COMP0004 Object-Oriented Programming
Programming Exercises 4

Purpose: More practice of using Java by working on editing and extending an already partly implemented program.

Goal: Complete as many of the questions as you can. If you are keeping up, you need to do at least the core questions. The additional questions are more challenging and are designed to stretch the more confident programmers. If you can't do them now, be prepared to come back and try them later on. You should complete at least the core questions and get them reviewed in a lab session.

Feedback: It is important that you get feedback on your answers so that you know they are correct, that you are not making common mistakes, that the program code is properly presented and that you are confident you have solved the problem properly. To do this, get your answers reviewed by a lab demonstrator during lab sessions.

Getting Started

These exercises require you to work with the example SimpleOrderSystem program. Create a project by cloning the code from the GitHub repository:

https://github.com/UCLComputerScience/COMP0004SimpleOrderSystem

There is a video on Moodle showing how to do this if you are not sure how.

Exercise Questions

Each of these questions involves working with, modifying and/or extending the SimpleOrderSystem application. Start by reading through the code so that you understand it.

The SimpleOrderSystem application as downloaded is not meant to be a complete or finished application. It is code that is still being developed. It will compile and run but there is no guarantee that it is free from errors! If you find any, fix them!

Q1. Add a command to display the total value of all orders for all customers. In class SimpleOrderSystem you will need to add a menu item, and you might add a method called overallTotal that will be called from the doOption method. The overallTotal method could iterate through each customer object, from the list in SimpleOrderSystem, adding up their total for all their orders.

Hint: But you have read through *all* the code to see what is already there haven't you?

To avoid having to enter example data manually every time you run the code to test it, add a method in class SimpleOrderSystem, where the main method is, to add example data by creating objects and calling methods.

Q2. A Customer's postcode needs to be stored separately from their address (e.g., number, street, town). Modify the Customer class and any other methods in other classes to support this.

Q3. Turn class Product into an abstract class and provide two or more subclasses for specific kinds of product. The method getDescription should be made abstract and the instance variable description removed from Product. Each subclass should support the common set of methods inherited from class Product, override the getDescription method, and can add additional

information. For example, you might have a Book class that adds title, author, publication date, etc., with getDescription returning a string containing the information in a suitable format. Think carefully about the constructors needed for each class. Update the rest of the program to support the addition of the subclasses.

Note, where possible the code in the program as a whole should still use variables of type Product and have an ArrayList<Product> to store the full collection of products. The subclass names only need to be used when objects are created in a new expression.

Q4. Add the ability to display a list of each order that includes a selected product. For each order, the customer name should also be displayed.

Q5a. Add the ability to load and save data to or from a text file. A representation of the value of each object can be a sequence of text lines in the file with a start and end marker. For example, class Book object values might be represented as:

book:1
title
author
publication date
end
book:2
title
author
publication date
end

… and so on.

Where one object links to other objects, for example a Customer has list of Orders, you will need to add some kind of identifier (e.g., an integer), to correctly connect the objects together. In the Book example, the first line of each book entry is denoted by the word book followed by the integer.

When loading data first think of the order objects need to be created and make use of temporary collections if needed.

b. (Challenge) Store the data in JSON format instead of just lines of text.

Q6. Add the ability to display the list of all orders for a Customer, showing each item in the order. In this and other questions you will probably want to improve the user interface to make the program easier to use. For example, instead of asking the user to type in the Customer's name you could provide a numbered list of names and let the user type in a number.

Q7. (Harder Challenge) Implement a graphical user interface using Java Swing to replace the command line interface.

You may be aware that the successor to Swing is JavaFX (https://openjfx.io/). See if you can implement your graphical user interface using JavaFX instead of Swing (be warned this is a serious challenge!).