COMP0004 Object-Oriented Programming Programming Exercises 2

Aim to complete as many questions as you can by the end of January

Purpose: Writing more Java programs to get familiar with the core language.

Goal: Complete as many of the exercise questions as you can. If you are keeping up, you need to do at least the core questions. The additional questions are more challenging and are designed to stretch the more confident programmers. If you can't do them now, be prepared to come back and try them later on. You must complete the core questions and get them reviewed in a lab session.

Feedback: It is important that you get feedback on your exercise answers so that you know they are correct, that you are not making common mistakes, that the program code is properly presented and that you are confident you have solved the problem properly. To do this, get your answers reviewed by a TA during a lab session.

See the additional notes on the COMP0004 Moodle site about using Java, and classes Input, FileInput and FileOutput.

Q1. Write a program to read in a text file and output the number of characters, words and lines it contains. Spaces, tabs, newlines and similar characters should all be counted as characters. Words should contain only a-z and A-Z. Hyphens, quotes, digits and any other characters are not part of a word. This means, for example, that words hyphenated like ‘on-time’ are treated as two words.

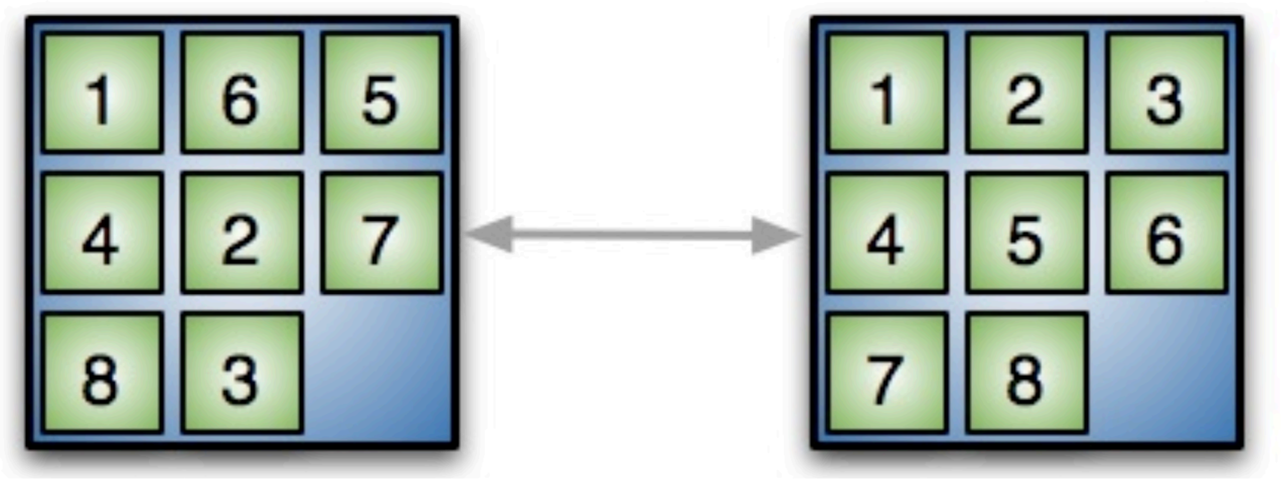
Q2. Write a program to read in a day, month and year, create an object to represent the date, and output the result as a formatted string (e.g., 2022-01-01). Use a class in the Java class library to represent and store the date.

Hint: There is a Date class but this is not what you want. Instead, look at the JavaDoc documentation for the Calendar class (e.g., Calendar.getInstance or Calendar.set), and also the DateFormat class for formatting dates to display. The JavaDoc explains how to use these classes, with some examples.

Q3 a. The Java Library class BigDecimal provides ‘immutable, arbitrary-precision signed decimal numbers’, or in other words numbers that can be as large as you want (subject to available memory) and not limited to 32 or 64 bit representations. Write a program to calculate factorial numbers of any size (or as big as can fit in memory).

b. Modify your prime number program from Exercises 1 Q17 to use BigDecimal rather than long.

4. You are probably familiar with a sliding block puzzle like the one illustrated below, where you have to slide the numbered blocks into order:



Determine a data structure to represent a sliding block puzzle and a recursive algorithm that will find the sequence of moves needed to slide the blocks into order from any starting position. Write a Java program to implement this. Note, use a text-based display, you don't need to display a graphical version.

5. The Java library class HashMap implements a dictionary or map class, that maps keys to values. For example, (key: 'blue' value: 123) maps the String 'blue' to the integer 123. The type of a HashMap to store the example would be declared as HashMap<String,Integer>, note the syntax and that you have to specify the key and value types. The JavaDoc for HashMap gives the methods to use a map and further information.

Write a Java program that uses a HashMap to implement a word dictionary, mapping words to their meanings (HashMap<String,String>). It should be possible to add words and meanings, get the meaning for a word, and any other operations you think necessary. Input and Output can be via the terminal, but words and meanings can be stored in files.

6. Consider a program that reads a Java source file (.java file) and verifies that the opening and closing braces (curly brackets) are correctly paired and nested. Beware of braces appearing in comments and string literals.

i) Write a basic one-class version of the program just using methods, variables, loops, etc. You can use the FileInput class to read the file, it does not count as the one-class.

ii) Write a version with several classes, one of which is a class BracePair to represent a matched pair of opening and closing braces. As braces can be nested, BracePair should also have an ArrayList<BracePair> of nested BracePairs, allowing zero of more nested pairs of braces to be recorded. As the program runs it builds a simple tree-like structure of BracePair objects representing the structure of the source code in terms of braces. If a complete tree can be built then it means that the braces are correct in the source code, otherwise the program should stop when it first finds that the braces are incorrect

Add additional functionality using the BracePair tree structure to generate a report on the level of nesting of braces, for example the number of pairs nested at depth one, two, etc., the deepest level of nesting, and the pair with the most nested pairs.