COMP 1020
Lab 3

Material Covered

- Multi-dimensional arrays
- Method structure and code design
- Working with existing code
- Converting numbers between 1d and 2d indexes

Notes:

- These exercises are intended to give you basic practice in using Java
- You should definitely do at least the Bronze and Silver exercises. You can do these
  - before the lab, on your own, using any Java programming environment you like, or
  - during the lab, with assistance from the lab TA, or
  - after the lab, on your own.
- For credit, you must demonstrate at least one working exercise (Bronze), in the lab.
- Make sure your TA has recorded your mark before leaving.
- The three questions are sequential – each builds on the previous one.
- Always try to complete as many exercises as you can on every lab. For Lab 1, everyone should try to complete at least the Bronze and Silver exercises.

Bronze:

Replace the 1D **`Food`** array in the **`VendingMachine`** with a 2 dimensional **`array`**. Update all methods to work with this (**`count`**, **`toString`**, **`add`**) in the **`VendingMachine`** class. The **`add`** method should now accept both a **`row`** and a **`column`**. It should still reject values that are outside of the range provided. You will want to comment out the **`testLab1`** & **`testLab2`** methods (for now) as we have changed the **`add`** method parameter list to instead accept a **`row`** and **`col`** value (rather then a single **`index`**).

Use the **`testLab3`** method to test that everything is working as intended. Note: I have not done a lot of extra boundary checking but you should assume that your methods check that the row and col provided is in bounds.

```
You can update the maxSize variable to instead be the following:
      // 5 rows of 4 items each is 20 total
      private static int maxFoods_rows = 5;
      private static int maxFoods_cols = 4;
```

See the **`expectedOutput`**.txt file for the expected output of the test classes.

Tip: All your row and col operations should be based off of the current size of the array, not hard coded in, as the max size variables above are subject to change.

Silver:

We would like to support backwards compatibility so create a method **add(Food newFood, int index)** to allow the **testLab1** & **testLab2** methods to continue to work. The **index** will be translated to the **row** and **col** required as a multidimensional array. You should also create the following utility methods to handle converting between an **index** value and a **row** / col value using the **/** and **%** operators.

```
// Useful utility methods you should create and test for Silver
// converts a row and col to an index value based on the current size of the array.
public int rowsColsToIndex(int row, int col)

// convert the index value to the associated row value
public int getRow(int index)

// convert the index value to the associated col value
public int getCol(int index
```

Update the **add(Food newFood, int index)** method to work with the 2d array. You will need to map the values using **/** and **%** functions. See below or the expected output for more details.

```
Eg Conversion of index to row and col values.
// Example output from testing the above methods in the testLabs_Silver
method.
// Item at row 0, col 3 would be index 3
Item [0,3] at index 3 is Name: Apple Calories: 120 Cost: 2
// Item at row 2 col 0 would be index 8
Item [2,0] at index 8 is Name: Test Snack Calories: 123 Cost: 9
//Item at row 4 col 3 would be index 19 (the last value).
Item [4,3] at index 19 is Name: Banana Calories: 180 Cost: 1
```

Run the **testLab3_Silver()** method to test.

Gold

Our test methods are beginning to take up a lot of space in the **VendingMachine**. Create a new class **TestLabs** and move the **main** method and all of the **test** methods over to that. You should then create your own **test** method **testLab3_Gold** similar to the ones I created, that will test the boundary conditions of adding to a **row**/**col** or **index** with the **add** methods. You should test adding a **Food** object to several different invalid locations in the **array** such as **row**/**col**/**index = -1** or values above the available range in the **array** such as **index 20** in a **4x5 array** (which should be out of bounds).