Working with the file system

Martin Pola

1 Navigate through the file system

Open a terminal and run the command to print the current working directory (see the slides!). List what's in your current directory by using the `ls` command on Unix-like systems, or `dir` on Windows. In IntelliJ, create a new module or project, and then create a class named `FileEditor`. See to that the class contains a main method like this:

```
1  class FileEditor
2  {
3      public static void main(String[] args) {
4          System.out.println("Working in: " + System.getProperty("user.dir"));
5
6          for (String a : args) {
7              System.out.println(a);
8          }
9      }
10 }
```

Using the `cd` command, navigate to the newly created folder. Compile file file and run it from your terminal:

```
$ javac FileEditor.java
$ java FileEditor
```

Try to run it with a few arguments and make sure that they're being printed:

```
$ java FileEditor arguments one two three
```

2 Read a file

Create an empty text file somewhere in your file system (i.e. in any folder/directory) and add a few lines of text to it. Make sure to have at least five lines.

In your `FileEditor` class, prepare a new `ArrayList` to hold strings.

Add code to read the content of your text file to one single string. Split the string on line breaks (e.g. using the `String#split` method) and add each line as a separate string to your `ArrayList` object.

When you have done so, you should be able to print the contents of the file, line by line, like this:

```java

// initialize the "lines" ArrayList

// code to read the contents of a file and populate the "lines" object

// print the contents
for (String line : lines) {
    System.out.println("This is a line: '" + line + "'");
}
```

3 Write to a file

Now, add a string to the end of your `ArrayList` object, e.g.:

```java
lines.add("A new last string at the very end");
```

Write some code to export all the contents from the list into one single Java string. Think about line endings. How would you handle those? What would you like your final string to look like, given a sequence of strings in your `ArrayList`?

Save the string to your text file, overwriting the previously stored data.

You can verify that you've done the right thing by looking at your text file again. If the contents remains the same, but with one extra line added, you've done it right!

4 A useful program – the line editor

Create a new program that lets the user give the name (path) of a file they wish to edit. You can use a `Scanner` object to ask the user for that information.

Then, open the file and print the contents, line by line. Add line numbers to the beginning of each line.

When the file has been printed, ask the user for the following two pieces of information.

- The line they wish to edit
- The new data that should be stored on that line

Save the data in two variables, edit the internal contents of your `ArrayList` for the specified line number (index in the list).

Convert the list to a single string, like in the previous exercise, and save it to the text file. Can you see that all other lines are kept as they were, with the user-specified line being edited?

5 Exporting `Book` objects

Now that you've been able to successfully read and write with files, add a method to your `Book` class to export an instance of the object to a file. Let the method take a string argument to specify the name (the path) of the output file.

Since you can only write strings (or bytes) to files, you would have to *encode* your `Book` data in some way – you can't export the instance directly. One idea could be to encode the data like this, separating the instance variables by tabs:

```
1 String encodedObject = productId + "\t" + title + "\t" + author + "\t" + price;
```

Create a `Book` object, populate it with data and call your new method to save it to disk.

6 Importing `Book` objects

Let's do the other way around – given a text file with information about a book, specified in the format from the previous exercise, create a `Book` object.

To do so, create an overloaded (a new) constructor that takes just one argument; a string with the name (the path) of the text file where the book information is stored.

In the constructor, open the file, read the data, split it on tab characters and copy the data to the instance variables in the `Book` class.