IN-COURSE ASSESSMENT (ICA) SPECIFICATION

Module Title: **Object-Oriented Development**	Module Leader: **James Fairbairn**
	Module Code: **CIS1056-N**
Assignment Titles: **GAS Package Prototype**	Deadline Dates: **Wednesday 11th January 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

Object-Oriented Development (CIS1056-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 - Games Access Services (GAS)

Games Access Services (GAS) currently allow the purchase of individual games for a one-off fee. GAS would like to expand their purchasing system to include subscriptions to packages that provide access to classic games. The packages would be purchased using monthly subscription fees allowing access to different categories of games (bronze, silver, or gold) for a defined number of months. The type and duration are listed in the table below:

Monthly Subscription

Package	Duration (months)			
	1	3	6	12
Bronze	6.00	5.00	4.00	3.00
Silver	8.00	7.00	6.00	5.00
Gold	9.99	8.99	7.99	6.99

To promote the service, discount codes would be issued that reduce the above subscription. The code format would be two letters, two digits, a letter, and a digit (e.g. `JF22L5`):

- The first two digits would be used to define how the code was issued (they can be any two letters).
- The third and fourth digits would indicate the year the code would be valid in.
- The fifth character would indicate the month the code expires (E= January to June and L = July to December).
- The final digit would indicate the how much discount (1%-9%).

The default payment term is monthly. However, customers can choose to pay the full subscription as a one-off fee to receive an additional 5% discount.

As well as entering the inputs identified above, the customer's name should also be input (e.g. "J Smith" up to a maximum of 25 characters). Once the validation of inputs and cost calculation has been completed, a summary should be output:

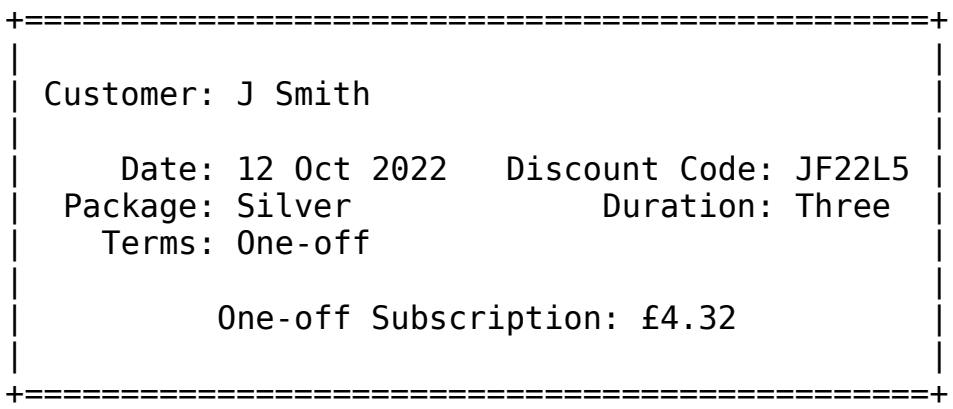

Figure 1: Summary Output

Notes:

- The duration must be displayed as a word (One, Three, Six or Twelve).
- For monthly payment the text "One-off Subscription" should be replaced with "Monthly Subscription"
- The subscription line must be centred.

Part 1 - GAS Java Prototype (70%)

Instructions

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

```
1. Enter new Subscription
2. Display Summary of subscriptions
3. Display Summary of subscription for Selected Month
4. Find and display subscription
0. Exit
```

Once an option is selected and processed, the menu should be redisplayed until the exit option is selected. Each operation is described in the next set of sub-headings.

1. Enter new Subscription

Ask the user to input the: customer's name, package, duration, discount code and payment terms (one-off or monthly). Note: All inputs should be validated.

- Calculate the one-off/monthly subscription.
- Display the subscription summary (it must use the format in Figure 1: Summary Output).
- Save (append) the subscription data to the text-based summary file (subscription.txt). Each line in the file must hold the details for a single subscription with the following information separated by spaces:
    - Subscription date (today's date - see §Technical Expectations for format).
    - Package ('B' for Bronze, 'S' for Silver, and 'G' for Gold).
    - Duration (numeric value).
    - Discount Code (save as '-' *if* a discount code is not provided by the user).
    - Payment terms ('O' for one-off or 'M' for Monthly).
    - Subscription (in pence) the raw one-off or monthly figure *after* discounts.

- Customer name.

2. Summary of Subscriptions

Prompt the user to select the current (current.txt) or sample (sample.txt) subscriptions, then show a summary including:

- Total number of subscriptions.
- Average monthly subscription fee.
- Percentage of each type of subscription (bronze, silver, and gold).
- Total number of subscriptions per month (assume the subscription file only includes data for the current year). Show an entry for all twelve months, even those that have an entry of zero.

Sample Output (note figures not accurate)**:**

```
Total number of subscriptions: 910
Average monthly subscription: 76
Average monthly subscription fee: £6.09

Percentage of subscriptions:
Bronze: 34.0
Silver: 34.8
Gold: 31.2

Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
83   89   74   72   81   84   100  99   62   91   90   75
```

3. Summary of Subscriptions for a given month

User is asked to select the current (current.txt) or sample (sample.txt) subscriptions, a valid month and a summary of the following is displayed:

- Total number of subscriptions.
- Average monthly subscription fee.
- Percentage of each type of subscription (bronze, silver and gold).

Sample Output (note figures not accurate):

```
Total number of subscriptions for May: 81
Average subscription fee: £6.09

Percentage of subscriptions:
Bronze: 34.0
Silver: 34.8
```

```
Gold: 31.2
```

4. Find and Display Subscription

Prompt the user to select the (current.txt) or sample (sample.txt) subscriptions, then enter text to search. The selected file should be searched for any matches (full or partial) against the customer's name. For example, entering NS would match against "F Townsend".

For *each* matching entry, the subscription summary (see Figure 1: Summary Output) should be displayed.

0. Exit

Display an appropriate message and exit the application.

Technical Expectations

A starter project `SubscriptionManager` is provided for you, simply extract the *a0123456_OO.zip* available alongside this ICA brief on Blackboard. Before opening the NetBeans project, ensure you rename "*a0123456*" to your student number. The prototype must be developed in **NetBeans 12**.

Data files must be named `current.txt` and `sample.txt`. The format of the files is identical. Ensure you never write to `sample.txt`.

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.

The implementation must include a `Subscription` class that represents a single subscription. You must demonstrate knowledge of basic object-oriented concepts including classes, objects, methods, and encapsulation.

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);
```

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 (20%)

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 **700 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%)
Java solution structure [15]	Sensible scoping throughout, appropriate data structures used, with little wasted code that fully adheres to the Google Java style guide.	Sensible scoping throughout, appropriate data structures used, with little wasted code, minor deviations from Google Java style guide.	Appropriate Java packages imported, appropriate data structures used but approach may not be the most efficient, variables created succinctly, some repeated code blocks, with some deviations from the Google Java style guide.	Few Java packages imported, some data structures may be inappropriate, variables haphazardly created, contains blocks of repeating code, with many deviations from the Google Java style guide.	Code is not in a runnable state. Contains syntax errors, or unresolved runtime errors.
	Excellent use of object-oriented concepts including classes, objects, methods, and encapsulation.	Good use of basic object-oriented concepts including classes, objects, methods, and encapsulation.	Reasonable attempt to use object-oriented concepts including classes, objects, methods, encapsulation but questions over aspects of implementation.	Limited use of object-oriented concepts with most of the solution relying on a procedural approach.	Code does not make use of object-oriented programming.
	Code is excellently documented using Javadoc.	A reasonable attempt has been made to document the code using Javadoc.	Appropriate code comments are spread through the solution.	Code comments are sparse.	Few useful comments in code.
	The solution throws no uncaught exceptions.	Exception handling covers exceptions from imported packages, file verification, user input, and type casting.	Exception handling is present but limited to file verification and type checking.	Exception handling is very limited.	Exception handling is not present or left as empty blocks.

Menu and enter new subscription [25]	*All* aspects implemented as required. Correctly uses abstract, serializable; private/ public/ protected modifiers; Subclasses use correct overrides; Subclasses with constructor using `super()`, all with correct logic.	*Most* aspects implemented as required. Correctly uses abstract, serializable; private/ public/ protected modifiers; Subclasses use correct overrides; Subclasses with constructor using `super()`, all with correct logic.	*Many* aspects implemented as required. Correctly uses abstract, serializable; private/ public/ protected modifiers; Subclasses use correct overrides; Subclasses with constructor using `super()`, all with correct logic.	*Some* aspects implemented as required. Some questions over use of abstract, serializable; private/ public/ protected modifiers; Subclasses and overrides; Subclasses may not call `super()`. Logic may not be entirely correct.	*Few or no* aspects implemented as required. Some questions over use of abstract, serializable; private/ public/ protected modifiers; Subclasses and overrides; Subclasses may not call `super()`. Logic may not be entirely correct, or erroneous.
Summary of subscriptions (including months) [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 subscriptions [10]	Successfully search file for matches; Correct subscription 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.
Non-functional requirements testing [20]	18+ Testing is organised, structured and exhaustive. 16+ Testing is organised, structured and substantive. 14+ 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.	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.	Report addresses the implementation risks, security, and legal considerations related data safeguarding in real world applications. Evidence is mostly limited to hypothetical situations.	Report addresses some of the implementation risks, security, and legal considerations related to data safeguarding in real world applications. Limited evaluation present.	Report does not address the implementation risks, security, and legal considerations related data safeguarding in real world applications.
Report	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 is very well presented, absent of proofing errors, word count +/- 10%.	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 is well presented, very few proofing mistakes, word count +/- 10%.	The performance of the prototype is described whilst referencing testing results. Bugs are identified but no possible fixes suggested. Report is reasonably presented, minor proofing mistakes, word count +/- 15%.	The performance of the prototype is loosely described but lacks evidence and conviction. Report is reasonably presented, does contain some proofing mistake errors, word count +/- 20%.	Report does not address the prototype performance. Report is poorly formatted, with little meaningful content. Word count +/- 30%.

Deliverables

You must submit your work as a ZIP file (i.e. *a0123456_OO.zip*) via *Blackboard* with the following directory structure:

- **Report**: this directory should contain your final report.
- **SubscriptionManager**: 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.

Before submitting your work to **Blackboard**, you must ensure that **your NetBeans project can be opened**, **builds**, and **executes** in the **school's Linux labs**. Please indicate which Linux lab computers you tested your work on.

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 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 an appropriate programming language.
7. Describe the legal and security issues related to software development.
8. Discuss own performance whilst constructing a software solution.

Document History

Revision 0 (25-Sep-22): Initial version of the 2022/23 assessment specification.