Practical 6: Simple File I/O

So far you have learned and practiced accepting data from the user through the keyboard (via the `System.in`), manipulating data in your programs (for example adding numbers, merging strings, creating and using variables etc.) and displaying data to the screen (via the `System.out`).

During all the above operations you have been using the computer's *short term memory* or *Random Access Memory (RAM)*. Once your program terminated, all the data used by your program was lost.

In this practical you will have the chance to save data permanently in a *file* that can reside in *long term memory* (hard disk, flash drive, network space etc.)

Please note that in order to be able to do this practical you should finish (and be completely familiar) with the tasks set in the previous practical (Practical 5 - Command Line I/O).

Task 1

Download and extract the **program_6.zip** that accompanies this practical sheet on the KLE.

Open the project in Apache NetBeans and **study**/run the code.

The project contains two classes called `Program` and `File`.

The `File` class is a template for a file object. Each object that derives from this class has only one **attribute** associated with it which is a variable of type `String`. This `String` variable is called `fileName` and it will hold the name of the file that you will be accessing (reading from and writing to). You must initialise this variable right after you create an object from the `File` class. Class `File` has some *behaviour* too. This is implemented in four methods called `appendText`, `readLine`, `createNewFile` and `displayContents`.

You are not expected to understand all the code in these methods at this point but you are strongly encouraged to make an effort to do so. Please ask a demonstrator for help.

As their names imply:
`appendText` is used to append text into the file,
`readLine` reads a specific line (given its number) from the file,
`createNewFile` creates a new (empty) file replacing the old one (if it exists) and
`displayContents` is used to print the entire contents of the file on screen with each line preceded with its number.

The `Program` class in the project contains a `main` method that uses the `File` class. You should be able to deduce how the `File` class is used in the `main` method. Please ask a demonstrator if you need help. Understanding the `main` method and the use (but not necessarily the implementation) of the `File` class is crucial before you proceed to the next task of this practical.

Before you move to the next task locate the file that the program creates and open it with a text editor (such as Microsoft WordPad or Word for example[1]).

Task 2

Use the above example to create a new program (i.e. in a new class) that will ask the user for text input and will subsequently **append** that input in a text file.

The text file should be **incremental**. In other words, every time you run your program the user's input should be **added as a new line** to what is already there.

Verify that your program is doing what it is supposed to by either displaying the contents of the incremental file every time your program runs or by finding and opening the incremental file in a text editor as you have done above. You could even create a completely new program whose only function is to display the contents of the incremental text file.

Task 3

Consider the following program that reads the system's date and time and prints it out to the console:

```
import java.util.Date;

public class Timestamp
{
    public static void main(String[] args)
    {
        Date timestamp = new Date();
        String timestampAsString = timestamp.toString();

        System.out.println("System time is: " + timestampAsString);
    }
}
```

Use the above example to modify your program in task 2 so that each user entry in the incremental text file is preceded with its timestamp (i.e. the date and time it was written in the file).

[1] You can open the text file in Notepad (the default text editor in Windows) also, however please note that Notepad does not interpret the escape sequence `\n` as a newline character and the contents of the file will not appear on different lines. If you want to use Notepad to open your text file then you will need to use `\r\n` instead of just `\n` to insert a new line character.

Task 4 (more challenging...)

Have a go at "breaking" your or my programs in this practical. Breaking in this context is a term often used to mean causing a program to terminate unexpectedly due to a **run-time error**.

This can be achieved, for example by entering data that the program does not expect (or is not asking for) in order to see if it can deal with that eventuality.

How about trying to access (read from or write to) a file that is not there? How can you do this?

If you manage to break the program, have a go at explaining what happens at the breaking point.

This may sound like a pointless exercise but such **testing** is a big part of *software engineering* (the formal approach followed in the design and implementation of software applications). During the testing of an application (which takes place before its release for actual use) specially employed people (the testers) try to break the running computer program. The results of their efforts are reported to the developers of the application who then try to improve it by making it more robust.