Tutorial 6: IO and Threads

CSE1100 - Introduction to Programming

1 Basic Reader

You are given a basic `Store` and `Product` class and a `StoreApplication`. The `StoreApplication` uses the `read` method in `Store` to read *resources/store.csv*. Implement the `read` method.

2 Basic Writer

You are given four basic classes. `WeatherReport` has a `write` method. This method should write the weather report to the given writer in the following format:

```
3/8/2020,RAIN,20C,1BFT,20mm
4/8/2020,SUN,25C,2BFT,12h
```

Here on August 3rd 2020 it was a rainy day. It was 20 degrees, there was a wind speed of 1 beaufort and 20 millimeters of rain fell. On August 4th it was a sunny day. It was 25 degrees, there was a wind speed of 2 beaufort and there were 12 hours of sun. One test is provided. Writing more tests is encouraged.

3 Complicated Reader

You are given the following classes:

- `CoffeeMenu`: This is a menu for a coffee bar. It contains all the coffees customers can order. It also contains all the additions a customer can put in their coffee, for example vanilla or sugar.
- `CoffeeItem`: This is an item on the menu, e.g. cappuccino or espresso. It contains a list allergies and a list of possible variations, like iced or double.
- `CoffeeVariation`: This is a variation on a coffee item.
- `Addition`: This is an addition a customer can potentially add to their order, regardless of what coffee they order.

Implement a the `read` method in `CoffeeMenu`. For the format see *resources/coffee.txt*. It starts with *COFFEES*, followed by the list of coffees. These in turn start with *COFFEE,name,price,optional list of allergies*, followed by the list of variations for that coffee. Each variation simply contains the name and extra cost. After all the coffees, the additions start with *ADDITION*. Each addition then is of the form *ADDITION,name,price,optional list of allergies*.

3.1 Testing

Verify your read method(s) work with tests. Write at least one test for reading every class.

4 Serializable

We want to make a brand new RPG in Java, so we have started with a `Character` class. This class has some attributes: `name`, `level` and `xpPoints`. It also has a special field `playerSecret`. This secret will be used for some internal magic in our game, but as you can see by the `equals` method it does not contribute to the equality of two characters. The secret will be randomly generated once for every character.

4.1 Reading and writing characters

Because we want to use this class in a game where it might make sense to not use too much storage space, we will store the characters as binary data. Have a look at the `testCharacterReadAndWrite` test. Try to understand what is happening.

Now run the test, it should fail. What did we forget in our `Character` class? Fix the `Character` class such that the test passes. Don't worry about the second test yet, we will fix that in **??**.

4.2 Keeping our secrets secret

Because we want to generate a new secret every time we save and load our game and because we do not want malicious users to have our secret, we do not ever want to save the secret to a file.

This is why we have the second test `testCharacterDoesNotSaveSecret`. Take a look at what it does and run it. This test will also not pass, but for a different reason: Java will by default store all fields of the class, including `playerSecret`. Luckily for us there is a keyword to fix this. Try to find out how we can fix our problem, without implementing a custom read/write method.