CPS209 Assignment 1:

E-Commerce System Simulator

Due Date: Mon, Mar. 21 11:59pm

You are to write parts of an E-Commerce System Simulator program. An Ecommerce system (like Amazon) sells various types of **products** to registered **customers**. Customers **order** a product they like, and the system **ships** it to them. The system must keep track of all its products, all its registered customers, all current orders and all shipped orders.

This programming assignment will increase your knowledge of array lists, objects and classes, inheritance, and interfaces. **You must do this assignment alone – no groups. Do not attempt to find source code on the web for this assignment. It will not help you and you risk extremely serious consequences. Your program will be checked for plagiarism.** Begin designing and programming early! This assignment is worth 10 percent of your mark. **If there is some part of the assignment you do not understand, please contact Prof. McInerney or Prof. Valenzano (via email or ask during lectures) and we will clarify the issue.**

NOTE: some parts of the assignment are using java concepts we have not yet studied (e.g. inheritance, interface Comparable and interface Comparator, the use of the *super* keyword). These topics will be covered in detail in the two weeks after the break. You may want to wait until these topics are covered before writing the part of the system that uses them. However, we greatly encourage you to start the parts that you can as soon as possible.

Skeleton code has been provided for you. This code compiles and runs! It defines the Java classes for you and your job is to basically fill in the methods for these classes. In addition, two of the basic system actions have been written for you: PRODS and NEWCUST. We suggest you begin by examining the code and understand how the flow of execution works for these two actions. Use print statements. Start in ECommerceUserInterface.java and see how it calls the methods in ECommerceSystem.java and how the Ecommerce system uses the support classes (e.g. Product, Customer etc.). Then, begin by implementing the simpler actions (see the marking scheme at the end of this document and use it as a guide. Also use the video as a guide). That is, write the code necessary to get a single action completely working (compiled, debugged and tested). Then move on to the next action. This way of "growing" your code piece by piece will minimize bugs and allow you to always have a working system to submit – even if you have not finished some of the more difficult actions. NOTE: if you submit code that does not compile, you are eligible for at most 1 or perhaps 2 marks out of 10.

Program Functionality Requirements:

Please view the video included with this assignment. Below is a description of the classes for your assignment and list of methods and variables. **NOTE:** We **have posted skeleton code for you of all the necessary classes. Some of these classes are complete, most are not. You are to fill in code for the methods according to the instructions below and according to the comments in the skeleton code for each class. This makes the assignment much easier to mark for the TAs and ensures you are adhering to proper Object Oriented design.**

Look at the constructor method of ECommerceSystem.java and you will see that we have created a list of 10 or so products as well as a list of 4 customers. **Please do not change or add to these as they will be used for marking!!** Please look closely at the

skeleton code and comments within. For this first assignment, you should always strive to use the methods outlined in the skeleton code rather than creating your own. In other words, we are controlling the overall design in this assignment. **You are permitted to add supporting methods and other instance variables if you deem it necessary**. Here is the list of classes you are to modify:

1. **Product:** class ***Product*** contains several instance variables (see the skeleton code) to model a product: Product ID, name, type, price and product options. For example a name might be "Acer Laptop". The ID (a string) is generated by the system (e.g. "913"). The options are a string that is used when a product is ordered (e.g. Paperback book vs Hardcover). There are different categories of Products (e.g. computers, books, clothing). This class has been completed for you.

2. **Customer:** class **Customer** models a registered Ecommerce customer. It contains the customer name, and ID (a string generated by the system) and a shipping address. **Almost all the code for this class has been written for you. You are to make class Customer implement the Comparable interface – see the skeleton code for comments. Note: this functionality is not needed except for the SORTCUST action – you may want to wait until we have studied Java interfaces before tackling this part.**

3. **ProductOrder:** class **ProductOrder stores** the order of a product by a customer. It contains an orderNumber (a string generated by the system), a reference to the product object ordered and a reference to the customer object. It also stores any product options associated with this product (e.g. for Books, the options contain "Paperback" or "Hardcover" etc) that were entered by the user. This class has been written for you.

4. **Book:** class Book is a subclass of Product. It also stores additional information (author, title). A book maybe in one of 3 formats: "Paperback", "Hardcover", "EBook". It also has additional variables to store the number of hardcover books in stock for this book, the number of paperbacks, and has an essentially infinite number of EBook copies. For this reason the get/set/reduce stockCount methods must be overridden to appropriately use the given productOptions so that the specific stock counts can be affected by setStockCount() and reduceStockCount(). The print() method must also be overridden such that the basic product information is printed (HINT: use the super keyword) as well as the additional information title and author. See the video.

5. **ECommerceSystem:** this class contains the bulk of the logic for the system. It maintains an array list of Product objects, Customer objects, ProductOrder objects and ProductOrder objects that have been shipped. Some of the methods have been written for you. Fill in the code for the other methods based on the comments. The constructor method has been written for you and Product and Customer objects have been created for you and added to the appropriate lists.

6. **ECommerceUserInterface:** This class has the **main() method and is the user interaction part. Some skeleton code has been provided for you with some comments.** In a ***while loop***, a scanner reads a line of input from the user. The input lines contain words (Strings). The first word is the action (e.g. custs, prods). Some actions require parameters. The code should prompt the user by print out what

parameter should be entered. See the video and look at the example code provided for you for action NEWCUST. Fill in the code based on the comments for all the other actions.

7. Design a subclass **Shoes** that is a special type of Product with some specific options. The user should be able to buy a pair of shoes that have sizes 6, 7, 8, 9, 10 and possible colors black or brown. Examine class Books to see how it overrides the various stockCount() methods inherited from class Product by using the productOptions string – which is set when the user orders a book and specifies the format (e.g. PaperBack). You also will need to create a new action ORDERSHOES in ECommerceUserInterface where the user can specify size and color and this information is stored in a single productOptions string.

8. ***BONUS***: Add an integer year variable (representing year published) to class Book. Create a new action called "BooksByAuthor" in class ECommerceUserInterface that specifies an author name. Add code to class ECommerceSystem and class Book such that all books by the given author are printed in increasing order of year published. Use proper OO design.

Grading (Out of 10 marks)

Highest Grade Possible: 11

Basic: actions CUSTS, BOOKS, ORDERS, SHIPPED	3 marks
Intermediate: actions ORDER, SHIP, CUSTORDERS, NEWCUST, CANCEL	4 marks
Advanced: Action ORDERBOOK and class Book functionality	1 mark
Advanced: action ORDERSHOES and class Shoes	1 mark
3 SORT actions (see code)	1 mark
BooksByAuthor action implemented using proper OO design	1 mark
Penalty for insufficient comments or bad program structure or bad use of inheritance and interfaces	Up to -1 marks

Submitting Your Assignment: READ CAREFULLY!!

- Inside each of your ".java" files have your name and student id as comments at the top of the file. Make sure your files can be compiled as is without any further modification!! Try compiling on a command line using javac.
- Include a README.txt file that describes what aspects of your program works and what does not work (or partially works). If your program does not compile, state this!! You may still get 1 or 2 (out of 10) part marks. The TA will use this README file as a guide and make fewer marking mistakes.
- Place all your ".java" file(s) and README.txt file into an archive (zip or rar archiving format only).

- Open your archive and make sure your all your files are there.
- **Do not include ".class" files!!!!**
- **Do not use any Package keyword in your java files!!!**
- Once everything is ready, submit it to D2L.