Northumbria University NEWCASTLE

Department of Computer & Information Sciences

# ASSESSMENT BRIEF

| | |
|---|---|
| **Module Title:** | Object Oriented Programming |
| **Module Code:** | PE7041 |
| **Academic Year / Semester:** | 2021-22 / TP2 |
| **Module Tutor / Email (all queries):** | Dr Mark C. Sinclair / mark.sinclair@northumbria.ac.uk |
| **% Weighting (to overall module):** | 100% |
| **Assessment Title:** | Sokoban |
| **Date of Handout to Students:** | 25th February 2022 |
| **Mechanism for Handout:** | Module Blackboard Site in Week 8 |
| **Deadline for Attempt Submission by Students:** | 27th March 2022, 23:59 BST |
| **Mechanism for Submission:** | Document upload to Module Blackboard Site |
| **Submission Format / Word Count** | Please upload your code as a zip file to one portal (labelled 'Code'), and your report as a single pdf file to the other (Turnitin) portal (labelled 'Report'). Your report should not exceed 3,000 words in length (not including tables and figures). |
| **Date by which Work, Feedback and Marks will be returned:** | 17th April 2022 |
| **Mechanism for return of Feedback and Marks:** | Mark and individual written feedback will be uploaded to the Module Site on Blackboard. For further queries please email module tutor. |

## Learning Outcomes

All the module learning objectives are covered by this assignment:

*Knowledge & Understanding:*

1. Demonstrate a systematic understanding of the principles, knowledge and skills required to design, implement, test and document programs written in an object oriented language.
2. Demonstrate a critical understanding of the essential principles and practices relating to object-oriented programming, including the need for standards, principles of quality, and appropriate software support.

*Intellectual / Professional Skills & Abilities:*

3. Critically evaluate the methods and conceptual tools used in developing solutions to programming problems.
4. Analyse, specify, design, implement and test a high-level solution to a programming problem using object oriented and general imperative programming language constructs, using appropriate documentation standards and software tools.

*Personal Values Attributes (Global / Cultural awareness, Ethics, Curiosity) (PVA):*

5. Effectively communicate development of a solution to a programming problem, including critical evaluation

The work is set in Week 08 of the module by which time you will have covered enough to make a start on the work.

## Module Learning Outcomes

### MSc Computer Science

If you are studying the MSc Computer Science, successful completion of this module will support you to evidence:

*Knowledge & Understanding:*

KU1. Demonstrate in-depth knowledge and critical understanding of the main areas of Computer Science, including the key areas of systems design, software development, security, databases and the web.

KU3. Apply knowledge and understanding of the software development life cycle.

*Intellectual / Professional Skills & Abilities:*

IPSA1. Appraise and judiciously apply a range of general computing principles, approaches, tools and methods.

IPSA3. Design and deploy general secure, standalone and web-based software applications.

*Personal Values Attributes (Global / Cultural awareness, Ethics, Curiosity) (PVA):*

PVA1. The ability to judiciously apply new skills and new knowledge as required.

PVA2. The ability to demonstrate creativity in problem solving and decision making.

PVA3. The ability to engage in critical self-appraisal of your own learning experience, personal strengths, limitations and performance.

## Scope

The assignment is an **individual** assignment.

You will be provided with some initial program source code as described in the section below.

You must extend the program to meet the specification below.

What follows is a detailed discussion of the program and how it is modelled and its functions, you will need to look very closely at the specification for the object-oriented (OO) program you are to create.

## Assignment

### Sokoban

This is a puzzle game based on a grid of cells. Each cell can be empty, a wall, or contain a box or the actor. The walls form a boundary, but there are usually some internal walls too. If there are N boxes (say, 6), then N non-wall tiles inside the boundary will be designated targets (say, 6). The aim of game is for the actor to push the boxes onto (any) of the targets. Only one box can be pushed at a time, and boxes cannot be pulled. If a box is pushed into certain cells, it will be impossible to push out ('wall stuck'), and if it is pushed against one or more other boxes, it may also become stuck. In addition, it is possible for the actor to block themselves into a limited area of the game (deadlocked).

There are ninety standard levels for the game, called screens, and these will be provided to you. A screen file uses a text format to represent the game: '#' for a wall; '$' for a box; '@' for the actor; '.' for a target; '+' for the actor on a target; and '*' for a box on a target. As an example, Fig.1 shows the first standard screen.

```
    #####
    #   #
    #$  #
  ###  $##
  #  $ $ #
### # ## #   ######
#   # ## #####  ..#
# $  $          ..#
##### ### #@##  ..#
    #     #########
    #######
```

Figure 1: Sokoban screen.1

Further discussion of Sokoban can be found on Wikipedia (https://en.wikipedia.org/wiki/Sokoban).

You have been supplied with code that implements Sokoban, including a random game player, and also a partially-implemented text-based user interface. The ninety standard screens are also included. The seven model classes are `Sokoban`, `Cell`, `Occupant`, `Actor`, `Box`, `Wall` and `Direction` (an enumeration); the text UI is `SokobanUI`; the two player classes are `Player` (an interface) and `RandomPlayer`; plus, a problem-specific exception class, `SokobanException`. The classes are illustrated in a UML class diagram (Figure 2).

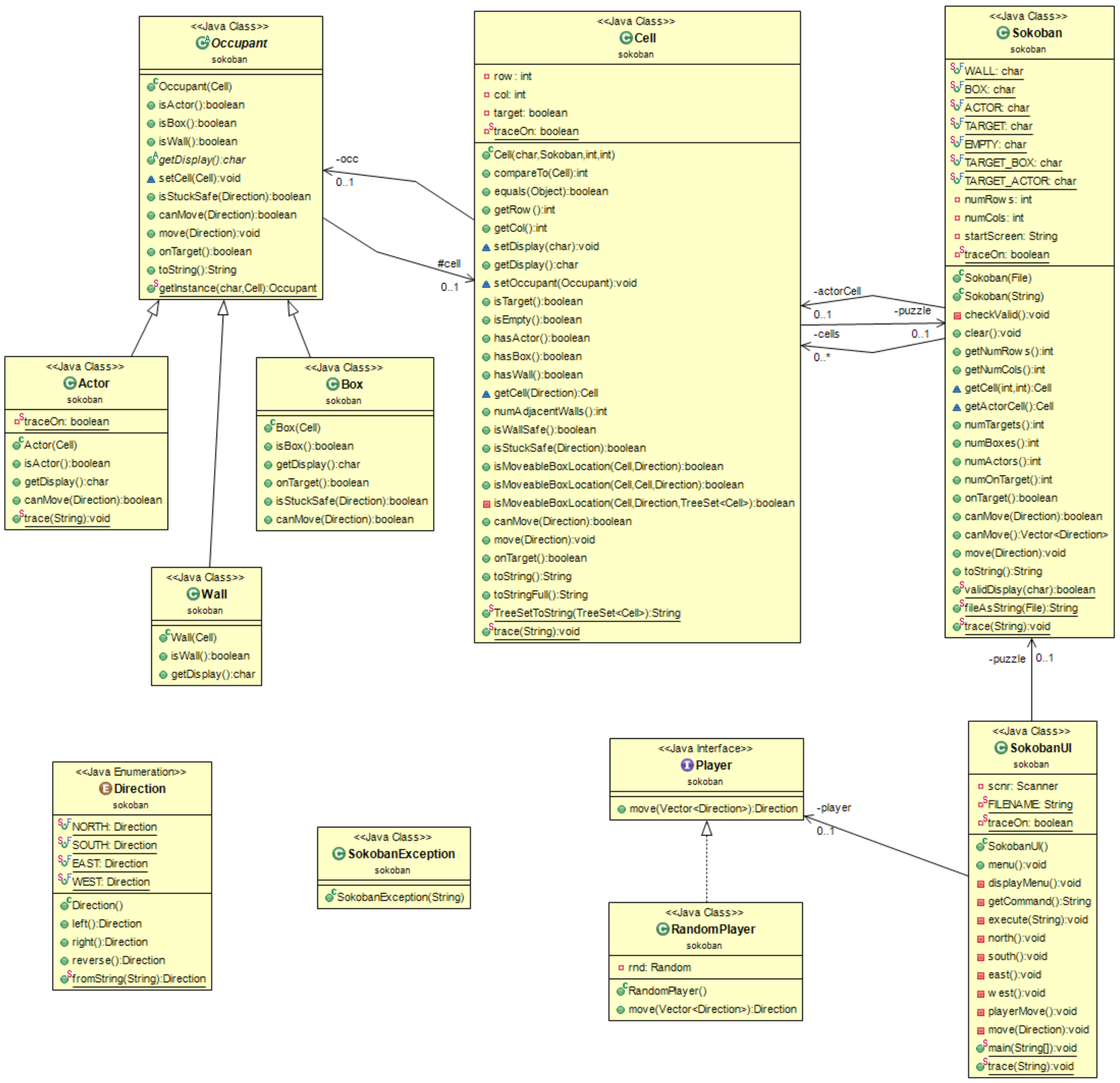

Figure 2: Class diagram for supplied code

## The Tasks

The tasks you are to carry out are as follows:

To complete the implementation of the text-based user interface to include the (as yet unimplemented) clear (i.e., reset the Sokoban game), undo (a single user move), save (the state of the game to a file) and load (the state of the game from file) methods. You should only need to make changes to the main user interface class (`SokobanUI`) to achieve this.

To design and implement a graphical user interface (GUI) for the Sokoban model code that allows the user to make moves, undo user entries, clear (reset) the game, save to, and load from file. At all times, the user should be protected from making illegal moves (the model code guards against moves caused by getting stuck due to walls or other boxes), and so e.g., provided with feedback on the possible moves available to the actor. The appearance of the interface is up to you, but an example is given in Figures 3 and 4. You should keep the user informed through the use of e.g., a status box, and may also consider the use of dialogue boxes where appropriate (such as file location selection).

Develop a test plan and test the functionality of your classes.

Write a report including design, testing and reflection (see below). The report should not exceed 3000 words.

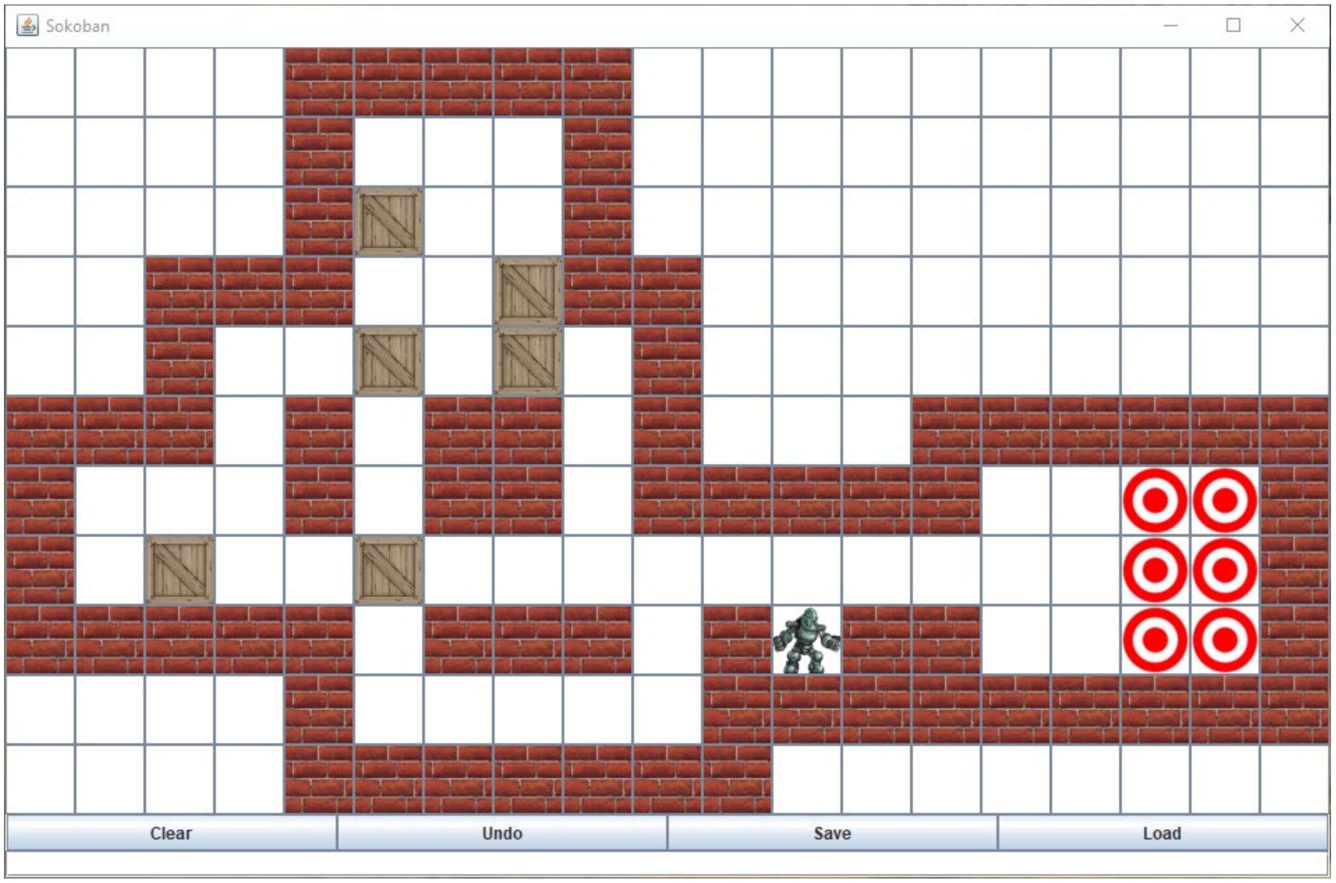

Figure 3: A possible GUI design (at start of game)

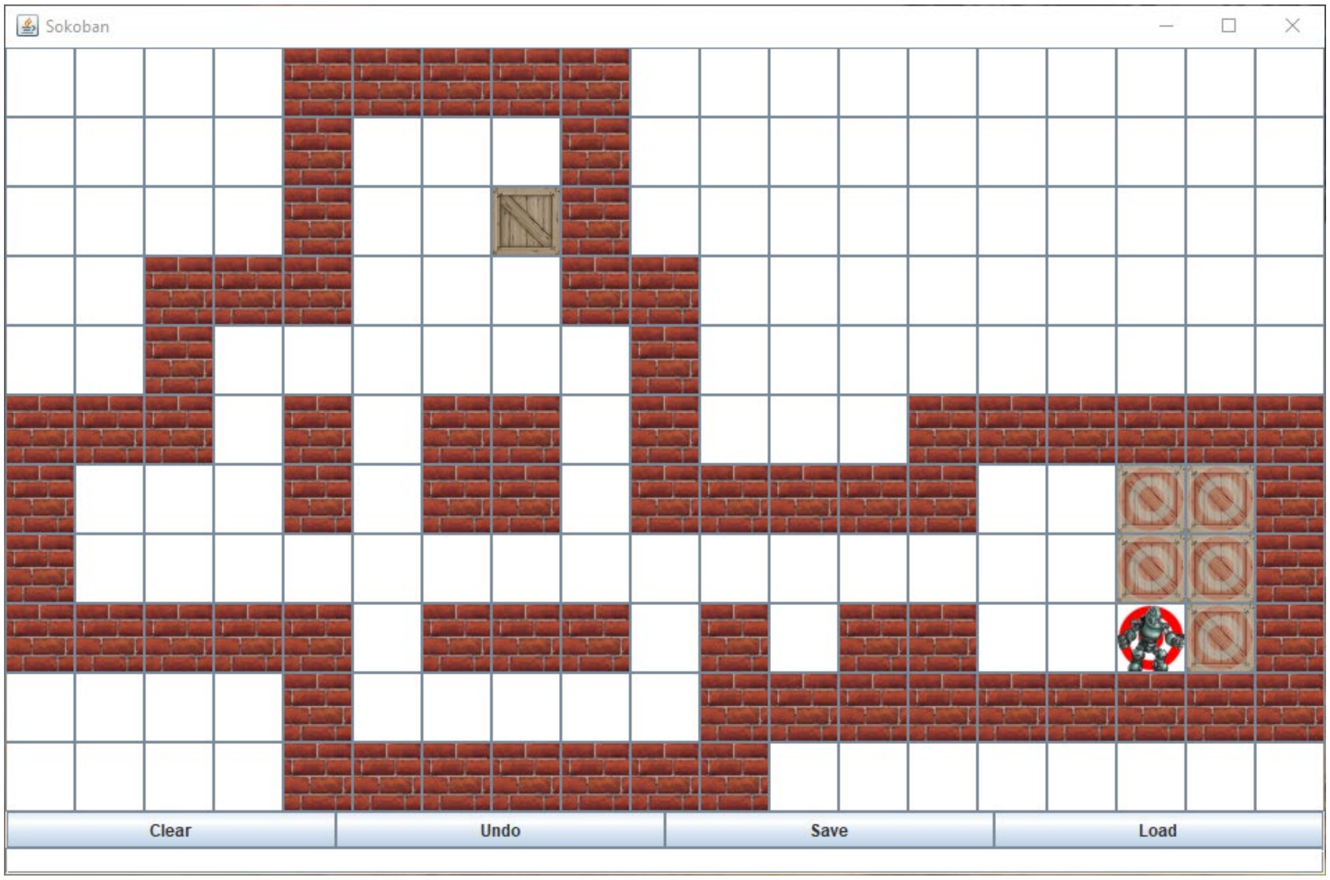

Figure 4: A possible GUI design (five boxes on target)

## Deliverables

### General Points

The development of your code is based on the object-oriented model, and you **must** make use of classes in developing your system. Each will have its own methods and properties though they could inherit from other classes.

All interactions must be made via just the text-based UI or just the GUI alone. You may design the GUI as you see fit; see the example depicted above.

You might consider writing custom dialogue(s) to input the user's preference and other information. This will require a little investigation on your part.

The program should consist of a number of classes each with well-defined functionality. There may be a driver class to set things going; GUI classes to provide the user interface; there may be additional classes for file handling.

The code you produce must adhere to the published course coding standards. Marks are awarded for code quality and appropriate defensive programming. Your code should be well commented and include Javadoc comments.

You will be expected to test parts of your program against a suitable set of situations. In your report you should describe your testing plan and results; also include your test results as screen shots as evidence.

## Design

You must produce design documentation. This will include class diagram(s) for the system, a short explanation as to the general purpose of each of the classes you have produced and a justification for any design decisions you have made (including options you discarded and why).

## Implementation

You must provide complete Java source code for your program. The code must adhere to the object-oriented style standards as defined for the module. Do not use a GUI editor to generate your code, but develop it by hand using the Java Swing API.

## Testing

You are expected to test your code using the strategies studied during the module. It is not necessary to test the supplied code, although you may choose to do so.

The testing section of your documentation should indicate the approach you have taken to verifying and validating your system. Just as you should not convey the design of your system by presenting the code or even listing the classes, you should not merely list the tests performed. Rather, discuss how tests were selected, why they are sufficient, why a reader should believe that no important tests were omitted, and why the reader should believe that the system will really operate as desired when in use.

You may not be able to create JUnit tests for all aspects of your classes, particularly the graphical user interface classes. It is sufficient to carry out well planned and documented manual testing.

Strategy: An explanation of the overall strategy for testing: black box and/or white box, integration, kinds of test beds or test drivers used, sources of test data, test suites. You might want to use different techniques (or combinations of techniques) in different parts of the program. In each case, justify your decisions.

Test Data: A set of tables showing the test data you used for each class, etc. The format of the test documentation should be as follows: for each test case in the tables,

- a unique test ID
- a brief description of the purpose of the test
- the pre-conditions for running the test
- the test data
- the expected result

### Reflection

You must provide a final critical evaluation of your work. The reflection section is where you can generalise from specific failures or successes to rules that you or others can use in future software development. What surprised you most? What do you wish you knew when you started? How could you have avoided problems that you encountered during development?

Evaluation: What you regard as the successes and failures of the development: unresolved design problems, performance problems, etc. Identify which features of your design are the important ones. Point out design or implementation techniques that you are particularly proud of. Discuss what mistakes you made in your design, and the problems that they caused.

Lessons: What lessons you learned from the experience: how you might do it differently a second time round, and how the faults of the design and implementation may be corrected. Describe factors that caused problems such as missed milestones or the known bugs and limitations.

Known Bugs and Limitations: In what ways does your implementation fall short of the specification? Be precise. Although you will lose points for bugs and missing features, you will receive partial credit for accurately identifying those errors, and the source of the problem.

This reflection should be one to two pages long.

### Deliverables

Submit a zip archive containing a complete BlueJ project for your program including Java source code with comments (this goes in the 'Code' portal).

A single document in pdf format (this goes in the 'Report' Turnitin portal) containing:

- design documentation as specified above;
- test documentation as specified above;
- your reflection report as specified above.

Two portals will be provided on the eLP (BlackBoard) for you to upload your work.

## Collaboration and Academic Integrity

You can discuss your work, but the submitted work **must** be your own individual work.

You can use some public domain code, but it must be clearly referenced. It must be a very small component (less than 5%) of the submitted assignment. Generally, it will **not** attract marks.

The assignments are designed to allow **you** to demonstrate **your** achievement of learning outcomes.

You **cannot** use the work of your fellow students or anyone else.

You are advised to read the University regulations concerning academic misconduct. More details can be found at: https://www.northumbria.ac.uk/about-us/university-services/academic-registry/quality-and-teaching-excellence/assessment/guidance-for-students/

## Marking Scheme

### Overall Mark

| Grade | Mark | Description |
|---|---|---|
| Distinction | 70-100 | Excellent work providing evidence to a very high level of the knowledge, understanding and skills appropriate to level 7. All learning outcomes met, many at high level. Marks at the high end of this range indicate outstanding work where all learning outcomes are met at a high level. Excellent in all the specific areas of the assessment criteria listed below for the assignment; evidence of successful independent learning as demonstrated by the implementation of optional features in the program; use of up-to- date material from a variety of sources; critical evaluation and creative use of theory. |
| Commendation | 60-69 | Commendable work providing evidence to a high level of the knowledge, understanding and skills appropriate to level 7. All learning outcomes met; many are more than satisfied. Good in all or most of the specific assessment criteria listed below for the assignment; evidence of independent learning; critical evaluation and creative use of theory. |
| Pass/Satisfactory | 55-59 | Satisfactory work providing evidence of the knowledge, understanding and skills appropriate to level 7. All learning outcomes are met. Satisfactory in all or most of the assessment criteria listed below. |
| Pass/Adequate | 50-54 | Adequate work providing evidence of the knowledge, understanding and skills appropriate to level 7, but only at a bare pass level. All learning outcomes are met (or nearly met and balanced by strengths elsewhere). Adequate in all (or most of, with balancing strength elsewhere) of the criteria listed below. |
| Marginal Fail | 40-49 | The program fails to achieve the basic pass criteria specified below. Work is not acceptable in providing evidence of the knowledge, understanding and skills appropriate to level 7. May be adequate in some, but not all of, the assessment criteria listed below. |
| Fail | 1-39 | Work is not acceptable and provides little evidence of the knowledge, understanding and skills appropriate to level 7. Few of the learning outcomes are met. Inadequate in terms of the various criteria given below as a basis for judging the work. |
| Fail | 0 | Work not submitted or work giving evidence of serious academic misconduct or work showing no evidence of the knowledge, understanding and skills appropriate to level 7. None of the learning outcomes are met. |

## Specific Criteria

### Basic functionality and code quality (15%)

| Grade | Mark | Description |
|---|---|---|
| Distinction | 11-15 | Excellent code that exceeds the functional specification for both text and graphical user interface; excellent structure, including use of meaningful variable names, Java coding conventions, @Override tags, visibility modifiers; relevant and accurate commenting including fully-tagged Javadoc comments. |
| Commendation | 9-10 | Good code that fully meets the functional specification for both text and graphical user interface; very good structure, including use of meaningful variable names, Java coding conventions, visibility modifiers; very good comments. |
| Pass/Satisfactory | 8 | Satisfactory code that fully meets the functional specification for both text and graphical user interface; good structure, including use of meaningful variable names, Java coding conventions, visibility modifiers; good comments. |
| Pass/Adequate | 7 | Code that meets most of the functional specification for both text and graphical user interface (areas that are lacking are compensated by other areas); reasonable structure; reasonable comments. |
| Marginal Fail | 6 | Code that meets some of the functional specification for both text and graphical user interface. |
| Fail | 1-5 | Code that does not run, but the source code shows potential to have met some of the functional specification for both text and graphical user interface. |
| Fail | 0 | No source code provided. |

### Quality of design and implementation (25%)

| Grade | Mark | Description |
|---|---|---|
| Distinction | 17-25 | Excellent design that fully demonstrates object-oriented programming design and implementation (assessed from both class diagram and description, plus the code itself), including classes, inheritance, attributes, and methods. |
| Commendation | 15-16 | Good design that demonstrates object-oriented programming design and implementation (assessed from both class diagram and description, plus the code itself), including classes, inheritance, attributes, and methods. |
| Pass/Satisfactory | 14 | Satisfactory design that demonstrates object-oriented programming design and implementation (assessed from both class diagram and description, plus the code itself), including classes, attributes, and methods. |
| Pass/Adequate | 13 | Adequate design that demonstrates object-oriented programming design and implementation (assessed from both class diagram and description, plus the code itself), including classes, attributes, and methods (areas that |

| | | |
|---|---|---|
| | | are lacking are compensated by other areas). |
| Marginal Fail | 10-12 | A design that demonstrates some object-oriented programming design and implementation (the class diagram and/or design description may be missing, so perhaps assessed from the code alone), including classes, attributes, and methods. |
| Fail | 1-9 | A poor design that demonstrates little object-oriented programming design and implementation (the class diagram and/or design description may be missing, so perhaps assessed from the code alone), including classes, attributes, and methods. |
| Fail | 0 | No evidence of object-oriented design and implementation. |

## Graphical User Interface (20%)

| **Grade** | **Mark** | **Description** |
|---|---|---|
| Distinction | 14-20 | Excellent graphical user interface, making full and appropriate use of a wide variety of widgets; correct use of action listeners and observer-observable; provision of excellent feedback to the user; use of optional dialogues such as file browser. |
| Commendation | 12-13 | Good graphical user interface, making use of a wide variety of widgets; correct use of action listeners and observer-observable; provision of good feedback to the user. |
| Pass/Satisfactory | 11 | Satisfactory graphical user interface, making use of a variety of widgets; correct use of action listeners and observer-observable; provision of reasonable feedback to the user. |
| Pass/Adequate | 10 | Graphical user interface, making use of a variety of widgets; correct use of action listeners and observer-observable; provision of reasonable feedback to the user (areas that are lacking are compensated by other areas). |
| Marginal Fail | 8-9 | A graphical user interface is provided, but may not be fully working/ interfacing to the model code, or providing adequate user feedback. |
| Fail | 1-7 | A graphical user interface of only limited functionality is provided. |
| Fail | 0 | No graphical user interface provided. |

## Testing (20%)

| **Grade** | **Mark** | **Description** |
|---|---|---|
| Distinction | 14-20 | Excellent test plan and report, full justification of tests selected, excellent coverage of program specification and implementation; clear reporting of test results (including screen shots) and corrective work for failed tests; some use of JUnit demonstrated. |
| Commendation | 12-13 | Good test plan and report, justification of tests selected, good coverage of program specification and implementation; clear reporting of test results (including screen shots) and corrective work for failed tests. |

| | | |
|---|---|---|
| Pass/Satisfactory | 11 | Satisfactory test plan and report, reasonable coverage of program specification and implementation; reporting of test results (including screen shots). |
| Pass/Adequate | 10 | Test plan and report, reasonable coverage of program specification and implementation; reporting of test results (including screen shots); areas that are lacking are compensated by other areas. |
| Marginal Fail | 8-9 | Test plan and report, some coverage of program specification and implementation. |
| Fail | 1-7 | Limited evidence of some testing covering program specification and implementation. |
| Fail | 0 | No testing performed |

## Report (20%)

| **Grade** | **Mark** | **Description** |
|---|---|---|
| Distinction | 14-20 | Excellent report with complete class diagram(s) and detailed description of classes, including design options considered; evaluation of the program; personal lessons learned; remaining bugs and limitations |
| Commendation | 12-13 | Good report with class diagram(s) and description of classes, including design options considered; evaluation of the program; personal lessons learned; remaining bugs and limitations |
| Pass/Satisfactory | 11 | Satisfactory report with class diagram(s) and description of classes; evaluation of the program; personal lessons learned; remaining bugs and limitations |
| Pass/Adequate | 10 | Report with class diagram(s) and description of classes; evaluation of the program; personal lessons learned; remaining bugs and limitations (areas that are lacking are compensated by other areas). |
| Marginal Fail | 8-9 | Poor report with some areas omitted, and much of the writing focused on generic accounts rather than specific to the student's actual program. |
| Fail | 1-7 | Poor report with very few areas included, and all of the writing focused on generic accounts rather than specific to the student's actual program. |
| Fail | 0 | No report provided |

## Feedback Techniques used in this Module

Individual written feedback will be provided for the assessment. Feedback will be given for each element of the marking scheme.

## Return of Feedback

Marks and written feedback for the assignment will be returned within three working weeks after the latest authorised submission.

## Late Submission

Unapproved late submission may cause the deduction of marks or the loss of all marks for the assignment. You can find more details at https://www.northumbria.ac.uk/about-us/university-services/academic-registry/quality-and-teaching-excellence/assessment/guidance-for-students/

## Referencing Style

Any commonly used referencing style is acceptable as long as it is used correctly and consistently. For technical work of this nature, you might prefer to use Harvard style.

## Academic Integrity

You must adhere to the university regulations on academic conduct. Formal inquiry proceedings will be instigated if there is any suspicion of plagiarism, ghosting, collusion, or any other form of academic misconduct in your work. Refer to the University's Assessment Regulations for Taught Awards especially the Academic Misconduct Policy if you are unclear as to the meaning of these terms. The latest copy is available on the University website. More details can be found at: https://www.northumbria.ac.uk/about-us/university-services/academic-registry/quality-and-teaching-excellence/assessment/guidance-for-students/