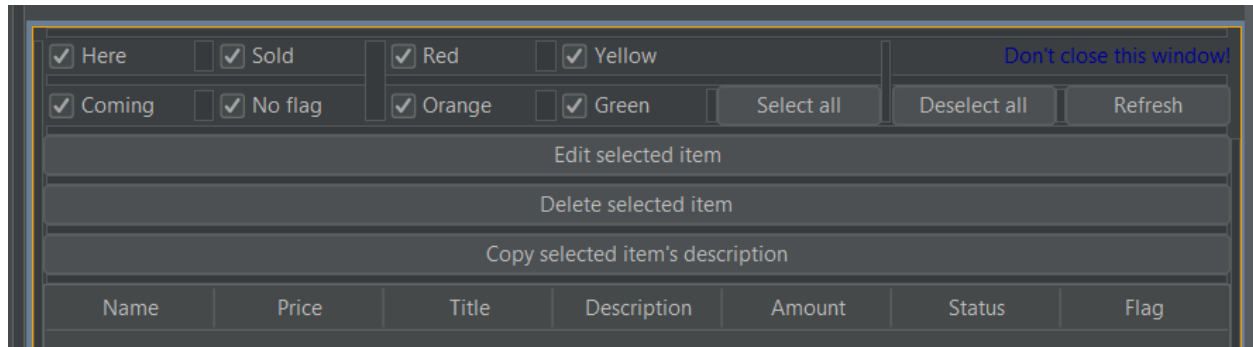


Techniques:

1. Usage of Apache NetBeans to create the GUI components

NetBeans allowed me to create graphical user interfaces more easily, and I was able to use its built-in functionality to save time and headaches. Although NetBeans took a very long time to learn, especially when it came to the JTable, it was very helpful overall. Below is an example of how NetBeans was used to create the GUI elements for the ShowItems window(Note: I'm using the dark theme, but nothing else is different.)



2. Usage of comma-separated-values(.csv) files to store data

The csv file(called "Data1") is stored inside of the project folder, and I used it to store the user's data across many sessions. The important aspect of the file is being able to read and write to it. Much of the code used was carried over from a project with similar functionality done in class prior.

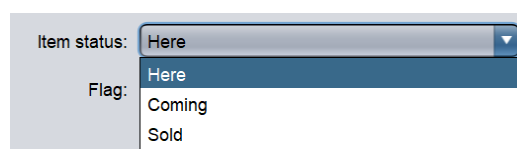
```
public static ArrayList<String[]> readData(String fileName) throws FileNotFoundException{
    Scanner myFile = new Scanner(new File(fileName));
    ArrayList<String[]>data = new ArrayList<String[]>();
    String[]row;
    while(myFile.hasNextLine()){
        String line = myFile.nextLine();
        row = line.split(",");
        data.add(row);
    }
    myFile.close();
    return(data);
}

public static void writeData(ArrayList<String[]> data) throws IOException
{
    FileWriter myFile = new FileWriter(new File("Data1.csv"));
    for(String[] row:data)
    {
        String line = "";
        for(String record:row)
        {
            line+=record+",";
        }
        myFile.write(line.substring(0,line.length()-1)+System.lineSeparator());
    }
    myFile.close();
}
```

3. Usage of Enums to store Item attributes

Learning to use Enums to store data proved to be very important because it allowed me to create dropdown menus for selection. The example below is the ItemStatus enum, which is used to store an item's status. An item can either be coming(in the mail), here(currently in inventory), or sold. The reason that I went with an Enum instead of a String was because an Enum allowed for the user to more easily switch between the different options.

```
1 package Package;
2 public enum ItemStatus{
3     Coming,
4     Here,
5     Sold;
6     public static ItemStatus stringToStatus(String status){
7         switch(status){
8             case "Coming":
9                 return ItemStatus.Coming;
10            case "Here":
11                return ItemStatus.Here;
12            case "Sold":
13                return ItemStatus.Sold;
14        }
15        return ItemStatus.Sold;
16    }
17    public static String statusToString(ItemStatus status){
18        switch(status){
19            case Coming:
20                return "Coming";
21            case Here:
22                return "Here";
23            case Sold:
24                return "Sold";
25        }
26        return "Here";
27    }
28    public String getStatusString(){
29        return this.toString();
30    }
31 }
```



Item status: Here

Flag: Here
Coming
Sold

4. Usage of ArrayList to store and parse data

An ArrayList of String[] was used throughout the project to convert back and forth from the .csv and the table displaying items. An ArrayList of Items was used to store the items in the user's inventory. The reason that an array was not used is because I didn't want to go through the trouble of creating a new array each time an Item was added or deleted. Below you can see the

ArrayList of String[] being read and created into an ArrayList of Items by the ItemManager in the void start function.

```
public void start() {
    ArrayList<String[]>itemData = new ArrayList<String[]>();
    try{itemData = readData("Data1.csv");}
    catch(FileNotFoundException e){
        System.out.println("AAAAAAAHAH! File not found!");
    }

    for(String[] row : itemData.subList(0,itemData.size())){
        int tempIndex = Integer.parseInt(row[0]);
        String tempName = row[1];
        double tempPrice = Double.parseDouble(row[2]);
        String tempTitle = row[3];
        String tempDescription = row[4];
        int tempAmount = Integer.parseInt(row[5]);
        ItemStatus tempStatus = ItemStatus.stringToStatus(row[6]);
        ItemFlag tempFlag = ItemFlag.stringToFlag(row[7]);
        Item newItem = new Item(tempIndex,tempName,tempPrice,tempTitle,tempDescription,tempAmount,tempStatus,tempFlag);
        items.add(newItem);
    }
}
```

5. Usage of switch cases instead of if statements in some places

I wanted to allow the user to toggle between what items in their inventory they're seeing. To do this, I made a selection menu. For example, the selections below would show all items except ones tagged with "Orange"(The color tags represent different labeled boxes).

<input checked="" type="checkbox"/> Here	<input checked="" type="checkbox"/> Sold	<input checked="" type="checkbox"/> Red	<input checked="" type="checkbox"/> Yellow
<input checked="" type="checkbox"/> Coming	<input checked="" type="checkbox"/> No flag	<input type="checkbox"/> Orange	<input checked="" type="checkbox"/> Green

Instead of using a clunky set of if-then statements, I used a shorter and more effective switch statement.

```
if(show){
    switch(item.getFlag()){
        case None:
            if(!none){show=false;}
            break;
        case Red:
            if(!red){show=false;}
            break;
        case Orange:
            if(!orange){show=false;}
            break;
        case Yellow:
            if(!yellow){show=false;}
            break;
        case Green:
            if(!green){show=false;}
            break;
    }
}
```