Programming Exercises - PRO1 - Session 24

Exercise 24.01

Write a program that asks the users to input names of all their friends. The program should write all the names to a file specified by the user. The first keyboard input is the name of the file, and the following inputs are names of friends. To finish the keyboard input type DONE. Then try to open the file using Notepad or similar to see its content.

An example run could be the following (bold text is user input):

Type filename: **friends.txt**
Type name of friend: **Bob**
Type name of friend: **Carl**
Type name of friend: **Dave**
Type name of friend: **DONE**

Now the file “friends.txt” should contain the following:
Bob
Carl
Dave

Exercise 24.02

Write a program that can read and print out all contents of any text file. The program should ask the user for the name of a text file, open that file, and then print out every line from the file.

If the file is “friends.txt” from the previous exercise (if that’s what you called the file) the output could be:

Contents of the file friends.txt:
Bob
Carl
Dave

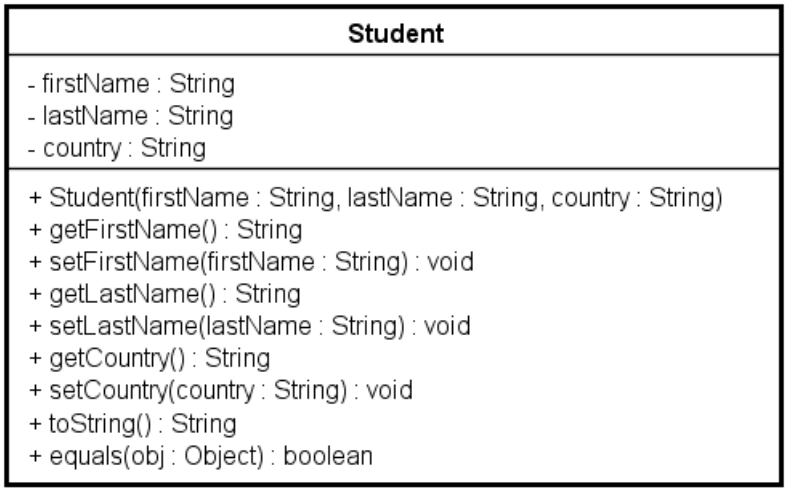
Exercise 24.03

Implement a program that generates some random integers and writes them to a binary file. First the program should ask the user to input how many integers it should make.

Then implement another program that can read the binary file and print its contents to the screen.

Exercise 24.04

Create a class Student, as shown in the class diagram below. The class should be serializable so that objects made from it can be saved in binary files (just add `implements Serializable` in the class header).



Now create a test class that creates some `Student` objects and saves them to a binary file. Afterwards create another test class that can read the created file and print out the information about the students. You can also do both in the same test class if you prefer.

Exercise 24.05

XML (Extensible Markup Language) is a popular text markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It's often used for storing data, or for exchanging data between various programs and web services. XML is structured very similar to HTML, in that it consists of various nested tags. Unlike HTML, there is not a fixed list of available tags, but rather custom tags are created based on the data that needs to be stored. The first line in an XML file is always an XML declaration in this format:
`<?xml version="1.0" encoding="UTF-8"?>`

Create a program that can generate an XML file with data from a list of `Student` objects.

E.g. from this data:

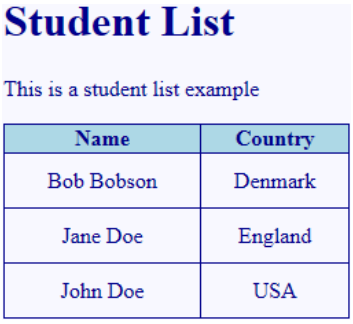
```
ArrayList<Student> list = new ArrayList<Student>();
list.add(new Student("Bob", "Bobson", "Denmark"));
list.add(new Student("Jane", "Doe", "England"));
list.add(new Student("John", "Doe", "USA"));
```

your program should be able to save the follow content to a text file named studentlist.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student>
    <firstname>Bob</firstname>
    <lastname>Bobson</lastname>
    <country>Denmark</country>
  </student>
  <student>
    <firstname>Jane</firstname>
    <lastname>Doe</lastname>
    <country>England</country>
  </student>
  <student>
    <firstname>John</firstname>
    <lastname>Doe</lastname>
    <country>USA</country>
  </student>
</students>
```

Exercise 24.06

The purpose of this exercise is to generate an HTML file with a table containing data from a list of `Student` objects. E.g. from the data below, we want to generate the HTML page shown to the right




```
ArrayList<Student> list = new ArrayList<Student>();
list.add(new Student("Bob", "Bobson", "Denmark"));
list.add(new Student("Jane", "Doe", "England"));
list.add(new Student("John", "Doe", "USA"));
```

The textbox below shows the template for an HTML file. It contains the following placeholders that can be replaced with the actual data we have:

- `$title`: the title of the web page.
- `$header`: the header title shown in the body.
- `$paragraph1`: a text/paragraph in the body.
- `$tableHeader1`, `$tableHeader2`: header titles for each of the columns in the table.
- `$tableData1`, `$tableData2`: data (name and country) for one of the `Student` objects in the list. Note that this line must be repeated for each of the Student objects.

```
<html>
  <head>
    <title>$title</title>
    <style>
      body
      {
        background-color:ghostwhite;
        color:darkblue;
      }
      table,th,td
      {
        border:1px solid darkblue;
      }
      table
      {
        border-collapse:collapse;
        width:30%;
      }
      td
      {
        height:40px;
        text-align:center;
      }
      th
      {
        background-color:lightblue;
      }
    </style>
  </head>
  <body>
    <h1>$header</h1>
    <p>$paragraph1</p>
    <table>
      <tr><th>$tableHeader1</th><th>$tableHeader2</th></tr>
      <tr><td>$tableData1</td><td>$tableData2</td></tr>
    </table>
  </body>
</html>
```

Save the content of the textbox in a file named “template.html”, and then write a Java program doing the following:

a) Create an `ArrayList` with a few `Student` objects of your choice (e.g. just by using the statements shown in the example above).
b) Read the template HTML file (as a text file) and add each line from the file as a separate string in an `ArrayList<String>`.
    a. When you read a line with one of the placeholders `$title`, `$header`, `$paragraph1`, `$tableHeader1`, and `$tableHeader2`, replace the placeholder with a proper value before inserting it in the `ArrayList`. Hint: use the method `contains` from the `String` class to check for a placeholder, and the `replace` method to replace the placeholder with the actual content, e.g.:

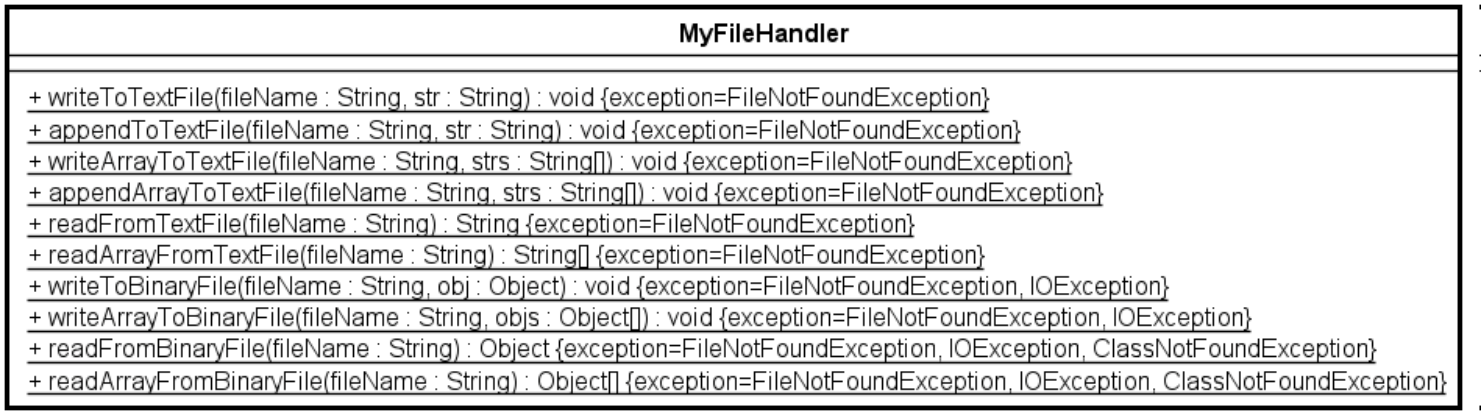
```
if(line.contains("$title"))
{
  line = line.replace("$title", "Student List");
}
```

    b. When you read the line with the placeholders `$tableData1` and `$tableData2`, repeat adding the line to the `ArrayList` for each of the objects in the list of students, each time replacing the placeholders with data from one of the objects.
c) After reading and replacing, your `ArrayList` should now contain the final HTML code for the list of `Student` objects. So, go through the `ArrayList` with a loop, and write one string at a time to a text file (with extension .html)
d) Open your newly created .html file in a browser, and rejoice ☺

Exercise 24.07 (optional)

To avoid writing all the low-level code for accessing files every time you need to read/write files, it could be smart to wrap it all inside a general-purpose reusable file class.

Implement the class `MyFileHandler` as shown in the UML class diagrams below.
All methods use a throws clause (as indicated by the curly brackets in the class diagram) so there are no try-catch blocks. Any class calling these methods will be responsible for catching the exceptions. All methods are also static (as indicated by them being underlined).



Afterwards, make a test class to test out the functionality of the class. You could e.g. also modify some of the previous exercises from today to now use the methods in the `MyFileHandler` class.