass/Fail) |  |  |

## Candidate feedback sheet

Candidate name

|  |  |
| --- | --- |
| **Task & date** | **Assessor comments/action plan**  (The assessor should sign after each feedback session) |
|  |  |

Candidate signature Date

Tutor/assessor signature Date

**1 Implement a Software Design Using Procedural Programming**

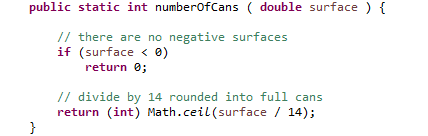
* 1. **Identify the program modules, data and file structures required to implement a given design**

I divided the program into two packages. Package “differentCoat” contains the main elements of the program. The “PaintShopMenu” class includes the “main” start method and main loop. The second most important class is Customer, creates and supports objects held in the ArrayList in “PaintShopMenu”. The “CalculateMenu” class contains methods that are used in main method. The remaining classes perform a service role for “CalculateMenu”. Full UML Class diagram is included in the Appendix 5.3.

Second package “differentCoatTest” includes several JUnit Test Case and collecting all test case JUnit Test Suite "All Test". Due to the required user response, I did part of the Black Box tests.

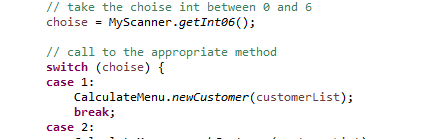
**1.2 Select, declare and initialize variable and data structure types and sizes to implement design requirements**

The basic object in the application is Customer. It consists of four name fields: String, phone: String, paintCans: int and id: final int. In "PaintShopMenu", the ArrayList of Customer is used. The dimensions of the rooms and the surface to be painted are expressed in double. In “PaintRequiredCalculator” class “numberOfCans” method, the surface is divided by 14, rounded up and cast into an int.



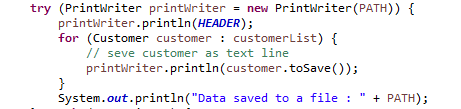
**1.3 Select and implement control structures to meet the design algorithms**

All customer responses are validated and in the event of a failure, it is requested to enter the data inside the loop again. All multiple selections are conducted in the switch statement. For example, choosing the main loop option in “PaintShopMenu”. Larger structures with methods of similar purpose are closed in separate classes. At the end, everything is packed in package.



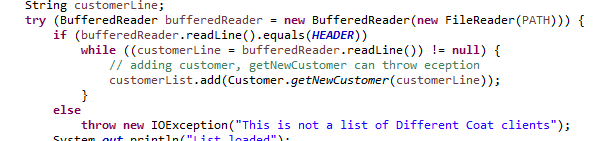
**1.4 Select and declare file structures to meet design file storage requirements**

For the ability to view and edit data in a text file I use to write PrintWriter and read BufferedReader. The record is made using the "toSave" method in the Customer class (using the separator as a tabulator). The readout is done with the help of the "getNewCustomer" method that returns the object, and the failure throws the exception. Both methods are in the "CalculateMenu" class.



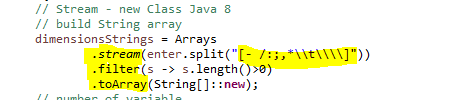
**1.5 Select and use standard input/output commands to implement design requirements**

For writing within try / catch with sources I use the "println" method. Similarly, when reading, I use "readLine" from "BufferedReader". To make sure of the correct file, I added the first line as a constant.



**1.6 Make effective use of operators and predefined functions**

Many times I created private methods to avoid copying existing fragments of code. RegEx was familiar with the validation of data from the user. Also interesting was the use of the new "Stream" class in Java 8 inside “getDimensions” method.



**1.7 Correctly use parameter passing mechanisms.**

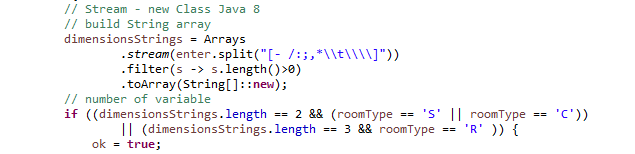
In every "CalculateMenu" method, a list of clients is passed be references. Similarly in the "MyScanner" class, the "getDimensions" method returns double arrays depending on the type of room. In the "getRoomTypeOrEmpty" method, I use the wrapper class "Character" to be able to return "null". This allows you to bypass editing this client parameter.



1. **Refine a procedural program to improve quality**

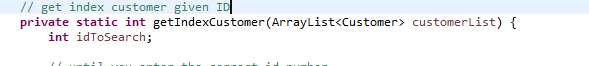
**2.1 Use an agreed standard for naming, comments and code layout**

In the code, I tried to use standard names that best describe the data and methods. Comments describe harder parts of the code. Formatting code in Eclipse is convenient and automatic, but for clarity, I formatted some fragments manually.



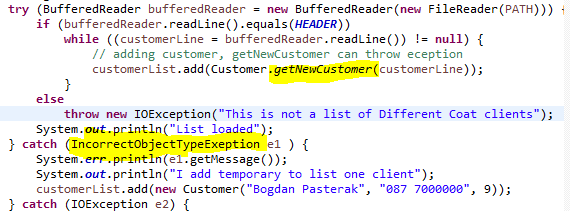
**2.2 Define user functions to replace repeating code sequences**

The best example is the private method "getIndexCustomer" in "CalculateMenu" used in "searchCustomer", "removeCustomer” and “editCustomer”. A similar role is played by almost all methods in "MyScanner", except that some of them use internal methods of other.



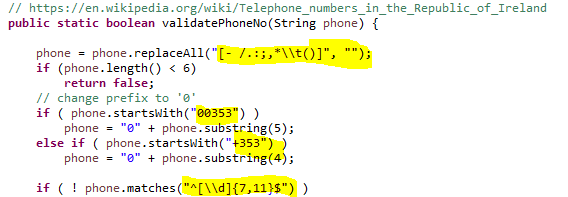
**2.3 Implement data validation for inputs**

Validation of data takes place on many levels. The basic validation starts in Customer objects via setters (name, phone, constructor and especially "getNewCustomer"). The next level is contact with the user. The entire "MyScanner" class was created for these needs. A separate standard is communication with the file system. Here I use my "IncorrectObjectTypeExeption" class inheriting from "Exception"



**2.4 Identify and implement opportunities for error handling and reporting.**

The challenge has been to validate phone numbers. I tried to take into account the extensive structure of the numbers. In addition to prefix (0, 00353 and +353) in Ireland a high diversity is allowed. It is even harder to control the user's responses.



Besides, my program allows you to edit data in a file. Here, the data is validated in all possible ways. The most difficult thing was the behavior of auto numbering of customers ID and protection against swaps in the list and file.

**3. Test the operation of a procedural program**

**3.1 Make effective use of available debugging tools**

One of the most interesting moments is to look for errors. The structure of the program looked correct, however, the result gave the unpredicted data. Using the debugger I found a place where the data was formatted incorrectly. It turned out that I forgot about the special negative sign function in RegEx. Normally I use it [a-zA-z0-9], but in  expression it returns all characters between 32(space) and 47(backslash). The dot (46) was to be used as part of the double number and was treated as one of the separators. Screenshot of this operation in Appendix 5.4

**3.2 Prepare a test strategy**

Overview: Creating an application for holding, searching and editing customer data from Different Coat.

Testing the program should take place in three stages. The lowest level will include automatic tests included in the program. They refer to individual elements of the system, above all the "Customer" class. There are also tests that check the correctness of the surface calculations and the number of cans of paint. They are also for testing phone numbers. Automatic tests are made using JUnit Test Case and Suite. Automatic tests are made using JUnit Test Case and Suite.

The next level of tests are black box tests. They should check communication with the program user as well as data writing and reading operations. They are made by hand. They should check the possible wide range of users' behavior. The program should respect the white marks at the beginning and end of the answers. Similarly, it should behave for the different types of separators. In the case of read and write methods, many incompatibilities must be considered. Starting from the lack of a file to read until the phone number is overwritten, the value is not valid.

The last level is tests with the client checking if the program meets his expectations. A few items not included in the specification were added to the program. This is a data read operation and editing of the customer's data in the list. Although they seem to be a logical consequence of the program structure, they strive for acceptance by the client. The same graphic arrangement of the message and responses as well as the style of communication requires the acceptance of the recipient of the program.

Automatic tests and black box tests will be performed in the Eclipse IDE. The automatic tests will be performed using the class 4 Junit.

**3.3 Select suitable test data and determine expected test results**

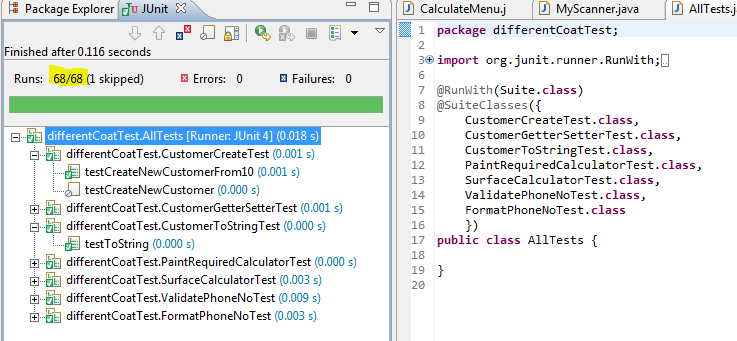
Automatic tests include classes “Customer”, “SurfaceCalculator”, “PaintRequiredCalculator” and partial “MyScanner” (methods “validatePhoneNo” and “formatPhoneNo”). Other classes and methods require interaction with the user. They were tested "manually". The automatic tests were carried out using the JUnit 4 Test Case and the JUnit 4 Test Suite grouping together. I have included test data in the table in Appendix 5.8

**3.4 Record actual test results to enable comparison with expected results**

The test results are presented in the form of a report in Appendix 5.7.

* 1. **Analyze actual test results against expected results to identify discrepancies**

The program was written in two parallel processes. After declaring the new method, I wrote the appropriate test methods. The next step is to send a code that meets the test requirements. Thanks to this program is free of errors.



* 1. **Investigate test discrepancies to identify and rectify their causes.**

My program passes the test. Appendix 5.7.

**4. Document a computer program**

**4.1 Create documentation to assist the users of a computer program – user manual**

The user documentation can be found in Appendix 5.6.

**4.2 Create documentation for the support and maintenance of a computer program – maintenance /technical document**

Software Documentation in Appendix 5.5

**5. Appendix**

5.1 Program Listing

5.2 Program Flow – output

5.3 ULM Classes Diagram

5.4 Debugging Screenshot

5.5 Software Documentation

5.6 User Documentation

5.7 Test Report

5.8 Test Data