PGCert IT: Programming for Industry

Input and Output (IO)

Before you start

The Git remote repository for today's lab is located at this link. **Remember to FORK yourself a personal copy** of this repository – don't just clone it!

Exercise One: Simple IO Problems

1. Consider the file input1.txt, which is located in the lab folder. What is the output of the following code?

```
private void fileReaderEx01() {
  int num = 0;
  FileReader fR = null;
  try {
    fR = new FileReader("input1.txt");
    num = fR.read();
    System.out.println(num);
    System.out.println(fR.read());
    System.out.println(fR.read());
    System.out.println(fR.read());
    System.out.println(fR.read());
    fR.close();
  } catch(IOException e) {
    System.out.println("IO problem");
  }
}
```

2. Complete the `printNumEsWithFileReader()` method in the `ExerciseOne` class, which should use a `FileReader` to read input2.txt and print out the following information:

    a. The total number of characters in the file, and:

    b. The number of e's and E's in the file.

3. Repeat step 2, but this time complete the `printNumEsWithBufferedReader()` method, and use a `BufferedReader`.

Exercise Two: PrintWriter & BufferedReader

1. In the ex02 package, you'll find the MyWriter class. Currently, this lets a user enter a file name, then type as many lines of text as they wish, using the keyboard. For this task, modify the start() method so that it properly uses a PrintWriter to write every line the user types to the file that the user specified.

    - **Hint:** Try to only open / close the file once, but you can write to it as many times as you need to.

    - **Hint #2:** You can check that your program is working as intended by looking at the output file in a text editor.

2. Now, complete the MyReader class, which should again prompt the user to enter a file name, then should use a BufferedReader to open the file, read all the text in that file, and print it out. Check that it works by reading one of the provided input text files, and / or your own output from part 1 of this exercise.

3. In the MyScanner class, create another program for printing out the contents of a text file. The functionality should be identical to that in part 2. This time, use a Scanner.

Exercise Three: DataInputStream & DataOutputStream

In the ex03 package you'll see the Movie class from a previous lab. In this exercise, we'll create two programs: the first will create a file containing data for lots of movies, while the second one will read that file and create Movie objects.

1. Complete MovieWriter's saveMovies method. This method should use a DataOutputStream to write the contents of the films array to the given file. **Note:** You should not make any assumptions about the number of movies in the array – your program (and the one in part 2) should work for any number of movies! It might pay to *write the number of movies to the file*, as well as the movies themselves.

2. Complete MovieReader's loadMovies method. This method should use a DataInputStream to read the contents of the given file and use it to create, fill and return an array of Movie objects.

Exercise Four: CSV Files

In this exercise we'll use something other than DataInputStream and DataOutputStream to load and save movies.

1. In the ex04 package you'll see Ex4MovieWriter, which extends exercise three's MovieWriter class, and expects a different implementation of saveMovies. Implement this version of the method so that it uses a PrintWriter to output a CSV-style file, with each line in the file representing a separate movie.

2. In the ex04 package you'll see Ex4MovieReader, which extends exercise three's MovieReader class, and expects a different implementation of loadMovies. Implement this version of the method so that it uses a Scanner to successfully read the files generated by part 1 of this exercise.

For more information on CSV files, you can visit this link.

Exercise Five: A Bulls and Cows Game

Introduction

The bulls and cows game is a code-breaking game designed for two or more players. Each player chooses a secret code of 4 digits from 0 – 9. The digits must be all different. The goal of the game is for each player to guess the other player's secret code.

The players in turn present their guesses to the opponents. The opponents respond by telling the players:

a. The number of bulls, i.e. the number of matching digits in their right positions, and
b. The number of cows, i.e. the number of matching digits but in different positions.

For example, the secret code is: 4321, the match responses for the following guesses are shown below:

```
Please enter your secret code:
4568
---
You guess: 1234
Result: 1 bull and 2 cows

Computer guess: 5940
Result: 0 bull and 2 cows
---
You guess: 1345
Result: 0 bull and 2 cows

Computer guess: 1279
```

```
Result: 0 bull and 0 cow
---
You guess: 4271
Result: 3 bulls and 0 cow

Computer guess: 4890
Result: 1 bull and 1 cow
---
You guess: 4281
Result: 4 bulls and 0 cow
You win! :)
```

More information about the game itself can be found here.

In this exercise, you need to design and implement a simple console program to play bulls and cows interactively against the computer. Both the player and the computer will have secret codes, and each will be trying to guess the secret code. Both the player and the computer only have seven attempts for guessing the secret codes. Before starting the game, the program should let the player enter the secret code for the computer to guess. You can use a simple random strategy (generating four unique digits randomly) for the computer to make its guesses. If the user entered an invalid secret code, the program should ask the user to try again. That is, you should use a try-catch block to recover from incorrect inputs made by the user.

You should work in pairs for this exercise.

Task One. Design the game

Design your classes and methods (i.e. create UML class and sequence diagrams) before implementing the game. You should apply the concepts you have learned so far in the course. Don't have everything in one class, and try to promote code reuse as much as possible. You **must** have at least one use of inheritance in your project.

You also need to be careful when designing the parts for secret codes. We may change our mind about replacing the four digits to four-letter words in future!

Task Two. Implement the game

Discuss with the tutors about your design. Once you are happy with the initial design, implement the first part of the game allowing the player to guess the computer's secret code. The computer randomly generates the secret code at the beginning of the game, which it then lets the player guess. Remember that when generating the computer's secret code, each of the four digits must be different. Note that the player only has seven attempts to guess the secret code. The prompt for player input, results for each guess and the final outcome (i.e. whether the player has won the game or not) should be displayed appropriately to the console.

Show the tutors your implementation so far and compare with your initial design. Note down anything that you like to change or you have changed from the initial design. Now, modify your code so that the player can now also enter a secret code when the game begins, which the computer must guess. Remember to verify that the player has chosen a valid secret code. The player and computer each take turns guessing the other's code. The game ends when either side successfully guesses the other's code (resulting in a win for that side), or when each side has made seven incorrect guesses (resulting in a draw).

Task Three. Reading guesses from a file

Modify your code so that before the game begins, the player is asked whether they wish to enter their guesses manually, or to automatically guess based on pre-supplied guesses in a file.

If the first option is chosen, then the game should progress in the same fashion as in Task Five above. If the second option is chosen, then the following actions should be taken:

Firstly, the player should be asked to enter a filename. If the player enters an invalid filename, they should be re-prompted until they enter the name of a file that actually exists. This file should then be read and interpreted as a text file, where each line contains a separate guess. For example, a sample file `input.txt` may contain the following text:

```
1234
4321
5830
8437
1489
3271
2530
```

You may assume that each line of the file contains a valid guess.

Once the file has been read, then the game should proceed as normal. However, when the player would be prompted to enter a guess, the next guess in the list of pre-supplied guesses should automatically be chosen instead. If there are no more pre-supplied guesses (for example, if the player needs to enter their fifth guess but the file only contained four guesses), then the player should be prompted as normal.

Task Four.

Once you have modified your game to read from a text file, the next thing you might want to do is to save the output of the game into a text file. Modify your game so that the user can save the result to a text file when the game finishes. You should let the user specify the text file name, and the location where the user likes to store the file.

(Extra) Exercise: Reading & Writing Objects

Java has a mechanism called **serialization**, which allows you to read / write entire Java objects from / to files.

For this exercise, *investigate* the use of this serialization mechanism, and use it to read / write the movie list from exercises three and four.

Hint: A good place to start would be to Google “Java serialization”. There are many excellent resources available online which explain how this works.

(Extra) Exercise: Other File Formats

XML and **JSON** are two widely used file formats. Java libraries exist which allow you to read / write files in both of these formats. Like Java serialization, they can be used to write entire Java objects - but in a format that’s human-readable and *platform agnostic*.

Investigate one or both of these formats, and use them to read / write the movie list from exercises three and four.