Informatics 1: Object Oriented Programming

Assignment 2 – Understanding and reviewing code

The University of Edinburgh
David Symons (dsymons@exseed.ed.ac.uk)

Overview

This assignment is designed to give you experience in reading and understanding code that is not your own. In so doing, you should learn to value well written code and gain an appreciation for documentation. You are set three tasks, described in the *Basic*, *Intermediate* and *Advanced* sections below. All your answers will be written up in the form of a report.

Marks will be awarded out of 100 and count for 20% of your overall grade for Inf1B. Each 20 credit course is designed to involve around 200 hours of work. Of those, 9 hours have been allotted for this assignment. Completion time may vary depending on your level of experience and the grade you are aiming for. While you are encouraged to aim high, the intermediate and advanced sections are optional and grades in the upper bands are expected to be rare. Details on marking criteria can be found in the eponymous section below.

Before you start, please read carefully the section on *Good Scholarly Practice*. The final section gives instructions on how to submit your work, which is **due by 16:00 on Friday 5th March**.

Basic: From code back to the specification

Usually, you are asked to write code to a given specification. Doing so can be seen as translating a natural language to a programming language. The translation process is prone to error and requires a lot of attention to detail. Once you have a prototype solution, it is useful to check that it does exactly what it was supposed to. One way of spotting disparities between what the code actually does and what was intended, is to have the programmer describe the program in their own words. Comparing this to the initial specification can reveal important differences. Your first task is to write such as a reverse translation i.e. work out the specification to which the given code appears to have been written.

Step 1: Download the code
The program to be analysed is called *PrecipitationGraph* and is included in Assignment2.zip, which can be downloaded from LEARN (under Assessment → Assignment 2).

Step 2: Get an overview of the code
Look at the different classes and their methods to gain an overview of the largest components that make up the program. Initially, focus only on the names of the methods and any comments describing what they do. At this stage, it can be counterproductive to read the code in the body of methods or comments pertaining to individual lines of code.

Step 3: Run the code
The class *Test.java* contains the main method, which exercises some of the features of the program. You can run this without having to edit the code. You may want to experiment a bit more by calling other methods or changing parameters. In so doing, you might break the code, which is deliberately imperfect. Take note of any errors you find for the intermediate task if you are attempting it.

Step 4: Look at the details of the implementation
Now it is time to get your hands dirty and dive into the code itself. Your aim should be to extract implementation details. Beware of errors in the code! If you find something that looks wrong, try to work out the programmer's intention even if a feature was not implemented correctly. Take notes while you are doing this, but do not correct any errors in the code.

Step 5: Make a list of the program's features
Use your notes and overall understanding of the program to compose a list of features that you think the programmer was trying to provide (whether they succeeded or not). Include this in your report in the form of short, technical bullet points. You can can assume the reader will be a programmer. This should not be a description of the implementation however. Another developer may want to follow a different approach, so state what the program should be able to do rather than how features should be implemented.

Step 6: Write a specification
Finally, write a matching program specification. This is essentially a prose version of the features, but it should be less technical. Imagine you are commissioning this piece of software and have never seen a single line of code. To make sure the programmers you are hiring produce something useful, you need to provide a detailed, unambiguous description of your requirements.

Step 7: Go beyond the code
Since you are reverse engineering the specification from the code, you can only see the features that are provided. The absence of some features may, however, be conspicuous. Try to come up with two additional features the company may require and explain why these would be useful in your report.

A template for the report is provided. See the *ReportTemplate* folder in Assignment2.zip.

Intermediate: Code review

Now that you have a specification, you can evaluate the provided code against it. Your job is to write a code review. This involves developers going over code written by other members of their team with the aim of finding errors and suggesting improvements. Your feedback should provide the creator of *PrecipitationGraph* with practical advice on how to improve their implementation.

You will be marked on the quality of your code review, which should be:
- Complete – cover all issues you can find
- Specific – focus on parts of the code that need to be improved, avoid giving vague feedback
- Constructive – propose actionable steps to improve the code
- Justified – provide explanations and give reasons for why a proposed change is beneficial

The following outlines the aspects to be covered in the code review. Please use a tabular format in your report as shown in the template. You can format the table differently if you wish, but there must be a clear separation between different issues.

Functional correctness
Evaluate the extent to which the program fulfils the specification (not including the requirements you added in step 7). Explain any issues you find and provide ideas for how to solve them. This is not asking you to find ways of crashing the program – that is more often an issue with robustness (to be reported under code quality below). Instead, you are looking for errors in the logic or cases in which the program does something different to what it was meant to do.

Code quality
List up to ten different issues you can find with the quality of the code. Explain why the code needs to be improved and make a suggestion for how to do so. If you find the same type of issue multiple times, only report one instance of it (preferably the worst offence).

Desirable qualities include, but are not limited to:

- Structure
  - A good design should split the task into more manageable chunks (think divide and conquer) e.g. by a meaningful division of the code into classes.
  - Modularity and reusability: Classes and methods perform well defined tasks. They can be seen as building blocks for solving many different problems, not just the task at hand. Long methods are a symptom of a too problem-specific design.
  - Code duplication is avoided.
  - Extensibility: New features can be added without redesigning the whole program.
- Robustness
  - The program is able to handle errors and invalid/unexpected inputs without crashing.
  - Correct use of access level modifiers (public/private) and static to restrict exposure.
- Technical/language proficiency
  - Use the appropriate features of your programming language. This means not restricting yourself to building everything using a few keywords when there is a more elegant solution. It also means not using a needlessly complex mechanism for solving a simple task.
- Readability (self-documenting code)
  - Use descriptive but concise names for variables, methods and classes.
  - Class names start with a capital letter, method and variable names with a lower case letter.
  - AllNamesAreWrittenInCamelCase
  - Exception: Static constants are written in SHOUT_CASE.
  - Use consistent formatting throughout all classes.

Code documentation
Give two example each of where:

- more comments would be helpful
- needless or uninformative comments clutter or obscure the code
- comments are actively misleading, distracting or inappropriate

Advanced: Legacy code

One of the developers on your team has gone into retirement and the job of maintaining their code has been dumped on you. In order to add a new feature, you must understand how the existing code works. The class *X.java* (in Assignment2.zip) is particularly obscure. Your predecessor seems to have been obsessed with efficiency and has no concept of code quality or documentation. All you can tell from context is that it must be some kind of data structure. You now have to look at the code to work out what it does. Hint: Look up bitwise operators and bit shifting.

For each method in *X.java*, do the following:

- State what the method does.
- Give it a more descriptive name.
- Explain in detail how the code works (not required for methods *m5* and *m6*).

Finally, answer the following questions:

- What kind of data structure is this and what would be a better class name?
- What are the advantages and disadvantages of the chosen data representation?
- Is there any justification for writing code like this (why/why not)?

Good Scholarly Practice

As with all assessed work, you must fulfil the University requirements for good scholarly practice. Details can be found here: https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Marking Criteria

Marks will be assigned in accordance with the University's Common Marking Scheme (CMS). See: https://web.inf.ed.ac.uk/infweb/student-services/ito/students/common-marking-scheme

Solving the task described in the *Basic* section is sufficient to obtain a pass grade (up to 49%).
Both basic and intermediate tasks must be solved for a 2nd or low 1st class (up to 79%).
You will need to tackle all three tasks to get an A1 or A2 (up to 100%).

Please note that attempting the more difficult tasks only removes grade capping. It does not guarantee a mark in the corresponding range. Your answers are graded on how complete, accurate and well presented they are. A good solution to one task is preferable to a poor attempt at all.

Report

A template for the report is provided, but you should still aim for the following qualities:

- Content
  - Answer questions directly without beating about the bush.
  - Do not just write down everything you know about the topic, stick to what was asked.
- Structure
  - Break the content down into well labelled sections.
  - If you are making an argument, ensure you organise your points into a logical sequence.
  - Try to make the text flow by avoiding frequent or abrupt changes of topic.
- Writing style
  - Should be clear and concise. Make the report as short as possible without cutting essential details.
  - Avoid repetition: Make your point once and make it well, but avoid re-stating it.

Submission

This assignment is **due at 16:00 on Friday 5th March.**

Please upload your report to LEARN (under Assessment → Assignment 2) as a single PDF file.

Resubmission

You can submit more than once up until the submission deadline. Your last submission before the deadline will be marked. Once the deadline has passed you can no longer change your submission.

Late submission

If you have not uploaded anything before the deadline, you can submit up to 7 days late. Be careful to submit the correct version, as resubmission is not possible. A lateness penalty of 5% per day will apply unless you have an extension and/or learning adjustment. The full policy can be found here: https://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests