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. The following three sections describe the tasks you need to complete for a basic, intermediate and advanced solution. All your answers will be written up in the supplied worksheet.

Marks will be awarded out of 100 and count for 25% of your overall grade for Inf1B. Completion time may vary depending on your experience and the grade you are aiming for. Of the 200 hours the course is designed to take, 9 have been allotted for this assignment. This is the estimated time for completing the basic and intermediate tasks. Marks above 69% are expected to be rare and require you to tackle the advanced section, which goes beyond the course in terms of content and time.

Before you start, please read the following sections below:

- *Good Scholarly Practice*
- *Marking Criteria*
- *Submission, late submission & extensions*

This assignment must be **submitted on Learn by 16:00 on Friday March 18th**.

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. The finer details can distract from the bigger picture at this stage, so 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 program

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: Write a technical description

Use your notes and overall understanding of the program to write a technical description of some of its key components. This should consists of a short statement of what each component does, followed by an explanation of how the implementation works. Use technical terms (e.g. class, method, loop, conditional, fields, arrays, data types, …) as though writing for another programmer. Add your description to the worksheet template. You can find this in *Assignment2.zip.* There are .odt and .docx versions to choose from. If you want to use a different editor, please recreate the templates as closely as possible. There are no extra marks for fancy formatting though.

In dataProvider.java describe:

- public int getRain(int month, int day)

In PrecipitationGraph.java describe:

- PrecipitationGraph()
- private int[] prepareData()
- private void verticalGraph(int[] array)

Step 6: Write a specification (non-technical description)

Compose a list of features that you think the programmer was trying to provide (whether they succeeded or not) and write a matching program specification. 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. You must not use any technical language at all! Also, avoid describing how the code you have been given works. Another developer may want to follow a different approach, so focus on the intended functionality and not on how something was implemented. Include your specification in the worksheet.

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. Some features are conspicuously absent. Describe two additional features the company will probably require and explain why these would be useful. Again, include your answer in the worksheet.

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 – avoid giving vague feedback and focus on parts of the code that need to be improved
- Constructive – instead of just criticising, 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. The worksheet has a table for you to fill in. There must be a clear separation between different issues. Do not lump everything together in one long piece of text.

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. The worksheet has a table for you to fill in what you think are the most severe logic mistakes in the program.

Code quality
See *CodeQuallity.pdf* (in the Assignment 2 folder) for a summary of desirable code properties and Java coding conventions. While there is broad agreement about what is good practice, there are some alternative styles. For this assignment, you must apply the rules stated in the provided PDF.

List different issues with the quality of the code. The worksheet has a table that already states the types of issues you must look for (do not change these). Some types of issue occur multiple times. Please only report the worst offence in each category. There are three columns for you to fill in.
- The "Code affected" column should identify which class is affected and give a more exact location if possible. For example the name of a method or a line number. If you want to refer to code by line number, you must use the line numbers in the provided code. Note that these may change if you e.g. add comments to the code to help your own understanding!
- The "Issue and proposed solution" column should provide a short and concise description of the problem and explain how it could be solved.
- The "Explanation/justification" column should give an objective reason for why your proposed fix is necessary. Personal preference is not a valid justification!

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 i.e. what would be a better name for class X?
- What are the advantages and disadvantages of the chosen data representation?
- What is categorically wrong with X.java?
- What aspect of the code can be justified and under which circumstances?

Advanced: Design patterns

The code for the first assignment loosely follows a design pattern known as Model-View-Controller (MVC). Read up on this design pattern and include a short summary of it in your worksheet. You should explain how the pattern is used and what it is good for. Reference your sources, a possible starting point for your research is: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm Guideline length is 400-500 words. Markers are instructed to stop reading after about 550 words because marker time is limited and you are also being graded on conciseness.

Good Scholarly Practice

Please remember the good scholarly practice requirements of the University regarding work for credit. See: https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct
This also has links to the relevant University pages.

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

The basic requirements correspond to the description of a pass grade (up to 49%).
Intermediate features are required to demonstrated the mastery required for a 2nd class (up to 69%).
You will need to tackle the advanced section for a distinction (up to 100%).

Please note that attempting the more difficult tasks only removes grade ceilings. It does not guarantee you a mark in the corresponding range. You still need to fulfil all requirements for the lower categories as well! We are using criteria-based marking for this assignment as explained in the first Wednesday live session. Review this for more information.

Submission, late submission & extensions

This assignment must be **submitted on Learn by 16:00 on Friday March 18th**. Please upload your completed worksheet to LEARN (under Assessment → Assignment 2) as a single PDF file.

Resubmission
You can resubmit as often as you like until the deadline. Your last submission before the deadline will be marked. If you have at least one on-time submission, you cannot submit late i.e. update your submission after the deadline.

Late submission
If you have not uploaded anything before the deadline, you can submit up to 6 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

Extensions and special circumstances
See links on the LEARN page (under Assessment) for information on how to apply.