IN-COURSE ASSESSMENT (ICA) SPECIFICATION

Module Title: **Java Programming Reassessment**	Module Leader: **James Fairbairn**
	Module Code: **CIS1010-N**
Assignment Titles: **Telephone Contract Manager Prototype**	Deadline Dates: **Friday 30th September 2022**
	Deadline Time: **4:00pm**
	Submission Method: Online (Blackboard) ☑ Middlesbrough Tower ☐

Online Submission Notes:

1. Please follow carefully the instructions given on the Assignment Specification

- When Extenuating Circumstances (e.g. extension) has been granted, a fully completed and signed Extenuating Circumstances form must be submitted to the School Reception or emailed to scedt-assessments@tees.ac.uk.

FULL DETAILS OF THE ASSIGNMENT ARE ATTACHED INCLUDING MARKING & GRADING CRITERIA

Java Programming (CIS1010-N)
Assignment Specification

Contents

Introduction

This document describes the assessment for the module. It is an in-course assignment (ICA) with a single submission. This is an individual ICA.

Background - Telephone Contract Manager Prototype

A small communications company requires a simple telephone contract manager prototype to demonstrate key features. The company offers the following packages:

Monthly Rates (in pence)

Package	Data Bundle			
	Low 1GB	Medium 4GB	High 8GB	Unlimited
Small 300 mins	500	700	900	N/A
Medium 600 mins	650	850	1050	N/A
Large 1200 mins	850	1050	1250	2000

Notes:

- Contract periods can be 1, 12, 18 or 24 months.
- Each contract has a reference number (e.g. JB123B – Two letter followed by three digits and a letter. The last letter can only be B for Business Account or N for Non-Business Account).
- Business customer receive a 10% discount and must take at least a 12-month contract.

- Charity customers receive a 30% discount for any length of contract
- Non-business customers taking a 12 or 18-month contract receive a 5% discount and 10% for a 24-month contract
- International calls are normally charged separately, but customers can choose to use the minutes included with package to make these calls. If this option is selected, the charges above are increased by 15%.
- All calculations are performed on the pence values and decimal portion disregarded.
- Contracts details also include the client name (eg: J Mason) – maximum of 25 characters

When required, summary outputs should be displayed using the following format:

```
+---------------------------------------------+
|                                             |
| Customer: J Mason                           |
|                                             |
|      Ref: JB123B         Date: 15 Oct 2021  |
|  Package: Large (1200)   Data: Low (1GB)    |
|   Period: 12 Months      Type: Business     |
|                                             |
| Discount: 10%     Intl. Calls: No           |
|                                             |
|      Discounted Monthly Charge: £7.65       |
|                                             |
+---------------------------------------------+
```

Notes:

- Monthly charge is centred and the word discounted **only** appears if a discount has been applied.
- If discount is not applied, the word None should be displayed instead of the percentage value.
- The border should be included with the summary.

Part 1 - Telephone Contract Manager Prototype (80%)

Instructions

A prototype console application (no GUIs or GUI dialogs) is required. Your prototype should display a menu similar to the following:

```
1.    Enter new Contract
2.    Display Summary of Contracts
3.    Display Summary of Contracts for Selected Month
4.    Find and display Contract
0.    Exit
```

Once an option is selected and processed, the menu should be redisplayed until the exit option is selected. Each operation is described below:

Option 1: Enter new Contract

- Ask the user for the inputs: client name, package, data bundle, reference, period and include international calls. Alternatively, the estimated minutes and data (megabytes) can be entered and the program automatically selects the appropriate package and data bundle. Note: All inputs should be validated.
- Calculate the total and discounted prices for the contract.
- Display the summary (it **<u>must</u>** use the format in the example).
- Create the appropriate sub class object
- Save (append) the data for the contract to the text-based summary file (contracts.txt). Each line in the file must hold the details for a single contract with the following information separated by spaces or tabs:
    - Contract Date (today's date - see technical details for format).
    - Package (1=Small, 2=Medium and 3=Large)
    - Data Bundle (1=Low, 2=Medium, 3=High and 4=Unlimited).
    - Period in Months
    - Allow international call from package minutes (Y or N)
    - Reference Number.
    - Monthly Charge (in pence).
    - Client Name.

Option 2: Summary of Contracts

User is asked to select the main (`contracts.txt`) or archive (`archive.txt`) contacts and a summary of the following is displayed:

- Total number of contracts.
- Number of contracts with High or Unlimited Data Bundles.
- Average charge of large package contracts.
- Total number of contracts per month (assume the files only include data for the current year). Show an entry for all twelve months even those that have an entry of zero.

A sample archive file can be downloaded from Blackboard. For the supplied archive file, the output should look similar to the following:

```
Total Number of contracts: 1000
Contracts with High or Unlimited Data Bundles: 300
Average charge for large packages: 12.77

Number of contracts per Month:

Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
91   67   83   84  103   71  103   68   80   65   99   86
```

Option 3: Summary of Contracts for a Selected Month

The user is asked to select the main or archive contracts and to enter a month. A summary of contracts (as specified in Option 2) should then be displayed for the specified month.

Option 4: Summary of Contracts for a Selected Month

The user is asked to select the main or archive contracts and enter the search text. The selected file should be searched for any matches (full or partial) against the reference or client name. For example entering **DE** would match against reference number "**DE**123N" and Client Name "C San**de**rson".

For each matching entry, the contract summary (see option 1) should be displayed.

Option 0: Exit

Display an appropriate message and exit the application.

Technical Expectations

The prototype must be developed using NetBeans 12 and JDK version 11. Name your project `ContractManager`.

Data files must be named `contracts.txt` and `archive.txt`. A sample of the `archive.txt` file can be downloaded from the Assessment section of the blackboard module page. The format of the files is identical. Ensure you never write to `archive.txt`. In the event you corrupt `contracts.txt` simply replace its contents (i.e. copy and paste) using `archive.txt`.

The implementation must include a `Contract` class that represents a single contract. This class should use appropriate encapsulation and not include any keyboard input or console output.

The following method will get today's date:

```
    public static String getDate()
    {
        Calendar cal = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy");
        return sdf.format(cal.getTime());
    }
```

You will need to `import java.util.Calendar` and `java.text.SimpleDateFormat`

Internally use integers for monetary values (hold them as pence). Display as pounds and pence by dividing the value by 100 and using the `printf` method to display. For example:

```
   System.out.printf("£%.2f ", amount / 100);
```

Validation on all user inputs is a must. As well as being correct, it must also be user friendly by concisely informing the user of the error and allowing a re-entry opportunity.

You code must be well documented throughout. For top marks you must document your code using Javadoc. For more information see this Oracle tutorial. To make your code easily readable by the module team you must adhere to the Google Java style guide. Deviation from this guide will result in lower marks.

Guidance

Design your solution using pseudo code or flow charts. Resist the urge to start hacking away immediately.

Save frequently and backup your work every day. Work methodically and proceed using small steps.

Work individually. Your code will be checked for plagiarism.

You are **strongly advised** to **retain all versions** of your **code** and your **development notes**, so that you can show the development of your work. This may be required as evidence in the event of any query about the authorship of your work

Part 2 - Testing Documentation (10%)

Instructions

Produce a black-box test plan containing the test cases applied to your solution to Part 1. The expected format of your test cases is shown in the table below (colour scheme is not necessary but headings are):

Id	Description	Steps	Expected	Actual	Result	Comment

Guidance

You should begin this documentation the same time as you start implementation of Part 1. It is much simpler to test as you develop (also known as *integration* testing) as opposed to 'faking' it before submission.

Remember to test all cases. It is very easy to test for success, but you must also test for failure, and your boundaries (your solution is likely to make use of multiple looping constructs).

You may want to consider using alternative tools such as Microsoft Excel to make managing and organising your test plan easier as the number of test cases grow.

When testing your prototype, we will look for correlation between your test plan to the solution. You will be marked down for tests that you have "passed" if they should fail under assessment testing.

Part 3 – Report (10%)

Instructions

The client is considering implementing the full system and would like your advice on the following:

- What security measures/features would need to be implemented to ensure the data in the system is safeguarded?
- What legal considerations need to be respected before the full system becomes operational?
- How could development and implementation risks be minimised?
- Current state of the prototype:
    - Which parts have been successfully implemented and those parts which have not been fully successful?
    - Provide a description of any known errors.

Your advice must be documented in a brief but professionally formatted report that is no more than **500 words** to be submitted with the prototype.

Guidance

When referencing code, to make code examples more readable you can copy and paste Java code using Visual Studio Code (a free code editor from Microsoft) into Microsoft Word. This will bring along Java syntax highlighting, and the monospaced font `Consolas`.

Stick to the word limit and ensure you cover each point in equal detail. You will lose marks if you fail to address all points, no matter how much detail you add to another to compensate.

Marking Criteria

Criteria	Excellent (70-100%)	Very Good (60-69%)	Good (50-59%)	Satisfactory (40-49%)	Fail (0-39%)
Create menu and enter new contract [38]	All aspects implemented as required: abstract, serializable; private/ public/ protected modifiers; Subclasses correct overrides; Subclasses constructor using super(), correct logic. Robust main method that does not throw uncaught exceptions. File Open Update Close implemented.	Most aspects implemented as required: abstract, serializable; private/ public/ protected modifiers; Subclasses correct overrides; Subclasses constructor using super(), correct logic. Working main method with evidence of validation and exception handling. File Open Update Close implemented.	Many aspects implemented as required: abstract, serializable; private/ public/ protected modifiers; Subclasses correct overrides; Subclasses constructor using super(), correct logic. Working main method with evidence of validation and exception handling, but data errors still possible. File Open Update Close implemented	Some aspects implemented as required: abstract, serializable; private/ public/ protected modifiers; Subclasses correct overrides; Subclasses constructor using super(), correct logic; Working main method with evidence of validation and exception handling, but exceptions still thrown. File Open Update Close implemented.	Few or no aspects implemented as required: abstract, serializable; private/ public/ protected modifiers; Subclasses correct overrides; Subclasses constructor using super(), correct logic. Partial main method. File An attempt at File Open Update Close.
Summary of contracts [20]	Appropriate file selected. Excellent logic/structure for file handling. All aspects implemented accurately.	Appropriate file selected. Good logic/structure for file handling. Most aspects implemented accurately.	Appropriate file selected. Reasonable logic/structure for file handling. Many aspects implemented accurately.	Appropriate file selected. Some attempt to provide logic/structure for file handling. Some aspects implemented accurately.	Unable to select the appropriate file. Little of no attempt to provide logic/structure for file handling. Very few or no aspects implemented accurately.
Find and display contract(s) [7]	Successfully search file for matches; Correct contract summary displayed; error message provided if no matches found.	Most aspects successfully implemented; correct summary displayed; appropriate error message provided if no matches found.	Many aspects successfully implemented; correct summary displayed; appropriate error message provided if no matches found.	Some aspects implemented successfully; partial summary displayed; no or limited message provided if no matches found.	Few or no aspects implemented successfully; limited or no summary displayed; no or limited message provided if no matches found.

Code efficiency and documentation [15]	Extremely efficient and well documented code. The code is well laid out, good use of Javadoc, excellent efficiency - internal methods, appropriate data structures used.	Very efficient and well documented code. The code is well laid out, good use of Javadoc, some efficiency employed - internal methods, appropriate data structures used.	Reasonable efficiency demonstrated, and reasonably documented code. The code is well laid out, good use of Javadoc, some efficiency employed. appropriate data structures used.	Some efficiency demonstrated, and some comments documented in the code. The code is reasonably well laid out, some use of Javadoc, some efficiency employed. appropriate data structures used.	Little or no efficiency demonstrated, and code not sufficiently commented The code is not well laid out, no use of Javadoc, limited or no efficiency employed.
Non-functional requirements testing [10]	9+ Testing is organised, structured and exhaustive. 8+ Testing is organised, structured and substantive. 7+ Testing is organised, structured and comprehensive.	A comprehensive black-box test plan that covers most test cases including success, failure, and range.	An acceptable black-box test plan showing individual test cases, but it is not sufficiently extensive.	A limited black-box test plan is evident showing individual test cases, but it is poorly formatted and/or contains few tests.	No evidence of testing provided.
Report [10]	Report critically evaluates the implementation risks, security, and legal considerations related data safeguarding in real world applications. The performance of the prototype is documented with an excellent evaluation covering solution applicability, code efficiency, supported with evidence from the test results. Any bugs have plausible solutions suggested.	Report evaluates the implementation risks, security, and legal considerations related data safeguarding in real world applications, evaluation includes facts and opinions of others with appropriate citations. The performance of the prototype is described with meaningful evaluation covering efficiency, results from testing, and possible bug fixes (if required) but they might not be entirely suitable.	Report addresses the implementation risks, security, and legal considerations related data safeguarding in real world applications. Evidence is mostly limited to hypothetical situations. The performance of the prototype is described whilst referencing testing results. Bugs are identified but no possible fixes suggested.	Report addresses some of the implementation risks, security, and legal considerations related to data safeguarding in real world applications. Limited evaluation present. The performance of the prototype is loosely described but lacks evidence and conviction.	Report does not address the implementation risks, security, and legal considerations related data safeguarding in real world applications. Report does not address the prototype performance.

Report [10]	Report is very well presented, absent of proofing errors, word count +/- 10%.	Report is well presented, very few proofing mistakes, word count +/- 10%.	Report is reasonably presented, minor proofing mistakes, word count +/- 15%.	Report is reasonably presented, does contain some proofing mistake errors, word count +/- 20%.	Report is poorly formatted, with little meaningful content. Word count +/- 30%.

Deliverables

You must submit your work as a ZIP file (i.e. *a0123456_JP.zip*) via *Blackboard* with the following directory structure:

- **Report**: this directory should contain your final report.
- **ContractManager**: this directory should contain the entire source code for the prototype.
- **Testing**: this directory should contain your testing documentation.

Unless otherwise stated all documentation must be submitted in .docx or .pdf format. A starter ZIP file is available on Blackboard alongside this specification.

Learning Outcomes

Personal and Transferable Skills:

1. Produce software documentation using source code comments.
2. Prepare reflective report to consider the effectiveness of the solution, workload, planning and development activities.

Research, Knowledge and Cognitive Skills:

3. Demonstrate knowledge of basic OO concepts including classes, objects, methods, and encapsulation.
4. Build an effective Java solution to a simple requirement specification using appropriate development methods.
5. Test a software application using a set of documented test cases.

Professional Skills:

6. Implement a simple application using the Java programming language.
7. Describe the legal issues related to software development.
8. Discuss own performance whilst constructing a software solution.

Document History

Revision 0 (28-Jul-22): Initial version of the 2021/22 reassessment specification.