Criterion C - Development

Introduction

This program performs two main functions as requested by my client, the Lost&Found team: generate a barcode to label lost items and keep track of the lost items. Through using Java's Swing tools in the Netbeans IDE and object-oriented programming, I created a graphical user interface that the client would be able to easily interact with.

Word count: 56

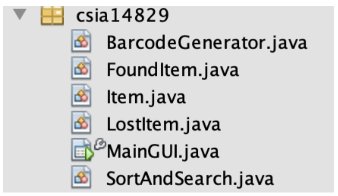
Summary List of All Techniques

- For loops and while loops
- Nested loops
- Parameter passing in methods
- Methods returning a value
- Array of objects
- User defined object made from an OOP "template" class
- Simple and compound selection (if/else)
- Sorting
- Searching
- Error handling
- GUI tabs
- Use of GUI features: jTextField.getText(), itemTypeComboBox.getSelectedItem(), etc.
- Inheritance between a superclass and subclasses
- Use of libraries (for barcode generation)
- Encapsulation of private methods and attributes with accessor and modifier methods
- Use of global variables such as "counter"
- Parsing values into appropriate data types

Word count: 0 (Bulleted List)

Structure of the Program

Overview of all classes:


csia14829
BarcodeGenerator.java
FoundItem.java
Item.java
LostItem.java
MainGUI.java
SortAndSearch.java

The main class of this program is the GUI class. When the user runs this main class, they can interact with the program by entering inputs and viewing outputs generated by the different algorithms of the program.

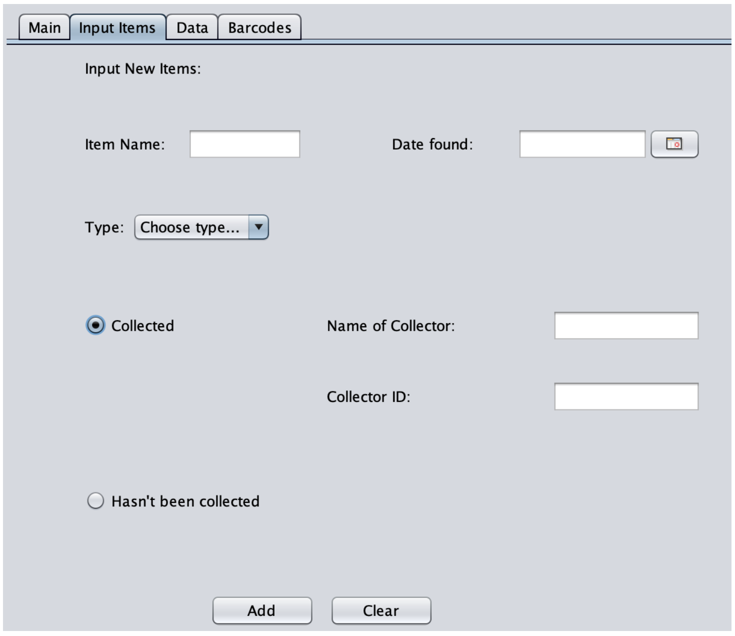
MainGUI.java

The program also includes object template classes which are used to store data. The superclass 'Item' is denoted with the data attributes common to all items that come to the Lost & Found; those such as date found, name of item, etc. In the program, instances of the 'Item' class (objects) are created based on user input. An array of objects is then created to handle all 'Item' type objects.

Item.java

The two subclasses that extend from 'Item' are 'LostItem' and 'FoundItem,' which denote the two types of items the Lost & Found team must keep track of. Since these subclasses extend from 'Item,' they contain all the methods and attributes of 'Item' and, hence, **inheritance** is used. These subclasses were made since if an item has already been found, the client requires more information to be stored in terms of collector name and collector ID. If it is lost, the client must know whether it has been lost for more than four weeks.

Required user input for when 'Collected':



Input New Items:

Item Name:

Date found:

Type: Choose type...

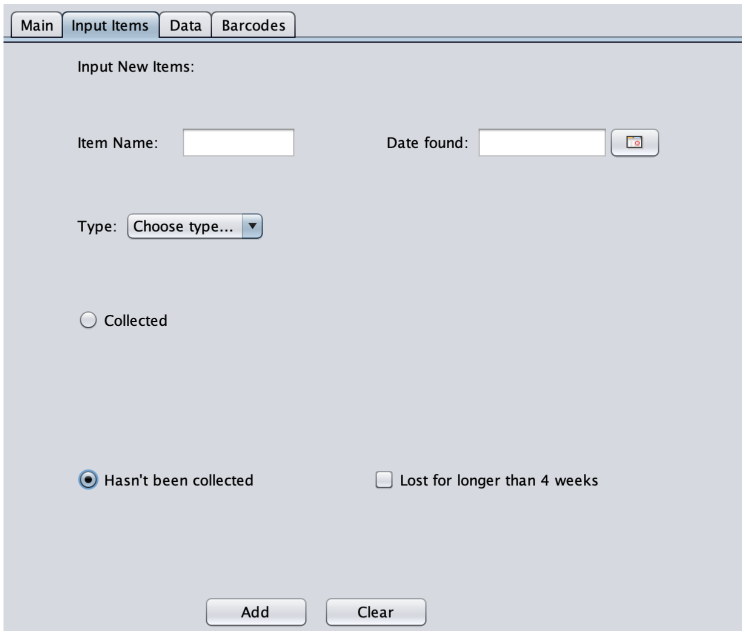
● Collected

Name of Collector:

Collector ID:

○ Hasn't been collected

Required user input for when 'Hasn't been collected':



Input New Items:

Item Name:

Date found:

Type: Choose type...

○ Collected

● Hasn't been collected

☐ Lost for longer than 4 weeks

This structure of classes allows a 'LostItem' object and a 'FoundItem' object to be treated differently as they should. However, the commonality provided by the superclass allows both to be stored in the same data structure.

FoundItem.java LostItem.java

The main GUI class also uses two other classes, enabling the user to use algorithms on the data that has already been inputted. These two classes are the 'SortAndSearch' class and the 'BarcodeGenerator' class. Hence, this program has **dependency**.

BarcodeGenerator.java SortAndSearch.java

Encapsulation is used since with all the classes used by the main GUI (e.g. 'SortAndSearch,' 'Item,' etc.), each class has private attributes and methods which can only be accessed after creating instances of the class. The attributes and methods are further accessed via accessor and modifier methods. With object oriented programming, I was able to use **abstraction** to deconstruct a problem into smaller problems which can be solved. The separate classes such as 'BarcodeGenerator' and 'SortAndSearch' are evidence of this as each of these classes contain specific operations which can then be used by the main GUI class.

Word count: 374

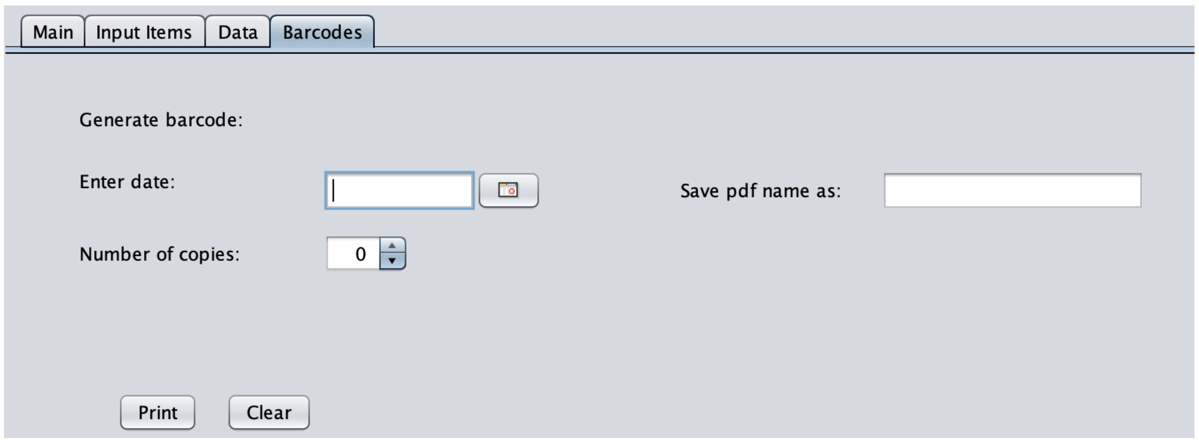
Data Structures Used

1. Array of objects

Arrays were used to store objects that represented individual items brought to the Lost & Found. Arrays were chosen as they can be easily indexed, meaning that sorting and searching processes are made more efficient as opposed to dynamic data structures. An array was also used as the Lost & Found receives more or less the same number of items each month.

Main Unique Algorithms

1. Barcode generator



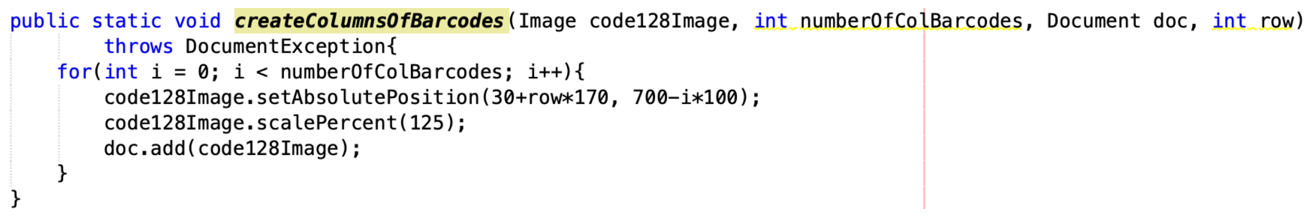
Use of Barcodes:

Since the Lost&Found team had to keep manually creating labels for 'date' the items were found, my client requested an automatic barcode generator. Initially, I had wanted the barcodes to encode more information than just date. However, I decided that that might be too difficult, so instead, this becomes a direction in which my project can extend towards in the future (encoding more information within barcodes). Hence, having these general barcode methods (below) adds to the program's extensibility.

Sample Output:

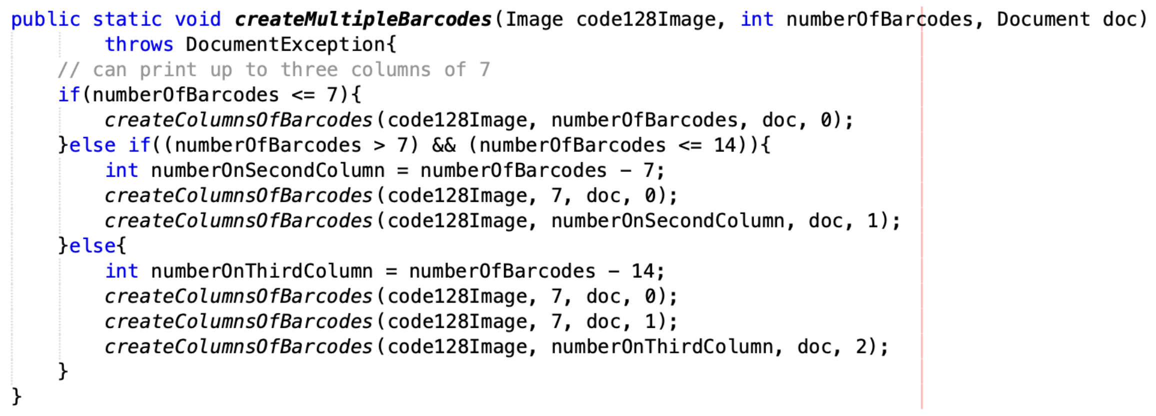
03-05-21

How It Functions:
The barcode generator algorithms are stored in the ‘BarcodeGenerator’ class. The class itself contains two methods that have the function of generating barcodes. The first method ‘createColumnsOfBarcodes’ generates a single column of barcodes.



```
public static void createColumnsOfBarcodes(Image code128Image, int numberOfColBarcodes, Document doc, int row)
        throws DocumentException{
    for(int i = 0; i < numberOfColBarcodes; i++){
        code128Image.setAbsolutePosition(30+row*170, 700-i*100);
        code128Image.scalePercent(125);
        doc.add(code128Image);
    }
}
```

The second method ‘createMultipleBarcodes’ calls the ‘createColumnsOfBarcodes’ method to create the multiple columns of barcodes accurate to the total number of barcodes the user requires. Since the space on each A4 sized pdf file can only contain 3 x 7 barcodes, the ‘createMultipleBarcodes’ method also takes this into account by using conditional statements.



```
public static void createMultipleBarcodes(Image code128Image, int numberOfBarcodes, Document doc)
        throws DocumentException{
    // can print up to three columns of 7
    if(numberOfBarcodes <= 7){
        createColumnsOfBarcodes(code128Image, numberOfBarcodes, doc, 0);
    }else if((numberOfBarcodes > 7) && (numberOfBarcodes <= 14)){
        int numberOnSecondColumn = numberOfBarcodes - 7;
        createColumnsOfBarcodes(code128Image, 7, doc, 0);
        createColumnsOfBarcodes(code128Image, numberOnSecondColumn, doc, 1);
    }else{
        int numberOnThirdColumn = numberOfBarcodes - 14;
        createColumnsOfBarcodes(code128Image, 7, doc, 0);
        createColumnsOfBarcodes(code128Image, 7, doc, 1);
        createColumnsOfBarcodes(code128Image, numberOnThirdColumn, doc, 2);
    }
}
```

2. Generating a pdf of barcodes

Within the ‘BarcodeGenerator’ class is the ‘createPDF’ method used to create the file onto which the barcodes will be placed. This method also calls the ‘createMulitpleBarcodes’ method inorder to create a pdf of barcodes. This method uses the ‘itextpdf-5.4.0.jar’ library and takes in Strings ‘pdfFileName,’ ‘myString,’ and integer ‘numberOfBarcodes’ as arguments, allowing the user to specify what String to encode, what name the file should be saved as, as well as the number of barcodes to be printed.

```java
public static void createPDF(String pdfFilename, String myString, int numberOfBarcodes) {
    Document doc = new Document();
    PdfWriter docWriter = null;
    try {
        docWriter = PdfWriter.getInstance(doc, new FileOutputStream(pdfFilename));
        doc.addAuthor("LostAndFoundTeam");
        doc.addCreationDate();
        doc.addProducer();
        doc.addTitle("Barcodes");
        doc.setPageSize(PageSize.LETTER);
        doc.open();
        PdfContentByte cb = docWriter.getDirectContent();

        Barcode128 code128 = new Barcode128();
        code128.setCode(myString.trim());
        code128.setCodeType(Barcode128.CODE128);

        Image code128Image = code128.createImageWithBarcode(cb, null, null);

        createMultipleBarcodes(code128Image, numberOfBarcodes, doc);

    } catch (DocumentException dex) {
            dex.printStackTrace();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        if (doc != null) {
            doc.close();
        }
        if (docWriter != null) {
            docWriter.close();
        }
    }
}
```

3. Adding Items: Creating Item objects during runtime based on user input

Based on user input, the main GUI class decides whether to store data as a ‘LostItem’ object or a ‘FoundItem’ object.

```java
// based on user's selection of radio button, program creates either a "LostItem" or a "FoundItem" object
if(collectedRadioButton.isSelected()){
    boolean claimedStatus = true;
    String collectorName = collectorNameTF.getText();
    int collectorID = parseInt(collectorIDTF.getText());
    // create new "LostItem" object
    FoundItem foundItem = new FoundItem(name, type, date, claimedStatus, collectorName, collectorID);
    // checks whether input is valid
    if((foundItem.getCollectorID() == -1) || (foundItem.getCollectorName().equals("not a valid input"))){
        errorMessagejLabel.setVisible(true);
        addedTextjLabel.setVisible(false);
    // if input valid, add to array
    }else{
        errorMessagejLabel.setVisible(false);
        itemsArray[counter] = foundItem;
        addedTextjLabel.setVisible(true);
        counter++;
    }
}else if(uncollectedRadioButton.isSelected()){
    boolean claimedStatus = false;
    boolean lostForMoreThanFourWeeks = lostForMoreThanjCheckBox.isSelected();
    // create new "LostItem" object
    LostItem lostItem = new LostItem(name, type, date, claimedStatus, lostForMoreThanFourWeeks);
    // add new "LostItem" object to array
    itemsArray[counter] = lostItem;
    addedTextjLabel.setVisible(true);
    counter++;
}
```

When adding items, there are also different case scenarios that the program must look for. For example, if the user selected ‘other,’ the program should take in user input from the jTextField instead of the jComboBox to make the data attribute for the object.

```
// if "other" is selected take user written input
if(itemTypeComboBox.getSelectedItem()=="Other"){
    type = otherItemTypeTF.getText();
}else{
// otherwise, take selected item from combo box
    type = itemTypeComboBox.getSelectedItem()+"";
```

All the objects are then stored in the correct index of an array via using a counter. The counter also enables the user to see how many items have already been stored.

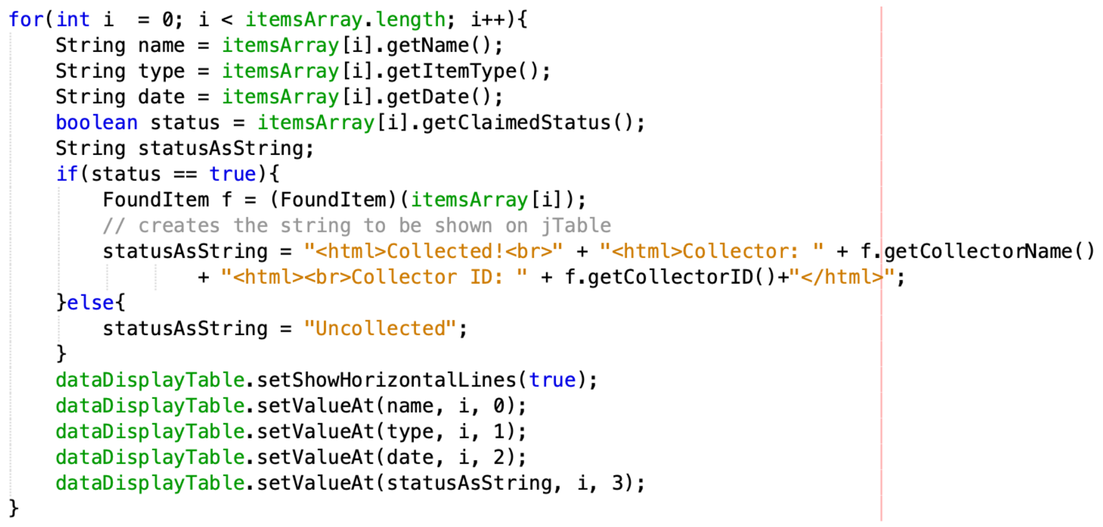
4. Sorting items (notably, sort by ‘Date’ and use of substrings and compound conditionals)

The program also includes Bubble Sort algorithms used to sort objects in the array. The ‘sortByDate’ method is the most complex out of the four. Since the ‘date’ data attribute in the object is stored as a String in the form “dd-mm-yy,” a substring method is first used to divide each ‘date’ into three integers: dd, mm, and yy. This is done so that the year, month, and date of the two ‘dates’ can be compared separately. Through simple and compound selections, two dates are compared accurately and the object switched if appropriate.

```
public void sortByDate(Item[] itemsQueue, int counter) {
    int n = counter;
    boolean sorted = false;
    while (!sorted) {
        n--;
        sorted = true;
        for (int i=0; i < n; i++) {
            int date1 = Integer.parseInt((itemsQueue[i].getDate().substring(0,2)));
            int month1 = Integer.parseInt((itemsQueue[i].getDate().substring(3,5)));
            int year1 = Integer.parseInt((itemsQueue[i].getDate().substring(6,8)));

            int date2 = Integer.parseInt((itemsQueue[i+1].getDate().substring(0,2)));
            int month2 = Integer.parseInt((itemsQueue[i+1].getDate().substring(3,5)));
            int year2 = Integer.parseInt((itemsQueue[i+1].getDate().substring(6,8)));

            if (year1 > year2) {
                switchElements(itemsQueue, i, (i+1));
                sorted = false;
            }else if(year1 == year2){
                if(month1 > month2){
                    switchElements(itemsQueue, i, (i+1));
                    sorted = false;

                }else if((month1 == month2) & (date1 > date2)){
                    switchElements(itemsQueue, i, (i+1));
                    sorted = false;
                }

            }
        }
    }
}
```

5. Display items in the jTable

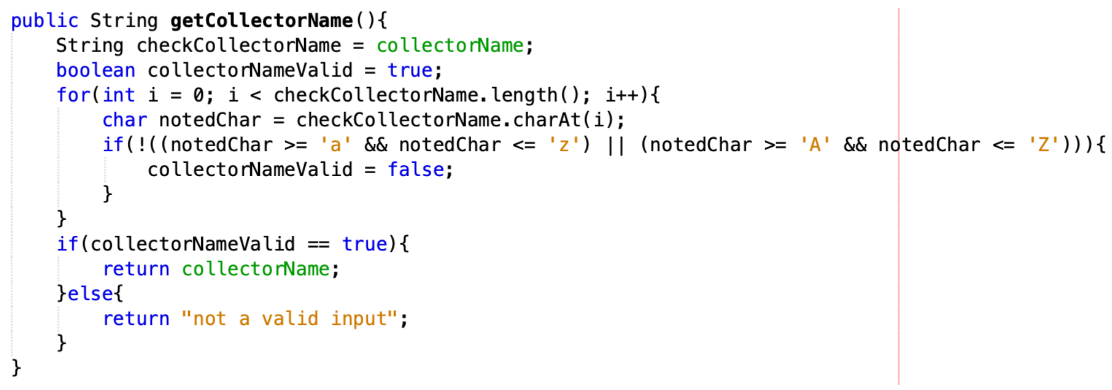
In this program, data about each item is displayed in a jTable. The simple algorithm used for displaying data utilizes a For loop to iterate over the array, filling in data for each row at a time. When an item is found, there is more data that needs to be displayed. Because of this, collector information is concatenated with “collected status” using html string formatting.



```java
for(int i  = 0; i < itemsArray.length; i++){
    String name = itemsArray[i].getName();
    String type = itemsArray[i].getItemType();
    String date = itemsArray[i].getDate();
    boolean status = itemsArray[i].getClaimedStatus();
    String statusAsString;
    if(status == true){
        FoundItem f = (FoundItem)(itemsArray[i]);
        // creates the string to be shown on jTable
        statusAsString = "<html>Collected!<br>" + "<html>Collector: " + f.getCollectorName()
                + "<html><br>Collector ID: " + f.getCollectorID()+"</html>";
    }else{
        statusAsString = "Uncollected";
    }
    dataDisplayTable.setShowHorizontalLines(true);
    dataDisplayTable.setValueAt(name, i, 0);
    dataDisplayTable.setValueAt(type, i, 1);
    dataDisplayTable.setValueAt(date, i, 2);
    dataDisplayTable.setValueAt(statusAsString, i, 3);
}
```

6. Error handling: checking whether all user input is valid (applies mostly to text fields)

Error handling for this program checks whether the user input is valid. This is important as user input is used in multiple algorithms. If an input is not valid, the program does not accept the value and shows an error message. An example (in this case, checking for student name) is shown below.



```java
public String getCollectorName(){
    String checkCollectorName = collectorName;
    boolean collectorNameValid = true;
    for(int i = 0; i < checkCollectorName.length(); i++){
        char notedChar = checkCollectorName.charAt(i);
        if(!((notedChar >= 'a' && notedChar <= 'z') || (notedChar >= 'A' && notedChar <= 'Z'))){
            collectorNameValid = false;
        }
    }
    if(collectorNameValid == true){
        return collectorName;
    }else{
        return "not a valid input";
    }
}
```

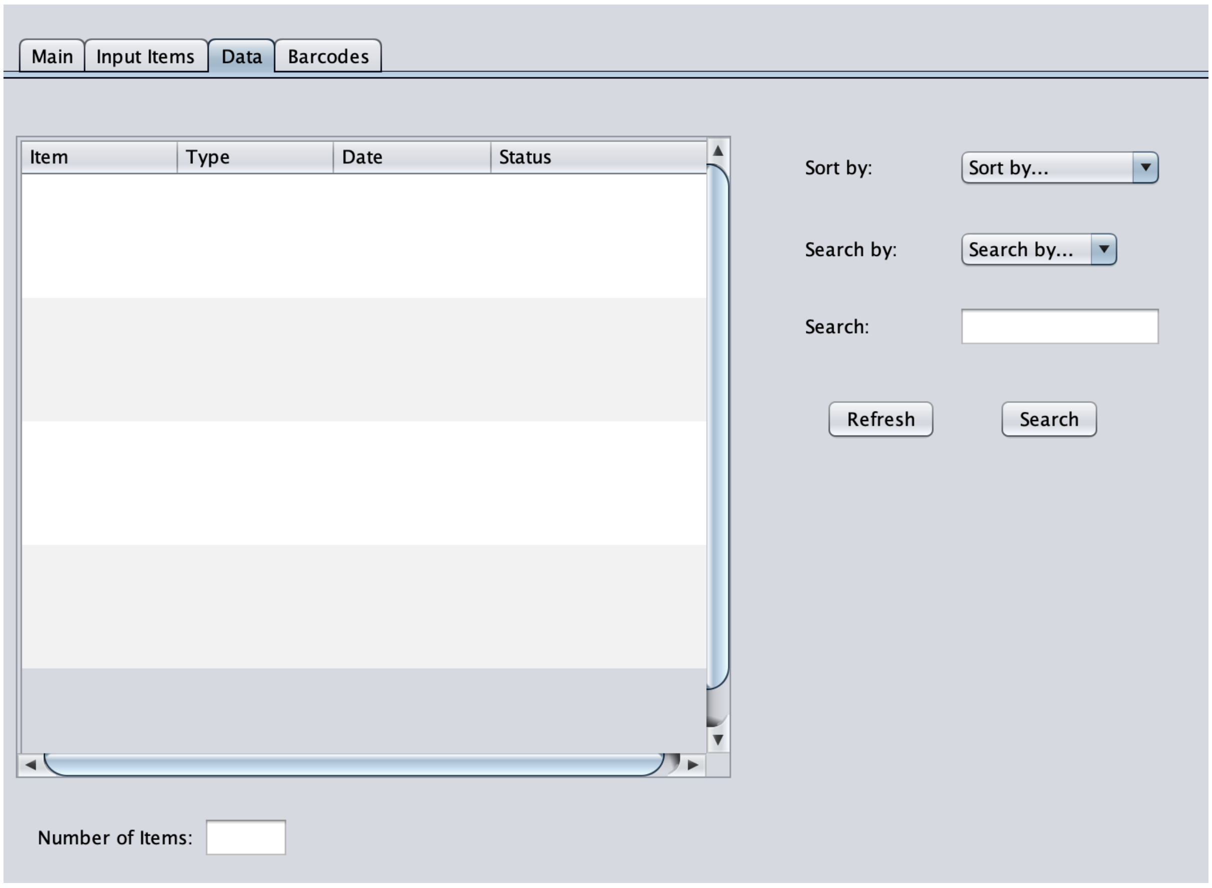
Word count: 612

User Interface/GUI Work

This program utilizes Java's Swing tools to allow the user to interact easily with the GUI. The components used are listed below:

- JTextFields
- JCombobox
- JButtons
- JLabels
- JTabbedPane
- JCheckBox
- JTable

<u>Ex. GUI Features on the 'Data' Tabbed Pane:</u>

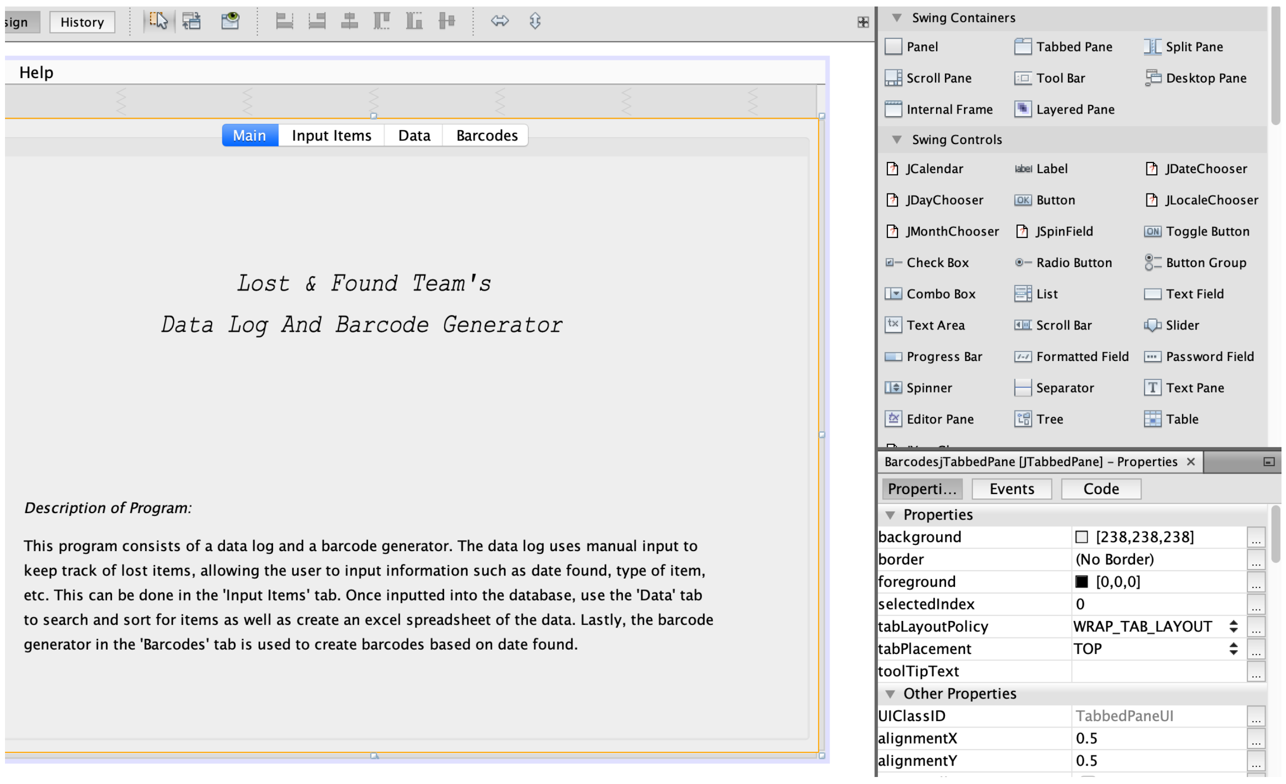


Word count: 22

Software Tools Used

The Netbeans Integrated Development Environment was used to code this program. This tool was chosen as it is user-friendly and allows the programmer to easily create functional GUI interfaces. Two libraries were used to write pdfs and barcodes: ‘itextpdf-5.4.0.jar’ and ‘barcode4j.jar’. Another library was used to aid users in selecting dates: ‘jcalendar-1.4.jar’.

<u>Programmer’s interface on Netbeans (including Swing tools, object properties, GUI view):</u>



Lost & Found Team's

Data Log And Barcode Generator

Description of Program:

This program consists of a data log and a barcode generator. The data log uses manual input to keep track of lost items, allowing the user to input information such as date found, type of item, etc. This can be done in the 'Input Items' tab. Once inputted into the database, use the 'Data' tab to search and sort for items as well as create an excel spreadsheet of the data. Lastly, the barcode generator in the 'Barcodes' tab is used to create barcodes based on date found.

BarcodesjTabbedPane [JTabbedPane] – Properties

Properties	
background	☐ [238,238,238]
border	(No Border)
foreground	■ [0,0,0]
selectedIndex	0
tabLayoutPolicy	WRAP_TAB_LAYOUT
tabPlacement	TOP
toolTipText	
Other Properties	
UIClassID	TabbedPaneUI
alignmentX	0.5
alignmentY	0.5

Word count: 58

Total word count: 1122