Assignment 3

ADTs and File I/O

Submit a single ZIP file called **assignment3.zip** containing your IntelliJ project. This assignment has 100 marks. See the grade breakdown that is posted on the assignment page.

The base code for this assignment contains some initial code for an electronic store program. This code includes an ElectronicStore class, a Product class, and a number of descendants of the Product class representing specific types of products. The ElectronicStore class currently uses an array to store up to MAX_PRODUCTS products. Within this assignment, you will be modifying the ElectronicStore to make use of ADTs for storage. You will also be adding customers into this program and implementing some additional functionality. The remainder of this document outlines the changes you should make to the existing code and the functionality you will be required to add. Throughout the implementation of your assignment, you should continue to apply the principles of OOP in a way that increases your code quality. There are two test classes provided on the assignment page that you should use to test your methods. These test classes perform minimal testing. Providing further testing of your own is advisable. You should include any additional test classes you have created with your assignment submission.

Customer Class:
Add a Customer class to the project that will be responsible for tracking the purchase history of a customer within the electronic store. The class must support the following constructors and methods:

1. Customer(String name) **–** Constructor used to initialize the customer with the given name.
2. String toString() – Should return a string of the format "***{name}*** **who has spent $x"** where *{name}* is the customer's name and x represents the total dollars that customer has spent on electronics.
3. void printPurchaseHistory() – Prints out a list of products the customer has bought. This list should contain each unique product once and should indicate how many units of each product the customer has purchased (e.g., 4 x Horn Pass Fridge with Freezer (Red, 111 watts)).
4. String getName() – A basic get method that returns the name of the Customer.

The above methods must be implemented following the given specifications. Note, however, that you will have to add additional state/methods into the Customer class to support the remainder of the assignment requirements (e.g., you will need to keep track of the purchase history for the customer).

ElectronicStore Class:
You will have to update the existing ElectronicStore class and add in additional state/methods. The final ElectronicStore class will keep track of both Product objects and Customer objects. Below is a summary of the changes and additions you must make:

1. Change the existing code so that is uses ADTs for expandable storage of Product objects instead of a fixed-size array.
2. Create a boolean registerCustomer(Customer) method that will add the given Customer object to the ElectronicStore. It should not be possible to add two customers with the same name (i.e., the customer names within a store are unique). This method should return true if the Customer was added successfully and false otherwise.
3. Update the boolean addProduct(Product) method in the included code to work with your ADT-based storage. It should not be possible to add duplicate products. For the purposes of this assignment, two products should be considered identical if their toString() methods return the same strings.
4. Add a List<Customer> getCustomers() method that returns a List of Customer objects that are registered with the ElectronicStore. This list does not need to be sorted in any way but should only contain one copy of each Customer registered within the store.
5. Add a List<Product> searchProducts(String x) method. This method should return a list of matching Product objects in the ElectronicStore. A Product should match the search query if the toString() method of the Product contains the given String x. This search should be done in a case-insensitive manner (e.g., “DO” matches a product with toString() of “dog”).
6. Add a List<Product> searchProducts(String x, double minPrice, double maxPrice) method. This method will work like the searchProducts(String x) method but must only include Product objects where minPrice <= Product Price <= maxPrice. If the minPrice or maxPrice inputs are less than 0, they should be ignored. As one example, searchProducts(“dog”, -1, -1) would be equivalent to searchProducts(“dog”). As another example, searchProducts(“dog”, 10, -1) would find all products containing “dog” in the toString() that have a price of 10 dollars or more.
7. Add a boolean addStock(Product p, int amount) method. If the given product exists in the store, this method should increase the stock amount of that product by the included amount. The method should return true to indicate that the product stock was updated and false otherwise.
8. Add a boolean sellProduct(Product p, Customer c, int amount) method. This should simulate selling the given amount of units of Product p to Customer c. The method should return false if the sale was not successful (p is not in store, c is not registered with the store, and/or there are not enough units in stock to complete the sale) and true if the sale was successful.
9. Add a List<Customer> getTopXCustomers(int x) method. This method must return a list of the x customers that have spent the most money at the store. This list should also be sorted from the highest to lowest amount spent. If x <= 0, return an empty list. If x is greater than the number of customers in the store, return a list with all customers sorted from highest to lowest amount spent.

10. Create a boolean saveToFile(String filename) method that saves the entire store state (including products and customers) to the file with the given name. The method should return true if the store was successfully saved and false otherwise. You are free to use any of the file input/output options covered in the lectures.
11. Create a static ElectronicStore loadFromFile(String filename) method that loads a store and its associated customers/products from a file with the given name. The method should return the loaded ElectronicStore object if the store is successfully loaded. If any errors occur during the loading process, the method should return null.