

IN-COURSE ASSESSMENT (ICA) SPECIFICATION

Module Title Object Oriented Programming (Semester 2)	Module Leader	Asher Rashid
	Module Code	CIS4037-N
	Submission Dates	Staggered across period from 21 st February 2025 to 7 th May 2025
Assessment Title Portfolio	Submission Method Online (Blackboard) <input checked="" type="checkbox"/> Middlesbrough Tower <input type="checkbox"/>	

Notes

- Please carefully follow the instructions given in this Assignment Specification.
- Each task in this ICA has a specific submission date and all tasks are separately submitted through Blackboard.
- Submissions made after the specified submission date, but not more than seven days after the specified submission date, will marked as late. Late submissions are restricted to a maximum mark equivalent to a pass mark.
- Any submission made more than seven days after the specified submission date, cannot be marked
- Extensions or MITS should be requested using the Extenuating Circumstances (EC) form available at https://www.tees.ac.uk/sections/stud/extenuating_circumstances.cfm
 - Short extensions requests, which allow up to seven additional days, are emailed to either Module Leader or Course Leader
 - Long Extension requests, which allow for more than seven days, are submitted to Course Leader or scdt-assessments@tees.ac.uk
 - MITS requests are submitted to SLSMitigatingCircumstances@tees.ac.uk
- If Extenuating Circumstances (extension or MITS) is granted, a fully completed and signed Extenuating Circumstances form must be emailed to scdt-assessments@tees.ac.uk or submitted to the School Reception.

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

Contents

1. Introduction	5
1.1 Overview.....	5
1.2 Tasks	5
1.3 Support	6
1.4 Programming Tasks Development Conditions.....	7
1.5 Referencing program code	7
1.6 Use of AI Tools	8
1.7 Avoiding Academic Misconduct	8
1.8 Marking.....	9
2. Task 01: Java Program [10%]	10
2.1 The Problem	10
2.2 Requirements	11
2.3 Marks.....	12
2.4 Submission	12
2.5 Notes	12
3. Task 02: Console Application [30%].....	13
3.1 Problem	13
3.2 Entity Class.....	13
3.3 Main Class: Start up	16
3.3 Main Class: Display fractions.....	17
3.4 Main Class: Set second fraction	17
3.5 Main Class: Add fractions	17
3.6 Main Class: Compare fractions.....	18
3.7 Main Class: Simplify fractions.....	18
3.8 Main class: Quit	18
3.9 Additional requirements	19
3.10 Marks.....	19
3.11 Submission	20
4. Task 03: Development Report [15%].....	21
4.1 Problem	21

4.2 Word Limit	22
4.3 Marks.....	22
4.4 Submission	23
5. Task 04: Event Driven Application [30%]	24
5.1 Problem	24
5.2 Teams and Roles	24
5.3 Phase 1: GUI	25
5.4 Phase 2: Data	27
5.4.1 Sample data	27
5.4.2 Entity Class	28
5.4.3 Loading data.....	30
5.4.4 JTable.....	31
5.5 Phase 3: Operations	32
5.5.1 Display Information.....	32
5.5.2 Display Image and caption	33
5.5.2 Check Stock	34
5.5.3 Sell Stock Operation.....	34
5.5.4 Update Stock Operation	36
5.5.4 Quit Operation	39
5.6 Phase 4: Data Merger.....	39
5.6.1 Stockton Scooters	39
5.6.2 Using new data.....	41
5.7 Phase 5: Quality Control [6 Marks]	41
5.7.1 Javadoc	41
5.7.2 Junit Testing	41
5.8 Additional Requirements.....	42
5.9 Marks.....	42
5.10 Submission	42
6. Task 5: Reflection [15%]	43
6.1 Problem	43
6.2 Sections.....	43
6.3 Content	43
6.4 Word Limit	45

6.5 Marks.....	45
6.5 Submission	46
Appendix A. Module Learning Outcomes	47
Appendix B. Key concepts.....	48
Appendix C. Academic Learning Support.....	49
C.1 Independent Learning	49
C.2 TU Library	49
Appendix D. Marking Criteria	50
Appendix E. Author, Presentation, Code Layout and Documentation	1
Appendix F. Test Plan.....	2
Appendix G. Guidance for written work	3
G.1. Structure	3
G.2. Presentation.....	3
G.3. Spelling and Grammar	4
G.4. Referencing	4
G.5. Cover Page and Table of Contents.....	4
G.6. Development Report Sections	4
Appendix H. Peer Review Form	7
Appendix I. Submission	9
Appendix J. Marking Rubrics	10
Appendix K. Extensions and Late Submission.	11

1. Introduction

1.1 Overview

The module assessment for Object-Oriented Programming is a portfolio of five tasks. Four of the tasks are individually attempted by every students. One of the tasks is a team based tasks, for which students will be allocated to a group.

Submission of assessment work for each task is staggered across the semester. To pass the module assessment, students need to score a minimum total of 50%.

1.2 Tasks

Three of the tasks are Java programming based and the remaining two tasks are essentially reporting based. Table 1 below summarises the assessment tasks.

Table 1: ICA assessment tasks

Task	Deliverable	Submission	Type	Marks
1	Java Program	Week 05	Individual	10 %
2	Console Application	Week 08	Individual	30 %
3	Development Report	Week 09	Individual	15 %
4	Event Driven Application	Week 12	Team	30 %
5	Reflection	Week 13	Individual	15 %

The emphasis for the programming tasks (1, 2 and 4) is for students to demonstrate:

- The use of the Java programming language.
- Applying modular programming techniques
- Validation and error handling
- The use of Java APIs.
- Good use of object-oriented programming techniques.
- Basic GUI creation using the Swing API.

- Selecting and implementing an appropriate design pattern to improve the overall architecture of the solution.
- Documentation
- Testing

The emphasis for the reporting tasks (3 and 5) is for students to individually demonstrate:

- Knowledge of Java language, concepts, constructs and principles
- Application of programming techniques.
- Research and Learning
- Ability to work individually or as part of a team.
- Communication skills

1.3 Support

This document specifies the problem and deliverable for each task.

Required programming knowledge will be delivered to students through a combination of taught sessions and independent learning. Learning materials will include Java applications, which utilise various aspects required by the programming tasks.

With regards to the reporting tasks, students are advised to take advantage of the academic skill services provided by the library. Further details are provided in Appendix A.

1.4 Programming Tasks Development Conditions

The work for Task 1 is to be developed using a text editor and JDK 17. For the Task1 submission to be marked, it must satisfy the following rules:

1. Opened and read using Debian Text Editor
2. Can be successfully compiled in a Debian 12 terminal using JDK 17
3. Can be successfully run in a Debian 12 terminal using Java 17

The work for Tasks 2 and 4, must be developed using Apache NetBeans 18 running JDK 17. For Task2 and Task4 submissions to be marked, they each need to satisfy the following rules:

1. Can be opened and files read, using Apache NetBeans 18
2. The project successfully clean builds in Apache NetBeans 18 running on JDK 17.
3. The project successfully runs in Apache NetBeans 18 running on JDK 17.

Any submission that does not adhere to the relevant rules above, can be scored zero marks.

1.5 Referencing program code

Students should deploy code, concepts and techniques for each programming task in accordance with what was taught in the module. If a student makes use of code not taught in the module, then comments should be provided which:

- References the source of the code used and or adapted
- Explains why the code was used (i.e. purpose) and what the code does (i.e. operation)

Failure to provide the above comments will result in zero marks being awarded for the relevant code and potentially the entire task.

1.6 Use of AI Tools

The use of (generative) AI software is not allowed for any task of this assessment. Failure to observe this rule will result in a referral for Academic Misconduct and a mark of zero being awarded

1.7 Avoiding Academic Misconduct

Academic misconduct is treated very seriously by the University, who define it as:

- **"Any activity or attempted activity which gives an unfair advantage to one or more students over their peers"**

(Taken from the Essential information for students on University regulations for academic year 2020/2021, p.1)

More information about activities, procedures and penalties relating to academic misconduct are listed in the TU Academic Misconduct Regulations available at:

- https://www.tees.ac.uk/docs/index.cfm?folder=Student%20regulations&name=Academic%20Regulations&folder_id=44

With regards to this module assessment students should take every care that their work is not the result of any of the following:

- Collusion
- Plagiarism
- Purchasing
- Use of Artificial Intelligence

All individual tasks (Tasks 1, 2, 3 and 5) should be the work of the student making the submission. Accordingly, students should not discuss or share their assessment work with other students. Students should not jointly work to solve the problems for

individual tasks. For the team-based task (Task 4) the above rules apply, but 'other students' will be any student(s) who is/are not a member of the team.

Further details are provided in Appendix B.

1.8 Marking

Tasks will be marked after submission deadline, with provisional marks and feedback provided to students through Blackboard. Tasks will be marked in accordance with a marking rubric, which students can find on the task submission page on Blackboard.

For Task 1, students need to submit the developed Java file and an explanatory text file. For Tasks 2 and 4 students need to submit the NetBeans Project folder. For Tasks 1,2 and 4, the assessment work should be archived into a zip file and this zip is what is submitted.

Always check before submission that the contents of the zip file can be safely extracted before you submit. If when marking, the contents of any submitted zip file cannot be extracted then the submission will be scored zero marks.

For Tasks 3 and 5 students are advised to submit a PDF version of their report / reflection. This ensures that the work looks the same on any computer.

Once any submission has been marked, **a revised attempt is not allowed**. If a student fails the overall module, they will then be eligible for a reassessment attempt in June and July.

2. Task 01: Java Program [10%]

2.1 The Problem

Create a java program that runs through a console. The program will calculate a result and outputs the result to the console. The input values for the calculation should be provided as command line arguments.

Students are free to select any calculation to base their application upon, but the calculation should be something expected to be known by GCSE / GED / IGCSE level students, for example:

Calculate the length BC using Cosine Rule

- <https://www.bbc.co.uk/bitesize/guides/zsgjxfr/revision/8>

Calculate the angle x using the sine rule

- <https://www.bbc.co.uk/bitesize/guides/zsgjxfr/revision/7>

Kinetic Energy

- <https://www.bbc.co.uk/bitesize/guides/zc6mv9q/revision/2#:~:text=The%20store%20of%20energy%20of,1%20%20m%20v%20>

Newton's Final Velocity

- <https://www.bbc.co.uk/bitesize/guides/zxd9w6f/revision/7>

Gravitational Potential Energy

- <https://www.bbc.co.uk/bitesize/guides/z46nf82/revision/7>

Further rules for selecting the calculation

1. Must not be calculation used in teaching materials
2. Must not be a calculation used in Task2 or Task 4
3. Must require at least two input values
 - Note at least three input values are required to score high marks

Details of the calculation and what inputs are to be provided (i.e. data type, range and sequence) should be provided in a text file, which is named "[readme.txt](#)".

2.2 Requirements

The application should only accept input through command line arguments. Any use of prompted or file input will result in no marks being awarded.

The name of the class should be no more than three words and describe the purpose of the calculation.

Validation should be provided as follows

- Only the exact number of command line arguments are provided by user
- Each command line argument underlying datatype is an integer
- Only positive non-zero values have been provided

Upon the application detecting a violation of a validation rule, the application should appropriately inform the user of the error and exit. The application should not display an exception message or trace.

If valid command line inputs are provided the application should display the result as part of a message. The start of the message should be a succinct description (i.e. label) for what the result is for. Next the calculation result is displayed to two decimal places. Finally, the message should end with either specific or general units of measurement.

Within the application there should be no use made of the following

- Package statement
- External or inner class
- External or inner interface
- User defined methods
- Lambda expressions

Students can make use of static class methods from Classes such as `Integer`, `Math`, etc. for example `Integer.parseInt()`. All coding should be in the `main` method and generally a comment be provided for every statement.

2.3 Marks

The available marks are shown in Table 2 below

Table 2: Task01 tasks

Component	Marks
Calculation choice and Text file instructions	2
Coding layout, naming style and comments	3
Code to obtain input through command line arguments	2
Program compiles and runs successfully without error	3
Total	10

Do note that Java compile marks will only be awarded for an genuine attempt at coding a solution for the task. A successful run means the program provides appropriate input validation and outputs correct result for the calculation.

2.4 Submission

The assessment work for Task1 should be submitted **before 4.00pm on Friday 21st February 2025** (Week 05).

2.5 Notes

In attempting this tasks students are being assessed for their ability to:

- Research and identify a suitable mathematical calculation
- Create a basic Java program using the fundamentals of the Java language
- Correctly make use of input through command line arguments
- Implement validation
- Code a calculation and format the result for output.
- Fully comment a program

3. Task 02: Console Application [30%]

3.1 Problem

Create a console application which has two classes

- The Main Class
- An Entity Class

The main class will provide the User interface, implement validation and make use of object instances of the entity class. The entity class will save data and provide methods to perform calculations on the stored data.

3.2 Entity Class

Students will design and code an Entity Class named **Fraction**, which will have fields to store the **numerator** and **denominator** of a fraction. The class will provide the following methods:

- A full constructor
- Accessor methods
- An **add** service method
- An **compare** service method
- A **simplify** service method
- A **toString()** method

The constructor should have parameters which are used to initialise the fields. The accessor methods simply return the value of the relevant field. There should be no mutator methods, (if mutator methods are included then marks will be lost).

The **add** service method accept a fraction object as a parameter and calculates and the result of adding the parameter fraction to the stored fraction. The addition

calculation should be implemented using basic fractional arithmetic and avoid any simplification.

For example, if the existing fraction object models $\frac{1}{2}$ and the parameter fraction object models $\frac{1}{4}$, the add calculation will be:

$$\frac{1}{2} + \frac{1}{4} = \frac{(1 * 4) + (1 * 2)}{2 * 4} = \frac{6}{8}$$

The method will return the result of the calculation using a two-element integer array, with the numerator at index 0 and the denominator at index 1. Thus for the above example the following array would be returned:

6	8
---	---

The **compare** service method accept a fraction object as a parameter and compares stored and parameter fraction values. The method returns an integer value to indicate the comparison result. Table 2, below specifies the possible return values.

Table 3: Compare method return values

Comparison result	Return value
Stored fraction is greater than parameter fraction	1
Stored fraction is equal to the parameter fraction	0
Stored fraction is less than parameter fraction	-1

Remember the calculation result should be returned as a new anonymous fraction object.

The **simplify** service method simplifies the modelled fraction into its simplest form. No parameters are accepted by the method. The method returns a three-element integer array. The first element contains the whole number of the simplified fraction, which could be zero. The second element contains the numerator of the simplified fraction. The third element contains the denominator of the simplified fraction.

Table 3, below provides example of simplifying modelled fractions.

Table 4: Fraction simplifying examples

Fraction	Return value		
$\frac{6}{3}$	2	0	0
$\frac{5}{3}$	1	2	3
$\frac{6}{8}$	0	3	4
$\frac{3}{8}$	0	3	8

The code to simplify a fraction should be entirely contained within the **simplify** method and only make use of:

- Standard variables
- Selection statements
- Iteration statements
- Arithmetical operations

The final service method requires the inherited **toString()** method to be overridden. The method, returns a string representation of the stored fraction. I.e. the numerator, followed by forward slash followed by denominator. For example "**7/8**", noting there should be no spaces between the numbers and the forward slash.

3.3 Main Class: Start up

The main class will declare two object instances of the Entity class at class-level. The first instance will be instantiated to model the fraction $\frac{7}{8}$ i.e. seven over eight. The second instance will be set to `null`.

When started the application should display a prominent main title followed by a subtitle for the menu which will have the following options:

- Display fractions
- Set second fraction
- Add fractions
- Compare fractions
- Simplify fractions
- Quit

The user should be prompted to select an option, which should be fully validated. If input is invalid, warn the user and redisplay the menu, but not the title. If the input is valid, match the input to the appropriate menu option and invoke a corresponding user defined method.

The first action of each user defined method should be to confirm which option was selected by the user, in the form of a subtitle. After a user defined method has been completed, except “Quit”, the menu should be redisplayed.

Students should ensure appropriate use of uppercase. Lowercase, line spacing, indentation and brackets are made use of in the presentation of content to the user throughout runtime.

3.3 Main Class: Display fractions

After the subtitle the operation will display as part of a suitable message the value of the first fraction. In displaying fractions, make use of the overridden **toString()** method in the Fraction class

If the second fraction is `null` then a suitable message should be displayed, otherwise its value should be displayed like the first fraction.

3.4 Main Class: Set second fraction

After the subtitle the operation uses prompted input to read the numerator and denominator for the second fraction. The input values should be validated as positive non-zero integers. If an input is invalid, the user should be warned, the operation cancelled, and the main menu redisplayed.

If both inputs are valid, instantiate the second fraction instance using the input values. Then confirm to the user the value of the second fraction as a fraction, making use of the overridden **toString()** method

3.5 Main Class: Add fractions

After the subtitle, the operation check if the second fraction has been set. If so it displays the fraction values and the add result in an equation format. For example:

- $1/2 + 1/4 = 6/8$

If the second fraction is `null`, warn the user.

3.6 Main Class: Compare fractions

After the subtitle the operation check if the second fraction has been set. If so it displays the result of comparing the first and second fraction. The message should display the fractions and indicate if first fraction is greater, less or equal to the second fraction. For example:

- `1/2 is greater than 1/4`

If the second fraction is `null`, warn the user.

3.7 Main Class: Simplify fractions

After the subtitle the operation will invoke the first fraction simplify method and output a message which displays both the original and simplified values. For example the original fraction is `9/6` then the message would be:

- `9/6 simplifies to 1 and 1/2`

If the integer component of the simplified fraction is zero, just display the fractional component

If the second fraction is `null` then a suitable message should be displayed, otherwise its value and the simplified value should be displayed like the first fraction.

3.8 Main class: Quit

After the user selects the quit option, the application should display a formal and polite goodbye message. After which the application should naturally terminate. Note:

- Simply stating `goodbye` is insufficient
- Do not force the application to terminate by using `System.exit()`.

3.9 Additional requirements

There should be only two java files within the project, each of which contain a single class, for which modular programming is applied. The only imported library should be the `Scanner` class.

Use the following names in your project:

- Project: `Fractions`
- Group ID: `ood`
- Package: `ood.fractions`
- Main Class: `Controller`
- Data Class: `Fraction`

3.10 Marks

The available marks are shown in Table 5 below

Table 5: Task02 marks

Component	Marks
Entity class has correct fields, constructor and accessors	3
Entity class has correct service methods	3
Correctly declared and instantiated object instances of the Entity class in the Main class	3
Correct NetBeans Project names, clean build and run	3
Command line menu works correctly (including quit)	3
Fraction operations work correctly	15
Total	30

Do note that NetBeans Project marks encompass correct project, package and class names, successful clean build and run without error. For a submission to be eligible for these NetBeans Project marks, the submission must represent a genuine attempt at developing a solution for the task.

3.11 Submission

The assessment work for Task3 should be submitted **before 4.00pm on Friday 14th March 2025** (Week 08). Students should archive their NetBeans Project into a zip file and submit the zip file.

If the NetBeans Project fails to open then the submission will be awarded zero marks. Accordingly students are advised to download what the submit (via Gradebook) and test that their project opens and runs in the Linux labs.

Submission of Java file(s) or a project from another development tool will also result in zero marks being awarded.

4. Task 03: Development Report [15%]

4.1 Problem

Student should write a development report on the development of their Task3 application.

The report should contain the following

- Cover Page
- Table of Contents
- Introduction
- Analysis
- Design
- Implementation
- Testing
- Conclusions
- Use of AI
- References

The reports structure follows the Waterfall development methodology. Student should make use of the URLs provided in Learning materials on Blackboard about methodologies.

Details of leaning more on reporting writing is at [Appendix C](#).

Information about writing the development report and what content is required for each section is at [Appendix G](#).

4.2 Word Limit

Students should be aiming for a word count in the range 1500 to 2000 words. Any report over 2000 words will be scored zero for presentation component. There is no penalty for word count being less than 1500 words, however it is likely than the any such report may not have sufficient content to pass.

The cover page, contents listing, footnotes, code listings, tabular data, headings, captions, citations (within round brackets), references section and any appendices are excluded from the word count.

4.3 Marks

The available marks are shown in Table 6 below

Table 6: Task04 marks

Component	Marks
Table of contents, Introduction, Conclusions and Referencing	3
Analysis	2¼
Design	2¼
Implementation	2¼
Testing	2¼
Presentation	3
Total	15

Remember any report over 2000 words will be awarded zero marks for the presentation component. However nay report significantly less than 1500 words, may not have sufficient content to be eligible for any presentation marks.

4.4 Submission

The assessment work for Task4 should be submitted **before 4.00pm on Friday 21st March 2025** (Week 09). Acceptable file formats are DOCX, DOC or PDF.

The filename of your report should be in the following format:

- *surname-userID-00D-Task03*

For example:

- *Rashid-u0018369-00D-Task03.pdf*

5. Task 04: Event Driven Application [30%]

5.1 Problem

Working in small teams, an Event-driven application is to be developed for “Boro Bikes”, a local supplier of children bicycles. The application will be used by the proprietor to manage the stock of bicycles, by entering when an item is sold or stock added. A Graphical User Interface (GUI) is required and data is to be loaded from a text file.

Various additional features are to be incorporated into the application, which will be added in a sequence of five distinct phases. It is recommended that a copy of the application is backed up once a phase has been completed.

5.2 Teams and Roles

Students will be allocated to a team of two or three, based on the time of their Lab Session. In each team there are three roles.

Table 7. Team Roles

Role	Duties
Leader	Delegates and monitors progress of work Organises meetings Supports Analyst and Tester
Analyst	Identifies requirements of a phase Translates requirements into a design
Tester	Identifies tests for a phase Monitors progress of testing

Students can opt to rotate roles or assigned fixed roles for the duration of Task04. In a team of two, students will share the team leader role and may rotate or fix the other two roles for Task04 duration.

Task04 is composed of a various phases in which the application will be incrementally developed. The coding within each phase is shared by all students in the team regardless of their assigned role.

Students should meet at least twice weekly to discuss development work done and to be done. At the first meeting a schedule should be agreed for the rotation of roles and future meetings. Brief notes should be kept of each meeting by the leader and shared with the team.

5.3 Phase 1: GUI

A standard java application should be created using the names in Table 3 below.

Table 8: Required NetBeans project naming

What	Name
NetBeans Project	BoroBikesSystem
Group ID	ood
Package	ood.borobikessystem
JFrame Class File	BoroBikesController.java

The Graphical user interface for the application makes use of a **JTable** and **TabbedPane**.

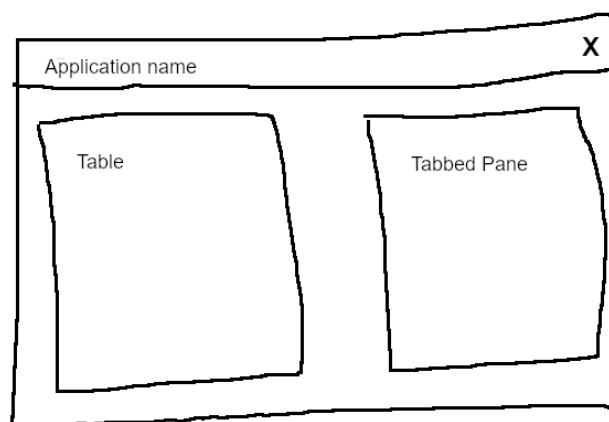


Figure2. Basic Design of GUI

The company want a elegant well balanced GUI. They do not want an user to be able to edit any displayed content or resize the GUI. They do not want the GUI or components within to behave inconsistently.

The **JTable** will display selected details for sale items, should not be editable by the user and only allow the user to select one row. The **TabbedPane** will contain three panels

1. Controls Panel
2. Information Panel
3. Image Panel

The controls panel will contain buttons to enable the user to:

- Sell an item, (the button will be initially disabled)
- Update stock level of an item, (the button will be initially disabled)
- Quit the application

In addition the controls panel will include a **JTextArea** will be displayed to keep a log which summarises sales and stock update made during the session.

The information panel will display the following information for the bike:

- Colour
- Age range (long version)
- Features

Students may use labels, text fields and or text area in any combination, to display the above information.

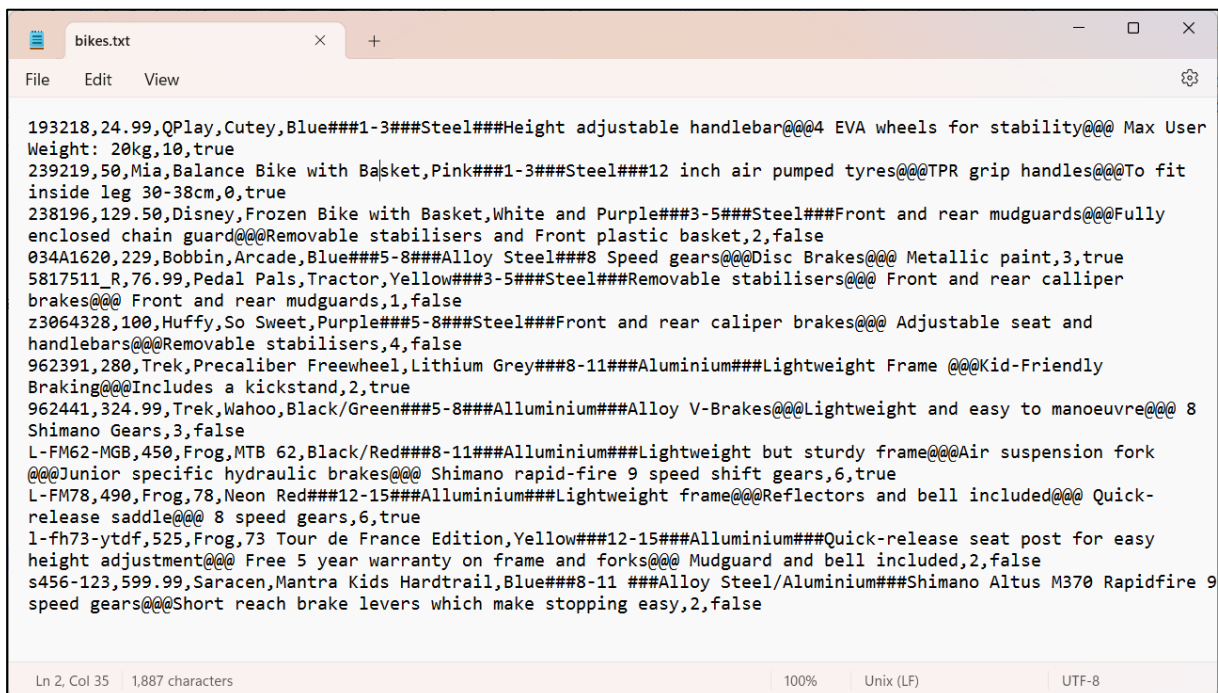
The image panel will be used to display an image of selected sale item in an label along with a caption in a separate label.

Students will need to decide on appropriate components, properties, sizing and alignment, as well as suitable use of panels. Text size of text in table and description and size of component for image may need to be updated when these are used in phases to follow. All components should be correctly named. There should be no redundant components, for example a label with no text which is not modified when program runs.

5.4 Phase 2: Data

5.4.1 Sample data

Boro Bikes have provided sample text data to be used by the application. The data is in text file **bikes.txt**, which is available from Blackboard.



```
193218,24.99,QPlay,Cutey,Blue###1-3###Steel###Height adjustable handlebar@@@4 EVA wheels for stability@@@ Max User
Weight: 20kg,10,true
239219,50,Mia,Balance Bike with Basket,Pink###1-3###Steel###12 inch air pumped tyres@@@TPR grip handles@@@To fit
inside leg 30-38cm,0,true
238196,129.50,Disney,Frozen Bike with Basket,White and Purple###3-5###Steel###Front and rear mudguards@@@Fully
enclosed chain guard@@@Removable stabilisers and Front plastic basket,2,false
034A1620,229,Bobbin,Arcade,Blue###5-8###Alloy Steel###8 Speed gears@@@Disc Brakes@@@ Metallic paint,3,true
5817511_R,76.99,Pedal Pals,Tractor,Yellow###3-5###Steel###Removable stabilisers@@@ Front and rear calliper
brakes@@@ Front and rear mudguards,1,false
z3064328,100,Huffy,So Sweet,Purple###5-8###Steel###Front and rear caliper brakes@@@ Adjustable seat and
handlebars@@@Removable stabilisers,4,false
962391,280,Trek,Precaliber Freewheel,Lithium Grey###8-11###Aluminium###Lightweight Frame @@@Kid-Friendly
Braking@@@Includes a kickstand,2,true
962441,324.99,Trek,Wahoo,Black/Green###5-8###Alluminium###Alloy V-Brakes@@@Lightweight and easy to manoeuvre@@@ 8
Shimano Gears,3,false
L-FM62-MGB,450,Frog,MTB 62,Black/Red###8-11###Alluminium###Lightweight but sturdy frame@@@Air suspension fork
@@@Junior specific hydraulic brakes@@@ Shimano rapid-fire 9 speed shift gears,6,true
L-FM78,490,Frog,78,Neon Red###12-15###Alluminium###Lightweight frame@@@Reflectors and bell included@@@ Quick-
release saddle@@@ 8 speed gears,6,true
l-fh73-ytdf,525,Frog,73 Tour de France Edition,Yellow###12-15###Alluminium###Quick-release seat post for easy
height adjustment@@@ Free 5 year warranty on frame and forks@@@ Mudguard and bell included,2,false
s456-123,599.99,Saracen,Mantra Kids Hardtrail,Blue###8-11 ###Alloy Steel/Aluminium###Shimano Altus M370 Rapidfire 9
speed gears@@@Short reach brake levers which make stopping easy,2,false
```

Figure 3. Contents of the input data file

Figure 3 above shows the contents of the bikes.txt data file. Student should note each record is a delimited string (using a comma as the delimiter), and some of the tokens are also delimited strings

5.4.2 Entity Class

Students will need to build a data class, named `Bike.java`, based on the sample data in the. The class should have following fields.

- **CODE** is a String value that uniquely identifies bike.
- **PRICE** is an double value that specifies the price in pounds and pence for each bike.
- **MAKE** is a String value which specifies the make of the bike.
- **MODEL** is a String value which specifies the model of the bike.
- **INFORMATION** is a String value which stores information about the bike.
- **STOCK** is an integer value which specifies the amount of the bike in stock.
- **ASSEMBLED** is an Boolean, which if true means the bike is sold fully assembled, false means some assembly is needed by the buyer.

Students should remember that in coding the data class, suitable java naming and house style should be applied. Assessors are required for each of the above fields, except for the **information** field. In terms of modifying values, the only mutator method should be for setting a new value for **stock**.

The data class, requires following service methods:

1. **Get Colour** method which parses the INFORMATION value to return the colour of the bike as a String value
2. **Get Short Age Range** method which parses the INFORMATION value to return the age range for the bike as stored.
3. **Get Long Age Range** method which parses the INFORMATION value to return the age range for the bike as a String message.
4. **Get Materials** method which parses the INFORMATION value to return the materials the bike is made of.
5. **Get Features** method which parses the INFORMATION value to return the features of the bike.
6. **Get Formatted Price** method which returns the PRICE value as a String formatted to two fixed decimal places and preceded by the £ symbol.

7. **Get Image Filename** method which returns the CODE value followed by `.jpg`.
8. **Calculate Total Sale** method is passed the quantity of the bike sold and calculates the total sale value of the purchase. The calculated value should use Get Formatted Price method, to format the calculated total sale value.

The value of the INFORMATION field is a delimited string value composed of four tokens, each separated by a triple hashtag pattern (###).

The first token represent the colour of the bike. The second token specifies the age range of the bike in years. This value is to be use in two formats as stated in Table 9 below.

Table 9: Short and long versions of age-range

Short version	Long version
1-3	Suitable for children aged one to three years old.
3-5	Suitable for children aged three to five years old.
5-8	Suitable for children aged five to eight years old.
8-11	Suitable for children aged eight to eleven years old.
12-15	Suitable for children aged twelve to fifteen years old

The short version is displayed in the JTable and the long version is to be displayed in the Features Panel.

The third token specifies the main materials the bike is made from. The fourth token specifies features of the bike

The value of the INFORMATION field for the first record in the data file is:

- `Blue###1-3###Steel###Height adjustable handlebar;4 EVA wheels for stability; Max User Weight: 20kg`

Table10 below shows the composition of the above value in terms of tokens.

Table10. Composition of example INFORMATION value

Token	Value
Colour	Blue
Age Range	1-3
Material	Steel
Features	Height adjustable handlebar@@@4 EVA wheels for stability@@@Max User Weight: 20kg

Note the value of the **features** token is also a delimited string, which uses a triple asperand (AT) pattern @@@ as the delimiter. When this token is processed each sub token should be trimmed and presented as a numbered item on a separate line, ending with a full stop. Table 11 below shows an example of presented value of features for the first record in the input file.

Table11. Composition of example features token

Features token value	Features presented value
Height adjustable handlebar@@@4 EVA wheels for stability@@@Max User Weight: 20kg	(1) Height adjustable handlebar. (2) 4 EVA wheels for stability. (3)Max User Weight: 20kg.

5.4.3 Loading data

In the main class, write a load data method which processes the input file, line by line. The tokens obtained from each line are used to create a Bike object which is then added to a suitable class-level data structure. The input file path and delimiter should be set as read-only class level variables.

The load data method is to be invoked from the main class constructor and should deploy appropriate validation and exception handling code.

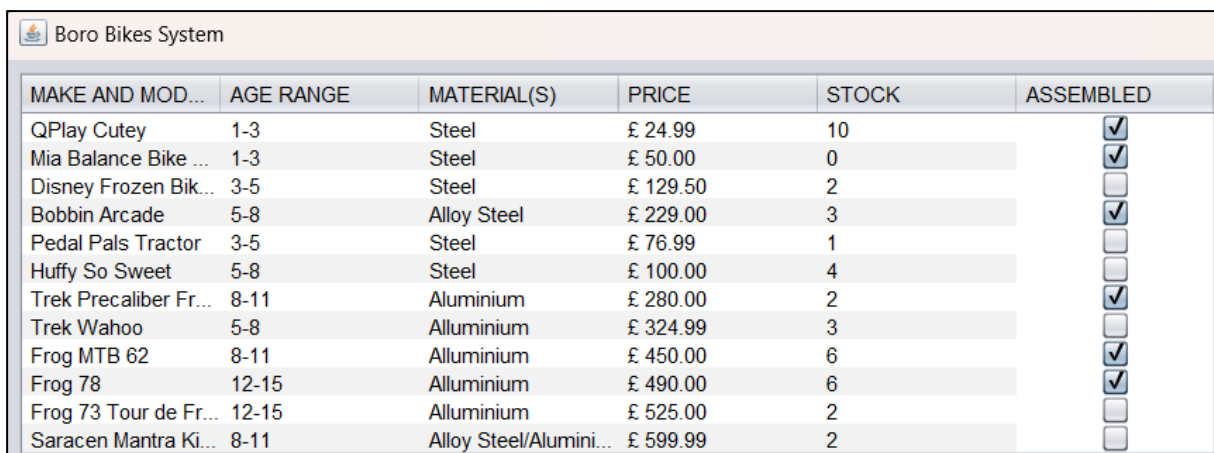
5.4.4 JTable

Add a new Java file named `BikeTableModel.java` which configures a table model in terms of number of columns and populating the underlying data structure of the table model. Students will need to decide which methods should be overridden.

The table should only display the following details for a Bike object:

- Make and Model (in one column, rather than separate columns)
- Age Range (short value)
- Material(s)
- Price (formatted to two decimal places and preceded by a £)
- Stock
- Assembled

Figure 4 depicts an example of how the table presents its data on startup



The screenshot shows a window titled "Boro Bikes System" containing a JTable. The table has six columns: "MAKE AND MOD...", "AGE RANGE", "MATERIAL(S)", "PRICE", "STOCK", and "ASSEMBLED". The "ASSEMBLED" column contains checkboxes. The data rows are as follows:

MAKE AND MOD...	AGE RANGE	MATERIAL(S)	PRICE	STOCK	ASSEMBLED
QPlay Cutey	1-3	Steel	£ 24.99	10	<input checked="" type="checkbox"/>
Mia Balance Bike ...	1-3	Steel	£ 50.00	0	<input checked="" type="checkbox"/>
Disney Frozen Bik...	3-5	Steel	£ 129.50	2	<input type="checkbox"/>
Bobbin Arcade	5-8	Alloy Steel	£ 229.00	3	<input checked="" type="checkbox"/>
Pedal Pals Tractor	3-5	Steel	£ 76.99	1	<input type="checkbox"/>
Huffy So Sweet	5-8	Steel	£ 100.00	4	<input type="checkbox"/>
Trek Precaliber Fr...	8-11	Aluminium	£ 280.00	2	<input checked="" type="checkbox"/>
Trek Wahoo	5-8	Alluminium	£ 324.99	3	<input type="checkbox"/>
Frog MTB 62	8-11	Alluminium	£ 450.00	6	<input checked="" type="checkbox"/>
Frog 78	12-15	Alluminium	£ 490.00	6	<input checked="" type="checkbox"/>
Frog 73 Tour de Fr...	12-15	Alluminium	£ 525.00	2	<input type="checkbox"/>
Saracen Mantra Ki...	8-11	Alloy Steel/Alumini...	£ 599.99	2	<input type="checkbox"/>

Figure 4. Presentation of table after application starts.

Student should note the ASSEMBLED column is set as a Boolean class, whereas all other columns are set as Strings. This should be coded within the Bike Table Model class.

In the main class write a method which initialises the table. This method should link the Table to an instance of the Bike Table Model class to the Table in the GUI. After which the column headers of the Table are set.

Students should implement a mouse click event handler for the table, which when invoked:

1. Sets a class level integer variable with the value of the selected row index of the table.
2. Displays information for the selected bike
3. Displays the image and caption for the bike
4. Checks the stock level for the selected bike

Note that **sell** and **update** operations are also dependent on a bike being selected from the table

5.5 Phase 3: Operations

5.5.1 Display Information

When the user selects a bike item from the table, the following details should be displayed in appropriate component(s) in the **information panel**:

- COLOUR
- LONG AGE RANGE
- FEATURES

Figure 5, below, shows an example of the information panel display after the first bike in the table was selected.

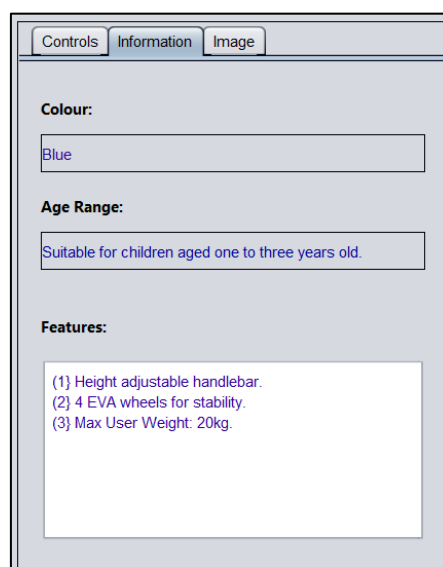


Figure 5. Example display for Information panel.

5.5.2 Display Image and caption

When the user selects a bike item form the table, the corresponding image and caption (i.e. Make and Model) should be displayed into the appropriate labels in the Image Panel

The photographs for all the garments are provided in a zip file, named **bike_images.zip**, which is available on Blackboard. A sub folder named **images** should be created in the NetBeans Project folder and the image files extracted into the sub folder, as depicted by Figure 6 below.

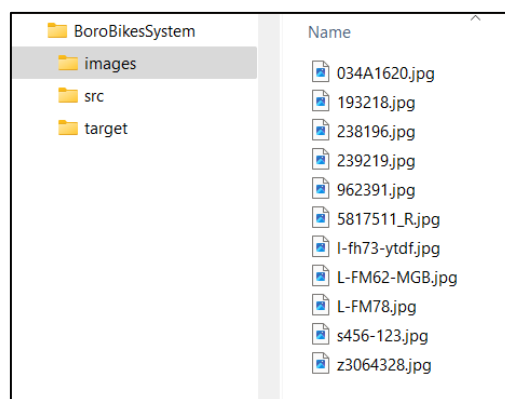


Figure 6. Contents of the images folder

In the main class code methods for the following:

- Load images when the application starts
- Display image and caption for an item selected in the Table. Figure 7, below, shows example display for the information panel after selecting the first bike.



Figure 7. Example display for Image panel

- If an image is not available, display the text “Image not available” in the image label and ensure that the caption label is to the empty string

Students are advised that at least one of the bikes does not have a corresponding image file.

5.5.2 Check Stock

Create two read only integer class level variables:

- Minimum Stock Level (value 0)
- Maximum Stock Level (value 10)

Create a method which checks the stock levels of a garment selected from the table.

- If the stock level is at the value of the Minimum Stock Level variable, then disable the **Sell** button, otherwise enable the component.
- If the stock level is at the value of the Maximum Stock Level variable, then disable the **Update** button, otherwise enable the component.

The Check Stock method should run anytime the user selects a table row, sells an item or orders stock for an item.

5.5.3 Sell Stock Operation

Modify the tool tip property for the **Sell** button, which displays the following message:

- Please select a bike, before processing a sale.

When an Item is selected, and the user clicks the **Sell** button:

1. The application will check if user has selected an item from the table, if not the user will be warned
2. If item has been selected, a pop-up dialog will request a quantity to be bought. An example of which is depicted in figure 8 below

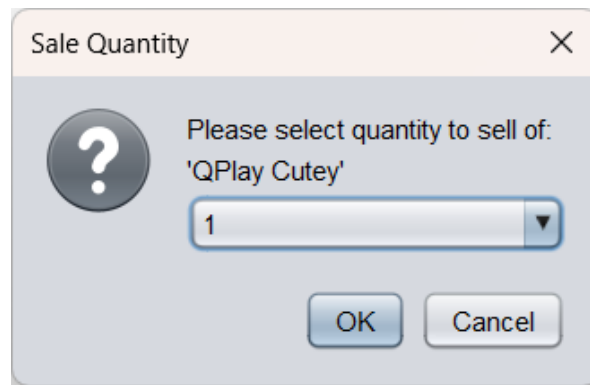


Figure 8. Input Dialog to get quantity to sell

3. If the user clicks cancel or closes the dialog, the operation should be quietly cancelled
4. The application will validate the input quantity as a positive non-zero integer, for which there is enough stock to purchase
 - Students can implement validation via an option list that displays quantities from minimum stock level to how much quantity is in stock
 - Alternatively, use a standard input dialog and providing additional code to validate the input and display suitable warning dialog if needed.
5. If there is sufficient stock:
 - Stock level in the bike data structure is amended
 - Stock column amended in the Table for the bike
 - Confirmation of sale displayed to user, as depicted in Figure 9 below

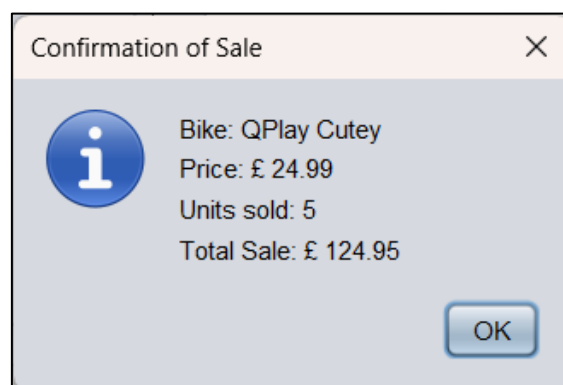


Figure 9: Confirmation of sale

- A summary of the sale is appended to the session log Text Area
 - Summary should start with the header **### BIKE SALE ###**
 - Then followed on the same line, but tab separated, by the **bike code** and **quantity sold**

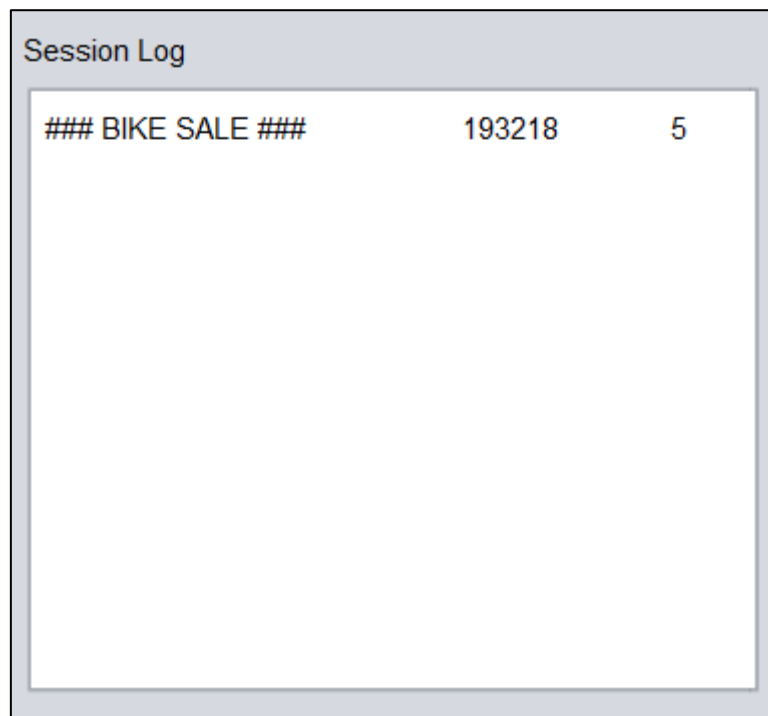


Figure 10: Logging of the sale

- Stock level is rechecked for the bike

5.5.4 Update Stock Operation

Modify the tool tip property for the **Update** button, which displays the following message:

- Please select a bike, before updating stock level.

When an Item is selected, and the user clicks the **Update** button:

1. The application will check if user has selected an item from the table, if not the user will be warned

2. Next the application will display a pop-up dialog which will request a quantity to update the stock by. An example of which is depicted in figure 11 below. Note the inputted quantity will be added to the existing stock value.

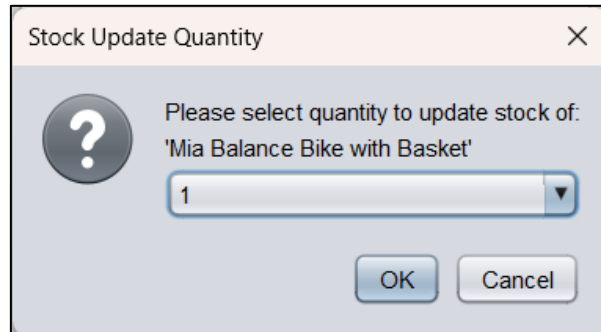


Figure 11: Dialog to get stock update quantity

3. If the user clicks cancel or closes the dialog, the operation should be quietly cancelled
4. User should only be allowed to input a quantity in the range one to a value which would make stock equal to maximum stock level.
 - Students can implement validation via an option list that displays integers in the valid range.
 - Alternatively, use a standard input dialog and providing additional code to validate the input and display suitable warning dialog if needed.
5. If the input is valid:
 - Stock level in the bike data structure is amended
 - Stock column amended in the Table for the bike
 - Confirmation of updating stock is displayed to the user, as depicted in Figure 12 below

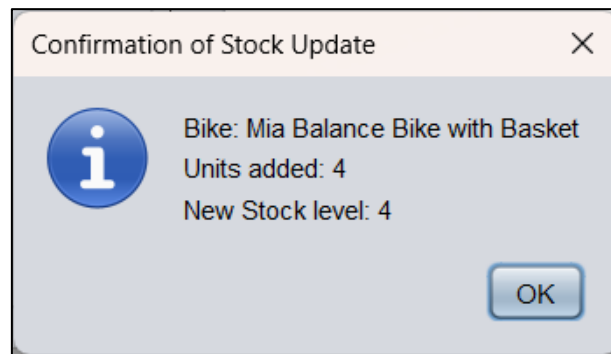


Figure 12: Confirmation of stock update

- A summary of the stock update is appended to the session log Text Area
 - Summary should start with the header **#BIKE STOCK UPDATE**
 - Then followed on the same line, but tab separated, by the **bike code** and **quantity added** to the stock

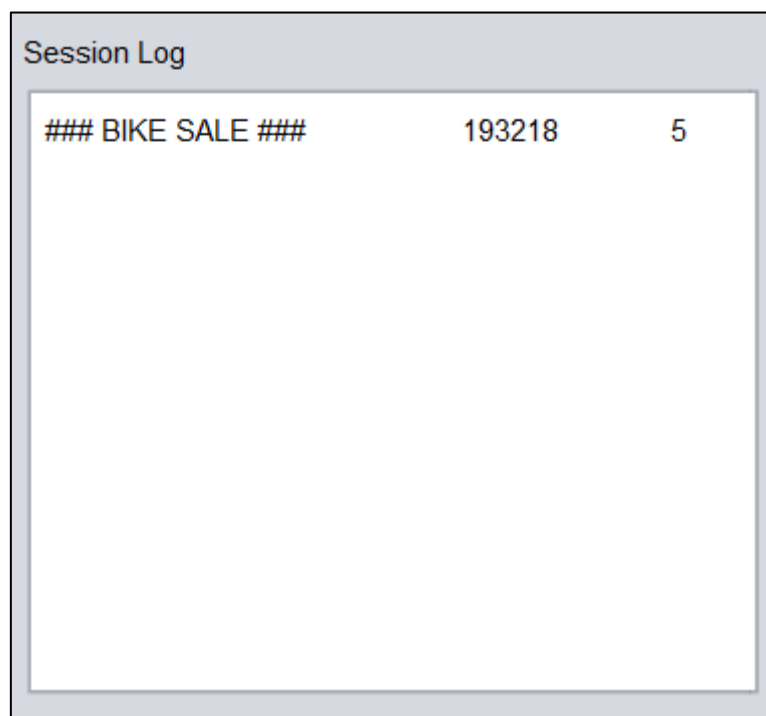


Figure 13: Stock update appended to session log

- Stock level is rechecked for the bike


5.5.4 Quit Operation

When the user opts to quit or close the application, a confirmation dialog should be displayed for the user to confirm their decision. If the decision is confirmed then the contents of the session log Text Area is written to an output file named `session.log`.

5.6 Phase 4: Data Merger

5.6.1 Stockton Scooters

As part of expanding its range Boro Bikes is to also stock garments from a mail-order business **Stockton Scooters**. A sample file data file name `scooters.txt` has been provided and is available to students on Blackboard.



```
662255,JD Bug,Classic Street 120,8-11,4500,Matt Black,10,Aluminim,true,Adjustable handlebar height;
Can be folded;Max rider weight up to 100kg
238198,Invert,FS2 Stunt Scooter,8-11,10950,Neo Chrome,0,Steel Alloy,true,Hi-Ten T bar with matching
Grips;Slick Stack IHC Headset;Rear Flex brake
541014,Chillafish,Scotti Glow Kids Tri-Scooter,3-5,6999,Blue/Black,2,Steel/Plastic,false,Attach and
remove the handlebar with no tools;Integrated footbrake;Fun LED lights - no batteries needed
880342,Skootz,Sonic Folding Inline Scooter,5-8,2500,Blue,4,Steel,false,EN71 tested;CE UKCA
compliant;Non-toxic plastic
322363c,Globber,Flow Element Lights,5-8,11000,Olive green,7,Aluminium,true,Deck has anti-slip grip
tape;Supports weight up to 100kg;Kickstand
880182b,Skootz,CoComelon Switch It Multi Scooter,3-5,2999,Orange/Yellow,9,Steel/Plastic,false,Comes
completed with 4 character plaques;Puncture proof EVA tyres;soft moulded grips for little hands
r100,Rampage,R100 Complete Stunt Scooter,8-11,9450,Black,1,Steel/Alloy,true,Composite Rear
Brake;holographic decals and inserts for a slick finish;Max Rider Weight: 80kg
```

Figure 14. Contents of scooters.txt data file.

The structure of the data file is different to that used by Boro Bikes. Stockton Scooters specifies the following tokens are on each line:

- **ID** – Saltburn Scooter stock code ID for the scooter
- **MANUFACTURER** – Manufacturer of the scooter
- **PRODUCT** – Model of the scooter

- **SIZE** – Age range in years for the scooter
- **PRICE** – Price of the scooter in pence
- **COLOUR** – Colour(s) of the scooter
- **STOCK** – Quantity of how much stock there is of the scooter
- **MATERIALS** – Materials scooter is constructed from
- **SELF ASSEMBLY** – Boolean value, which if true means buyer has to assemble the scooter
- **DESCRIPTION** – Description of features for the scooter

Stockton Scooters have also provided a data class named [Scooter.java](#) which is modelled in Table 12 as a class diagram.

Table 12. Class Diagram for Scooter Entity Class

Scooter
<ul style="list-style-type: none"> - id: String - manufacturer: String - product: String - size: String - price: int - colour: String - stock: int - materials: String - selfAssembly: boolean - description: String
<ul style="list-style-type: none"> + Scooter(String, String, String, String, int, String, int, String, Boolean, String) + getId(): String + getManufacturer(): String + getProduct(): String + getSize(): String + getPrice(): int + getColour(): String


```

+ getStock(): int
+ getSelfAssemble(): String
+ getDescription(): String
+ increaseStock(int): void
+ decreaseStock(int): void

```

Note separate methods are provided to change the stock level, meaning there are no mutators. The Entity class is classed as complete and should not be modified, but when deployed, students can include an appropriate package statement. Images have been provided in a zip file. These images should be extracted and copied into the existing **images** sub folder within the NetBeans Project folder.

5.6.2 Using new data

Deploy a suitable design pattern to allow the application to run with both Boro Bikes and Stockton Scooters data. This will require a copy of the Scooter entity class to be added to the project and one additional new class to be created. In addition, either a modification of the existing load data method or coding or a new load method will be also required.

5.7 Phase 5: Quality Control [6 Marks]

5.7.1 Javadoc

Write and generate Java doc for the following classes

- Boro Bikes Entity Class (Phase 2)
- Table Model Class (Phase 2)

Test that the appropriate Java doc can be viewed.

5.7.2 Junit Testing

For both valid and Invalid cases, write JUnit Test s() for following:

1. Phase2: Data Loading
2. Phase3: Check Stock

5.8 Additional Requirements

Some phases will be marked by testing in runtime. Therefore, students should ensure that the application clean builds and runs. Also ensure the project is efficient:

- Only code the class(es) stated in the phases
- Ensure there are no unused imported libraries
- Ensure there are no unused variables, objects, methods and classes.
- No use of functional interfaces, lambda expressions and or inner classes/methods

5.9 Marks

The available marks are shown in Table 11 below

Table 11: Task04 marks

Component	Marks
Phase 1: GUI	5
Phase 2: Data	5
Phase 3: Operations	10
Phase 4: Data Merger	5
Phase 5: Quality Control	5
Total	30

For a submission to be eligible for the marks, the submission must represent a genuine attempt at developing a solution for each phase of the task.

5.10 Submission

The assessment work for Task05 should be submitted **before 4.00pm on Wednesday 30th April 2025** (Week 12). A group submission link will be set on Blackboard, meaning only one team member needs to submit on behalf of their team.

6. Task 5: Reflection [15%]

6.1 Problem

Students are required to reflect on the role(s) they undertook for Task 04 and to reflectively compare their experience of individually developing Task 01 / 02 against the team approach to develop Task 04. Students can present their reflection in either in essay or report format. There is no need to deploy a reflection model or include any appendices.

6.2 Sections

Regardless of format the reflection should include the following

- Cover Page (Template available on Blackboard)
- Introduction
- Team Role(s) Reflection
- Comparing individual and team development
- Conclusions
- Use of AI
- References
- Peer Review

6.3 Content

The introduction should identify what will be reflected in the report and provide an overview of the activities that will be discussed.

In the role reflection, the reflective discussion of each introduced activity should address:

- What specific work was done for the activity for the role
- What was learnt in carrying out the work
- Problems or issues that arose

If a team decided to allocate fixed roles, then at least two activities should be discussed for the undertaken role. If a team rotated roles, then a different activity for two of the roles should be discussed.

For the comparison of individual and team tasks, identify an item of work comparable to each work example previously discussed for each of the Team roles. For these work items reflect whether overall you prefer working individually or as part of a team, and explain why. Note there is no right or wrong answer to this question.

In the conclusions describe one aspect of your performance in Task04 that you could improve, if you were to work on a team programming project in the future. Discuss what you could do to help facilitate the improvement and what would be the impact of the improvements for yourself, the team and project.

Students should provide relevant evidence to support their discussion, which can include:

- Emails, Messages, communication log
- Meeting schedules and minutes
- Analyses and Designs
- Test Cases
- Code (before and after changes)
- Runtime testing (before and after changes)

For use of AI follow the guidance at [Appendix G.6](#). Citations and References should be in the 'Cite Them Right' (Harvard) Reference Style. The Peer Review form is available at [Appendix H](#).

6.4 Word Limit

Students should be aiming for a word count in the range 1500 to 2000 words. Any submission over 2000 words will be scored zero for presentation component. There is no penalty for word count being less than 1500 words, however it is likely that any such submission may not have sufficient content to pass.

The cover page, contents listing, footnotes, code listings, tabular data, headings, captions, citations (within round brackets), references section and any appendices are excluded from the word count.

6.5 Marks

The available marks are shown in Table X below

Table X: Task05 marks

Component	Marks
Introduction and conclusions	3
Reflection of Team role(s) undertaken	3
Reflective comparison of individual and team development experience	3
Presentation	3
Peer Review	3
Total	15

Remember any reflection over 2000 words will be awarded zero marks for the presentation component. However any report significantly less than 1500 words, may not have sufficient content to be eligible for any presentation marks.

6.5 Submission

The assessment work for Task05 should be submitted **before 4.00pm on Wednesday 7th May 2025** (Week 13). Acceptable file formats are DOCX, DOC or PDF.

The filename of your reflection should be in the following format:

- *surname-userID-00D-Task05*

For example:

- *Rashid-u0018369-00D-Task05.pdf*

Students are reminded to always check what they have submitted, via Gradebook. If the submission cannot be read, then it will be awarded zero marks,

Appendix A. Module Learning Outcomes

The following tables provide the learning outcomes for 'Computer Technologies and Operating Systems' module.

Personal and Transferable Skills	
1.	Produce appropriate software documentation to communicate the programming concepts and techniques underpinning their solutions.
2.	Demonstrate a sound understanding of the Java API and the ability to make good use of the information provided therein when building solutions.
3.	Attain teamwork skills through participation in a team-based application development activity to solve a practical problem.

Research, Knowledge and Cognitive Skills	
4.	Analyse complex and/or incomplete problem specifications, justify the design and technical methodologies used, and recognise and argue for alternative approaches
5.	Demonstrate a systematic and critical understanding of Object-Oriented concepts, event handling and the development of Graphical User Interfaces.
6.	Select appropriate programming techniques, and Object-Oriented concepts, and critically evaluate their effectiveness in a specified scenario.

Professional Skills	
7.	Design and implement efficient Java solutions to unfamiliar problem specifications and critically evaluate the processes and facilities used in an autonomous manner.
8.	Participate effectively in a peer review process and reflect on own/team contribution.

Appendix B. Key concepts

In attempting the ICA tasks, most if not all of the following concepts will be utilised.

- Console and GUI applications
- Input and Output
- Validation
- Selection and Loops
- Methods
- Command Line Menu
- Object-oriented programming techniques
- GUI creation using the Swing API
- Event handling
- Arrays and Array Lists
- Reading from input file(s)
- Writing to output file(s)
- Error and Exception Handling
- Design Patterns
- Testing
- Documentation
- Teamwork
- Report & Reflection writing

The learning materials will introduce students to most of the above. However, students will be expected undertake research to gain specific knowledge to complete the tasks. In addition students can make use of the services provided by the University's Learning Hub with regards to teamwork and written communication skills.

The use of any code not taught in the module must be referenced. Within Java program files the reference is presented as a comment; containing the where code was obtained, e.g., a URL, along with an explanation of the copied code. Within written work, apply the 'Cite Them Right' variant of Harvard referencing style.

Appendix C. Academic Learning Support

C.1 Independent Learning

The module is a postgraduate module and as such students are expected to undertake substantial independent learning, for example:

- Reading lecture slides/notes before the lecture
- Watching any lecture recording in advance of the lecture
- Completing practical work before start of following practical
- Gaining academic skills through independent learning

C.2 TU Library

Students are expected to take up opportunities to gain academic skills in their own time. For example, staff will not be teaching students on how to write a reflection, instead students will need to make use of the library to gain reflective writing skills.

The library provides help through its Learning Hub:

- https://libguides.tees.ac.uk/learning_hub/home

At the Learning Hub there are three main sources of help:

1. Succeed Workshops: <https://libguides.tees.ac.uk/workshops>
2. Tutorials: https://libguides.tees.ac.uk/learning_hub/drop-ins
3. PASS: <https://libguides.tees.ac.uk/pass>

A calendar of workshops can be found at:

- <https://libguides.tees.ac.uk/c.php?g=89233&p=4990709>

Recommended Workshops to support Students on the first year of their MSc include: Studying at a UK University; Academic Writing; Avoiding Plagiarism; Using References; Report Writing; Reflective Writing.

Appendix D. Marking Criteria

Tasks 1, 2 and 4 (70%)				
Programming Practice	Distinction	Merit	Pass	Fail
	<p>The code is written to a very high and consistent standard, elegantly specified. The code is easy to read and understand by someone other than the original author.</p> <p>Comments are provided, clear descriptions of algorithmic implementations that are not immediately obvious by the code itself.</p> <p>The code is clearly demonstrating that it is very easy to extend and maintain.</p>	<p>The code is written to a good standard, although not always consistent.</p> <p>The code is easy to read and understand by someone other than the original author.</p> <p>Comments are provided, although it maybe inconsistent in clarity or usefulness.</p> <p>The code appears to be generally easy to extend or maintain with little effort.</p>	<p>It performs reasonably, although not consistently. The code is written to a reasonable standard, although the author may not have always considered that others would need to read and understand their code in the future.</p> <p>Some of the requirements have been met.</p> <p>Comments are present, although they may be inconsistent in their usefulness or appropriateness.</p> <p>It is not immediately clear, or it looks like quite a lot of effort would be required to extend and/or maintain the current code-base</p>	<p>Code is poorly written, lacking in in structure. There is little or no consideration that other programmers may need to be able to read and understand the code in the future.</p> <p>Few, if any, useful or appropriate comments are provided.</p> <p>The current code-base is messy and would require considerable effort to extend and/or maintain - probably a complete rewrite would be required.</p> <p>.</p>

Tasks 3 & 5 (30%)				
Development Report	Distinction	Merit	Pass	Fail
	<p><i>Full (but concise) explanation of the problem domain and the objectives of each task; clear insight into main challenges.</i></p> <p>Very good discussion of development work for the three applications. The content is clearly selective and relevant, in order to satisfy the word limit. Code snippets concisely discussed and appropriate issues raised.</p> <p>Appropriate improvement discussed in terms of what benefit it would bring. Relevant methods identified for changes. Appropriate code suggested to implement the change.</p> <p>Choice of figures and tables compliments the discussion and are all appropriately numbered and captioned.</p>	<p><i>Technical authorship presents a clear explanation of the problem; discussion/critique shows a good appreciation of main issues & objectives.</i></p> <p>Good discussion of development work for the three applications. The content is relevant, Code Blocks are discussed, and issues potentially raised.</p> <p>Appropriate improvement identified. Some discussion of benefits and changes Relevant classes identified for changes. Appropriate pseudo code suggested to implement the change.</p> <p>Good range of figures and tables to support the discussion with some attempt to number and caption.</p>	<p><i>Technical authorship presents a clear explanation of the problem; discussion/critique shows a good appreciation of main issues & objectives.</i></p> <p>Adequate discussion of development work for two or three of the applications. The content is more general than specific. Classes are discussed, and issues potentially identified.</p> <p>An improvement is identified, for which a basic discussion of benefits and or changes is provided. Suggestions to implement the change, lack detail.</p> <p>Limited range of figures and tables provided, lacking numbering and or captions.</p>	<p><i>Technical authorship is below the standard expected of a Masters degree.</i></p> <p>Unsatisfactory discussion of development work for any or all of the applications. The content is irrelevant or too general. Limited identification or required changes, with either no or inappropriate issues raised.</p> <p>Basic improvement identified, but not suitably discussed as to its benefits and or how it could be implemented</p> <p>If any figures and tables are provided, they do not support the discussion and may be provided in lieu of discussion, for example screenshot of code with no discussion.</p>

Appendix E. Author, Presentation, Code Layout and Documentation

For Tasks 1, 2 and 3, the student name and ID should be entered as the author comment before the start of the main class. For Tasks 4 and 6 a cover page should include the document title followed by student name and ID. For task 5 the name and ID of all students in the team should be provided as an author comment before start of the main class.

A console application gives limited options for user friendly presentation of output. Students should look to make use of Uppercase/Lowercase, Line gaps, Indents and Formatted Output to improve presentation. Students may apply simple borders, but do check the task details if this is necessary. Do not use Font Metrics, attempt to change text colour or import a third-party library to improve the output.

For GUI applications, the Frame should be sized so the user does not need to change the size when application runs. Also the application should not automatically resize when running. GUI components should be sized and sited to present a neat, balanced and minimalistic layout. For text choose a size to suit the component displaying it and its purpose. Avoid over use of large text sizes and or Bold. Only hyperlinks should be underlined.

Always place a comment at the start of a class to describe purpose and special feature of the class. Code layout should be as taught in class but if in doubt refer to the Google style guide. Remember to make use of code groups, e.g., importing libraries. Place a comment before the start of a code group and a line gap after end of code group. Students will need to use their own judgment about appropriate commenting and line spacing for code inside block statements, such as control structures.

Appendix F. Test Plan

For each programming task students should test that the task works as would be expected. For tasks 1, 2 and 3 the students should construct test plans for each new and or revised feature of the application. For Task 5 the tester for a phase produces the test plan.

The test plan will use the following format, where each row is considered an IEEE test case:

Test Number	Description	Values / Input	Expected outcome	Actual outcome

Once the test plan has been completed, students should use it to check the design and implementation. If any of the tests fail, the application will need updating and re-test (keep the original entry that failed in the plan along with the new entry for the re-test).

It is advisable to keep test plans, screenshots of testing and the code before or after any changes for use in the written tasks.

Appendix G. Guidance for written work

G.1. Structure

The body of a report or essay is to be structured into sections and each section split into sub sections. For essays section headers are optional, but advisable, there are no formal sub sections in an essay. For reports, both sections and sub sections start with a numbered header.

The Introduction is the first numbered section (i.e. 1. Introduction) and References is the last numbered section. Sections are to start on a new page.

Within a section the content is split into sub sections, for example usually the first sub section of the Introduction is to state the purpose of the document, (i.e. 1.1 Purpose). Sub sections do not start on a new page, unless there is insufficient space left on the page for the new sub section header and first paragraph to be fully displayed. Appendices are treated as sections but are lettered instead of being numbered.

G.2. Presentation

For written submissions, paragraph text should be no smaller than font size 12. Captions can be smaller than paragraph text size, but headings should be larger. For example: Bold and 18 for section headers and Bold and 14 for sub section headers.

All text, except captions, should be left aligned, use 1.5 line spacing. Ensure there is full line gap after each paragraph. Use a sans serif font face such as Aptos, Arial Calibri. Only underline text if the text is an active hyperlink.

Code listings, Figures, Tables and their captions are horizontally centred across the page. There should be no paragraph text to the left or right. Tables are captioned/numbered above the table. Code listings and Figures are captioned and numbered below. If presenting a screenshot of Code or a Table it is captioned and numbered as a figure. Numbering starts at 1 and is incremented sequentially regardless which section the item is in.

G.3. Spelling and Grammar

Ensure default proofing language is set to British English or English (UK).

G.4. Referencing

Citations and References should be in the 'Cite Them Right' (Harvard) Reference Style.

G.5. Cover Page and Table of Contents

A cover page will be provided on Blackboard for students to use or to adapt for both the report and reflection. The cover page is the first page of the document but should not display a page number.

Reports, but not essays, require a Table of Contents. The table of contents should start at the beginning of the second page of the report and lists the numbered sections and sub sections in the report along with their page numbers.

G.6. Development Report Sections

Introduction Section

This section should provide the purpose of the report and an overview of the problem for the task.

Analysis Section

Use this section, to explain the analysis undertaken on the problem and what requirements were identified in terms of code, behaviour and testing. For example retaining leading zero's in binary numbers is a key requirement. Any research done for the Analysis, can also be discussed.

Design Section

Use this section, to explain decisions made in designing a solution to the problem. It would be apt to present UML class diagrams of the Main and Data classes. Select some of the requirements and present/discuss code designs (e.g. Algorithms, Pseudo-code, Flow charts, etc.). In particular student should discuss their design to retain leading Zero's in binary numbers. Any research undertaken for the design, can also be discussed.

Implementation Section

Discuss the implementation of the data class diagram into code; only one bitwise service method needs to be discussed. Discuss how the designs for selected requirements were implemented. Any coding research, can also be discussed.

In discussing code avoid presenting/discussing more than a few lines of code at a time. If presenting code as text, caption and number the code as a Code Listing. For example "Code Listing 1". Body of Data Class; note this goes below the code. If presenting a screenshot of code, it is captioned and numbered as a figure.

Testing Section

Discuss testing of the application regarding selected requirements. Each test should have a (IEEE style) Test Cases, presented as a table. Tables are numbered and captioned above the table. Provide screenshots of input and output to support each test case.

Conclusions

Use this section to reflect on your experience of working on Task 3. If you had any issues discuss these and explain how they were or not overcome. If you had no issues, explain why this may be, for example describe previous programming experience

Use of AI

At present the University is developing an use of AI template. Until this template is available, students should under this section state

- No use of (generative) AI tools were used in researching, planning and writing this report and development the solution

References

Use this section to alphabetically list references for any citations in the previous sections. If you have no citations then state under the section header:

- No use of publish material or previous work was made use of in writing this report and developing the solution.

Appendix H. Peer Review Form

H.1 Overview

The last section of each student's Reflection document, a completed peer review form should be submitted. Simply copy and paste the below form into your reflection and complete it. Three marks will be awarded for correctly completed form

H.2 Teams Members

The student completing the form will be listed as Student A.

	Name	ID
A		
B		
C		

H.3 Fixed Team Role Peer Rating

If the team decided to allocate fixed teams roles for Task 04, then the following peer rating form should be used.

<i>Category</i>	A	B	C
Role Performance			
Coding work			
Support provided			
Total			

Students have only three points in total to award for each category. If a student believes each team member performed equally for a specific category, then award one point to each student for the category.

Otherwise students will need to award points for a category on the basis

- Two-points for who performed the best
- One point who was next
- Zero for the student who performed worst.

Only award whole (i.e. integer) points, i.e. no half points.

H.4 Rotated Team Role Peer Rating

If the team decided to rotate the teams roles for Task 04, then the following peer rating form should be used.

<i>Role</i>	A	B	C
Team Leader			
Designer			
Tester			
Total			

Students have only three points in total to allocate for each role. If a student believes each team member performed equally for a specific role, then award one point to each student for the role.

Otherwise students will need to award points for a role on the basis

- Two-points for who performed the best
- One point who was next
- Zero for the student who performed worst.

Only award whole (i.e. integer) points, i.e. no half points.

H.5 Confidentiality

As with all submitted assessment work, submitted peer rating forms are confidential and will not be shared with other students by staff.

Appendix I. Submission

All assessment work must be submitted, to Blackboard unless specified otherwise, before it is marked. When students test their programming assessment work, before submitting, do ensure the NetBeans Project has the correct main class set. Also do ensure the Project clean builds and executes using the specified version of JDK. Failure to do so will lose marks.

If a student's work been submitted but not yet marked, the student has the right to make a revised submission any time before the submission deadline.

When submitting applications to Blackboard, archive the Project folder into a zip file and submit the zip file. Always check before submission that the contents of the zip file can be safely extracted before you submit. If when marking, the contents of any submitted zip file cannot be extracted then the submission will be scored zero marks.

The most reliable method to archive a NetBeans project folder is:

1. Locate the NetBeans Project folder using a file management program ('[File Explorer](#)' in Windows, '[Files](#)' in Debian. or '[Finder](#)' in MacOS)
2. Right click on the NetBeans project folder
3. Select the Archive / Compress option
4. Ensure you select [zip](#) as the Archive / Compressed file type

For Task 01, either right click on the parent folder containing the Java and text files. Alternatively highlight both files and right click on any of the files.

Appendix J. Marking Rubrics

Details of how both programming and written tasks will be marked is made available on Blackboard through a rubric. Students should always refer to the rubric as a check list for determining the quality of their assessment work and make required amendments before submission.

Appendix K. Extensions and Late Submission.

Students who face circumstances beyond their control may also be entitled to apply for an extension. Such circumstances must be genuine, for example: illness, delayed visa, family emergency, 'Blackboard not being available', 'Logon problems', etc..

Evidence may be required to support an extension request. The front page of this document has details about extensions.

Do note that extensions will not be granted simply to 'give more time', 'work related reasons' or 'failure of students own IT equipment' i.e. there must be a genuine reason and evidence provided when applying.

If a student is not eligible for an extension, they can submit their work up to seven days after the specified submission date and time. Such submission are known as late submission and capped to a maximum of 50% of the available marks.