COMP0004 Object-Oriented Programming Exercises 2

Example Answer for Q1

Q1. *Write a program to read in 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.*

This answer takes the approach of creating a class to provide the service of counting the characters, words and lines in a text file. An object can be created, given a file to read from, do the counting, and provide access to the results.

The counting strategy is to read the file one character at a time, then check if the character is a letter (a-z or A-Z) and finally to check for a newline. At each stage the relevant counter is incremented, with the counters represented by the instance variables `characters`, `words` and `lines`.

To determine if there a word has been found a `boolean` flag called `inWord` is used, which is set to false when a character is not a letter, and set to true when a character is a letter and the flag is false. Setting the flag to true denotes the start of a word, and the word count can be incremented.

Class Character provides a method called `isLetter` to determine whether a character is a letter, and this is used in preference to writing a method. This works well enough for a-z and A-Z used in English, but the definition of what is a letter is more complicated than you might think. Take a look at the Javadoc documentation for class Character to find out more.

```
public class Q1
{
  private int characters;
  private int words;
  private int lines;
  private boolean inWord;

  public Q1()
  {
    characters = 0;
    words = 0;
    lines = 0;
    inWord = false;
  }

  private char getCharacter(Input input)
  {
    char c = input.nextChar();
    characters++;
  return c;
}

private void checkForLetter(char c)
{
  if (Character.isLetter(c))
  {
    if (!inWord)
    {
      words++;
      inWord = true;
    }
  }
  else
  {
    inWord = false;
  }
}

private void checkForNewline(char c)
{
  if ((lines == 0) || (c == '\n'))
  {
    lines++;
  }
}

public void processText(Input input)
{
  while (input.hasNextChar())
  {
    char c = getCharacter(input);
    checkForLetter(c);
    checkForNewline(c);
  }
}

public int getCharacterCount()
{
  return characters;
}

public int getWordCount()
{
  return words;
}

public int getLineCount()
{
  return lines;
}
```

```
  public void displayCounts()
  {
    System.out.println("Lines: " + lines);
    System.out.println("Words: " + words);
    System.out.println("Characters: " + characters);
  }

  public static void main(String[] args)
  {
    Input input = new FileInput("sample.txt");
    Q1 q1 = new Q1();
    q1.processText(input);
    q1.displayCounts();
    input.close();
  }
}
```

The main method simply opens a file called sample.txt if it exists. You might want to replace that with code that asks the user to input the filename, but what we have here is sufficient to run the code on some sample text.

On Unix based systems there is a utility program called 'wc' (word count) that *'… displays the number of lines, words, and bytes contained in each input file …'*. This has been available since the very early days of Unix.

The output of the 'wc' program was compared with the output of the Java code to check that the code was producing correct answers. The character and line counts were found to be consistently equal, but the word count could vary by plus or minus a few words. The issue here is what is defined as a word. The Java code above simply assumes a sequence of one or more letters with no other characters. Apostrophes, hyphens and other punctuation are treated as breaks between words. This means, for example, that "it's" will be treated as two words. The 'wc' program has its own definition of what is a word, which is not exactly the same, explaining the small differences between the programs. Read the 'wc' manual page to find out what it does.

There is one other issue that the Java code above does not fully address, and that is: what is a newline? On Unix-based systems newline is the single character '\n', while on Windows newline is represented by two characters '\r\n'. Given that both representations contain '\n' this doesn't actually affect counting lines, but the issue is mentioned here as it can cause problems in some applications where the difference matters. For example, the '\r' could be incorrectly treated as a character in a word when processing a text file created on Windows.

The issue is recognised in Java as the System class provides a method that returns the end of line representation for the operating system that the Java program is running on:

```
String endOfLine = System.lineSeparator();
```

This allows a Java program to be written so that it will work correctly with the representation in use on the machine it is running on.

You might wonder what ‘\r’ and ‘\n’ actually mean. They are ‘carriage return’ and ‘line feed’, which make sense if you think about manual typewriters and the teletype printers in use as terminals when Unix was being created. The carriage return operation moved the carriage position, where the next character is typed, back to the start of the line on the left. The newline operation moved the paper up one line. The combination meant that the next character printed would be at the start of the next line. Hence, ‘\r’ and ‘\n’ are control characters for the teletype or printer to position the printing position and are not visible characters that are printed.