COMP2013-Developing Maintainable Software Coursework

Last Update: 14 Dec 2021 (Added section "Uploading your coursework to Moodle")

Number of Credits: 75% of a 20 credit module. This coursework is contributing 75% to your overall grade and will be marked out of 100.

Recommendations: We recommend dedicating approximately 40-60 total hours on the coursework. We expect 30-40 hours of work for those who are aiming for a pass (40+) and 60 or more hours of work for those that are aiming for a first (70+). Please keep in mind that the skill level varies quite a bit in a class with over 300 students, and that the exact number of hours depends on your individual skill level. In order to help you, we have less required lectures in the last two weeks of term and some of the lab sessions are dedicated to your coursework.

Deadline: The deadline will be most likely the second week in December. The exact deadline will be posted on Moodle

Assessment: The marks will be split as follows:

- 15% for git use (e.g. push, branch, merge, providing .gitignore)
- 30% for refactoring
- 30% for additions
- 15% for documentation (Readme file + source code documentation)
- 10% for the demonstration video, explaining your refactoring activities and additions

Questions: Questions can be asked either on Teams or in person during the lab. We will try to answer them as quickly as possible. Please read the questions that have been already asked before you post yours, to avoid duplication.

Summary: This coursework is about maintaining and extending a re-implementation of a classic retro game called Breakout. The new implementation has never been completed, but at least it runs, once it is set up properly. More information about the original Breakout game and its history is available on Wikipedia (https://en.wikipedia.org/wiki/Breakout_(video_game)). In addition, you will find many opportunities on the internet to play the original game online.

Requirement Specification

Basic Requirements (to pass the assessment with 40%):

1. Set up a PRIVATE git repository on the school's GitLab server and use it actively for version control activities. The URL is: https://projects.cs.nott.ac.uk/

2. Do some basic maintenance of the delivered code base (this should include things like providing a meaningful Readme.md file and Javadocs, organising files in a meaningful way into packages, breaking up large classes in a meaningful way to support the idea of single responsibility, improving encapsulation, etc.)
3. Extend the delivered code base by adding:
   3.1 A module-info.java file
   3.2 A START screen (using JavaFX) with a colour theme choice for the GAME screen and a button that allows going to the GAME screen.
   3.3 A SCORE pop-up, appearing at the end of each round, showing the scores from each round

Additional Requirements: For higher marks: In addition to the previous, do some of the following:
A1. Refactor the code by adding some design patterns to enhance maintainability
A2. Organise the code to adhere to the MVC design pattern
A3. Translate code from Swing to Java
A4. Create a permanent high score list (using a file to store scores and player names)
A5. Add interesting levels to the game (based either on your own ideas or on the original game)
A6. Add game object pictures (sprites) to the game, for representing ball/paddle/bricks
A6. Add meaningful JUnit tests
A7. Use build files (Maven or Gradle)
A8. Come up with your own groundbreaking idea ... surprise us :)

Java Version + IDE: You have to use Java 12 or higher and JavaFX 12 or higher for the implementation. The project files you are submitting need to be compatible with IntelliJ.

JavaFX: Please be aware that you have to know Swing in order to understand the source code. You should know this from Y1, but we also do some revision later in the module. Then, for the refactoring and extension, you will need to know JavaFX. We will look at JavaFX for two full weeks, starting in the middle of November. If you want to start beforehand, you can also find details about Swing and JavaFX at the following website: https://www3.ntu.edu.sg/home/ehchua/programming/#Java (check out the section "Java Programming - Part II"). If you want to install JavaFX, please have a look at https://openjfx.io/. Here you will find information about how to install JavaFX and how to use it from the command line or from within IntelliJ. JavaFX also offers a mailing list where you can ask questions.

Version Control: To start, please download the given source code from Moodle. Set up a project (we would recommend that you set up a Java project first, to see the original game running, or wait until the lectures in mid-November, when we explain how to set up a Swing project in a JavaFX wrapper) in IntelliJ and embed the files that you just downloaded. Note that the zip file you downloaded from Moodle only provides source code and resources but no project files. It is good practice to ignore IDE-specific generated files for source control. Add a .gitignore file to your project, ensuring that you follow this good practice as well. Set up a remote git repository at the school's GitLab server (https://projects.cs.nott.ac.uk). It needs to be set to PRIVATE. Then follow the setup instructions provided in GitLab to "Push an existing folder"

(i.e. do an initial push to upload files from your local to your remote repository). Now you are ready for coding with version control.

Object-Oriented Implementation: The implementation needs to adhere to some rules. We require these rules, because we want a good product to be delivered. When we look under the hood, he does not want to see a pile of scrap metal, but rather a finely tuned, carefully crafted machine. You are required to follow **Bob's Concise Coding Conventions**. See Moodle (or Bob's web page) for a copy of this document. And yes, points will be deducted for magic numbers and other rules that are not followed.

Assignment Submission and Organization

This section describes which files need to be submitted for assignment and how they should be organised. Remember, there are a LARGE number of students in the class, thus the organisation of your submission is very important. You are required to create a root folder called **COMP2013surnameFirstname**, where "surname" is replaced with your surname, likewise for "firstname", The COMP2013surnameFirstname folder contains (and organises) digital copies of all of the files that compose the assignment.

Assignment Report: A **Readme.md** file (max. 500 words), documenting the work you conducted (highlighting the key changes you made for maintenance and extension, where you made them, and why you made them). WARNING: If you do not mention it here, do not blame us later if we miss it.
Your report contains the following information:

1. Your name and student number,
2. How to compile the code to produce the application,
3. Where your Javadoc documentation is stored (the path to the directory)
4. A list of features that are implemented and are working properly,
5. A list of features that are implemented and are not working properly,
6. A list of features, if any, that are not implemented with an explanation of why they were not implemented,
7. A list of new Java classes that you introduced for the assignment,
8. A list of Java classes that you modified from the given code base,

The report also informs the reader if any unexpected problems arose during the course of the assignment. Feel free to add any information which you feel is relevant.

Design Diagram: A file called **Design.pdf** contains a high-level class diagram that shows the structure of the final version of your game (considering only classes (excluding fields and methods, unless they are relevant for understanding design principles/patterns), interfaces, relationships, and multiplicity). If you use software to reverse engineer your class diagram, make sure the delivered diagram is correct and follows the above requirements.

Project Implementation Files and Folders Description: A zip file containing your ENTIRE LOCAL PROJECT FOLDER. It needs to be possible to OPEN AND RUN your project in IntelliJ. To avoid disappointment later, test your final version on a different computer. This should help to uncover hardcoded path dependencies, which was a major issue in previous years. Name your zip file **SurnameFirstName_JavaVersion.zip**, where JavaVersion represents the Java version you used. Here is an example: "SiebersPeer_17.zip". Place the .zip file in a folder called **project** that resides in the **COMP2013surnameFirstname** folder described above.

Source Code Documentation: A copy of your generated Javadoc documentation is required. Javadoc produces a series of linked HTML web pages in order to facilitate the browsing of project implementations. The HTML web pages produced by Javadoc are placed in a folder called **javadoc** that resides in the **COMP2013surnameFirstname** folder described above. Recall that the Java output contains: (1) a brief description of each class, (2) a description of each method including input and return parameters, and (3) the original source code. The new Java classes you write use the following **author tag** convention:
@author FirstName Surname.
All modified Java classes from the previous code base use the following author tag convention:
@author FirstName Surname-modified.

In addition to reading your README file, we will look at the Javadocs to find out how you maintained and extended the game. If it is not obvious from there, we might miss it. Also, we have only a limited amount of time to look at each coursework submitted. So, please make sure to provide informative but concise Javadocs.

Demo via Screen Capture: Use screen capturing software to demonstrate the features of your application. The video should be approximately 3 minutes in length. The video file needs to be saved in MPEG or MP4 format. Also, you can use MPEG2 and MPEG4 compression formats. Your screen capture demo has to be called **surnameFirstnameDemo.mp4** (or .mp2 or .mpg) and resides in the **COMP2013surnameFirstName** folder described above. Note that the demo video will be the primary way in which we assess whether or not your software is working. The demo video shows your software running and then (for the main part) explains your refactoring activities and additions. You also have to highlight the two achievements you are most proud of.

Folder and File Organisation
Thus, when completing the submission of the assignment, you have a directory structure in your **COMP2013surnameFirstname** folder that looks like this:

README.md
Design.pdf
surnameFirstnameDemo.mp4
project/SurnameFirstName_JavaVersion.zip

Uploading your coursework to Moodle

For this coursework, Moodle limits uploads to a single file of up to 250MB. Before you upload your coursework, please create a zip archive of your COMP2013surnameFirstname folder and then upload that zip archive to Moodle (https://moodle.nottingham.ac.uk/mod/assign/view.php?id=5629781).

Assessment

To give you an idea about what we are looking for when we do the marking, we provide a draft marking scheme as an appendix. Please note that this is only a guide for us and does not guarantee you marks when you have done certain things. There is always an aspect of quality that needs to be considered as well, which often accounts for up to 50% of the mark. We also reserve the right to revise the marking scheme (within limits), if we see the need for this during the marking process.

Penalties (besides late submission penalty, which follows University of Nottingham standards):

- Using incorrect document formats (word instead of pdf) or video formats (swf instead of mp4) will lead to a penalty of -2 each
- Failing to comply with any naming conventions requested in this task sheet will lead to a penalty of -2 each.

Important Hints Notes

Source code: This coursework is about maintaining and extending existing code. So, for the maintenance part YOU HAVE TO USE THE CODE WE PROVIDE (Breakout_Clone v1.zip) as a basis, and not write your own game from scratch, or use source code from other campuses or the internet.

Interview: Make sure you understand what you are writing in your code and Javadocs. We reserve the right to briefly interview you if we think that you do not understand it.

Changes: Please note that we may make some small modification to the coursework specs. We will announce these on Moodle and keep a change log there as well.

Feedback: Dr Siebers and Dr Laramee will of course be happy to answer questions and give high-level interim formative feedback on your assignment. If you get stuck, please get in touch! However, we might refuse to answer very detailed technical questions, or very general questions like "What do you think about my project so far?".

Tips: Please be aware that there will also be a lot of useful tips and answered questions on the COMP2013 Moodle page, in particular in the Announcement section, in the labs and in the Teams COMP2013-DMS Questions channel.

Plagiarism: You are gently reminded that we are at liberty to use plagiarism detection software on your submission. Plagiarism will not be tolerated, and academic offences will be dealt with in accordance with

university policy and as detailed in the student handbook. This means you may informally discuss the coursework with other students but your submission must be your own work. Please also note that it is not permitted for you to copy and paste text from another source without correct referencing. If you are unclear about this process, please discuss with the module convenors during one of our lab sessions or at the end of a teaching session.

Good luck, have fun, code well :).

References

- (Laramee, 2007) R.S. Laramee. **Bob's Concise Introduction to Doxygen**. Technical report, The Visual and Interactive Computing Group, Computer Science Department, Swansea University, Wales, UK, 2007. (available online).
- (Laramee, 2010) R.S. Laramee. **Bob's Concise Coding Conventions (C3)**. Advances in Computer Science and Engineering (ACSE), 4(1):23–26, 2010. (available online).

Appendix 1: Draft Marking Scheme

Please keep in mind that for each item, marks are given for quality as well as delivery, e.g. for the category "Start screen with colour theme choice, using JavaFX (compulsory)" a basic Start Screen with a very complicated approach to defining a colour theme will give you 3 marks, while a well thought through and attractive looking start screen with an easy to handle colour theme choice will give you the full 5 marks. Adding some "welcome music" to the start screen or anything else exciting will then give you some extra marks in the "Anything else exciting (reward) or bad (penalty)" category.

Tasks	Marks	Comments
GIT (15%)	15	
Project exists	5	
Commits	2	Very few (below 20); good use (20+); extensive use (50+)
Regularity	2	
Commit messages	2	Need to be expressive
Branching/merging	2	
Meaningful .gitignore	2	Needs to do the job correctly
REFACTORING (30%)	30	
Meaningful package naming/organisation	2	
Basic maintenance (e.g. renaming classes; encapsulation; deleting unused resources)	4	
Supporting single responsibility by splitting up classes	2	Class sizes S (good) - XXL (bad)
MVC	3	Needs more than just reorganising files
Other patterns	3	
Meaningful JUnit tests	4	
Correct use of build tools (Maven or Gradle)	1	
Complete translation from Swing to JavaFX	10	Big job; only attempt if you feel confident you can do it
module-info.java file (compulsory)	1	
ADDITIONS (30%)	30	
Start screen with colour theme choice, using JavaFX (compulsory)	5	
Button to move to game	1	
Permanent highscore list (or high score popup for half the marks) using JavaFX	4	Store high scores and player names
Enter highscore player's name	2	
Game object pictures (sprites) for ball/paddle/bricks	4	
Additional (playable) levels	2	
Anything else exciting (reward) or bad (penalty)	12	Additional features (e.g. https://poki.com/en/g/brick-breaker)
VIDEO (10%)	10	
Showing software running	2	Don't use too much time for this
Explaining refactoring activities and extensions	5	Explaining refactoring activities usually requires a look at IntelliJ
Highlighted two achievents most proud of	1	
Video sound	1	
Timing (approx. 3 minutes) correct	1	
DOCUMENTATION (15%)	15	
Readme.md: highlighting the key changes (maintenance + extensions) + where + why	3	Either using Bob's or Peer's proposed format
Javadocs: Created (and deposited in the correct location)	2	
Javadocs: New ones added (half marks if only comments added)	2	
Javadocs: Complete (half marks for substantial amount)	2	
Javadocs: Informative and concise	2	
Class diagram: Something meaningful present	2	Classes/Interfaces with correct relationships/multiplicity
Class diagram: High level	1	Beware of complex auto-generated diagrams
Class diagram: Conforms with code (of the final program)	1	

Appendix 2: Technical FAQs and Tips

Coursework - how to approach it...

Peer: I have been ask by a student, how I would approach the coursework. Here is my ad-hoc answer: The way I would approach this is to set up the local and remote git repository. Then I would inspect the code, perhaps using a debugger. While inspecting the code I would add any kind of understanding I gained as comments or Javadocs to the code. I would perhaps consider using a reverse engineering tool for creating a class diagram, but I would not want to rely on it, and would also create one manually, while inspecting the code. Once I know the code a bit better I would do some basic maintenance, e.g. organising files in a meaningful way into packages, breaking up large classes in a meaningful way to support the idea of single responsibility, improving encapsulation, and sorting out other things that are easy to do. Then I would perhaps create a new class diagram, which shows my planned alterations and extensions. Then I would go for the more difficult maintenance and extension tasks. Then I would add some surprises :).

Bob: In addition to what Peer stated, I would also start applying the debugging guidelines we went over in the debugging lab. Start inserting test and m_trace flags into the classes to understand the flow of control starting with the main method. Also, I would definitely create some auto-generated diagrams of the code using built-in diagram generators in IntelliJ or some other IDE.

How do Swing and JavaFX interact?

See the following links for some details. If you find more (useful) information on the topic, please post the related links on the TEAMS Question channel

https://stackoverflow.com/questions/35906779/converting-from-swing-to-javafx

https://docs.oracle.com/javafx/2/swing/jfxpub-swing.htm