Files and Exceptions

Objectives

The objectives of this practical are to:

- Read from files and the ‘keyboard’
- Use exception handling
- Write new exception classes

Reference Material

This practical session is based on the *Files and Exceptions* chapter material.

Overview

There are three parts to the exercise. The first deals with inputting employee names and ages; the second reads the contents of a file from disk; the third introduces a user defined exception class and throws that exception if the Employee’s age is too low (or too high).

Practical

Reading and Writing to a File

1. Create a new class `EmployeeFileTest` for this code, in your existing project and package. Give it a `main()` method – this will be your executable class.
2. Define a `static` method to receive keyboard data back from console, called `getInput()`, taking a `String` parameter for a prompt question. In your method, print a message to the console asking the question, using `System.out.println()`. Declare and create an `InputStreamReader` using `System.in` as its constructor argument. Declare and create a `BufferedReader` using your `InputStreamReader` reference as its constructor argument. Use the `readLine()` method of `BufferedReader` to retrieve your keyboard input and that is the string the method should return. Try and compile, it will fail. The `IOException` needs to be handled or the buck passed to the calling method. Pass the buck! Declare the method as throwing `IOException`.
3. Declare two new `static` methods: an `inputEmployee()` method which will when coded ask for a new employee's details to be entered; and a `showEmployees()` method which will when coded display the contents of the file.

4. From `inputEmployee()` call `getInput()` twice. Call the `getInput()` method, asking for a name, storing the return value in a variable `name`. Then call the `getInput()` method, asking for an age, storing the return value in a variable `strAge`. Convert the return value to an `int` using static method `Integer.parseInt()` and store in the variable called `age`, don't worry about any non-numeric data at this stage. Compile.
5. This method does not compile now as you are calling a method that throws but does not catch an `IOException`. You must catch it or pass the buck! Pass the buck back to `main()`. Try and compile, `inputEmployee()` will compile but `main()` won't.
6. Introduce `try` and `catch()` blocks. Wrap the two calls in `main()` in a `try` block and catch the `IOException` with a block that does nothing i.e. `{}`. Ensure the code compiles. Having seen that you are not actually forced to do anything display "Problem with file handling" to `System.out`. Now run the application supplying a name and a numeric age.
7. Now let's handle non-numeric data. Run the application again this time supplying a non-numeric age. Result: the JVM exits with a stack trace due to the `NumberFormatException` thrown by `Integer.parseInt()`. Code a `catch` block now, that prints:

   ```
   e.getMessage() + " is an invalid age!"
   ```
8. Compile and test again with non-numeric age and see the message.
9. In `showEmployees()` you will read data from a `.txt` file. Somewhere on your file system, create an empty `.txt` file, and declare a `public static final String` variable in your code, with the file's path. Declare and create a `FileInputStream` with your filepath as a constructor argument. Declare and create a `InputStreamReader` using your `FileInputStream` reference as its constructor argument. Declare and create a `BufferedReader` using your `InputStreamReader` reference as its constructor argument.
10. Read a line of the file into a `String` variable which you call `data`. While `data` is not null (`readLine()` eventually returns `null`) print it out to console and read the next line into `data`. Call `close()` on the `BufferedReader` when the loop exits. Compile the code, several errors appear referring to two different exception types. We can 'pass the buck', so you only need to code `throws IOException` as it is the superclass of `FileNotFoundException`. The code now compiles and runs and displays the employees.
11. In `main()`, test ‘File not found’ situation. The `showEmployees()` method potentially throws either an `IOException` or a `FileNotFoundException`. Write code immediately after the catch of `IOException` that is already there a catch of `FileNotFoundException` displaying a suitable 'Not Found' message. Try and compile, it won't as 'code not reachable'. Place the catch immediately

before the more generic catch of `IOException` up above. Test this out with a spurious file name.

12. Code your own `Exception` class called `UnderAgeException`. It should
    - be a nested (inner) class inside `Employee`
    - extend `Exception`,
    - have a `private int age` variable,
    - have a `public int getAge()` method,
    - have a constructor that receives an `int` and a `String`, storing the `int` as the instance variable `age`, and passing the `String` to the superclass constructor.
13. In `inputEmployee()` deal with the age less than 18 problem. If the age is less than 18 then create and throw an instance of `UnderAgeException` passing the invalid age into the constructor. Compile: It fails because the exception is a checked exception that must be handled somewhere. Change the signature of this method so it also `throws UnderAgeException`.
14. Thus, in `main()` catch `UnderAgeException` to display a message with the invalid age that is encapsulated in the exception plus the contents of the `String` passed into the constructor when it was thrown from `inputEmployee()`. Test now supplying an under-age employee.
15. Introduce a `finally` block. We have not ensured that the `BufferedReader` is closed. Place `bfr.close()` in a `finally` clause. Wrap the rest of the method in a `try` block. Compile. It fails because `bfr` is declared in and is local to the `try`. Ensure that the `BufferedReader` is declared (just declared) prior to the `try`, you will also be asked to ensure that the reference is initialised (to `null`) prior to the `try` block. Experiment with this until it works.
16. Put the whole of `main()` in a `while` loop, so that they can potentially enter lots of employees. Keep prompting them "Type 'quit' to exit" and keep looping until they type in 'quit'. But allow them to type ‘quit’ in any case.
17. Write code to output all employees back to the file, so that the data is persisted.