The final state of my program consists of four classes (Wine class, CustomerAccount class, CustomerAccountException class and a class containing the main method).

The Wine class contains two constructors. One that takes no parameters and when initiated, includes an array as an instance variable. A separate constructor creates wine objects with instance variables – name, quantity and price per bottle (ppb) – resembling each transaction. I initially didn’t consider using an array, thinking that using the name of the wine from the user input as a reference to a new wine object would be enough. However, I realised this could cause problems for example, the user could potentially enter the same wine name on more than one occasion but with different quantities and prices, therefore re-point pre-existing references to new objects. I chose to use an array to overcome this problem, storing each new object from each transaction individually in a separate index.

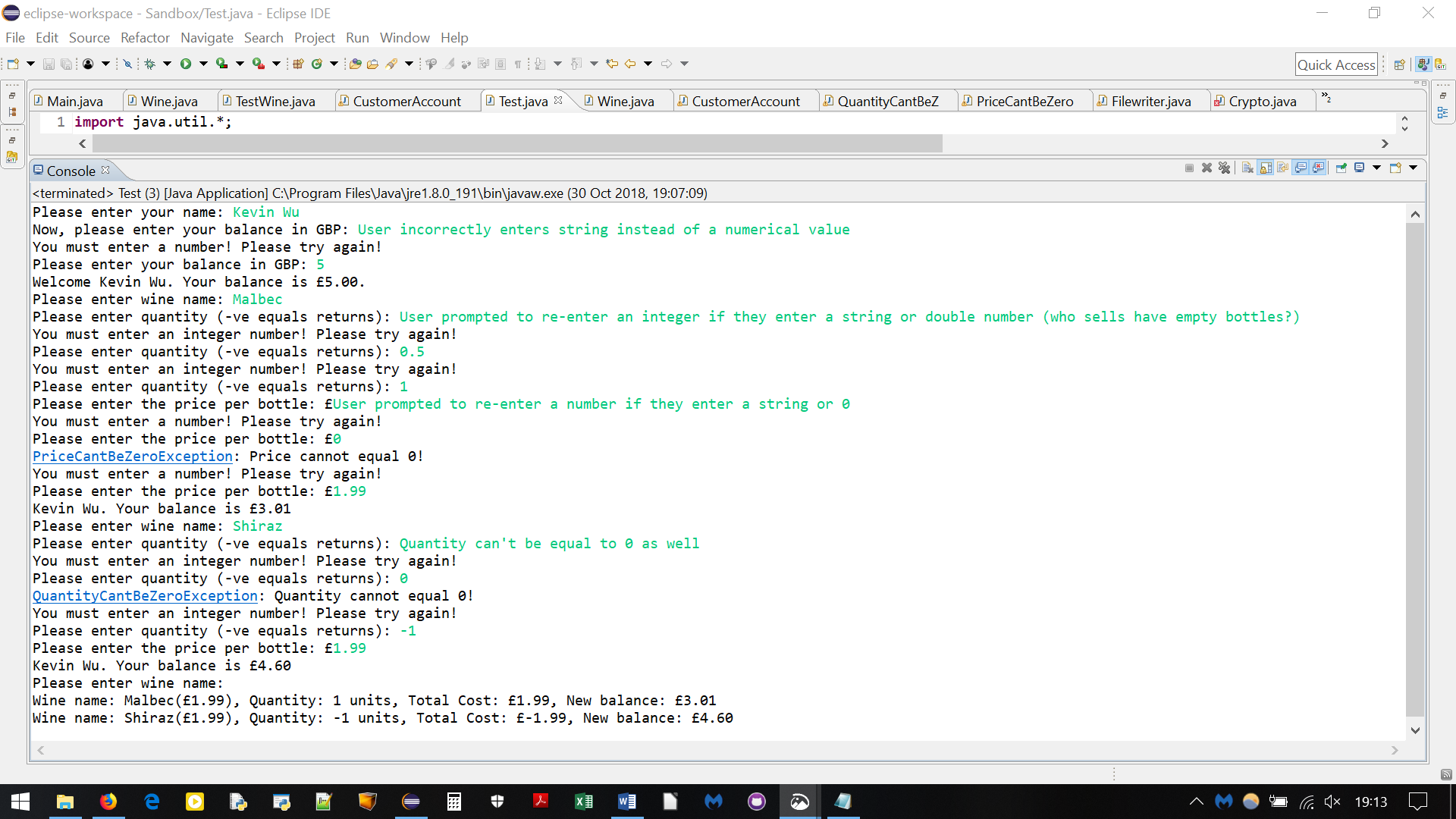
The wine class has a method to add Wine objects. It checks that the array has space before adding another wine object – name, quantity and ppb – by considering the number of transactions carried so far and checking for equality with the size of the array. If there isn’t enough space, it’ll jump to the expandArray() method which grows the array size by one – (Figure 6)

Finally, the wine class has a print method, which prints the transaction history to a file on the desktop. This methods re-sizes the array so that it only contains the transaction/wine objects in each index leaving behind no ‘null’ ones (i.e. if the customer doesn’t make 999 orders). If it tried to print without doing so, a null pointer exception would be thrown – (Figure 5). An object reference of the type CustomerAccount (i.e. the user details) is passed to it so that can access methods within that class. I also included a System.out.println() to print to console just to make sure I was getting the right details. I initially struggled to get the correct running balance as the getBalance() method would return the final balance after all transactions had been completed. I overcame this by resetting the customer’s balance back to the initial balance (i.e. number user came up with at the start) and going through the transactions with the wine objects stored in the array one by one again.

In the print method, the user is given the chance to input the directory and name of the file that they’d like to write the output file to if the String “filename” or (“outputFile” as it’s called in my program) is incorrect. The method catches this as an IO exception or if the user specifies a directory rather than a file – (Figure 7). I was concerned that the user may accidentally specify a file path that already contains a file of the same name and had considered coming up with a method to check that the path is writable and to prompt the user whether they’d like to overwrite a file of the same name or if they preferred to input another different file path (by using ‘y’ or ‘n’) – this turned out to be far more complicated than I initially hoped so these checks were left out of the final program.

Various methods involving user input were initially in the main method but I moved them to this class for clarity. An object reference of the type CustomerAccount is initially set up in main to have access to methods within that class. A new object is returned containing the username and initial balance

The ‘CustomerAccountException’ has been included to throw a unique exception when the user completes simple mistakes such as entering £0 for price or quantity or entering a fractional number for quantity (Figure 1). I did have concerns that the user may not enter a valid string for a name (i.e. mistakenly include special characters, unnecessary spaces and/or number) and used a for loop to check a string made of special chars (Figure 3).



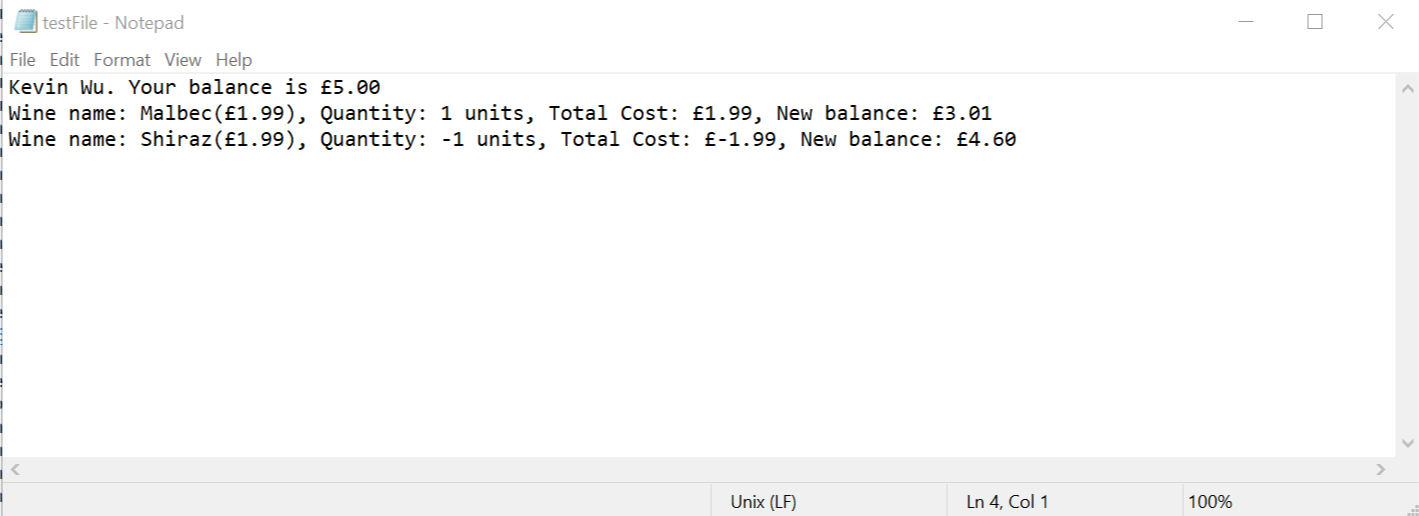


Figure 2: Output from inputs shown in figure 1 (user errors not displayed in output file)

Figure 1: Eclipse Console showing user input and potential errors

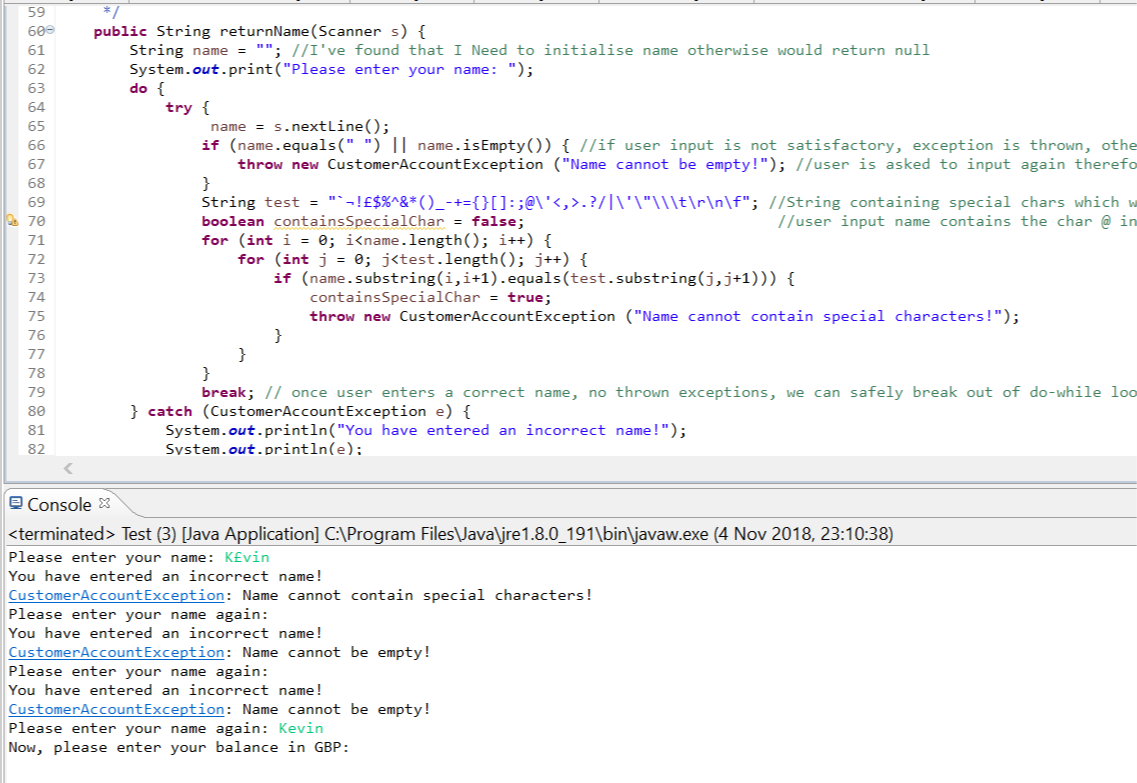


Figure 3: Checking valid name input

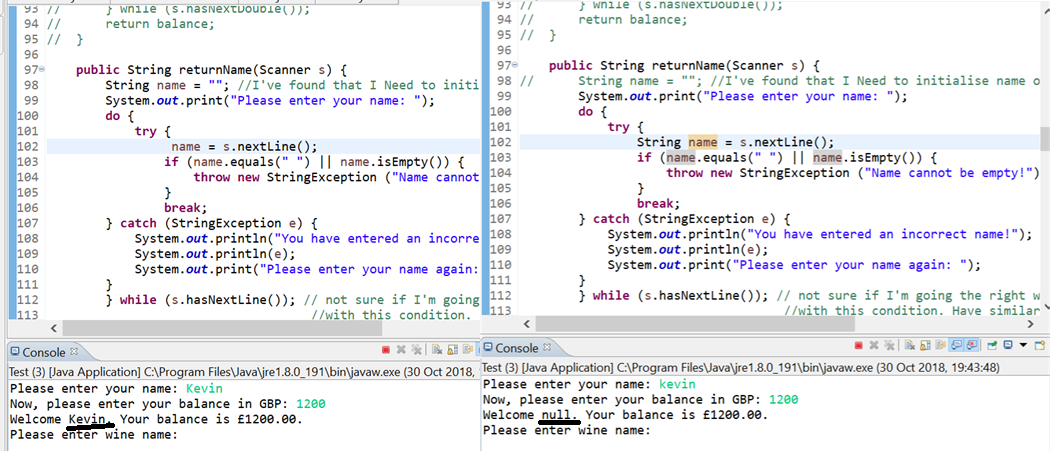
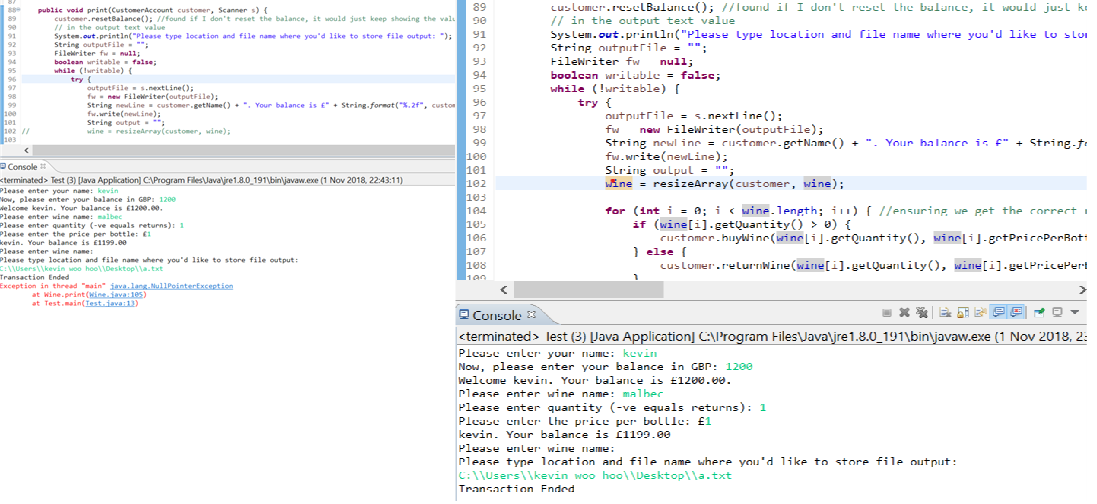


Figure 4: String name requires initialisation otherwise defers to a null value

Figure 5: Null Pointer Exception when trying to print an array containing null values - requires the resize method

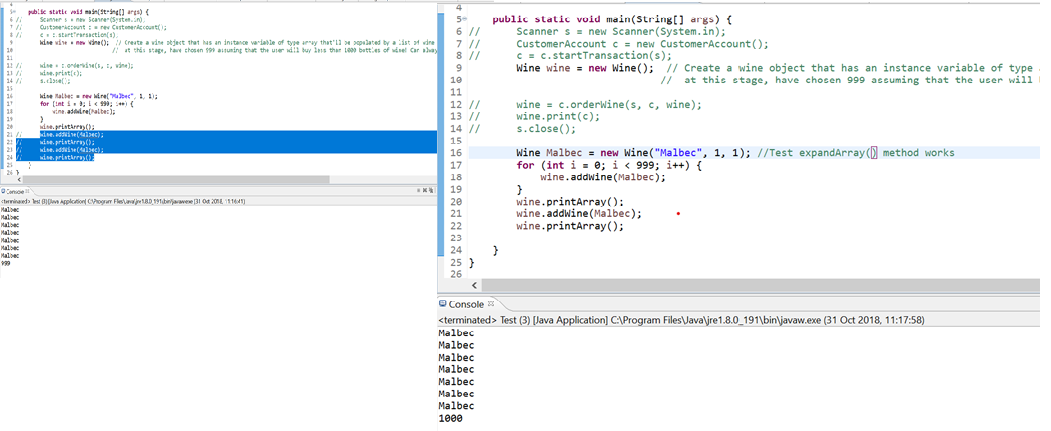
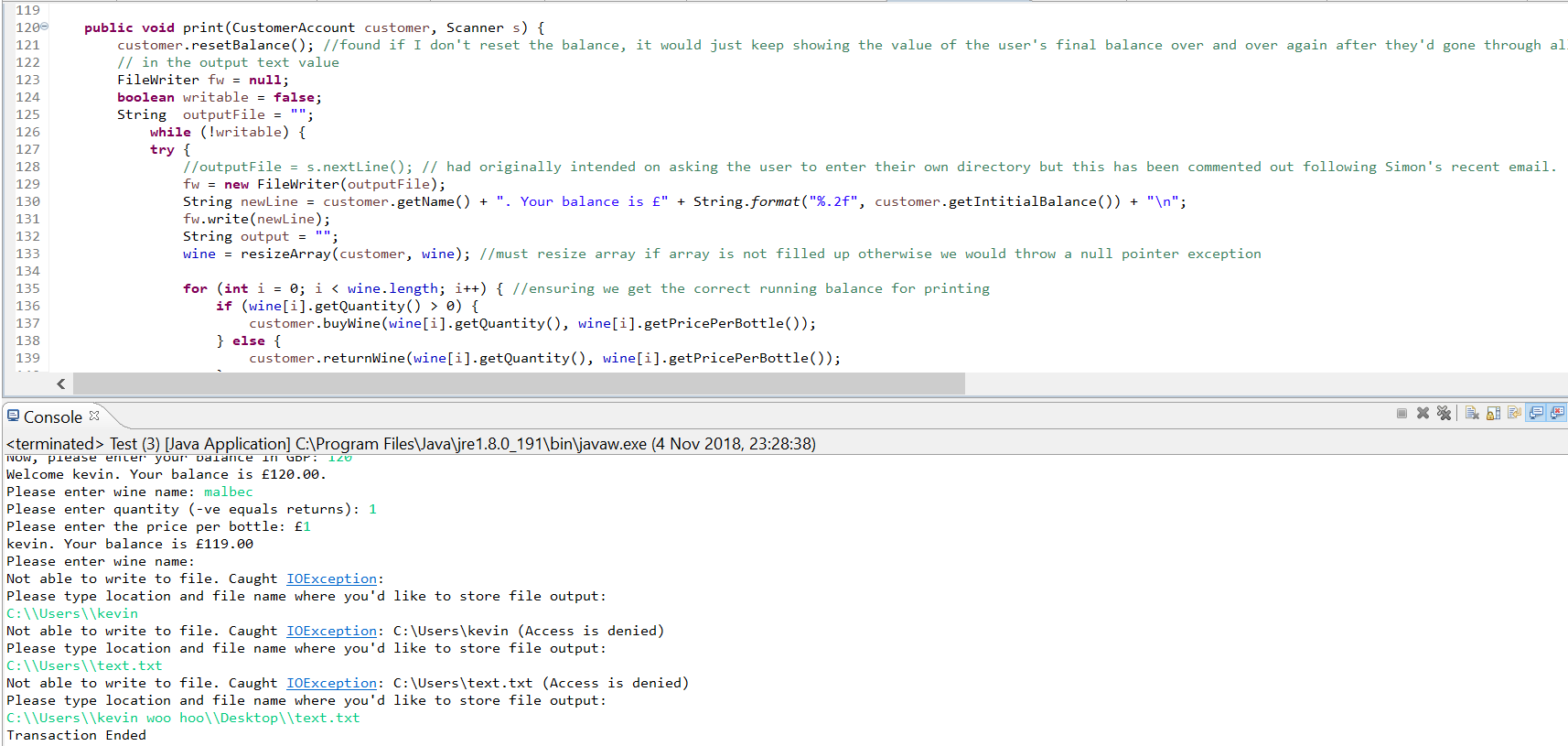


Figure 6: Checking expandArray() by printing it out

Figure 7: Catching the IOException if filename is incorrect and then letting user decide where to store the file.