Module	**5CS019 Object Oriented Design and Programming**
Module Leader	Dr. John Kanyaru
Semester	1
Year	2021-22
Assessment	Design, Programming, and Testing, Code Documentation
% of Module Mark	80%
Due Date	This is the main assessed task in this module. Please aim to complete your work by 17/12/2021. The key to this task is to demonstrate your ability to write object oriented programs using Java, and the ability to use JUnit to test your classes. The source files provided are in two categories. First, the test cases. You are to write implementation code to pass the tests. Second, implemented code which needs completing. Please read internal comments as they do advise which methods to implement or test.
Deliverable - what to hand-in	Your submission will be a single .zip folder containing your source files, JUnit Tests, and a test coverage screen grab, and your UML class diagram. Please see the section on Submitting your work for further details.
Submit when?	By latest 12pm on submission date. Please submit via Canvas.
Pass mark	40% is required overall to pass the module.
Feedback	You will be provided with feedback on your work during scheduled workshops.
Plagiarism & Cheating	Cheating is any attempt to gain unfair advantage by dishonest means and includes **plagiarism** and **collusion.** Cheating is a serious offence. You must work individually as this is not a group coursework. **Cheating** is defined as any attempt by a candidate to gain unfair advantage in an assessment by dishonest means, and includes e.g. all breaches of examination room rules, impersonating another candidate, falsifying data, and obtaining an examination paper in advance of its authorized release. **Plagiarism** is defined as incorporating a significant amount of un-attributed direct quotation from, or un-attributed substantial paraphrasing of, the work of another.

Coursework Overview

This coursework contributes 80% of your mark for the module.

You are required to develop a simulation of workers processing orders. This is an individual coursework involving the design (UML Class Diagram), implementation (Java) and testing (JUnit) of an application.

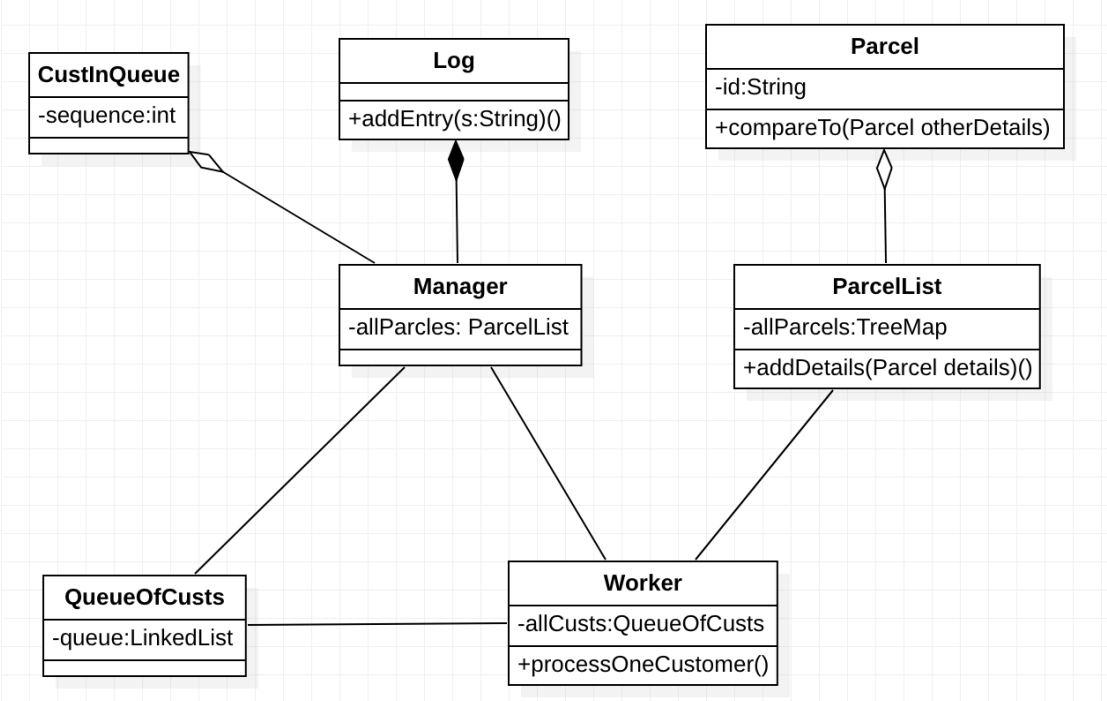
The coursework has 2 sets of requirements

- Functional requirements, which describe WHAT the application should do
- Software engineering requirements, which describe HOW you should develop the application.

More precise details are provided in the following sections.

Please read the complete document before starting.

To provide a common starting point, you are provided with a UML class diagram, two CSV files (Custs.csv, Parcels.csv), and Java source files with incomplete constructors and methods. Please read the comments in these source files so you understand what you ought to achieve. **The comments indicate the methods that you must write implementations for or test cases for.** You're to complete the UML class diagram with fields and methods as appropriate. Here is the initial UML class diagram. Again, you are required to complete it with appropriate setter and getter methods. Furthermore, you will be emailed a class name that you will use in place of Parcel class.



The **.java** source files that form the basis of your implementation, and test cases, as well as data files (**.csv** files**)** are provided in a zip folder on Canvas.

General Marking Scheme

80 – 100% Excellent. All elements completed. All tests and implementations very detailed and thorough, high test coverage.

70 – 80% Very Good. Almost all elements completed.
Mostly detailed and thorough, high test coverage.
60 - 70% Good. Most elements completed, fairly detailed and thorough, high test coverage
50 – 60% Adequate. Most elements done, some rather basic, test coverage medium.
40 – 50% Fair. Quite a bit omitted, elements done at a rather basic level, test coverage low.
<40% Fail. Very little has been achieved.

Functional Requirements

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. You are required to develop a simulation of workers processing orders. Details of the customer who placed the order (apart from their ID) are ignored for the purposes of this program. It is okay if you want to add further customer details, and a customer class.

Your warehouse only contains one type of goods (e.g. pens), although there may be different attributes such as size, colour or category. For the initial code provided, the item is a Parcel, **but you're to replace this with the class emailed to you**.

Warehouse items have a unique item ID, further details, a price, and how many are in the warehouse. For your program, details of about 10 items will be enough, e.g.

ID	Colour	Type	Price of One (£)	Quantity
401	Red	Ballpoint	2.00	50
402	Blue	Ballpoint	2.00	60
501	Red	Marker	1.50	10
502	Blue	Marker	1.50	0
503	Green	Marker	1.25	5

Each order has a unique order id, a customer id, and is for a quantity of one particular item. The price of the order is not determined until the order is processed. The table below is an example of orders :

Order Id	Customer ID	Item Id	Quantity
1	C101	402	10
2	C200	503	10
3	X59	402	3
4	C44	401	4
5	C101	401	1

You should decide what format IDs have. For the class provided to you, it is required that you include at least **two** other attributes in addition to item id and price of each item. The ID format should be something that you can write code to check (e.g. "3 digits in the range 400 – 999", or "a letter which is C or X followed by 2 or 3 digits", etc.). You will need to perform unit tests to verify the ID format you choose. Only do this for CustomerID.

The basic price of an order is the quantity multiplied by the price of one item. You should make your price calculation a little bit more complicated. For example, discounts for large orders, discounts for customers with a particular type of ID, discounts for items on special offer. Part of the assignment is to test your calculation, so it should not be too simple or too complicated! You should perform unit tests to verify that the price of an order is calculated correctly. This is in addition to the other unit tests described in the provided source code.

Input for the program is a text file containing details of the items in the warehouse, and another text file containing details of all the orders. These text files do not need to be updated.
The output should be a text file consisting of: a list of processed orders, including the cost; a list of orders that cannot be processed because there are not enough items in the warehouse; some counts and totals of your choice.

A graphical user interface is not required to produce this functionality up to this point.

To summarise, your program should:

- Initialise the collections of orders and items in the warehouse by reading from text files.
- Process each order one by one, updating the collections of items in the warehouse in memory, and write a report to a text file.

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

Software Engineering Requirements

Start with the initial program and UML diagram which is provided. Replace the Parcel class with the class emailed to you. Your software engineering tasks are to:

1. Choose the most appropriate data structures (chosen from lists, maps, sets). When making your decision, imagine that you are running a real system with a large number of orders and a large number of different types of items in your warehouse. You may alter some or all of the data structures in the provided code.

2. Decide what additional classes or methods you need, and what alterations you need to make to existing classes.

3. Use validation and exception handling in the constructor of at least one of the classes, to ensure that the objects of that class that you create are valid. For example, format of ID, and/or length or values of other parameters. This means that when you read the text file, you don't check the values for correctness – leave this to the constructor.

4. Write Javadoc comments describing each method you implement and each test case you perform. The comments should inform the reader what the method does, or what the test checks.

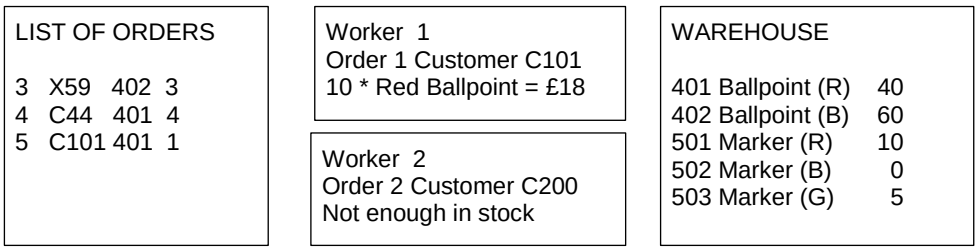
5. At the top of the class emailed to you, write Javadoc comments (maximum 250 words) of how you have ensured the confidentiality and security of your designs and implementation during the development process and submission. For confidentiality, consider how you have especially ensured exception handling is not revealing confidential information (e.g., of your file system). For security, consider how the use of encapsulated fields has enhanced the security of the objects in your application.

The initial step should be to understand the design, redesign it as necessary before writing the code or testing the code. Read internal comments so you know which methods to test, or which methods to implement. Add your own internal comments to describe what you have done.

Adding a GUI.

A GUI should show the state of the list of orders still to be processed, the quantity of items currently in the warehouse and the details of the order currently being processed by the

workers. Each panel should be updated (use appropriate event handlers) whenever the data which is displayed in that panel is altered. One possible example is shown below.



This is only an example, and you might choose a different way to present your data.

Marking Scheme

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

The weighting of each aspect is shown below:

Functionality:

```
20    Accurate order processing. Evident in source code
      (e.g., Price calculation, updating totals)
15    Contents of output text file
```

Class design and code

```
10    Good OOP design, clear modular well-commented code (Javadoc)
5     Javadoc on confidentiality, integrity and availability
10    Detailed UML class diagram – classes with logical relationships
and detail (methods and attributes)
```

JUnit Tests

```
15    JUnit tests of all the required methods
10    Validation and exception handling inside constructors (e.g., to
check appropriate length and format of customer ID or other ID.)
```

Adding a GUI

```
15     Use of GUI components to display the state of orders, workers,
and remaining stock in the warehouse.
```

Submitting your work

Before submitting your work, make sure you have done the following:

1. Create a new folder somewhere.

2. In that folder place:

   A screen grab of your eclemma coverage report. This should appear in your status window at the bottom of the eclipse screen. Your test coverage should be >90%

   If it doesn't click 'Window->Show View->Other->Java->Coverage'.

   Save this in a file called 'coverage.png'.

3. Copy your main package with sub-packages for implementation (e.g., itemsSrc) and tests (e.g., itemsTests) with all their contents for submission.

4. Ensure to provide your test coverage (as a png), and your final UML class diagram as a png.

5. zip your submission folder using the .zip format.

6. Submit your .zip folder.

For your information, you are provided with a summary of the test case classes and implementation source classes.

NB – 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 Canvas as we go along.