CPS209 Assignment 2:

Extended Ecommerce System

Due Date: Thursday April 14 11:59pm

You are to extend the Ecommerce System program from assignment 1. This programming assignment will increase your knowledge of Java Collections, File I/O and Exceptions, 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 email your professor as soon as possible and they will clarify the issue.**

NOTE: some parts of the assignment are using java concepts (specifically Maps) that we have not yet studied in the lectures. You may want to wait until this material is covered in the lectures before attempting this part of the assignment.

Program Functionality Requirements:

We will add additional functionality to assignment 1 so that the system more closely simulates a real Ecommerce system like Amazon. You will be adding the idea of a **Cart** to the system. A cart is a virtual realization of a shopping cart you might use in a grocery store. In Amazon, each customer has their own cart. A customer adds products to their cart. A customer can then order all the items in the cart. In this system, a separate product order will be generated for each item in the cart. Customer may remove items from cart and print the current items in the cart.

Below is a description of the new functionality for your assignment and some new methods and modifications to the existing functionality as well as modifications to the user interface. **NOTE:** You are encouraged to **extend and modify your own assignment 1 (A1) code and keep the classes essentially as they were except for changes and additions specified below. You are permitted to add other instance variables and other methods if you think they are necessary.**

NOTE WELL!!: if you want to extend Prof. McInerney's solution to assignment 1 rather than use your own A1 code, you are permitted to do so. It will be posted on April 1 or thereabouts (once everyone has handed in their assignment 1).

1. **ECommerceUserInterface: Add 4 new actions (commands): ADDTOCART, REMCARTITEM, PRINTCART, ORDERITEMS.** The new actions take the following arguments:
   a. **ADDTOCART:** String productid, String customerID, String productOptions
      - Adds a product to the customer's cart
   b. **REMCARTITEM:** String customerId, String productId
      - Removes a product from the customer's cart
   c. **PRINTCART:** String customerId
      - Prints all the products in the cart
   d. **ORDERITEMS:** String customerId
      - Creates a product order for each product in the cart. For example, if there are 5 items in the cart, then 5 separate product orders should be created. Empties the cart of all items once the orders are created.

2. **Cart:** design a class **Cart** and class **CartItem. Add a Cart object to the instance variables in class Customer.** This way each customer object has a Cart object (created when a Customer object is created). A Cart object should contain a list of **CartItem** objects. Class **CartItem** contains a reference to a Product object and a reference to a productOptions string. Create appropriate variables and methods for these two new classes.

3. **ECommerceSystem: the following is a list of additions and changes required for the ECommerce class:**
   a. Create methods to support the 4 new actions (ADDTOCART, REMCARTITEM, PRINTCART, ORDERITEMS). Use existing methods as a guide. We suggest you write, test, and debug one of these actions + methods at a time before moving on to the next.

   b. Replace the error message variable ***errMsg*** and the return types of the methods in ECommerceSystem with exceptions. That is, if an exception occurs (for example, in method **orderProduct()** the customerId parameter may not match any customerId in the array list of customers) then **throw a new exception.** Create custom exception classes (for example: class UnknownCustomerException) by extending class RuntimeException (see the Exception slides for examples of this). Place these new exception classes in the ECommerceSystem.java file at the bottom of the file, outside of the Ecommerce class. Note: the return type for **most** of the public methods in class ECommerceSystem that currently set the error message variable (for example, method createCustomer()) can now be changed to **void**. Update the code accordingly in class ECommerceUserInterface. Here is a list of exceptions that should be supported: Unknown customer, unknown product, invalid product options, product out of stock, invalid customer name, invalid customer address, invalid order number.

   c. Catch the thrown exceptions in class **ECommerceUserInterface**. It might be convenient to place the **catch**() {…} code block(s) together in one part of the file and use a single **try** {…} that surrounds the if statements inside the while loop – it is your design choice. The catch() block(s) should print any message contained in the exception object to the screen and then continue the loop waiting for the next user input.

   d. **File I/O**: replace the code in the constructor method (that currently creates a fixed set of Product objects) with code that reads the product information from a file, creates Product objects and then adds them to a list (or map – see Map requirement) of products. Use the posted file **products.txt**. Look inside this file. It has a fixed format for each product. That is, each product "record" takes 5 lines. The first line is the product category. The 2nd line is the product name, the 3rd line is the price, the 4th line contains stock count information. For a non-Book and non-Shoes product this line contains a single integer (string). For a Book, this line contains two integers (Paperback and Hardcover counts). Finally the 5th line contains additional product information. For most products it is an empty line. For a Book, this line contains the title, author and year of publication and this information is separated by colon ':' characters. Hint: make use of the Scanner class method **useDelimiter()** to break the line into its constituent strings. Or, write code to read in this line, then go through the string character by character until a ':' is reached. Note: we suggest you create a private method that reads the products.txt file and returns an array list of

product objects. Inside the ECommerce constructor method call this private method, surrounded with try{…} catch(IOException e) {…}. If an IOException occurs in the private method, print out the IOException message and exit program using System.exit(1).

e. **Map:** replace the array list **products** with a **Map** called products. The map should "map" a string productId key to a Product object. You will need to replace all the code in ECommerceSystem that currently loops through the products array list with code that uses a map. Maps will be explained in an upcoming lecture. **Note**: once you use a map, the sortByPrice() and sortByName() code will need to be updated: use the map to generate all product objects, add them to a temporary arraylist, sort this arraylist then print the list. **That is, unlike assignment 1, the sortByPrice() and sortByName() actions will now not only sort products but also print the products in sorted order. For this reason, change the names of the actions SORTBYNAME and SORTBYPRICE to PRINTBYNAME and PRINTBYPRICE respectively.**

f. **Product Order Statistics:** add functionality to your system to keep track of the number of times a product was ordered. Use a **map** to achieve this. Add a command to **ECommerceUserInterface** called "**STATS**" that will print out a formatted list of each product name, its id and the number of times it was ordered. Print this list in order of products ordered the most to products ordered the least.

4. **BONUS:** Write code to support the capability of adding customer ratings (ratings from 1 to 5) to a product. Create new actions in ECommerceUserInterface to add a rating to a product and print the ratings for a product. For each product, keep track of the number of 1 star ratings, the number of 2 star ratings etc. Add functionality to your system so that a user can list products of a certain specified category with ratings above a specified threshold. For example, print all books with an average rating > 4. In a README file, provide a description of your actions and how they work for ratings. You are permitted to use Java streams for the bonus but it is not necessary.

Grading (Out of 10 marks)

Note: Highest Grade Possible: 11 out of 10

File I/O and exception handling for reading the products.txt file and creating Product objects and storing in a map (or array list if you are not using a map)	3 marks
Good use of customized exceptions instead of **errMsg** variable. Exceptions thrown and caught properly	2 marks
Cart and CartItem functionality	3 marks
Correct use of a map for product objects (all array list code	1 marks

properly replaced)	
Use of a map to support the printing of product order statistics.	1 mark
Bonus	1 mark
Penalty for insufficient comments or bad program structure or bad use of exceptions, maps etc	Up to -1 marks

Submitting Your Assignment: READ CAREFULLY!!

- Use D2L to submit your assignment.
- 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!!
- Include a README.txt file that describes what aspects of your program works and what doesn't work (or partially works). If your program does not compile, state this!! You will still get part marks. The TA will use this file as a guide and make fewer marking mistakes.
- **Do not include ".class" files!!!!**
- **Do not use any Package keyword in your java files!!!**
- Once everything is ready, submit it.