CS2001 160003253 25th September 2017

Overview

This week's practical was to implement the method bodies in the classes provided (following the interfaces we were given), which represent a system for vending machine product management. In addition, we had to write tests with the use of JUnit.
As an extension, I added parameters to the buy/add methods to make it easier to change the quantity of items in the vending machines. I also created an interface for customers and implemented it, along with a system for payment. Finally, I created a simple GUI for a possible instantiation of the package.

Design & Implementation

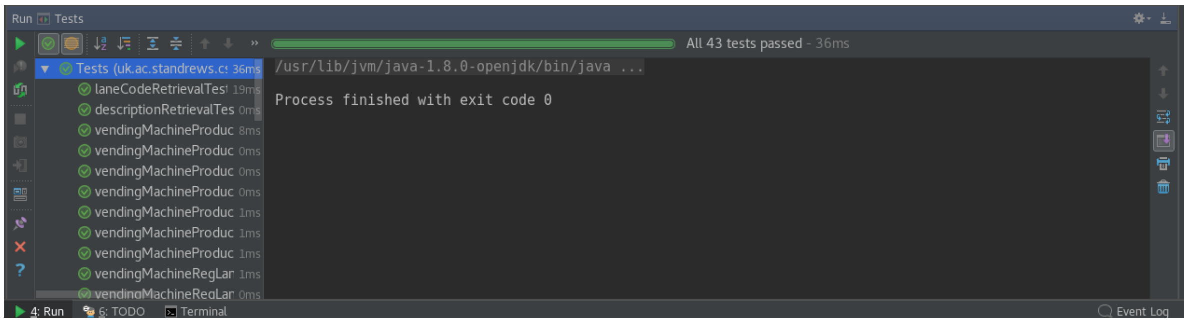
After reading all the method descriptions in the interfaces, I started off by writing the setup and teardown methods for my Tests class. I then added a few simple tests for the methods in the VendingMachineProducts class and some for registering products within vending machines.
I soon realised that I needed to implement the methods in order to grasp the interaction between the classes.
I started off with the shorter classes, adding any attributes I found fitting, and implementing their getters and class constructors.
The biggest challenge for me was how to connect products, vending machines and records together. My approach was not exactly elegant, as I chose to add a HashMap in the VendingMachine class that stored vending machine products as keys, and product records as values. This meant that when I had to access lane codes, or the methods from the ProductRecord class, I needed to search the objects within the Map. I made a private method for searching lane codes to avoid code repetition. In my implementation, the ProductRecord class never needs to be directly accessed, only through the VendingMachine class. Once I implemented all the necessary methods, I ran the tests I made in advance, and was happy to find that they passed. I then proceeded to write more tests, and ran all the tests after adding each one.
Once I confirmed that my base code worked well, I decided on what extensions to do. I remembered that I didn't like using the add/buy methods multiple times during tests, so I included a quantity parameter in both methods and modified the tests accordingly.
Upon confirming that it worked, I created a Customer class and added prices to the products. Expanding the practical was quite challenging, because I had to modify every class in some way or another: adding a price attribute to VendingMachinePruduct, a constructor for ICustomer into Factory, and strongly modify both buy methods in the VendingMachine and ProductRecord classes. This rendered most of my tests useless, therefore I commented out their vast majority and wrote new ones/modified the old tests to see if the code still worked properly.
As I still had time, I attempted to create a GUI. I used NetBeans to arrange the different elements of the user interface visually, then moved the VendingMachineGUI class into the package and added implementations for the methods. The GUI is divided into 3 main panels. The one on top is for the customers, where one can add money to their account. The middle section shows the products in the vending machine, and enables the customers to buy them, if they have enough money. The one on the bottom is there to manage the stock, and to keep track of the sales (for reference, see Examples).

Testing

For the base code, I first wrote tests for basic functionality and Exceptions, then went on to focus on the edge cases. My code passed all stacscheck tests.
For the first two extensions, I had the same approach, but I had to test the GUI manually, by trying to enter invalid inputs, and checking that the various labels change their contents as they should.

Examples

JUnit Tests output:



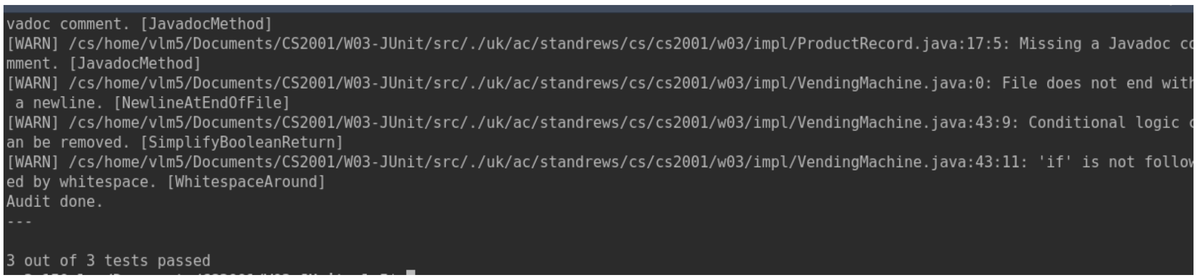
```
All 43 tests passed - 36ms

Tests (uk.ac.standrews.cs 36ms
    laneCodeRetrievalTest 19ms
    descriptionRetrievalTes 0ms
    vendingMachineProduc 8ms
    vendingMachineProduc 0ms
    vendingMachineProduc 0ms
    vendingMachineProduc 0ms
    vendingMachineProduc 1ms
    vendingMachineProduc 1ms
    vendingMachineProduc 1ms
    vendingMachineRegLar 1ms
    vendingMachineRegLar 0ms

/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...

Process finished with exit code 0
```

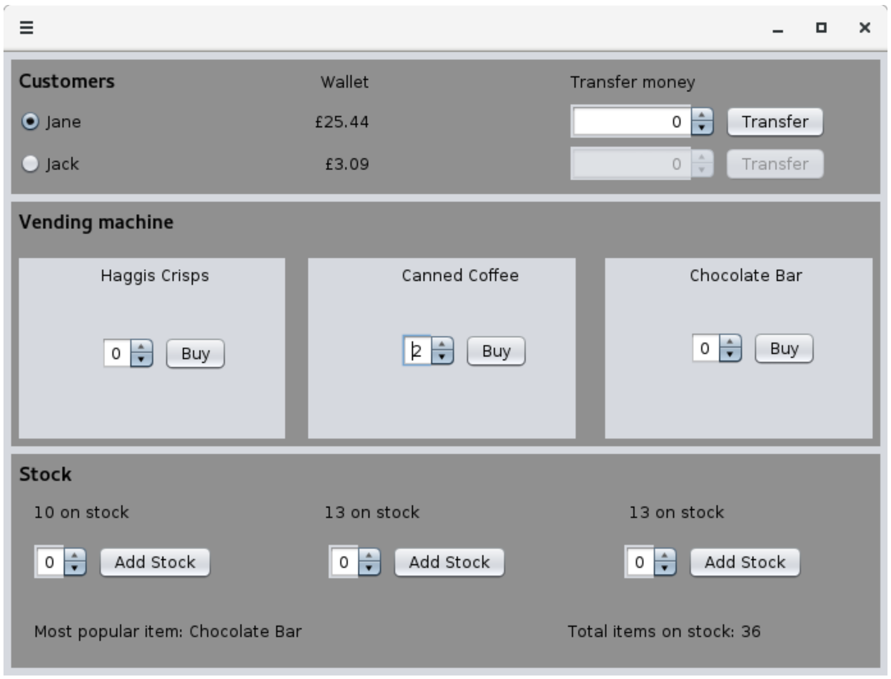
Stacscheck automated checking output:



```
vadoc comment. [JavadocMethod]
[WARN] /cs/home/vlm5/Documents/CS2001/W03-JUnit/src/./uk/ac/standrews/cs/cs2001/w03/impl/ProductRecord.java:17:5: Missing a Javadoc co
mment. [JavadocMethod]
[WARN] /cs/home/vlm5/Documents/CS2001/W03-JUnit/src/./uk/ac/standrews/cs/cs2001/w03/impl/VendingMachine.java:0: File does not end with
 a newline. [NewlineAtEndOfFile]
[WARN] /cs/home/vlm5/Documents/CS2001/W03-JUnit/src/./uk/ac/standrews/cs/cs2001/w03/impl/VendingMachine.java:43:9: Conditional logic c
an be removed. [SimplifyBooleanReturn]
[WARN] /cs/home/vlm5/Documents/CS2001/W03-JUnit/src/./uk/ac/standrews/cs/cs2001/w03/impl/VendingMachine.java:43:11: 'if' is not follow
ed by whitespace. [WhitespaceAround]
Audit done.
---

3 out of 3 tests passed
```

GUI Extension:



Jane £25.44 0
Jack £3.09 0

Haggis Crisps 0
Canned Coffee 2
Chocolate Bar 0

10 on stock 0
13 on stock 0
13 on stock 0

Most popular item: Chocolate Bar
Total items on stock: 36

Difficulties

Debugging caused me the most difficulties, as it often seemed as if nothing was wrong with my code, and yet it failed the tests. In particular, the getMostPopular method always returned null, and after spending almost an hour on it, I found out that I simply typed in the wrong variable into an if statement. It was a valuable life lesson, as I wrote that piece of code late at night.

I found it challenging to work with so many classes, and implementing the interface was often frustrating, as I would have chosen a different approach at times. I was very satisfied once I figured out a way to implement it.
The GUI was also tricky to use at first, but the code I needed to write was quite simple once I figured out how the Swing library worked.

Conclusion

Overall, the program fulfils its intended purpose in keeping track of the various products stored in vending machines.
Writing the JUnit tests forced me to think of many scenarios of my code potentially failing, which is definitely a useful experience when debugging code in the future - although writing the tests themselves was somewhat monotonous. JUnit helped me pinpoint the methods where the code failed, so it’s a useful tool, that I may use in the future for bigger projects.
The extensions were interesting to do, as I modified the interfaces and classes considerably. I tested them again, although less thoroughly this time, because I knew that most of the code worked already.
It was my first time implementing a GUI. I have used some before in Python, but I’ve never made any myself, so NetBeans really simplified the positioning of the various objects. It took care of all the visuals, so I could focus on getting the code working behind it, which I believe works well. The GUI itself is very simplistic as the user can’t add machines or customers, only work with the ones I provided, and it doesn’t use all the methods from the other classes either. But having so few features meant that I could do more thorough testing, and focus on minor details.
This practical was definitely a useful experience, as I managed to make progress quickly and had time to do a lot of experimentation.