Criterion C:

Introduction:

The GUI creation interface and its Swing tools in Netbeans was used to develop and create the GUI in Java program. Netbeans was also used to detect any mistake or glitches in the code such that I, the creator, can edit and fix them to improve the program

Summary list of all programming techniques utilized:

- Making a 2D array of objects
- GUI tabs
- Method returning a value
- Sorting by different attributes of an array of objects
- For loop
- Recursion
- Simple and compound selection
- Encapsulation
- Using a flag value (such as -999)
- Searching and matching values in an array
- Editing information within a 2D array
- Get and sets

Structure of Program

What:

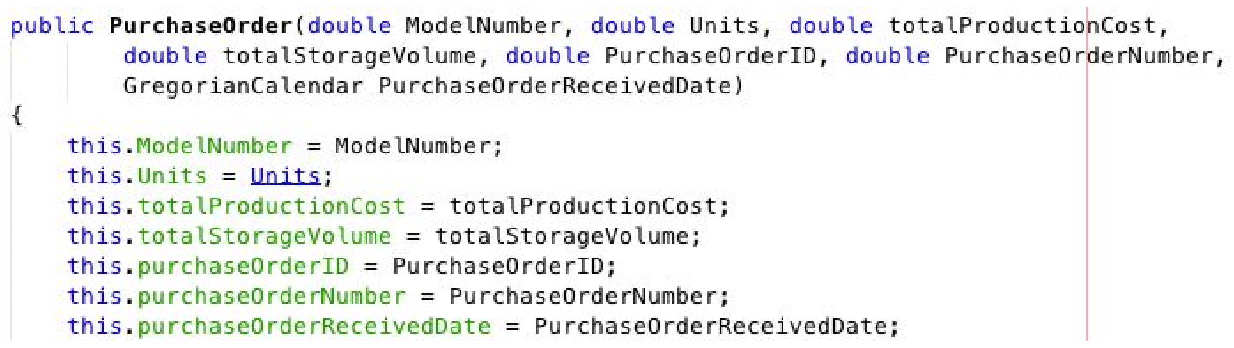
There are 2 main classes in this program. The first is Product, which contains the product model number, the volume of each unit, the estimated production cost of each unit, and the estimated production time of each unit. The second is purchaseOrder, which contains once again the product model number, the number of units ordered, purchase order ID, and purchase order number. The way these classes will interact with each other is that based on how many units are ordered in “Purchase order”, that number of units will be multiplied by the estimated production time and cost, thus giving the user the estimated production time and production cost for the purchase order of a particular number of a certain product model.

Why:

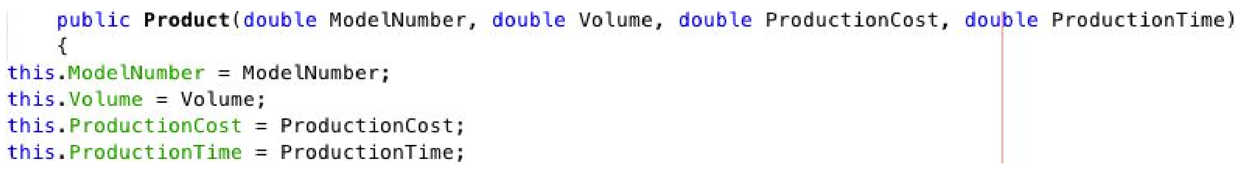
I chose to divide the classes into what they are as it would make the most sense contextually. Considering that this is a program meant to increase the operating efficiency of a firm. By creating 2 classes, product and purchase order, I can create a program where the user (the firm) can input specific info on each product model the firm offers production for into an array, then have the purchase order class use the same getProductModelNumber function in order draw upon the inputted data on a certain product model from the product class, and use these drawn values to calculate new values within the purchase order array such as calculate total costs and production time for each purchase order.

Data Structures used:

- Array of objects
- 2d arrays



```java
public PurchaseOrder(double ModelNumber, double Units, double totalProductionCost,
        double totalStorageVolume, double PurchaseOrderID, double PurchaseOrderNumber,
        GregorianCalendar PurchaseOrderReceivedDate)
{
    this.ModelNumber = ModelNumber;
    this.Units = Units;
    this.totalProductionCost = totalProductionCost;
    this.totalStorageVolume = totalStorageVolume;
    this.purchaseOrderID = PurchaseOrderID;
    this.purchaseOrderNumber = PurchaseOrderNumber;
    this.purchaseOrderReceivedDate = PurchaseOrderReceivedDate;
```



```java
    public Product(double ModelNumber, double Volume, double ProductionCost, double ProductionTime)
    {
this.ModelNumber = ModelNumber;
this.Volume = Volume;
this.ProductionCost = ProductionCost;
this.ProductionTime = ProductionTime;
```

Why:

I used arrays and arraylists for in my coding because it creates a large amount of convenience when programming. Because the type of program requested by my client is one that requires a large amount of user input data, not using arrays would mean that I would have to go through a very tedious process of creating a class and variable for every single product. By having an array and arraylist however, I greatly simplify the coding process by having the user simply add an element into the array each time instead of manually creating a class or variable. Moreover, information can be easily extracted from an array. This was utilized when I extracted the volume, ProductionCost, and ProductionTime from the product array to be used for calculation of totalProductionCost and totalStorageVolume to be stored in the PurchaseOrder array.

Algorithms:

Because this program is meant to simplify the calculations of net production costs and production time as well as warehouse management for the firm, some algorithms to calculate these values were used.

The main values that are to be calculated using algorithms include:

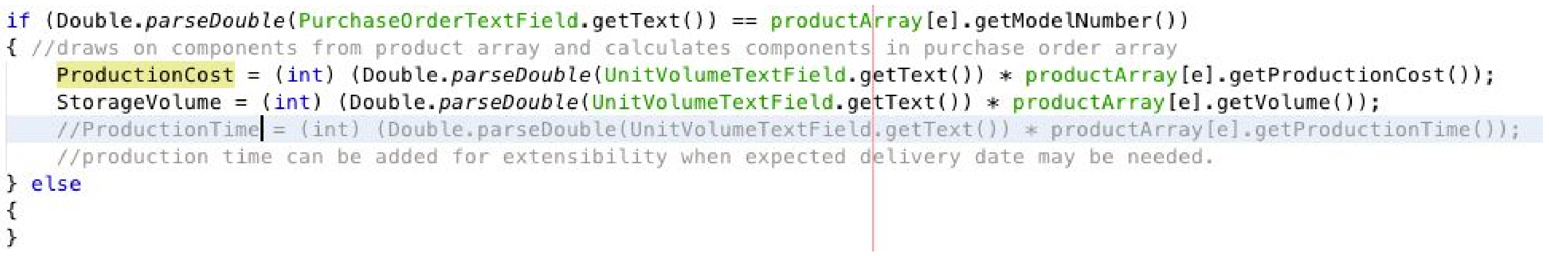
- Total storage volume of purchase order
- Total production cost of purchase order

The following algorithms are used to calculate each value:

Total production time = volume per unit * units ordered

Total production cost = production cost per unit * units ordered

Because these 2 values are to be input into the purchaseOrder array, the volume per unit and production cost per unit is taken from a specific product in the product array.

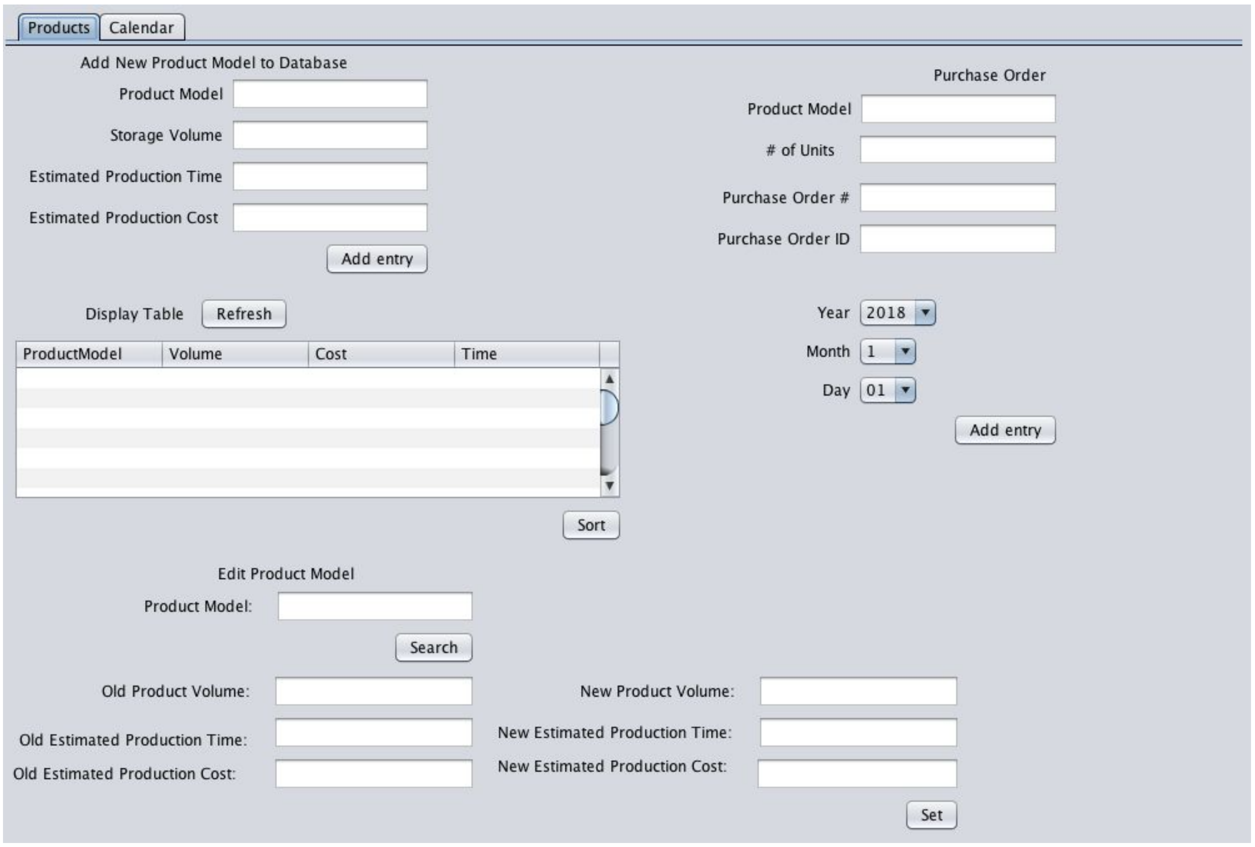


```java
if (Double.parseDouble(PurchaseOrderTextField.getText()) == productArray[e].getModelNumber())
{ //draws on components from product array and calculates components in purchase order array
    ProductionCost = (int) (Double.parseDouble(UnitVolumeTextField.getText()) * productArray[e].getProductionCost());
    StorageVolume = (int) (Double.parseDouble(UnitVolumeTextField.getText()) * productArray[e].getVolume());
    //ProductionTime = (int) (Double.parseDouble(UnitVolumeTextField.getText()) * productArray[e].getProductionTime());
    //production time can be added for extensibility when expected delivery date may be needed.
} else
{
}
```

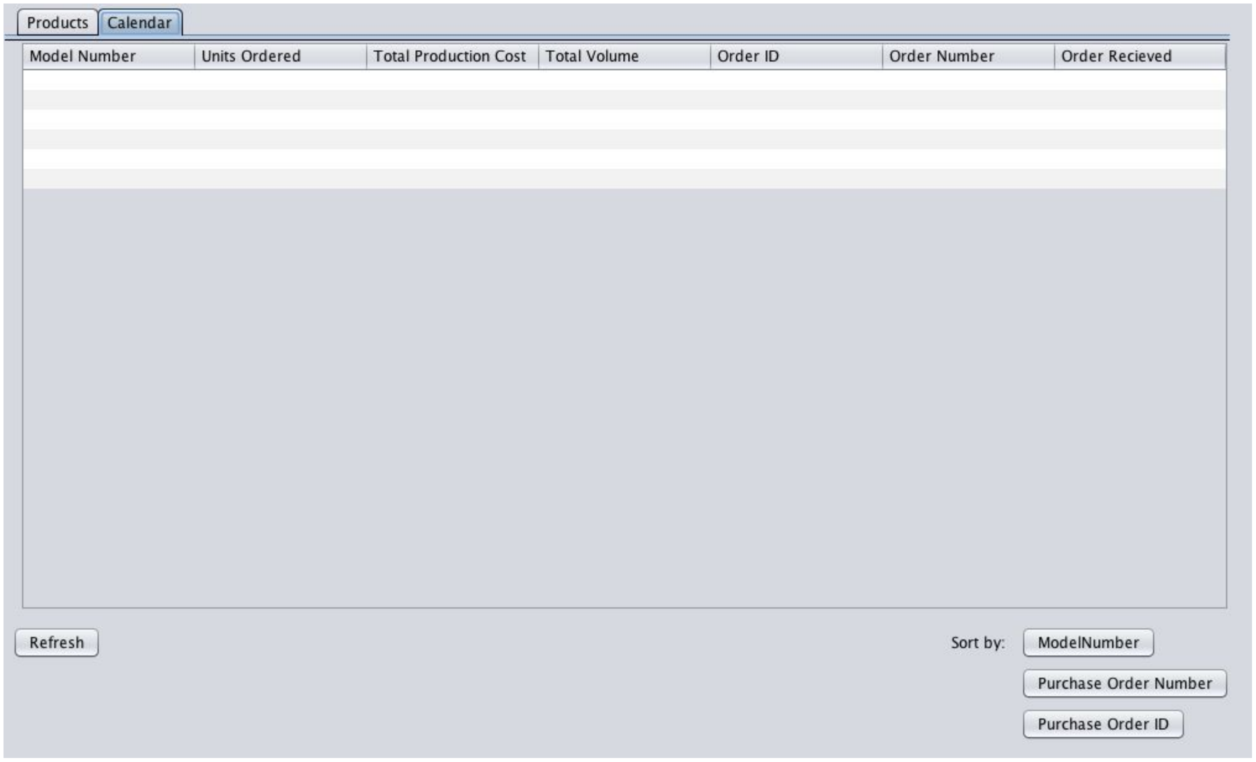
In this case, the value within a text field for user input is taken and compared to every product with the same model number in the product array. If a match is found, the production cost and storage volume per unit of that product model is then taken from that item in the array and multiplied by the number of units the user inputs into the UnitVolumeTextField.

Contextually, the these are the equations as the total production time for each purchase order depends on how many units were ordered and how long it takes to produce each one, the total production cost is the total cost of the raw materials + labor costs. The remaining warehouse storage space is such because each storage unit can only store the same product type while still staying below the total volume of the storage unit. Because of this, (total warehouse storage space / number of storage units) is used to indicate the storage space per storage unit, and the portion that then divides this by volume per unit is to indicate how many units can be stored while still staying below the max volume. This number is then rounded down such that it makes sense in the real world as one would not store half a product.

User Interface and GUI



ProductModel	Volume	Cost	Time



Model Number	Units Ordered	Total Production Cost	Total Volume	Order ID	Order Number	Order Recieved

What:

As it can be seen from the screenshots above, the user interface for the program includes Java swing components such as combo boxes, text fields, labels, tables, buttons, and tabbed panes.

Why:

The labels are there mainly to guide the user on what each part of the program is or what it is supposed to do. Buttons also serve a similar purpose to the labels where they tell the user what pressing the button does to guide them, but to also execute certain lines of code that allows to user to do what the program is meant to do. The table are used to represent the information regarding the product model or purchase order within each array in an easy to understand format to the user. The combo boxes are there to simplify the user input process for the user thereby hopefully increasing the extent to which an employee can use the program efficiently. Tabbed panes are used to make the user interface seem cleaner as well as smaller such that the entire program can fit within the screen of a computer monitor (program would be too big to fit on the screen with 2 tables otherwise).

Software tools used:

- Netbeans

Netbeans was useful is that it allowed for the easy creation of a UI interface for the programs. By using netbeans, the process of creating and editing the GUI to fit the needs of the client was much easier in addition to being fast. Moreover, netbeans allowed for checking mistakes while coding, which was also useful when refining the program.

Word Count: 1083