Northumbria University NEWCASTLE

Department of Computer & Information Sciences

ASSESSMENT BRIEF

Module Title:	Object Oriented Programming
Module Code:	PE7070
Academic Year / Semester:	2023-24 / TP3
Module Tutor / Email (all queries):	Lauren Scott / lauren8.scott@northumbria.ac.uk
% Weighting (to overall module):	100%
Assessment Title:	Minesweeper
Date of Handout to Students:	23rd April 2024
Mechanism for Handout:	Module Blackboard Site in Week 1
Deadline for Attempt Submission by Students:	7th July 2024 23:59, GMT
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:	5th August 2024
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

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 analysis, systems development, operating systems, security, networking, databases and the internet

KU2. Demonstrate a comprehensive knowledge and critical understanding of essential facts, concepts, principles, theories, methods, techniques and tools in the application and management of a range of current and emerging secure computing and information technologies

KU4. Apply a detailed knowledge and understanding of the software development life cycle with respect to generalised information systems

Intellectual / Professional Skills & Abilities:

IPSA1. Systematically identify and analyse complex problems of a familiar and unfamiliar nature and offer appropriate strategic solutions using a range of effective methods and tools

IPSA4. Systematically apply a range of techniques, tools and knowledge in the analysis, design, construction, testing and maintenance of high quality enabling solutions to Computer Science problems in a variety of both real world and theoretical contexts

IPSA5. Critically examine the ways of defining, promoting, controlling and validating the attainment of quality in the field of Computer Science, including security

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

PVA1. Learn independently, enhancing your existing skills and developing new ones to a high level, enabling you to sustain your own continued professional development

PVA2. Demonstrate creativity in problem solving and decision making in complex and unpredictable situations

PVA3. Effectively and professionally communicate information, ideas, arguments, problems and their solution in written and oral form to specialist and non-specialist audiences

PVA6. Engage in critical self-appraisal of their 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

Minesweeper

Minesweeper is a puzzle where there are a number of dangerous squares (mines) spread out across the board. The objective of the player is to determine where the mines lie, and flag them, to warn others. Then to find the safe squares of the board. The game provided allows the player to flag mines, or guess empty squares.

```
Col    0 1 2 3 4
Row  0 ? ? ? ? ?
Row  1 ? ? ? ? ?
Row  2 ? ? ? ? ?
Row  3 ? ? ? ? ?
Row  4 ? ? ? ? ?
Please select an option:
[M] Flag a mine
[G] Guess a square
[S] save game
[L] load saved game
[U] undo move
[C] clear game
[Q] quit game
```

Figure 1: Minesweeper project - UI(file: em1.txt)

You have been supplied with code that implements the Minesweeper game and also a partially implemented text-based user interface, found in Minesweeper.zip,. The three model classes provided are Assign,

Minesweeper, and Slot. The text-based user interface is UI, and they can be seen in the class diagram (Figure 4).The game is designed to work with supplied puzzle files, these have been included in the Levels folder.

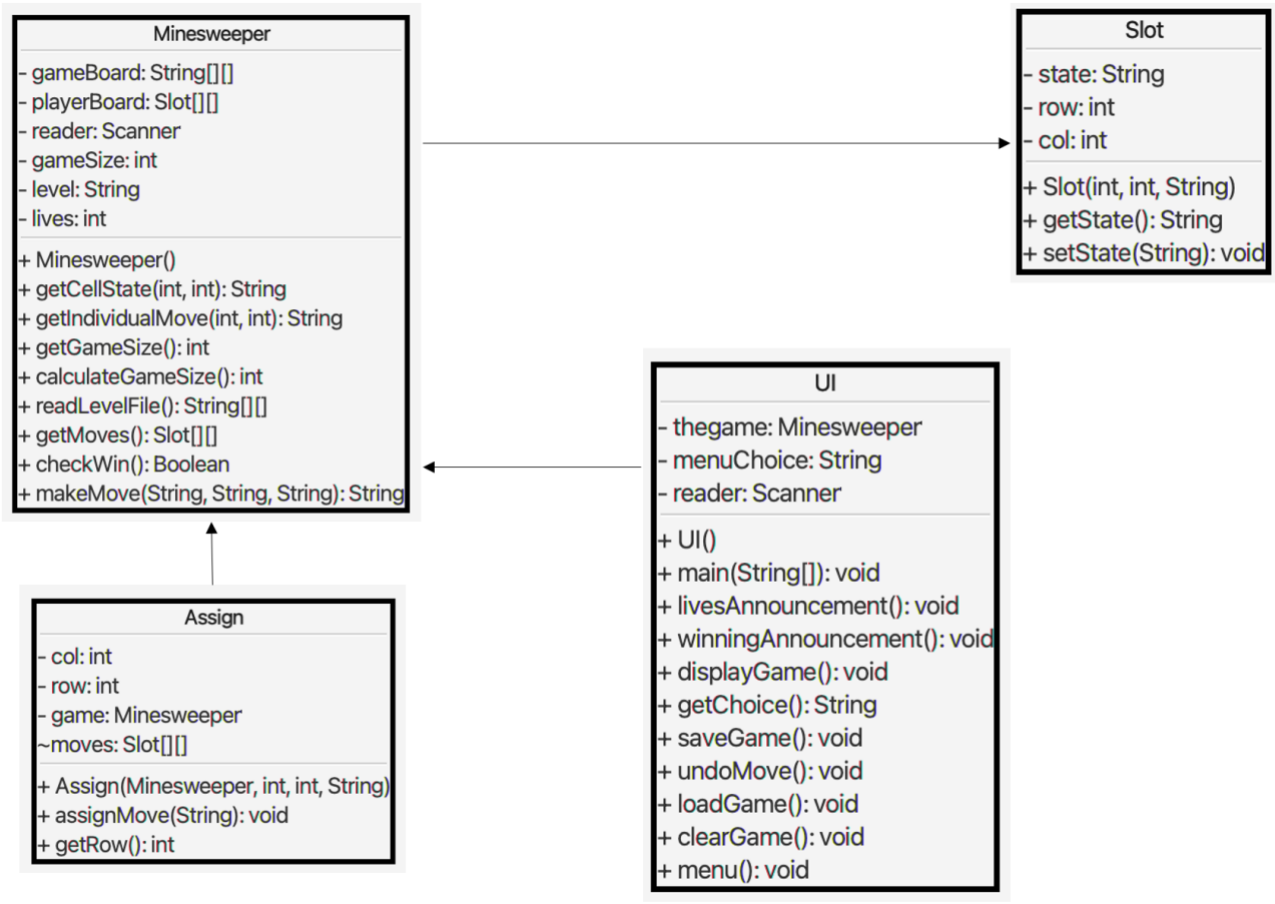

Figure 2: Class diagram for supplied code

The Tasks

The tasks you are to carry out are as follows:

- Provide a full test plan and test log for both the sample code provided (to determine initial functionality), and any further alterations made to the code.

- Complete the implementation of the text-based user interface to include the (as yet unimplemented and therefore failed tests) clear (resetting the Minesweeper game), undo (a single user move, then a further move if called again and so on), save (the state of the game to a file) and load (the state of the game from a file) methods. You should only need to make changes to the main user interface class (UI) and the helper class Assign to achieve this. You **must** use the provided model code, you are not permitted to make your own game from scratch.

- To design and implement a graphical user interface (GUI) for the Minesweeper model code that allows the user to make moves, undo user moves, clear (reset) the game, save to, and load from a file. The individual slots will need to indicate whether they are unknown, empty, flagged, or an adjacent space. The GUI should at least give feedback on whether the game has been won. The appearance of the interface is up to you, but an example is given in Figure 3 (of a game in progress). You **must** use observer-observable to update the GUI as the model changes.

- Write a report including design documentation for the full system, including a completed class diagram, descriptions of ***all*** classes, and any relevant design decisions made when creating the GUI; and an individual reflection covering the development of the project and evaluating the work.

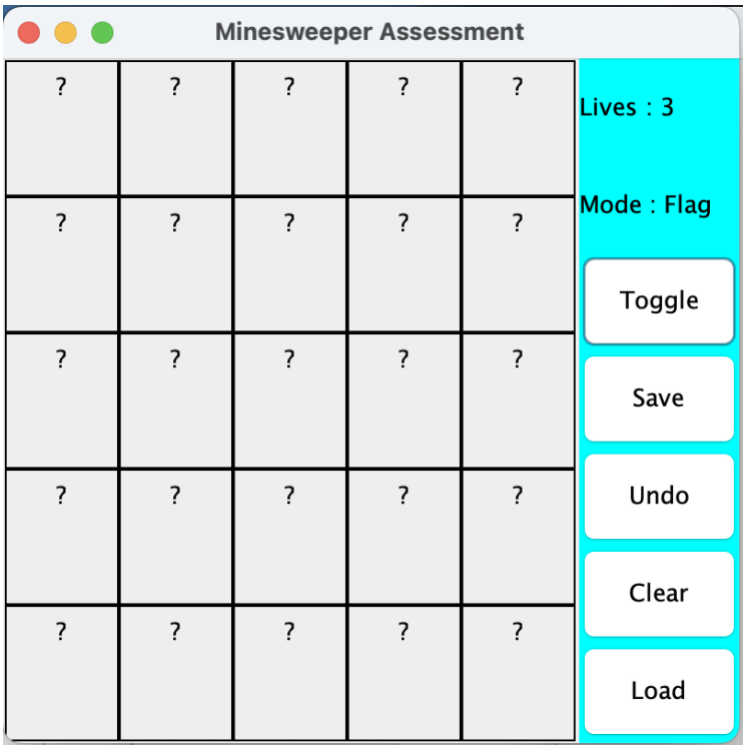

Figure 3: A possible GUI design (at start of a puzzle)

Deliverables

General Points

The development of your code is based on the object-oriented model, and you **must** make use of classes provided 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 (do not try and mix text-based UI and GUI). 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 GUI should consist of a number of classes each with well-defined functionality. There may be a driver class to set things going; multiple GUI classes to provide the user interface; there may be additional classes for e.g., 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.

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/

Northumbria University NEWCASTLE

Department of Computer & Information Sciences

Marking Rubric

Total marks available: 100	No Submission 0 marks	Fail 1-30% 1-6 marks	Marginal Fail 30-50% 6-10 marks	Pass/Adequate 50-60% 10-12 marks	Pass/Satisfactory 60-70% 12-14 marks	Commendation 70-80% 14-16 marks	Distinction 80-100% 16-20 marks
Basic Functionality and code quality Total marks: 20	No source code provided	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	Code that meets some of the functional specification for one or both text and graphical user interface.	Code that meets most of the functional specification for one or both text and graphical user interface (areas that are lacking are compensated by other areas); reasonable structure; reasonable comments.	Satisfactory code that meets most of 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.	Good code that fully meets all of 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.	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.
Graphical User Interface Total marks: 20	No graphical user interface provided.	A graphical user interface of only limited functionality is provided.	A graphical user interface is provided but may not be fully working/ interfacing to the model code or providing adequate user feedback.	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).	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.	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.	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.
Quality of design and implementation Total marks: 20	No evidence of object-oriented design and implementation .	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.	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.	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).	Satisfactory design that demonstrates object-oriented programming design and implementation (assessed from the class diagram, description of the classes, plus the code itself), including classes, attributes, and methods.	Good design that demonstrates object-oriented programming design and implementation (assessed from the class diagram, description of the classes, plus the code itself), including classes, inheritance, attributes, and methods.	Excellent design that fully demonstrates object-oriented programming design and implementation (assessed from the class diagram, description of the classes, plus the code itself), including classes, inheritance, attributes, and methods.
Testing Total marks: 20	No testing performed	Limited evidence of some testing covering program specification and implementation.	Test plan and report, some coverage of program specification and implementation.	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.	Satisfactory test plan and report, reasonable coverage of program specification and implementation; reporting of test results (including screen shots).	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.	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.
Evaluative Report Total marks: 20	No report provided	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.	Poor report with some areas omitted, and much of the writing focused on generic accounts rather than specific to the student's actual program.	Report with evaluation of the program; personal lessons learned; remaining bugs and limitations (areas that are lacking are compensated by other areas).	Satisfactory report with evaluation of the program; personal lessons learned; remaining bugs and limitations.	Good report including design options considered; evaluation of the program; personal lessons learned; remaining bugs and limitations.	Excellent report including design options considered; evaluation of the program; personal lessons learned; remaining bugs and limitations; and consideration of relevant academic literature.

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 four 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/