Advanced Software Engineering (F21AS)
Coursework 2021

Overview

This coursework contributes 100% of your mark for the course, and involves the design and implementation of an application. The application has been chosen to be complex enough to enable you to try out various software engineering features, whilst small enough to fit in the time available. The scenario is based around a simulation of a coffee shop.

You should work in groups of 4-5, with the group collaborating over the design and implementation of the application. Larger groups will be expected to achieve proportionally more.

All group members are required to contribute to design, implementation and report writing. To ensure fairness, each group member will be asked to assess the contribution of other members. Given differences in educational and vocational backgrounds, some variance is expected in the level of contribution. However, if the contribution of a group member is significantly below what is expected, then the marks for each student in the group will be adjusted, taking into account each member's contribution.

The application is developed in **two stages**:

- Stage 1 follows on from materials you will be taught in Part 1 of the course, and concentrates on planning, iterations, data structures, exceptions and unit testing.
- Stage 2 follows on from materials you will be taught in Part 2 of the course, and focuses on threads, design patterns and agile methods.

The requirements of the application are outlined in the following sections of this coursework specification. Each stage has two sets of requirements:

- Functional requirements, which describe **what** the application should do.
- Software engineering requirements, describing **how** you should develop the application.

The following deliverables are required:

- A **group development plan**, handed in before your group begins implementation work.
- **Two group reports**, describing your group's work, to be written by the whole group.

The following table shows when each of these deliverables is due, and how much they are worth:

Stage	Deliverable	Weight	Deadline
Stage 1	Group development plan*	50%	Thursday 11th February (Week 5)
Stage 1	Group report		Thursday 4th March (Week 8)
Stage 2	Group report	50%	Thursday 1st April (Week 12)

* The development plan will be assessed alongside the Stage 1 group report, so you will not receive separate feedback on this. All deadlines are strict and late submission penalties apply.

Stage 1

Stage 1 is designed to assess your understanding of planned iterative development and your knowledge of data structures, exception handling and unit testing, which are all taught in the first half of the course. You are required to develop a simulation of a coffee shop. This first stage develops the basic functionality, which you will then extend in Stage 2.

Functional Requirements

For Stage 1, your application should load in details of a menu and a list of existing orders, show a simple interface for processing new orders, and generate a summary report when exited.

1. A text file should be provided that shows the details of each item that a customer can order, including a brief description, the item's cost, the item's category, and a unique identifier.
   - There should be at least three categories: e.g. beverages, food items, other items
   - The identifier should be more than just a number. For example, it could include a string that indicates the item's category, e.g. a cheese sandwich might be FOOD255.
2. Another text file should be provided that contains a list of existing customer orders. Each order should include a timestamp, a unique identifier for the customer, and details of the item ordered. If a customer orders multiple items, these will appear as multiple orders with the same customer number.
3. These text files are read at the start of the application, and your code can assume that they are correctly formatted, e.g. the right number of commas in a CSV file. You should check that any identifiers are valid according to your rules.
4. Once the application is running, it can accept new orders from customers. These should be entered by the user using a simple GUI. The GUI should allow one or more items to be selected from a list of available items, and then display a bill. To make things more interesting (including your unit tests – see later) you should come up with some rules for discounts, e.g. get 20% off when you order a beverage and two food items.
5. Before the application exits, it should generate a report. As a minimum, this should list all the items in the menu, the number of times each item was ordered, and the total income for all orders.

Software Engineering Requirements

These are the software engineering requirements for Stage 1:

1. Your application should be implemented using Java.
2. Develop your program using **planned iterative development**. In this stage you should do all the design before writing the code. Decide on all the classes for this stage, their instance variables and methods. Try using CRC cards to help with class design, and use other diagrams where appropriate. Make a plan to divide the work between you, in such a way each person can work independently where possible. How will you test your code? Decide when and how often you need to meet or be in contact. How will you integrate your work?
3. Base all your decisions just on the requirements for this stage. Do **not** use agile development at this stage, and do **not** plan ahead to Stage 2. However, do take notes about the

development process and your experiences, since you will be asked to summarise this in your report.

4. Your program should read the data from the files at the start, and store it into appropriate data structures. When reading the files, don't think ahead to the GUI or reports; just store the data so that it can be accessed easily (e.g. is a list suitable? would a map be useful?). Then, write methods to analyse the data, which is likely to involve using more data structures. When making your decisions, imagine that you have a large number of customers, orders and menu items.
5. Use version control. Your group should set up a repository, and a link to this repository should be included in your report. Note that we will check the commit history, so make sure this reflects your individual contributions.
6. Use exceptions to catch errors in the data. Each group should decide what makes valid data (e.g., length, range, number of characters, etc.) If an error is found, just continue without that line of data. Provide suitable data to check that your program is working correctly, e.g. input files with some errors.
7. You should throw exceptions in the constructor of at least one class, to ensure that the objects of that class that you create are valid (e.g. menu item identifiers), and you should write at least one of your own exception classes.
8. Use JUnit to test some of your constructors and/or methods, particularly ones involving calculations, e.g. the method that applies your discount rules. You could try test-driven development for these methods. If you create a JUnit test for a method, you should test all the paths in the method, not just one.

Development Plan

One person in the group should submit this document, which gives details of your class diagram, chosen data structures and work plan, **before you start coding**. This should be done by the end of Thursday in Week 4. Submission is mandatory and your group will be penalised if it is not submitted on time. This document should be a **maximum of 3 pages** in length.

Group Report

The report will consist of several sections:

1. Names of group members and a summary (no more than one page) explaining who did which parts of the application and which parts of the report.
2. A link to your repository.
3. Does your program meet the specification, or are there some bits missing or bugs outstanding? Either provide a single sentence "This program meets the specification" or provide details of problems that you know about.
4. Suitable UML class diagram(s) showing the associations between the classes, and the contents of each class.
5. Explain which data structures you used, which classes they are used in, and why you chose them. (Some tools for creating UML diagrams hide the data structure of associated classes. Ensure the reader knows what these are.)

6. What decisions has your group made about the functionality of the program? (This is just information, not a technical report). Include the format of any identifiers and details about how discounts are applied.
7. Testing: What did you test using JUnit? Give details of which JUnit tests relate to which methods. (No need to print all the code, the marker will look at this electronically). Which types of exception did you use, in which method, and for what purpose?

Try to keep the writing concise. Your report should be **no longer than 12 pages**. Your development plan, group report and application for Stage 1 will be marked according to the following criteria:

Criteria	Weight	A (70-100%)	B (60-69%)	C (50-59%)	D (40-49%)	E/F (<40%)
Development plan and design decisions	25%	Clear, well thought out, well justified, and submitted by the deadline	Mostly clear, thought out and justified, submitted by the deadline	Some issues with clarity, planning or justification, but submitted by the deadline	Significant issues with planning or justification, or not submitted on time	No real indication of planning or thinking, or not submitted
Functionality	30%	Functional requirements have been met	Some requirements not fully met	A number of requirements are incomplete	Significant limitations in functionality	Very little achieved
Implementation and coding	25%	Good OOP design, clear modular well-commented code, clear class diagrams, appropriate use of version control	Generally good OOP design and coding with readable class diagrams, appropriate use of version control	Some issues with OOP design or coding, class diagrams lack clarity, limited use of version control	Significant issues with OOP design and coding, class diagrams poorly presented or absent, poor use of version control	Poor OOP design and coding, incomprehensible class diagrams, no use of versionl control
Exception handling and testing	20%	Effective use of exceptions, thorough unit testing and test data	Generally good use of exceptions and unit testing with sensible test data	Some issues with the use of exceptions, unit testing and test data	Significant limitations with exception handling and testing	No useful exception handling and testing

Submission

All reports and code should be submitted through Vision using the links provided in the Assessment section of the course. Late submissions will be marked according to the university's late submissions policy, i.e. a 30% deduction if submitted within 5 working days of the deadline and no mark after that. If you have mitigating circumstances, please submit the form available at: https://www.hw.ac.uk/students/studies/examinations/mitigating-circumstances.htm
Note that all submissions will be checked for plagiarism.

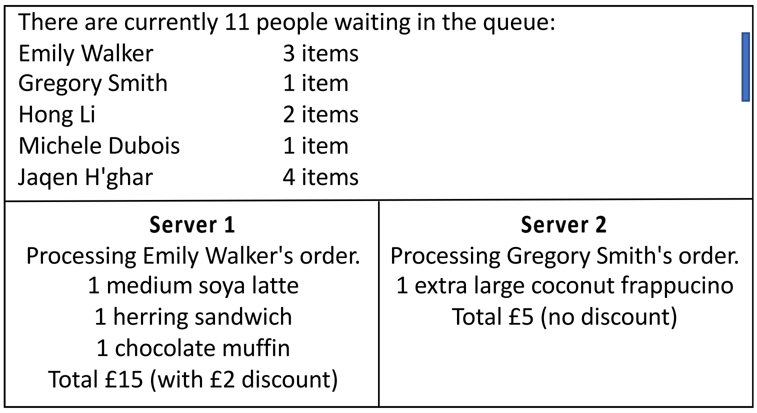
Stage 2

Stage 2 is designed to assess your command of thread-based programming and design patterns, and your understanding of agile development, which are all taught in the second half of the course. In this stage, your application will be extended to simulate customers queuing at the counter and having their orders processed by coffee shop staff. This will involve the addition of threads and design patterns, and the development of a suitable GUI to show the state of the simulation.

Core Functional Requirements

These are the **core** functional requirements for Stage 2. Also see the **extended** requirements section below.

1. As a minimum, the simulation should consist of multiple serving staff and a single queue of customers that are waiting to be served. When passengers arrive at the coffee shop, they join the back of the queue. When they reach the front of the queue, they will be processed by the next available member of serving staff.
2. As for Stage 1, the application will read in details of existing orders when it starts up. These orders should then be gradually added to a queue. When a member of serving staff becomes free, they will begin to process the order at the front of the queue. Once the order has been processed (which should take a finite amount of time), they move on to the next order in the queue.
3. Once the queue is empty, the coffee shop closes, a report is generated, and the program exits.
4. The GUI should show the customers and their orders waiting in the queue, and details of what each member of serving staff is currently doing (e.g. processing a particular order) throughout the simulation. One possible example is shown below. Make the timing slow enough that you can watch the displays changing.
5. A log should be kept which records events as they happen, e.g. a customer is added to the queue, an order is processed. This log should be written to a file when the application exits.



There are currently 11 people waiting in the queue:

Emily Walker	3 items
Gregory Smith	1 item
Hong Li	2 items
Michele Dubois	1 item
Jaqen H'ghar	4 items

Server 1	**Server 2**
Processing Emily Walker's order. 1 medium soya latte 1 herring sandwich 1 chocolate muffin Total £15 (with £2 discount)	Processing Gregory Smith's order. 1 extra large coconut frappucino Total £5 (no discount)

Software Engineering Requirements

These are the software engineering requirements for Stage 2:

1. You should experiment with **agile methods**. For your initial overall plan, simply decide how many iterations you will have and which features will be developed in each iteration. Then plan each iteration when you start work on it. Plan, design, develop and test each iteration without

considering features in the future iterations. You may need to refactor your code at the end of each iteration, before continuing.

2. Spend some of the time trying out agile techniques such as pair programming, stand-up meetings and time-boxing, and decide whether they are practical for you to use in your project or not. Note down your observations.
3. Use **threads** and ensure that they are synchronized where necessary and do not interfere with one another (see the learning materials in Part 2 of the course). For the core functional requirements, you should have one thread for each member of serving staff, and one thread to add customers to the queue. You are encouraged to add more threads to the application when you develop extensions and also consider more advanced scenarios like the producer-consumer model.
4. Java provides thread-safe versions of some of its collection classes. However, **you should not use these**, since this will limit your ability to demonstrate your knowledge of threading.
5. Your design should include **design patterns**. As a minimum, you should use the Singleton pattern to implement your log class, and the observer and MVC patterns in your GUI (see the learning materials in Part 2 of the course). You are free to use other design patterns.
6. Use packages appropriately.
7. Continue to use version control.
8. The final application should be exported to a **jar file**, from which the program can be run.

Extended Functional Requirements

You are also asked to **extend the core requirements**. Marks will be awarded based on the complexity of your extension(s) and the knowledge and understanding required to implement them. The following are some suggestions, in rough order of complexity. You are free to come up with your own, though you may want to check with your lecturer before starting work on them:

1. Allow the user to alter the speed of simulation using runtime controls.
2. Allow members of serving staff to be added and removed during the simulation. This could be done by the user (e.g. by pressing buttons) or automatically based on the queue length.
3. Allow customers to order online in advance. These orders could be handled using an additional queue that has priority over the main customer queue.
4. Add in baristas or kitchen staff. These could be represented by additional threads that interact with the serving staff threads either directly or through a queue of items to be prepared/cooked/assembled.

Group Report

The group report for stage 2 should include the following:

1. A brief description of the functionality that your system provides and what it does not do, i.e. does it meet the specification, are there some bits missing or bugs outstanding. How did you extend the core requirements?
2. UML class diagram(s) showing the associations between the classes, and the contents of each class – possibly not both in the one diagram.
3. Details about how you developed your program using agile processes. What features did you include in each iteration? Which agile techniques did your group use?
4. An explanation of how and where threads are used in your application.

5. An explanation of how and where design patterns are used in your application.
6. Sample screen shots.
7. A brief comparison of your development experiences in Stage 1 and Stage 2. Did plan-driven or agile development work better for your group? What problems did you encounter, and what might you do differently next time?

The technical sections of your report should be written for someone who has studied the material on this course but is not familiar with the coursework. You should include diagrams (including UML diagrams where appropriate) and snippets of code where relevant. Try to keep the writing concise. Your report should be **no longer than 15 pages**. Your group report and application for Stage 2 will be marked according to the following assessment criteria:

Criteria	Weight	A (70-100%)	B (60-69%)	C (50-59%)	D (40-49%)	E/F (<40%)
Design *(including use of threads and patterns)*	30%	Well designed with appropriate use of design patterns and ambitious use of threads	A generally good design with appropriate use of design patterns and threads	A reasonable design, but limited or incomplete use of threads or patterns	Significant design issues, limited use of threads or patterns	Poor design, little indication that threads and patterns were understood and/or used
Functionality *(including core requirements, extensions, user interface, and demo)*	30%	Core requirements complete, an effective user interface, one or more complex extensions, all features working in application	Core requirements complete, a usable interface, one or more non-trivial extensions, a generally working application	Some missing requirements, or limited extensions, or significant usability issues, or significant features not working	Limited functionality with missing core requirements or no extensions or major usability issues, major issues with the application	Poor functionality, very little achieved, application not working
Implementation *(including code quality, readability and comments)*	20%	Clear modular well-commented code, threads and patterns correctly implemented	Generally good coding, threads and patterns correctly implemented	Some issues with coding or with how threads and patterns are implemented	Significant issues with coding or with how threads and patterns are implemented	Poor coding, poor or absent thread and pattern implementations
Software documentation *(including technical writing, UML diagrams)*	10%	Precise, concise technical writing and UML diagrams that give a clear presentation of the developed software	Generally good technical writing with readable UML diagrams that gives a fairly clear overview of the developed	Technical writing lacks clarity or conciseness, UML diagrams lack clarity, it is not clear how some aspects work	Significant issues with technical writing, poor or absent UML diagrams, it is unclear how the software works	Poor writing, incomprehensible UML diagrams, it is very unclear how the software works
Development report *(including iterations, agile techniques, Stage 1/2 comparison)*	10%	A clear, concise and complete report of the application's development and the tools and techniques	An accurate report of the application's development and the tools and techniques that were used	A reasonable report of the application's development, with some lack of clarity, details or conciseness	A poor or limited report of the application's development	Little or no reporting of the application's development

Submission

See instructions for Stage 1 submission.