**KV4001 Week 8File Lab**

The purpose of this lab is to reinforce the material developed in the lecture of file IO. You will also need to make effective use of exception handling techniques. Start by reviewing the lectoral examples. You should ask a lab tutor if there is anything you do not understand.

First complete any outstanding exercises from the lectoral.

Exercises 1 – 7 each develop the previous exercise. When writing the code try and write it so that it handles any possible exceptions either using try / catch or by throwing them.

1. Write a file ‘W8Ex1’ to write the following hard-coded data representing   
   first name, last name, age & result to a text file ‘Ex1.txt’. The name of the file should be supplied as a parameter to the constructor. You may wish to add the ‘.txt’ in the constructor itself.

Ansel Adams 20 49.5

Betty Boothroyd 45 92.3

1. If you have not already done so write W8Ex2 to amend W8Ex1 to test whether the file already exists and if it does to throw an IllegalStateException. Test this by attempting to create the file twice.
2. Write W8Ex3 to amend the previous exercise to produce the same output as the previous exercise’ but obtaining the data using the Scanner class (Hint: using System.in).
3. Write W8Ex4 to amend the previous exercise so data is input until ‘XXX’ is entered.
4. Write W8Ex5 to amend the previous exercise so that if the file already exists, it creates a file with the same filename but an ‘a’ added e.g. if the file was ‘Ex5.txt’ then the resulting file would be ‘Ex5a.txt’. Your code should amend whatever file name is supplied to the constructor.
5. Write W8Ex6 to be able to read any of the text files you have created in the previous exercises and write the contents of the file to the terminal window.
6. Write W8Ex7 to amend the previous exercise so that it reads a file but writes the data to a new text file.

**Help File System – Challenge Exercise**

Here you are going to develop part of a small application that will provide a simple file based help system. Because it is also about working with strings you should start by looking at the PowerPoint slides attached to the lab folder.

**Setup**

Create a new BlueJ project called HelpSystem.

**The Problem Specification**

The application you are going to develop will allow a user to obtain help information on a range of

predefined topics. The actual help text will be held in a text file. As part of the development you will need to put such a file or files together. They can be built by using a simple text editor. One of the first requirements is to define the structure of the help files. A user of your program will be able to:

1. Select a help file
2. List all the topic titles in the help file
3. Obtain help on a given topic
4. Quit the program

Users will communicate with the system via a standard text user interface.

**Help File Structure**

The help file will consist of one or more blocks of text. A block of text has the following format:

|  |
| --- |
| <nameOfTopic>  Help line 1  Help line 2  ...  Help line n  <\nameOfTopic> |

A help file will consist of an arbitrary number of such blocks.

Please note that the various characters ‘<’, ‘>’ & ‘\’ will be used by the program to identify the beginning and end of the help topics and are required.

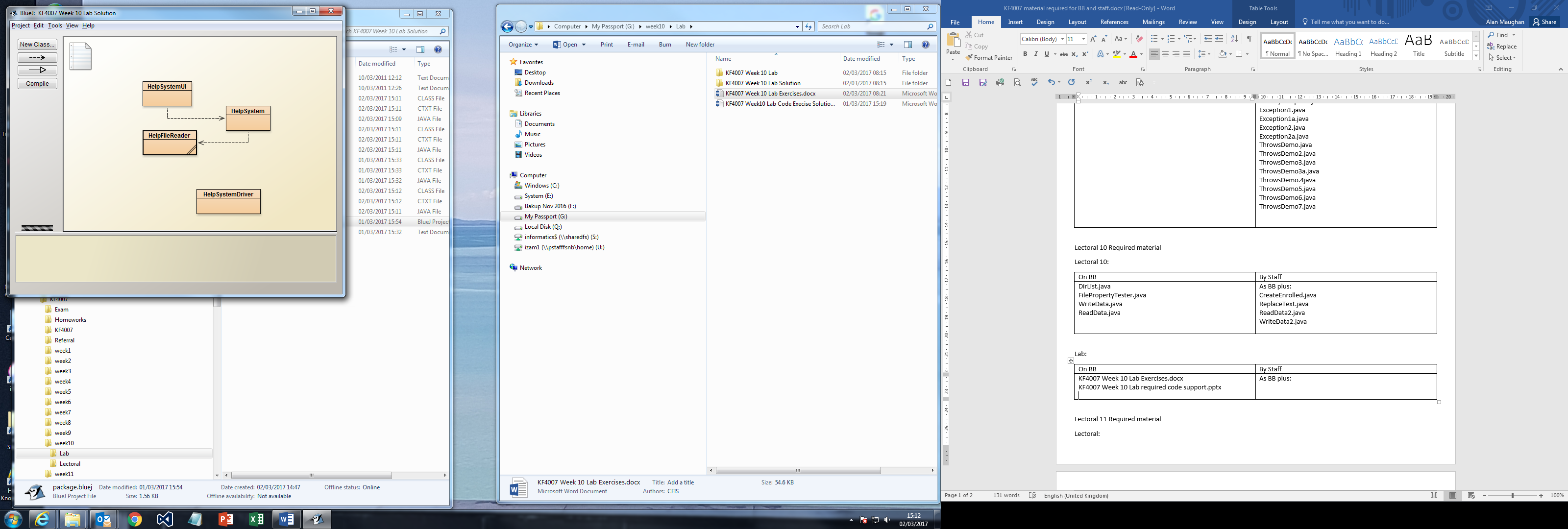
**Task 1**

Page **2** of **4**

Using a text editor, create a sample help file consisting of three blocks. You will use this during your development of the application. This file should be created in the HelpSystem project folder.

**Program Structure**

Normally you would need to think about the structure for your application. But in this example it is given, see figure 1.



*figure 1 : System Architecture*

There are three classes:

|  |  |
| --- | --- |
| _Pic4 | HelpSystemUI  o The text user interface  HelpSystem  o The class that looks after the help responses  HelpFileReader  o A service class encapsulating the essential file handling |

You are provided with full implementations of the first class. During the lab the HelpSystem class will be built. You will need to add code to the HelpFileReader class.

**Task 2**

The given files are available on Blackboard. Copy them and add them to your project.

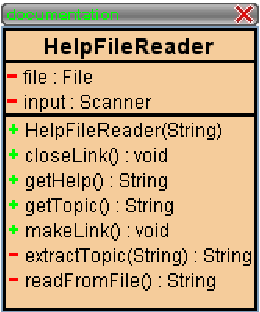
**Task 3**

Page **3** of **4**

Look at the HelpSystemUI class and make sure you understand what it is doing. In particular you should examine carefully the command methods. They contain code that invokes methods, yet to be implemented in the HelpSystem class. Write down the HelpSystem methods you identify.

**Task 4**

Open the HelpFileReader class. This class is heavily based on the work you did in the lectures. It may be represented as:



The class is partially complete. This class is based on the

classes you looked at in the lectures. The extra methods are specific to the current application. These are

|  |  |
| --- | --- |
| _Pic7 | getHelp() getTopic()  extractTopic(String) |

You need to add the code for each.

***Task 4 a extractTopic(String)***

The format of a “topic line” is <nameOfTopic>. The purpose of this method is to extract the nameOfTopic string and return it. To achieve this you will need to look at the String API for suitable methods.

***Task 4 b getTopic()***

This method needs to read lines from the file until it detects a line whose first character is a ‘<’. Since the combination ‘<\’ indicates an end of topic it is necessary to check for a second character being a ’\’.

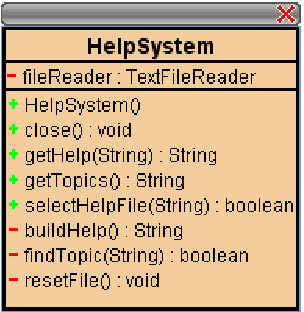
When a topic line is found the topic string should be extracted (using your method from Task 4a) and returned. If the method fails to find a topic line an empty string should be returned.

***Task 4 c getHelp()***

This method reads lines from the file at its present position until it detects a line beginning with the character ‘<’ at which point it returns the lines already read as a String.

**Task 5 Implementing the HelpSystem Class**

This is where the bulk of the work takes place. The following figure shows what you will be expected to produce:



The template for this class is given on Blackboard. Add it to your project. Three key methods are:

* selectHelpFile(String)

o enables the user to pick the file that contains the current help data

o may be used at any time whilst the application is running

* getHelp(String)

o obtains the help lines for a given topic.

o the name of the topic is defined by the parameter

o outputs a message if the topic help cannot be found

* getTopics()

o returns a list of topics

You are dealing with sequential file access. After dealing with an action you will be part way through the file. For the next action you most probably need to start at the beginning again. To do this you need to close the link and reopen it. This is the purpose of resetFile().

The close() method is used to ensure that the file is actually closed at the end of processing.

Your task is to implement the class.

**Extra Tasks**

Two possible areas for you to look at are:

1. The HelpFileReader is based very heavily on the FileReader class (ReadData2 in BetterReadData.java) developed in the lecture. A possible option when implementing the above would have been to make HelpFileReader extend FileReader. Rewrite your classes to do this.
2. Setting up the help files using a text editor is easy but error prone. Develop an application to allow the user to produce help files.

Page **4** of **4**