# COMP 1020
# Lab 2

MATERIAL COVERED

- More advanced Objects.
- Method structure and code design
- Working with existing code
- Arrays

Notes:

- These exercises are intended to give you basic practice in using Java (as opposed to Processing or Python).
- 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. Everyone should try to complete at least the Bronze and Silver exercises while Gold will push your skills and help you study the material in more detail.


## Bronze:

Continuing from where we left off last week, our next we will upgrade the **`VendingMachine`** class. We are going to extend it so we can create and use **`VendingMachine`** objects. This is a more intuitive structure since we can imagine our **`VendingMachine`** being a physical object that would contain some set of **`Foods`**. It will also allow us to create multiple vending machines, each with their own array of **`Food`**.

Since our **`main`** method is **`static`**, and **`static`** means not instance, we will then need to create at least one instance of our **`VendingMachine`** object in order to access the instance methods that we will create.

While I have typically been separating the functionality into different classes to avoid confusion, there is nothing wrong with having **`VendingMachine`** being both an Object class with instance methods and variables, while also containing **`static`** methods (such as **`main)`** that can create new **`VendingMachine`** objects for testing.

Add the following variables to your **`VendingMachine`** class

- An instance variable that is an array of **`Food`** objects.
- A **`static int`** variable with a value of **`10`**. This will be our max array size for all our vending machines and will be used in the **`VendingMachine`** constructor to specify the array size. Since it is static all instances will share it. If we want to change the size of all the arrays in all machines, you just change this value.
- You should then add a constructor to the **`VendingMachine`** class that simply creates a new array of the specified size.

Once you have that in place, complete the **toString**, **size**, **addFood** and **countFood** methods as outlined in their comment blocks. When you are done that, you should be able to run the test method provided and get the same output as the example.

> *Tip: The array is used differently in this class then the assignment. For the lab, the array will contain several null entries and new Food objects will be added to a specific index instead of in order. You have to be extra careful with this usage to always check if the value in the array is null before calling it.*


## Silver:

Next, we will add nutritional information to our **Ingredients** class. Add 4 new instance variables of type **double**: **carbs**, **fat**, **protein** and **servingSize**. These will store information about our **Ingredients** nutritional information. All are in Grams (make a note in your comments) and the nutritional information is defined in terms of grams per serving, so will need to be converted to the correct values based on the **amount** actually used.

Add another **constructor** to **Ingredients** that includes parameters for each value.

Add the following methods to **Ingredients**:

Proportion of a serving this **Ingredient** contains as defined by (**amount** / **serving**Size)

Returns the proportion of a serving size this **Ingredient** represents, or 0 if it is undefined. (It should not crash if the old version of the constructor is used and the nutritional information is not set.

*Eg. If you have a serving size of 60g, and you have an amount of 30g, your proportion would be .5.*

**public double getProportion()**

Calculate calories per serving for this **Ingredient**.
Protein has 4 calories per g, fat has 9 calories per g, carbs have 4 calories per g as well.

**public double getCaloriesPerServing()**

Total calories for the amount of food defined in this object. That is, find the calories per serving and the amount of servings you have and calculate the total calories from that. You should be able to use the methods already created to calculate this easily.

**public double getCaloriesAbsolute()**

Returns the following String (with values):

**"Serving Size: [amount] Protein: [amount of protein] Fat: [amount of fat] Carbs: [amount of carbs]"**

**public String getNutrition()**


## Gold

For gold, add a test method to the **VendingMachine** to verify your **Ingredients** class. You should then add additional methods to **Food** that will take the total information in each **Ingredient** and give a similar set of methods as you defined in **Ingredient**. The **Food** versions of the methods will calculate the sum totals of all **Ingredients** within that **Food** object at the given amounts.