**AS ALWAYS, THIS WORK IS TO BE THE EFFORT OF ONLY THE INDVIDUAL HANDING IT IN. CODE SHARING, COPYING or COLLABORATING ON A SOLUTION ARE STRICTLY FORBIDDEN. YOU ARE NOT TO DICSUSS THE PROGRAM WITH ANYONE OTHER THAN THE INSTRUCTOR OR TAs**

**Get Your Popcorn, Peanuts, Candy!**

**Due Friday, April 28, 11:55 p.m.**

**ALL METHOD NAMES AND THE NUMBER AND ORDER OF PARAMETERS MUST FOLLOW THIS DESCRIPTION EXACTLY.**

The Seaside Stadium wants to manage information about the sales by the vendors who walk through the seats selling snacks. The vendors sell popcorn, peanuts and candy, but each vendor only sells one type of snack. Vendors are also grouped in the stadium by section, and your program will reflect that organization. You must implement the Vendor and SectionSales class definitions as specified below. **The methods must be named EXACTLY as specified and the number, type and order of parameters must be exactly as specified.**

Though the description may look long, the code itself is not very long. Rather, it emphasizes the organization and structure of the data and interaction between modules.

**All instance variables are to be private. Use good programming style and each class and method must have its own comment briefly describing what it implements.**

**Vendor class**

This class keeps track of the sales of an individual vendor as well as the sales of all the vendors combined. The class must contain the following:

An array of 3 elements the holds the total dollar value of sales of all the vendors for each type of snack. The first position is the total dollar value of sales of popcorn from all the vendors, the second for all peanut sales and the third for all candy sales.

Each vendor also has a name.

When an instance of the vendor class is created, it will be provided with **(in this order)** the vendor’s name, an integer indicating what snack they are selling (1- popcorn, 2-peanuts, 3-candy), the price for the item they are selling (since the prices can change due to promotions, what type of event they are being sold at etc.) and the initial number of units of the snack they are selling.

Your class must provide the following methods:

**sellSnack**- takes an integer parameter indicating how many units of the snack the vendor sold to a particular customer (for example, if the vendor sells candy, this integer will represent the number of boxes of candy sold to a customer). Since it is assumed that the vendor will not sell what they do not have, an error message should be output if an attempt is made to record selling more snacks than the vendor currently has with them.

**refill**-takes an integer parameter. A “refiller” will visit the vendors in the stands from time to time to see if they are running low on their snack. If so, they will be given more of their product. The refill method is called to reflect a refill of snacks and the integer parameter is the number of units the vendor is given in the refill.

**getSales**- takes no parameters and returns the dollar value of the vendor’s sales

**getName**- takes no parameters and returns the name of the vendor

**getType**- takes no parameters and returns an integer indicating what type of snack the vendor is selling (1-popcorn, 2-peanuts, 3-candy)

**displayAllTotals**- takes no parameters and displays the total sales of all vendors for all snacks

**displaySnackTotals**- takes one integer representing a snack type (1-popcorn, 2-peanuts, 3-candy) and displays the total combined sales of all vendors for that snack.

**displayVendorData**- takes no parameters and displays the vendor’s name, the snack they are selling, the total number of units of the snack they have sold and the total dollar value of their sales.

**SectionSales class**

This class manages the information for all vendors in a particular section and maintains an array, where each element represents an individual vendor in the section.

When an instance of the SectionSales class is created it will be provided with (in this order) a String indicating the section name and an integer indicating the maximum number of vendors that will be in that section.

The SectionSales class provides the following methods:

**addVendor**- takes four parameters which must be in the following order: a String indicating the name of the vendor, an integer indicating the snack they are selling (1-popcorn, 2-peanuts, 3-candy), the price for the item they are selling and the initial number of the snack they have to sell. addVendor must create the new vendor and assign it to the next available position in the array representing the list of vendors in the section. If the section already has reached its maximum number of vendors, as error should be output.

**vendorSale-** takes two parameters: the first is a String, which is the name of a vendor, and the second is an integer, which is the number of units the vendor sold of their snack. An error should be output if the name of the vendor passed as a parameter is not in the list of vendors for the section.

**vendorRefill-** takes two parameters: the first is a String, which is the name of a vendor, and the second is an integer, which is the number of units the vendor has been given as a refill. An error should be output if the name of the vendor passed as a parameter is not in the list of vendors for the section.

**displaySectionVendorData**-takes no parameters and displays the section name and the vendor data (name, the snack they are selling, the total number of units of the snack they sold and the total income of their sales) for each vendor in the section.

**dispalySectionAllTotals**- takes no parameters and displays the section name and the individual totals for each of the snacks sold in its section, that is, the total sales of all popcorn, of all peanuts and of all candy sold by the vendors in the section.

**displaySectionSnackVendors**- takes one integer parameter representing a snack type (1-popcorn, 2-peanuts, 3-candy) and displays the section name and the names of all vendors in the section selling that snack and the dollar value of each of vendor’s sales of that snack.

**displayAllSectionSnackTotals**- takes one integer representing a snack type (1-popcorn, 2-peanuts, 3-candy) and displays the total sales of all vendors in the stadium for that snack for the entire stadium.

**The main:**

Write a main that, by creating instances of the classes and comprehensive methods calls and error testing, insures that your code is working correctly.

**Important Notes:**

Be careful with the array of vendors. If your code uses dot notation on an array element that is null (does not have a vendor object assigned to it), it will crash your program. So, if the vendor list can contain at most 5 vendors and only 2 have been created, using the fourth array element to access a method in the vendor class will crash your program. It is perfectly valid to only have 2 vendors, just make sure that those are the only array elements that your code ever uses to make method calls. Reference variables that have not been assigned values via new, are initially set to null. Thus, trying to use such a reference variable to call a method causes a runtime error that will likely mention a “null pointer exception”.

Be sure to error test any value that your program uses as an array index to make sure that the index value is not out of range and outputs an appropriate error message if it is out of range.

As always, use good coding practice with meaningful variable names, do not use literal values the represent important values in your code, rather assign them to meaningful variable names. Now that you now about **final**, use it.

Drop your file containing the main and the Vendor and SectionSales java files into moodle.

IT IS EXTREMELY IMPORTANT, FOR TESTING PURPOSES, THAT ALL CLASSES AND METHODS ARE NAMED AND SPELLED (INCLUDING UPPER AND LOWER CASE USAGE) EXACTLY AS SPECIFIED AND TAKE EXACTLY THE NUMBER AND TYPE OF PARAMETERS SPECIFIED IN THE ORDER SPECIFIED.