Programming Design and Implementation (COMP 1007) Assignment

Connor Kuljis, 19459138, 31st May 2020

Introduction: This is a document that will attempt to justify and provide reasoning to the overall design choices made - and provide some insights on class models and challenges faced.

Contents

Approach to Data Validation

One aspect of the approach to data validation in this assignment is to minimise repetition of handling and validating user input. To aid this I have made use of a separate UserInterface class, in conjunction with method overloading to validate user input from the keyboard. The class submodules 'userInput' can validate the following datatypes: Integer, Double, Long, Float, String and Character. The submodules/methods take in 3 parameters. A short string message to display to the user, a minimum value and a maximum value. These modules help mitigate the chances of error and exceptions where the flow of the program is disrupted. This approach also underpins the idea of making programs more readable by other humans. Instead of mutiple lines of code and nested loops, it can be done in a more readable fashion on one line, avoiding repetition and reducing the total number of lines of code in the program.

Another aspect in my approach to data validation is the idea that there is upper and lower level exception handling. Exceptions could occur anywhere in my code, but wrapping everything in a try catch block can be hard to read and confusing. My solution to this problem is to isolate a small areas to my program that does use exception handling, and if needed, re-throw exceptions to the caller - which can then loop untill no exception is raised. This is most evident in the FileIO class where the program is reading/writing to a file, based on user input. In my readFile method, (which opens a given file based on a file name, and attempts to create a multidimensional array from it) I can catch and re-throw some custom exceptions such as: FileNotFoundException, IllegalArgumentException and NumberFormatException. This allows the upper level caller to handle responsibilty and simply catch the base Exception class.

Validating Images vs Kernel Images According the spec, when importing an image as a csv, we must validate each element in the array, being within the values 0 and 255 inclusive. This created an issue as if a kernel csv with negative integers was read in, an exception would be thrown. My solution was to create two different parsing methods within the FileIO class *parseStringToInt* and *parseKernelStringToInt*. Additionally Images must be rectangle in size (n x m), whereas kernels must be validated to be square (n x n). Because a square is a perfect rectangle, I was able to apply the same methods to check if the dimensions of the array. When reading the kernel file I can further check if the array is square with a seperate method, whilst maintaining that the kernel has no jagged edges. Further breakdown is included within the inline comments.

Design and Justification of Classes and Class Functionality

My reasoning and design of my classes mainly centers around the assignment specification and avoiding complicating things further than needed. In total there 7 classes, 2 of which are model classes - Image and Date.

Image class

Object oriented approach to an image object. The image has a single classfield and is generally constructed by passing in a 2D array of integers via the alternate constructor. This allows easy creation of images objects via FileIO. Within the class it also includes public submodules to handle the convolute() and smoothing() operations. I prefer this as I can directly perform those operations on the object, and can convolute/ smooth the image multiple times without creating many more arrays.

Date class

A class for creating dates. Allows for validation of dates and dates that coincide on a leap year. The date is created from an 8 digit integer, and will throw an IllegalArgumentException if the date is invalid. This date class is incorporated in the FileIO class when naming files. I had decided to use the class as we may create multiple date objects when naming files and provides a way to validate dates.

Menu

This is the root file from where all other java classes are called. The main functionality stems from my Menu class which is a base skeleton of the program which branches off and calls other small sub-menus for further functionality. This gives it a more readable structure and avoids having multiple nesting

ofcase/switch statements and makes it more cohesive at the expense of more coupling.

DetectEdges

This class has only one function, which is to provide a further menu for reading files - either as a png or csv. It is deeper level sub menu that is in its own class to further streamline and isolate functionality. It is a stripped out version from one of the prior assignment tasks and already had a suitable menu built.

FileIO

FileIO is a class of static methods and functions to safely read, write and name different image formats. It handles a majority of user input and includes exception handling for most major exceptions.

PDIMath

Class of mathematic functions for use within the program. Learning how to create a private library rather than relying on the Java API.

UserInterface

Class to handle input and output to the user. An approach to data validation and method overloading.

Importance of Static and Model Classes

Model classes can be used as a template for creating objects. Static classes often provide common functionality through various methods in a class. Model classes, such as the Image class, help manage interactions in a straight forward way. It allows me to get information from an image or update the image. Whereas static classes can simplify and allow for greater re-usability of methods.

Real World Implication

There are many implications of the convolute algorithm, it could also be used to remove or isolate lines in an image. Imagine an image with horizontal and vertical lines, but we just want singular lines running in one direction. By applying a convolution with either horizontal or vertical kernels we could remove any of the opposing lines. This could be used in the context of maps, diagrams or topography, eg: a train map, circuit diagram, satellite image, barcode.

Challenges Faced in Design and Implementation

One of the challenges I faced was *naming*, more specifically having meaningful names wether it be for variables, methods, arguments, classes, directories or files. It was clear to me once dealing with many different class files with multiple methods, especially looking earlier practicals, things can get messy and and without proper comments can sometimes make little sense; even if they fit the style guide. I have learnt to use longer, intention revealing names, creating names where the length of name is relative to the scope of the program and understanding the purpose of good naming.

Another obstacle was the grand design/implementation and the linking of different submodules together. When I first started I wasnt quite sure where things needed to go and in what order. Just a general sense of direction. But by breaking things down in to small tasks helped me overcome this obstacle.

Testing. Many times when refactoring, stripping/ adding functionality you are not sure if what you have done will break the program, or give you wrong output even if it compiles. This leads on to the point of testing as I spent a lot of time just testing the program by hand, compiling it, running it, then exploring through the menu by hand trying to find any exceptions, then going back - adding a print statement to double check if the program is working optimally. I found it hard to write test harnesses, especially in regards to user input. In the end i created a testMain file to skip user input and hard code some input values for singular methods.

Self Reference Notice

SELF-REFERENCE NOTICE: This assignment may consist of adpated versions of previous assignment practicals and assignment questions. Prior work is denoted by a line comment at the beginning of a class file. All work is my own.