CSCI 235 PROJECT 0

TEAMWORK IS PREFERRED FOR THIS ASSIGNMENT

Hunter Inc is a company that carries exactly 20 model of shirts. These shirts are sold in its retail stores throughout the nation. Each of the company's retail store must carry between 5 and 20 of those models. A shirt sells for exactly 5 dollars.

A store can only sell items that it carries and cannot sell beyond the quantity it currently has in stock. The company itself cannot distribute to its retail stores more shirts than it currently has.In short,there is no back order. Furthermore, if a retail store requests a quantity (in a particular model) than the company currently has, Hunter Inc will just deliver what it has at hand to complete the order.

The first part of your assignment consists of writing an interface file (*Inventory.h*) and an implementation file (*Inventory.cpp*) for a class named **Inventory** that has as private section:

(1) a static array **corpInventory[20]** that represents for each model, the number of shirts available for distribution at the headquarter.

(2) a static array **corpQtySold[20]** that represents for each model, the quantity that has been sold by Hunter Inc nationwide.

(3) an integer variable **models** that represents the number of models available in a particular store.

(4) an array of int **stock[20]**that represents for each model of shirts, the quantity that a particular store has available. If a store does not carry a model, its index must be set with -1.

(5) an array of int **sold[20]** that represents the number of shirts sold per model in a store. If a model is not sold or not carried by a store, its index must be set to 0.

(6) an integer variable **money** that represents the total amount of money made daily by a particular store.

The public section of the class Inventory must have

(1) exactly one constructor **Inventory()** that randomly determines the number of models and the number of shirts carried by a store . The number of models must be between 5 and 20, while for each model that a store sells, the initial stock available (ie number of shirts for that model) must be between 1 and 75. Indexes of models that have not been chosen must be

initialized with value -1 (to indicate that a particular model is not currently available a store)

(2) a static function **void setCorpInventory()** that sets the initial stock available at the headquarter of the company.This is the mutatator of the array corpInventory.It must be initialized for each index, with a random number between 600 and 2000.

(3) a function **int getModelSold(int mdl)** that returns number of shirts sold of a particular model by a store.

(4) **void salesSimulation()** that
- randomly simulates a daily sale for each model available,
- update for each model, the number of shirts sold by a store and by the company in a day.

As a reminder, a store cannot sell beyond the quantity it currently has and only sells models that it carries.

(5) **int getDailySales()** that returns the amount of money made in a day by a store.

(6) a friend function **int* grossDailyQtySold()** that returns the number of shirts sold per model by the entire company.

(7) a friend function **int* grossDailyUpdate** that returns the number of shirts per model still available at the headquarter after all the requests from all the stores have been fulfilled.

(8) a friend function **int* grossSales** that returns the total amount money made each day per model by the entire company.

Write a driver program**DailyReport.cpp** that does the following:

- initializes the static array corpInventory[20]. Each index must be initialized with a number between 600 and 3000 included. Recall that a static variable can exist before any object has been instantiated

- initializes the static array corpQtySold[20]. Each index must be initialized with zero.

- a dynamic array that indicates number of stores Hunter Inc currently has. This number must be enter manually by the user and must be between 30 and 50.
  Example Inventory* stores = new Inventory[nmbOfStores]; $30 \leq nmbOfStores \leq 50$

After the initialization and instantiation phase described above, simulate one day sale for each stores:(*stores[i].salesSimulation()* for all i)

At the user's request, and for any given store, you must display for any given store, the number of shirts sold, the number of shirts still available, and the amount of money made that day.

[**Remark about the salesSimulation method**, For each store during your salesSimulation, do not forget to update the stocks at both end: sold and corpQtySold.]

Your second set of statistics will consist of displaying for a given model, the number of shirt sold by each store and the amount of money made.

The last statistics will be of displaying for all the models, the number of shirts sold by the whole company (hint grossDailySales()) and the amount of money made per model, then the cumulative amount of money made by Hunter Inc in a day.

Extra Credit Assume that at the end of each day, the headquarter is able to produce for each model a random number of shirts between (600 and 2000) that will be added to its inventory. Also assume that at the end of each day, each store will place an order to replace all the shirts that have been sold. the number of shirts requested by a store this time can be up to 5 more than what have been sold.

Simulate a 7-day sale and produce the same statistics as above.

Programming Rules

- You must always start your program with the " HONOR CODE " by which you are bound
- You are strongly encouraged to use notions covered in class or in CSCI 127 and CSCI 135. However, if you choose another route, feel comfortable enough to be able to explain your work.
- Your program should be the result of personal effort not a copy from someone else's work. Feel free to communicate with me about your progress but do not expect me to write your codes. I will be more than happy to give you hints and directions to go around obstacles you may be facing.
- Your program should be professionally documented.
- The output of your program should be readable and understandable by ordinary human beings who lack mind-reading capability and who have not read the assignment or the program.
- Every program must follow commonly accepted stylistic guidelines regarding the use of blank lines, white space, indentation, and naming of program entities such as variables, classes, functions, and constants.
- Your program must be consistent in its use of typographical format for distinguishing types, variables, functions, and constants. For example, you might decide that all type names should begin with an uppercase letter

and all variables begin with lowercase letters, or that all variables use underscores to separate the words in the name, as in number_of_scores and first_author, or that they use changes in case, as in numberOfScores and firstAuthor.

- **Grading**

  Correctness 55% *Design* 25% *Style* 10% *Documentation* 10%