CS6311
Final Exam

The Great Purchase Kiosk

Objectives:

- Apply the programming techniques taught this semester to implement a small software application

Exam Structure / Grading:

Because this exam is a chance for you to demonstrate what you've learned, it will be structured different in that:

- The entire exercise must be done individually without consultation of any aspect of this project or code with anyone other than the instructor ('anyone' includes other professors, TAs, tutors, coworkers, friends, relatives, etc). If you have a clarification question regarding this project, please feel free to contact the instructor directly.
- The exam will have 2 parts:
    - Part 1 (74% of the score) will have requirements similar to the homework
    - Part 2 (26% of the score) will require you to apply the techniques you've learned so far in the homework with less structure / details being provided.
- If Part 1 earns fewer than 75% of its possible points (fewer than 56 of the 74 possible), then Part 2 will not be graded at all. Note that this means it will be far better to complete Part 1 correctly, rather than complete a little bit of each of the parts with lots of errors in each.
- Any project submitted with compiler errors will receive a 0 – please don't hand in code that doesn't compile.
- Your work will be graded on functionality, appropriate output, and code style.

Background Information:

The owners of a local electronics store, Great Purchase, have asked you to develop an application to help at their stores. This application will present a simple, text-based user interface and allow both customers and employees to interact with the inventory. There is a sample output file provided on Moodle to give you an idea of what this application will do (in addition to the description provided in these PDF documents)

Part 1 (74 points)

Be sure to read through all items before you begin. The following descriptions are not written in any particular order. You should apply the iterative development techniques we've stressed this semester to develop your code in an appropriate fashion.

Organization

All classes in your Eclipse project will be stored in one of the following packages only:

```
edu.westga.cs6311.kiosk.controller
edu.westga.cs6311.kiosk.model
edu.westga.cs6311.kiosk.tests
edu.westga.cs6311.kiosk.view
```

You are encouraged to develop a set of *informal* tests for your model classes and place that code into the tests package, however no points will be awarded for testing (we'll focus much more on *formal* testing in the Spring semester)

Package - Model

Develop a class named `Computer` such that it defines:

Instance Variables:

- The computer's Stock Keeping Unit (SKU) number (String)
- price (double)
- the number currently in stock (int)

Methods:

- A 3-parameter constructor that accepts the values to be stored in the instance variables. Be sure to error check each parameter so that if invalid values are passed, the default value for that data type is stored (the Empty String for String variables and zero for numeric variables). Use these values to initialize the instance variables.
- Accessors ('getters') for all instance variables
- A mutator ('setter') for the number in stock *only*. This method will accept the new number of computers in stock and not return anything. If the new quantity passed is negative, simply return without changing anything.
- A method named `purchase`. This method should accept the number of this item being purchased and not return any value. All this method should do is deduct the specified number of units from the number in stock. Be sure to provide error checking for the number to be purchased to be sure that (a) it isn't less than 0 and (b) there are enough on hand to be purchased. If either of those conditions fails, then this method should return without any items being purchased. If the conditions are met, then the number in stock should be reduced appropriately.
- A `toString` method used to return a `String` representation of the computer. This `String` must list each of the data members with the number of characters specified

and exactly one (1) blank space between each value. Note that there is a dollar sign character before the price and the price should be listed with the decimal place in the same column for each item as follows:

<SKU> 6 characters, left justified	Blankspace	$	<Price> 7 characters	Blankspace	I	n		S	t	o	c	k	:	Blankspace	<number> 3 characters

For example:

H	P	1	2	3	4		$		6	9	9	.	9	9		I	n		S	t	o	c	k	:			1	5
D	L	3	3	3			$	1	3	4	9	.	2	8		I	n		S	t	o	c	k	:		1	2	0

** It is OK to assume that the SKU will be no more than 6 characters, the price won't be more than $9,999.99, and there won't be more than 999 of each item.

HINT: There is a method in the `String` class named `format`, which will work similar to `printf` (`format` returns a `String`, whereas `printf` prints the formatted data on the console). Please see the textbook and/or Java API for more details on how this can help you here.

Develop a class named `InventoryManager` such that it defines:

Instance Variables:

- An `ArrayList` of `Computer` objects

Methods:

- A 0-parameter constructor that instantiates the instance variable
- `startStore` – This method is going to be used to add standard items to the store's inventory:

  15 HP1234's that cost $699.99 each
  120 DL333's that cost $1,349.28 each

- `findComputer` – This method will accept a String holding a computer's SKU. It will then search the inventory and return a Computer that matches that SKU. You can assume that each item SKU is unique. Note that this search will not be case sensitive (so "HP1234" is treated the same as "hp1234" or "hP1234"). If no such computer is present, this method will return `null`.
- `addComputer` – This method will accept 3 parameters to create a Computer. It will then create the object and add it to the inventory.
- `getLeastExpensive` – This method will search the inventory and return the least expensive computer. If there are no computers in stock, this will return null.

- `getMostExpensive` – This method will search the inventory and return the most expensive computer. If there are no computers in stock, this will return null.
- `getTotalInStock` – This method will calculate and return the total number of computers in stock (so if there are 15 HP1234's and 120 DL333's, then this method would return 135).
- `getAveragePrice` – This method will calculate and return the average computer cost. Be sure not to round this value. If there are no computers in stock, then this will return 0.
- `toString` – This method will return a `String` containing a complete description of the inventory. If the inventory is empty, this `String` should say so; otherwise it will include a listing of each Computer on its own line (hint: use the `Computer`'s `toString` method), followed by a line telling the number of different computer models, followed by a line listing the total computers in stock, and finally three lines listing the most and least expensive computers and the average computer price (be sure to format this average to exactly 2 decimal places). For example:

```
HP1234 $ 699.99 In Stock:  15
DL333  $1349.28 In Stock: 120

Number of computer models: 2
Total quantity of all computers: 135
Most expensive computer:  DL333  $1349.28 In Stock: 120
Least expensive computer: HP1234 $ 699.99 In Stock:  15
Average computer cost: $1024.64
```

Package - View

Develop a class named `ManagerTUI` such that it defines:

Instance Variables (these are the only acceptable instance variables):

- An `InventoryManager` object
- A `Scanner` object

Methods:

- A 1-parameter constructor that accepts an `InventoryManager` object and initializes the instance variables.
- `runManager` – This method serves as the 'director' of the user interface by calling on private helper methods to do their work, as appropriate.
- A variety of other `private` helper methods that will provide the functionality for allowing the user to interact with the program and display the necessary output. Every method should perform exactly one cohesive task and should have an appropriate name to describe this task. As a general rule, each menu option will have at least one method associated with it.

Application Functionality

- The user will interact with the application using a console-based, textual user interface (TUI). This interface should be menu driven. To keep things simple, menu options will be numbered so that the user can type a single integer value as a menu choice (instead of typing in a character or word). See the sample output for an example on how this might be done.
- There are a number of methods defined to return `null`. Be aware that null is a programmer's term and should never be displayed to a user. Instead, be sure your application presents appropriate words and feedback for 'regular people' (non-programmers)
- Menu options include:

  1. Start a new store

     This should call the `startStore` method to add the standard computers to a store.

  2. Add a new computer to the inventory

     The application will allow the user to enter the computer's SKU, price, and number in stock, then this `Computer` will be added to the inventory. Once this action is complete, a confirmation message should be printed on the screen describing the computer that was just added.

     You can assume that the user will enter the appropriate data type (integer, floating point number, etc) but you aren't guaranteed what value will be entered. If the user enters a price or quantity less than or equal to 0, the application should force the user to try again until they enter a positive value.

  3. View the current inventory

     This should display a listing of each computer on its own line and include the total number of computer models, the total quantity of all computers in stock, the most and least expensive, and the average computer price.

  4. Quit the application

     The application will display an appropriate message to thanking the user for using the application before closing appropriately.

  Again, you can assume that the user will enter an integer value as a menu choice, but you don't know what integer they will enter. Be sure that if they choose a menu option that's not available that you display an appropriate message and show the menu again.

Package - Controller

Develop a class named `GreatPurchaseDriver` such that it defines a `main` method as the entry point into the application. It will create an instance of the `InventoryManager` class. It will then pass this object to the `ManagerTUI` constructor and use the `ManagerTUI` object to call `runManager`.

When all code is completed and you have the correct functionality, close Eclipse and use 7-Zip to zip your entire project folder. Give your file the name CS6311*YourLastName*FinalPart1.zip. Upload the Zip file to the appropriate link in Moodle.

Part 2 is posted on Moodle.