COMP9103 Software Development in Java

OBJECT-ORINETED DESIGN

The key topic for this week is Software Development procedure File IO.

- write code to read from text files
- write code to write to text files
- read objects from binary format files
- write objects to binary format files
- write code that handles basic exceptions
- declare, instantiate and use ArrayList objects to manage a collection of data items
- write a text-menu-based driver class that allows the user to direct the order of execution of tasks

Tasks

1. Download the provided zip file, extract it to your workspace, add the project to the workspace, and look at the contents.

2. In particular, consider the file parts.txt which contains a set of records representing computer parts or equipment that are for sale by the computer store.

   The file's contents follow a specific format: The first line of the file is an int value, specifying how many records will follow in the file after that line. Each of the records consists of 3 separate lines. The first line of three for each record is the part's name, the second line is the part's description, and the third line is the price of the part.

   The parts can be represented in the program by objects of the Part class type, which has been provided to you. Ensure that you understand how to use the Part class.

3. Start to write a ComputerShop class that includes the features described below. Remember that you will probably need to include exception handling code. As you come to each task, you might want to update a separate Driver class (see task 4) to make sure it calls the method, and tests that the method is working correctly:
   a. The constructor of ComputerShop should create an ArrayList in which the parts will all be stored. Initially (at constructor time) there should be no parts. Ensure that the driver class creates an object of the ComputerShop type, which will be used for calling the subsequent methods.

   b. The loadPartsDataCleanly( ) method should replace any contents that are in the ArrayList, with Part objects created from data which is read in from the parts.txt file, and setting the quantity in stock to be initially 5 for each part. Ensure that your driver class calls this method before any of the remaining methods or described behaviour is done.

   c. The showParts( ) method should display each of the Parts that the shop sells, reporting all details of the parts including the quantity currently available.

   d. The reportParts( ) method should do the same as for showParts( ) except the output should be written to a text file named parts-report.txt

e. The addPart( ) method should accept a Part object reference as the parameter, and if valid will store this part at the end of the ArrayList.

f. The savePartsBinary( ) method should save the current state of all parts that are in the ArrayList (including any that have been added by task e), to a binary file named parts.dat, using techniques for serialization. If you need to alter other code, this is fine.

g. The loadPartsBinary( ) method should restore the state of the ArrayList to be referring to the same objects that it had at the time the data was serialized. So for example, if the first part was "Wireless Keyboard" and there was a quantity of 4 of them at serialization time, then after this method is called, the system would report that there are just 4 Wireless Keyboards in stock. The only parts that should be in the ArrayList, are the ones that were serialized.

4. Commence writing a text-menu-based driver class that starts by creating a new ComputerShop object, and then address the following aspects. Note that it may require you to make further adjustments to the ComputerShop class:

   a. Initially, the ComputerShop object should fill itself with data about Parts read from file. It should aim to read the data from the binary file, but if that is missing it should try to load the initial data cleanly from the text file. If it can't read even from that text file, it should tell the user, and exit.

   b. One of the actions on the menu should call the showParts( ) method to show on the screen details about all the parts.

   c. Another of the actions on the menu should call the reportParts( ) method, to produce the text file containing the report.

   d. Another of the actions on the menu should be to Exit the program. If this is selected, the program should save the current data about parts to a binary file, so that next time the program is started, that data can be restored from the binary file and the program can continue on as though it had never ended.

   e. Another of the actions on the menu should be to "Sell a part". This should first present the user with a list of all the parts that are sold by the computer shop, and ask the user to select one. If there is at least one of the selected item in stock, then display a message indicating success and reduce the quantity on hand by one. Otherwise, report a suitable message to the user.

   f. Another of the actions on the menu should allow the user to specify the details of a new kind of part that the shop will sell, and record the information in memory. Thereafter, that new part will be included in reports, lists, etc., including on future runs that load data from the binary file.