F21AS ADVANCED SOFTWARE ENGINEERING COURSEWORK

COURSEWORK OVERVIEW

This coursework contributes 50% of your mark for the course. The other 50% comes from an examination.

You are required to develop an application based on orders of food in a restaurant. You should work in groups of three, with the group collaborating over the design and implementation of the application.

The coursework has 2 sets of requirements

- functional requirements, which describe WHAT the application should do
- software engineering requirements, which describe HOW you should develop the application.

The basic application is developed in 2 stages, first concentrating on planning and testing, the second adding in threads and patterns. You write an individual report which gives technical details of some sections of the application, summarises the contribution of each team member, and discusses the development of the whole application.

More precise details are shown in the following sections.

Your mark for the group component will be the group mark scaled by your contribution to the group. This contribution will be assessed by the lecturer according to your own report and the reports of the other members of your team.

You are welcome to discuss the work with other students, but not to copy.

Please read the complete document before starting. In an individual report at the end of stage 2, you are expected to compare the methodologies that you used and discuss any resources used for both stages.

Weightings are shown here, deadlines are shown on Vision

Stage 1	Development Plan		Week 4 approximately
Stage 1	Basic Application	20%	Week 7
Stage 2	Threads & Patterns Presentation (tba) and report	30%	Week 12

GENERAL MARKING SCHEME

80 – 100%	Excellent. All elements completed. All very detailed and thorough.
70 – 80%	Very Good. Almost all elements completed. Mostly detailed and thorough.
60 - 70%	Good. Most elements completed, fairly detailed and thorough
50 – 60%	Adequate. Most elements done, some rather basic.
40 – 50%	Fairly Poor. Quite a bit omitted, elements done at a rather basic level
30 – 40%	Very poor, only a few items completed at a very basic level
<30%	Fail. Very little has been achieved.

THE FIRST STAGE OF THE APPLICATION.

FUNCTIONAL REQUIREMENTS STAGE 1

This application has been designed 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 orders for food in a restaurant.

This first stage concentrates on analysing the food orders, and on planning and testing. It should not involve much user input, just input and output using text files.

You have a list, in a text file, of orders for one evening in the restaurant. For each table, each dish ordered is listed separately, although there can be multiple orders per dish. For your program, you will need at least 30 orders, and only a few tables.

Input for the program should be :

- A text file containing details of the orders with table number – not all the orders for one table will necessarily be together, but there won't be duplicate orders from the same table for the same dish
    - E.g. table 1, dish Chicken Soup, quantity 3
    - There won't be any other orders for Chicken Soup from table 1
- Another text file containing details of the menu - dishes, category and prices.

You can make up your own orders, dishes, categories and prices. Examples are shown below.

Table ID	Name	Quantity
3	Roast Beef	2
1	Smoked Salmon	1
1	Carrot Soup	1
1	Roast Beef	2
3	Chocolate Cake	2
2	Carrot Soup	2
2	Roast Beef	2
3	Coffee and Mints	2

Dish Name	Price	Category
Smoked Salmon	6.00	Starter
Chicken Balerno	8.00	Main
Roast Beef	10.00	Main
Chocolate Cake	4.50	Dessert
Coffee and Mints	2.50	Drinks
Carrot Soup	3.50	Starter

It would be a good idea to allocate a sequence number to each order, for easy identification. See the section on data structures in the Software Engineering Requirements.

Output for the first stage is a single text file consisting of these reports:

- A menu, with the category as a heading, and dishes in alphabetical order.
- A list of the number of times each dish has been ordered overall.
- A list of the dishes on the menu that have not been ordered.
- The cost of all orders for each table – assume table is only used once per evening.
- A few other statistics of your choice.

It should also be possible to enter the table number and get a bill for that table, displayed in a window. This is the only GUI requirement. Otherwise, a graphical user interface is not required in Stage 1.

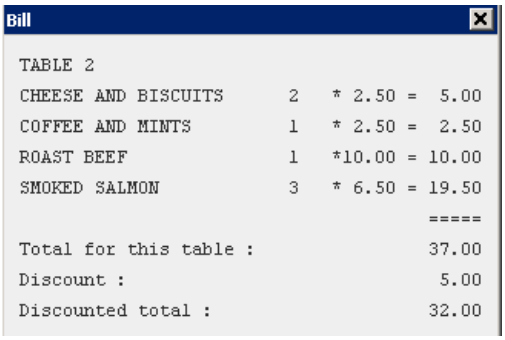
Examples of output and bills are shown below. Yours don't have to look exactly like this. It's in 2 columns just to save space in this document.

```
MENU
====

STARTER
  CARROT SOUP          3.50
  SMOKED SALMON        6.50

MAIN
  CHICKEN BALERNO      8.00
  ROAST BEEF          10.00

DESSERT
  CHEESE AND BISCUITS 2.50
  LEMON SORBET         4.00

DRINKS
  COFFEE AND MINTS     2.50
```

```
FREQUENCY REPORT
================
CARROT SOUP                2
CHEESE AND BISCUITS        3
COFFEE AND MINTS           4
LEMON SORBET               2
ROAST BEEF                 5
SMOKED SALMON              4

DISHES NOT ORDERED
==================
CHICKEN BALERNO
```

```
TABLE SUMMARY
=============
TABLE 1
CARROT SOUP          2   * 3.50 =  7.00
CHEESE AND BISCUIT   1   * 2.50 =  2.50
COFFEE AND MINTS     2   * 2.50 =  5.00
LEMON SORBET         1   * 4.00 =  4.00
ROAST BEEF           1   *10.00 = 10.00

=====
Total for this table :           28.50
Discount :                       10.00
Discounted total :               18.50

. . . . repeated for each table . . .
.
```

Followed by grand total



To summarise, your program should:

- Initialise a list of orders and the menu by reading from the text files. Then analyse the orders.
- Allow the user to enter a table number, and display the bill.
- Write the reports to a text file.

And you must follow the instructions in the 'Software Engineering Requirements Stage 1' in the next section.

SOFTWARE ENGINEERING REQUIREMENTS STAGE 1

Your software engineering requirements in this stage are:

1. Your initial program should contain at least the following classes: a class for a food order, representing one line in the input text file; a class for an iterable collection of orders; a class representing a menu item including price and category; a class for a collection of menu items; a manager class.

2. Choose the most appropriate data structures (try to use lists, maps and, sets), both for storing the initial data and for subsequently analysing the orders. When making your decisions, imagine that you have a large number of orders.

    a. When you read the text file of the orders, create a list that can be used as a base dataset in stages one and two (it's inefficient to re-read a file in a program). It would be a good idea to allocate a sequence number to each order, for easy identification. You may wish to put the orders into other data structures later.

    Refer to the 'concordance' example in the collections lecture for ideas on how to count and sum the orders, rather than using the frequency example in Foundations.

3. 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, use other diagrams where appropriate. Make a plan to divide the work into iterations, and divide this work between you. Decide when and how often you need to meet or be in contact. Members must do some work independently and then merge their work. How will you integrate your work?

    ***One person in the group should submit a document giving details of your workplan and chosen data structures to Vision BEFORE you start, by week 4 or 5. These decisions won't be marked but you will receive feedback on them **

4. Use some form of version control.

5. Error checking using exceptions should be done when reading from the text file. Each group should decide what makes valid data (e.g., length, range, number of characters etc).
    a. Some errors will be found as each line is read from the text file and split into its components.
    b. Throw exceptions in the constructor of at least one class, to ensure that the objects of that class that you create are valid.
    c. Write at least one of your own exception classes.
    d. Once the text files have been processed, only continue if all data was valid.

6. Provide suitable data to check that your program is working correctly e.g. input files with and without errors, data for all types of discount.

7. Use JUnit to test some of your constructors and/or methods, particularly ones involving calculations. 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. Similarly, checking that it fails when it should is not enough - you need to test when it works as well!

a. The calculation of the cost of the bill must be in a separate method so that you can test it easily. Make it a little complex, for example apply a discount for one table, and/or for a total which exceeds a given amount.

THE GROUP REPORT STAGE 1

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. 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.

3. Suitable UML class diagram(s) showing the associations between the classes, and the contents of each class.

4. 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.)

5. What decisions has your group made about the functionality of the program? (This is just information, not a technical report). Include:
    a. How discounts are calculated.

    b. Details of each of your own tests are NOT required, but we would like to know:

        i. What errors in the input files do you check for?

        ii. Which errors did you catch with exceptions? Give details of errors, classes and methods and type of exception here.

    c. What did you test using JUnit? Give details of which tests relate to which methods. (no need to print all the code, the marker will look at this electronically).

6. A development report focussing on the development of the project including

    a. Your development schedule (as submitted to Vision at the start), showing the breakdown into sections, allocation per person and group meetings.

    b. How successful was this approach, and what problems did you encounter? How much did your schedule vary from the original development plan? Did using your iterations work? It's ok to admit that your original plan was imperfect!

SUBMISSION STAGE 1

Your report should be handed in by the deadline shown on the front page of this document.

You are required to submit, in hard copy to the CS coursework box outside EM1.25, a report containing purple/mauve CS front sheet plus your group report as described above.

Additionally, create a **zip** file of your source code and input files. Submit via Vision. This source code will be inspected during marking. Please supply a file which is suitable for easy import into eclipse.

We will keep the reports, so please keep a copy for yourself.

MARKING SCHEME STAGE 1

Your application will be marked according to the implementation, description and usability of each aspect.

The weighting of each aspect is shown below (weight 20% of course, adds up to 70):

30	Overall functionality - reports to text file including choice of data structures. Includes good OOP design, clear modular well-commented code, clear class diagrams
20	Text file error checking including use of exceptions JUnit tests for errors and other methods.
20	Discussion on development plan

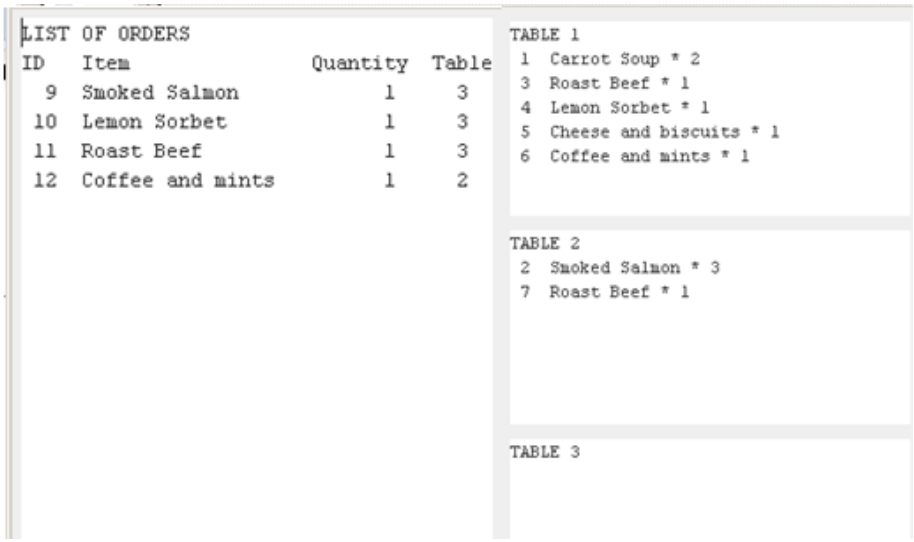
THE SECOND STAGE OF THE APPLICATION

In this part of the coursework, you will extend your application using threads and patterns, and include a graphical user interface.

FUNCTIONAL REQUIREMENTS STAGE TWO

You are required to develop a very simple simulation of food being ordered in a restaurant and then being ready for collection. When the simulation starts, the orders received are displayed 'in the kitchen', gradually appearing over time. One by one, these orders are displayed permanently at the destination table instead of in the kitchen.

See a sample program display below. Orders on the left are orders received but still in the kitchen, not yet delivered to the table. Your display doesn't have to look exactly like this.



At a certain point, the restaurant will not accept any new orders. All existing orders are produced and delivered to the tables.

The simulation ends once the last order has been delivered to the table. At that point, the user can request a bill for a single table, or request that a complete report (as in Stage 1) is written to a text file. This report should be based on orders used in the simulation, which may not be all the ones in the input text file.

There are lots of simplifications in this simulation. For example, we assume:

- One group of guests per table for the whole session.
- No-one wants a bill until the restaurant closes.
- Customers can't change their mind
- The simulation doesn't show dishes being cleared from a table
- Dishes are delivered in order of the order arriving in the kitchen.
- Etc.

Once the system is working as required above, extend the system in two ways of your choice. For example:

- Include an extra thread to show food orders moved to a hatch waiting for delivery to a table, then after a delay being delivered to a table from the hatch.
- Have 2 waiters delivering dishes to the tables (or orders to the kitchen). Don't make your waiters deliver dishes AND orders – that's quite complicated.
- Dishes ordered are generated randomly rather than being taken from a text file.
- Alter speed of simulation using runtime controls.
- In some way, simulate what goes on in a restaurant a little more closely. BUT, don't make this too complex. Have a sensible order in your input file.
- Anything else (maybe check with your lecturers about suitability).

More complex extensions get more marks!

And you must follow the instructions in the 'Software Engineering Requirements Stage 2' below.

SOFTWARE ENGINEERING REQUIREMENTS STAGE TWO

Continue to use version control.

Use threads

- One thread to simulate the orders which are sent to the kitchen (gradually take each order from your input file).
- One thread to simulate the orders being sent from kitchen to table (gradually remove each order from the kitchen).
- Java GUIs include their own built-in threads.

Your design should include patterns, in particular:

- Use the Singleton pattern in a Log class which is used to record every event in the simulation (i.e. start, order arrives in kitchen, order moves to table.) The Log file is finally written to a text file and should be easy to read.
- You could use the Singleton pattern to allocate a sequence number to each order.
- Use the Observer and/or MVC pattern in your GUI components.

As you develop your system for Stage 2, you should experiment with agile processing. Don't make an initial overall plan. Divide the features that you need to implement into several increments, and don't plan beyond the current increment. Do, however, plan each increment! Choose features to incorporate in an increment then plan, design and develop each increment without considering features in the future increments. You may need to refactor your code at the end of each increment, before continuing. Spend some of the time trying out Pair Programming. Consider other agile ideas such as stand-up meetings and time-boxing, and decide whether they are practical for you to use in your project or not.

The final application should be exported to a jar file, from which the program can be run.

GROUP REPORT STAGE TWO

The report will consist of a report consisting of several sections:

a) A brief description (< 1 page) 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 application (brief here, more detail below)?

b) UML class diagram(s) showing the associations between the classes, and the contents of each class – possibly not both in the one diagram

c) Explain how threads are used in your application.

d) Explain how patterns are used in your application.

e) Explain your extensions to the application.

f) Explain how you used version control and whether this was successful. Include the commit history if possible.

g) Write a development report. Include details about how you developed your program using agile processes? What features did you include in each iteration? Was this approach successful?

h) Sample screen shots

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 and snippets of code where relevant.

THE INDIVIDUAL REPORT COVERING STAGE 1 & 2

Your individual report should focus on the development of the project and contain details of the following:

- A comparison of the methodologies used in stage 1 and stage 2. How successful were these 2 approaches, and what problems did you encounter? Which did you prefer? You could include:
    - How successful was it to do all the design first in stage 1? And how did that compare to refactoring in stage 2?
    - How easy it was to divide up the work and combine each person's work together. If there were any problems, how were these resolved or how could they be resolved in the future.
    - What techniques and tools did you use in each stage? E.g. pair programming, CRC cards, tool for UML diagrams, version control. Were these successful?
    - Which parts of the application did you find were the most difficult to design and implement, and why was this?
    - Individual and group decisions- Did these turn out to be good or bad decisions? Did you agree with majority group decisions?

- What resources did you refer to when developing the project (lecture notes, books, websites, friends etc). Which of these resources were useful or not? E.g.
    - How did the course material fit into your existing knowledge?
    - What material was the most challenging?
    - How did you manage to resolve any problems with understanding the material?

- A one page summary of the contribution which each person made to the group and which sections of the system they worked on. You should include a weighting for each person in the group. E.g. Ann 25% Mary 35% Tim 40%

PRESENTATION WEEK 11

Arrange a group presentation with the lecturer by the end of week 11– this can occur before you have completed your reports.

At this presentation, you should demonstrate your program and explain how you have extended the basic specification. You should also talk about some technical details of your choice (for example, use of patterns, or how threads are used). It's quite likely that you will want to use a presentation program such as Powerpoint. Aim to talk for no more than 10 minutes, and be prepared to answer some questions. Each person should make a contribution to and participate in the presentation.

SUBMISSION STAGE TWO

Your group report and individual report should be handed in by the deadline shown on the front sheet.

You are required to submit, in hard copy to the CS coursework box outside EM1.25.

Your group report as described above, including the purple/mauve CS front sheet

Each person should submit an individual report as described above, including a purple/mauve CS front sheet

We will keep the reports, so please keep a copy for yourself.

Additionally

a) One person in each group should submit your source code and input files via Vision. Please either submit an zipped eclipse project or a zip file containing just the source and input files in a format which can be imported into eclipse easily (not rar or nested zips for example). This source code will be inspected during marking.

b) One person in each group should submit your jar file of the completed application. The marker should be able to run your program from the jar file.

MARKING SCHEME STAGE TWO

The weighting of each aspect is shown below (weight 30% of course, adds up to 120). The marks may be scaled according to your contribution within the group.

Group application

```
Functionality, including OOP design and clear well commented
code: (60)

5     see orders added to the list of kitchen orders over time
5     see orders removed from kitchen and delivered to table
5     Stopping new orders on request (at end of session)
5     GUI appearance
5     Creating a log file of events
5     Requesting a bill for a table
5     Creation of output text file, closing of program.
5     Jar file

20    extra features
```

Group report

```
Technical report – marks on content and quality of
explanation.  (30)

5     Use of patterns
5     Use of threads in basic stage 2 application
10    Explanation of extensions to application
      including use of patterns and threads
5     Discussion of version control
5     Clear class diagrams
```

Group presentation (10)

```
5     Clarity of explanations including demo
5     Presentation materials
```

Individual report (20)

```
15    Discussion of methodologies
5     Discussion on Resources
      Details of each person's contribution
```

Note – it is normal in a software engineering project to need to clarify details with the customer. I hope that this specification is fairly complete, but there are likely to be questions. Any clarifications to this specification will be posted on Vision under Assessment as we go along.