Criteria C

Total words: **879**

Introduction *words: 44*

I developed a program that allows my client to take in and store the information associated with different donation contribution forms. New contributions can be added and then edited and exported to Excel. I used the Netbeans IDE to design a GUI interface for ease of use, and an OOP approach to allow for encapsulation, polymorphism, and inheritance.

Summary of Programming Techniques *words: 0*

- parameter passing

```
donorNameTF.setText("");
amountEstimateTF.setText("");
dueDiligenceDateChooser.setDate(null);
dDCommentsTF.setText("");
budgetYearLeftChooser.setYear(2021);
budgetYearRightChooser.setYear(2021);
bYCommentsTF.setText("");
```

```
contributionTable.setValueAt(contributions.get(index).getDonorName(), 0, 0);
```

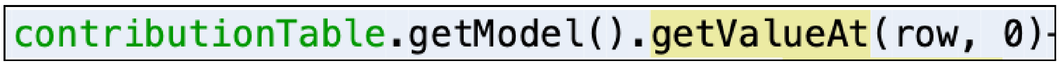
- random number generation

```
Contribution[] finalArray = {genArray[rn.nextInt(2)], ncArray[rn.nextInt(2)], cArray[rn.nextInt(2)]};
```

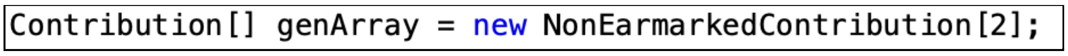
- for loop, while loop, and nested loops, and a method returning a value

```
while (!sorted) {
    sorted = true;
    for (int i=0; i < n-1; i++) {
        String country = contributionArray.get(i).getGeoInterestName();
        if (country.equals("non-earmarked contribution")) {
            Contribution temp = contributionArray.get(i);
            contributionArray.set(i, contributionArray.get(i+1));
            contributionArray.set(i+1, temp);
            sorted = false;
        }
    }
    n--;
}
```

- arrays, 2d arrays, and an array of objects

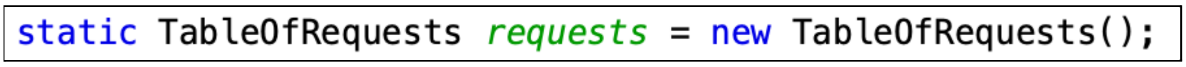


```
contributionTable.getModel().getValueAt(row, 0)
```

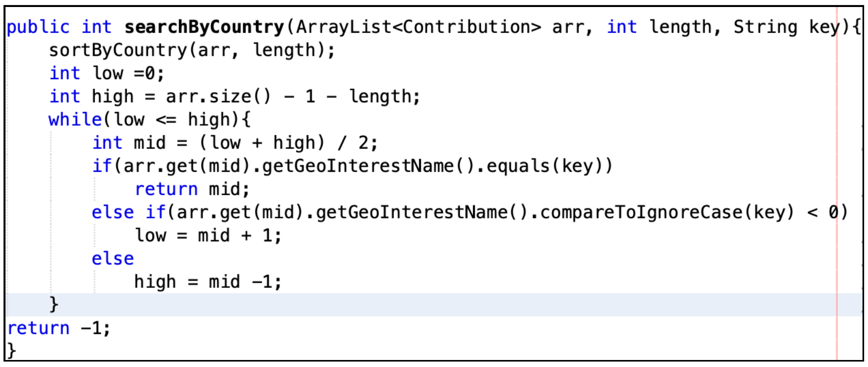


```
Contribution[] genArray = new NonEarmarkedContribution[2];
```

- user defined objects made from an OOP "template" class

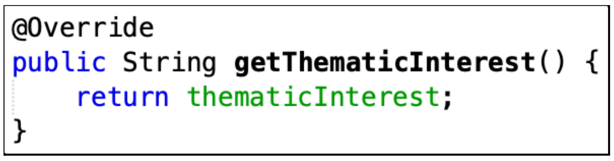


```
static TableOfRequests requests = new TableOfRequests();
```



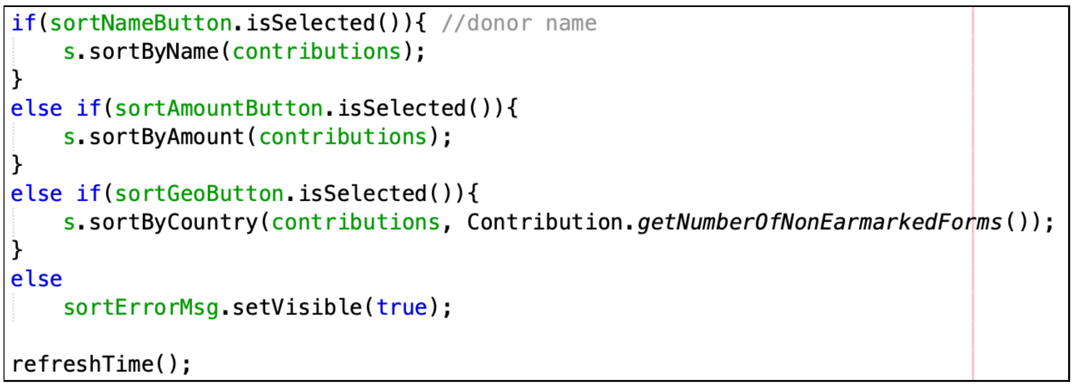
```
public int searchByCountry(ArrayList<Contribution> arr, int length, String key){
    sortByCountry(arr, length);
    int low =0;
    int high = arr.size() - 1 - length;
    while(low <= high){
        int mid = (low + high) / 2;
        if(arr.get(mid).getGeoInterestName().equals(key))
            return mid;
        else if(arr.get(mid).getGeoInterestName().compareToIgnoreCase(key) < 0)
            low = mid + 1;
        else
            high = mid -1;
    }
return -1;
}
```

- encapsulation of private methods that work on public attribute of a "template" class



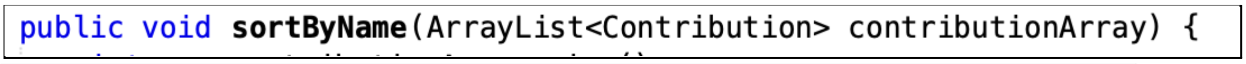
```
@Override
public String getThematicInterest() {
    return thematicInterest;
}
```

- simple and compound selection (if/else)

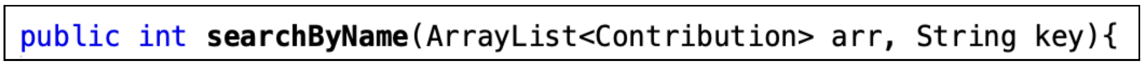


```
if(sortNameButton.isSelected()){ //donor name
    s.sortByName(contributions);
}
else if(sortAmountButton.isSelected()){
    s.sortByAmount(contributions);
}
else if(sortGeoButton.isSelected()){
    s.sortByCountry(contributions, Contribution.getNumberOfNonEarmarkedForms());
}
else
    sortErrorMsg.setVisible(true);

refreshTime();
```

- sorting (bubble sort) and searching (binary search…)

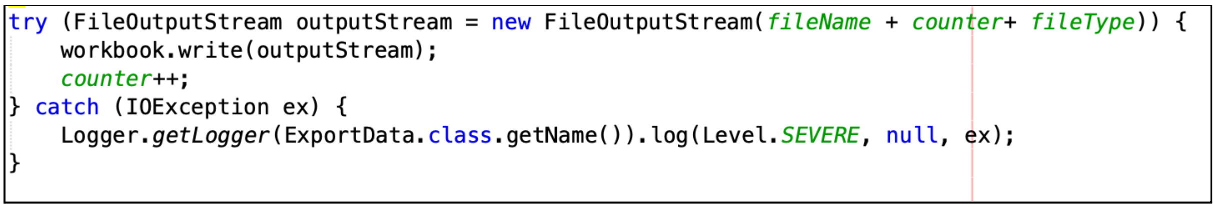


```
public void sortByName(ArrayList<Contribution> contributionArray) {
```



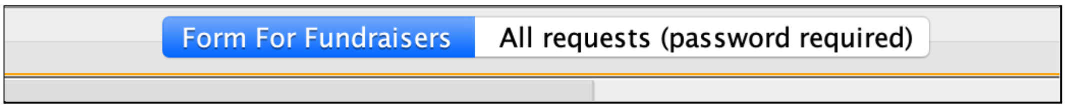
```
public int searchByName(ArrayList<Contribution> arr, String key){
```

- saving to a file and catching an IOException

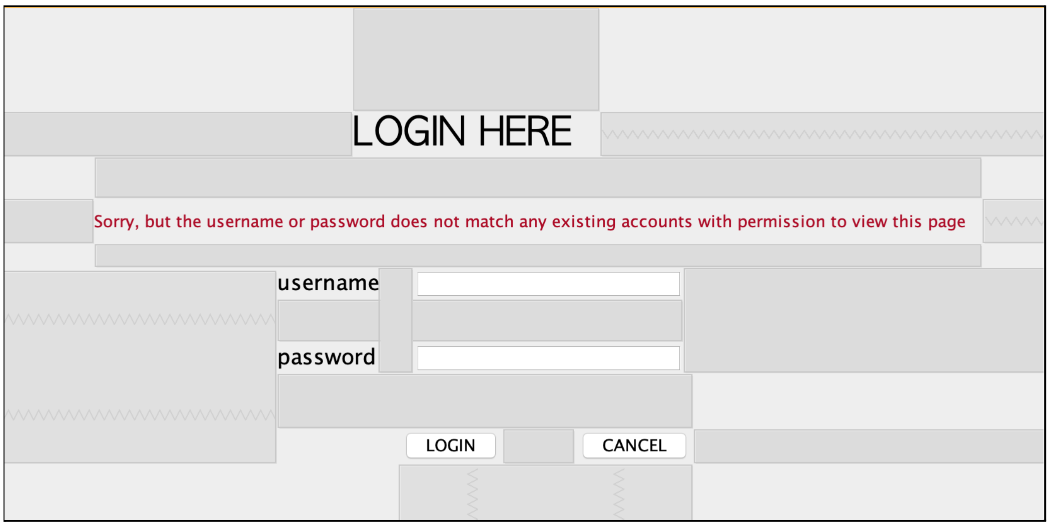


```java
try (FileOutputStream outputStream = new FileOutputStream(fileName + counter+ fileType)) {
    workbook.write(outputStream);
    counter++;
} catch (IOException ex) {
    Logger.getLogger(ExportData.class.getName()).log(Level.SEVERE, null, ex);
}
```

- GUI tabs

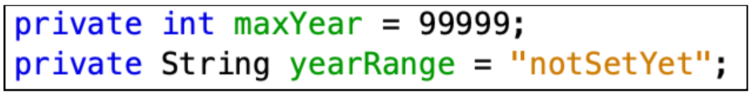


- GUI popup menus



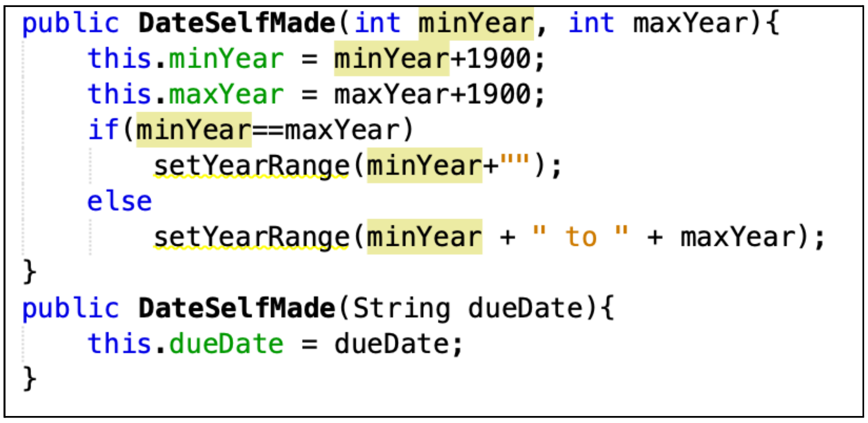
Sorry, but the username or password does not match any existing accounts with permission to view this page

- Use of a flag value (such as -999, or "not set yet")



```java
private int maxYear = 99999;
private String yearRange = "notSetYet";
```

- overloaded constructors, which work differently depending on the parameters sent

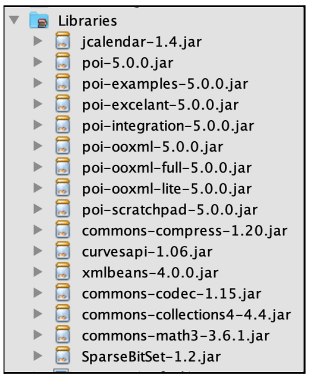


```java
public DateSelfMade(int minYear, int maxYear){
    this.minYear = minYear+1900;
    this.maxYear = maxYear+1900;
    if(minYear==maxYear)
        setYearRange(minYear+"");
    else
        setYearRange(minYear + " to " + maxYear);
}
public DateSelfMade(String dueDate){
    this.dueDate = dueDate;
}
```

- inheritance between a superclass and a subclass

```
public class NonEarmarkedContribution extends Contribution{
```

- use of some other specialized library you imported



```
Libraries
  jcalendar-1.4.jar
  poi-5.0.0.jar
  poi-examples-5.0.0.jar
  poi-excelant-5.0.0.jar
  poi-integration-5.0.0.jar
  poi-ooxml-5.0.0.jar
  poi-ooxml-full-5.0.0.jar
  poi-ooxml-lite-5.0.0.jar
  poi-scratchpad-5.0.0.jar
  commons-compress-1.20.jar
  curvesapi-1.06.jar
  xmlbeans-4.0.0.jar
  commons-codec-1.15.jar
  commons-collections4-4.4.jar
  commons-math3-3.6.1.jar
  SparseBitSet-1.2.jar
```

- linked list, or any other ADT, like binary search tree class…

```
public static ArrayList<Contribution> contributions = new ArrayList<Contribution>();
```

- ADT methods

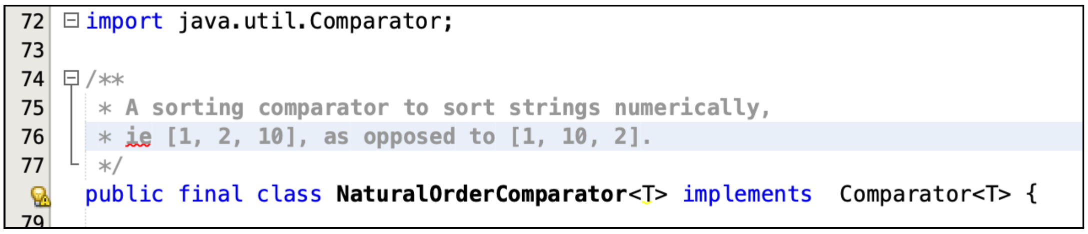
```
contributionTable.setValueAt(contributions.get(index).getDonorName(), 0, 0);
```

```
contributions.add(newNonContribution);
```

```
contributionTable.setValueAt(contributions.get(index).getDonorName(), 0, 0);
```

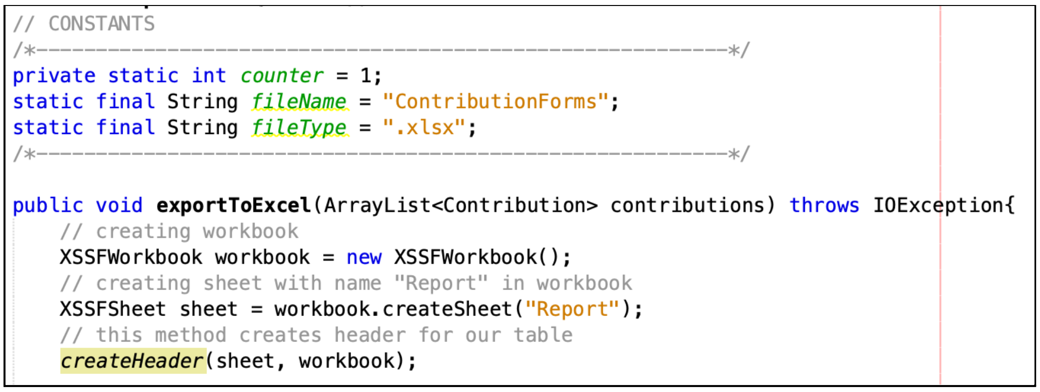
Unique tool implementation *words: 124*

The imported class NaturalOrderComparator is used to sort the estimated amount of donation money. This required a special class because the amount estimated is set as a String value, with the potential of including non integer characters. So by passing the amount attribute of the Contribution ArrayList, I can use the NaturalOrderComparator to ensure no errors are thrown when my program is run.



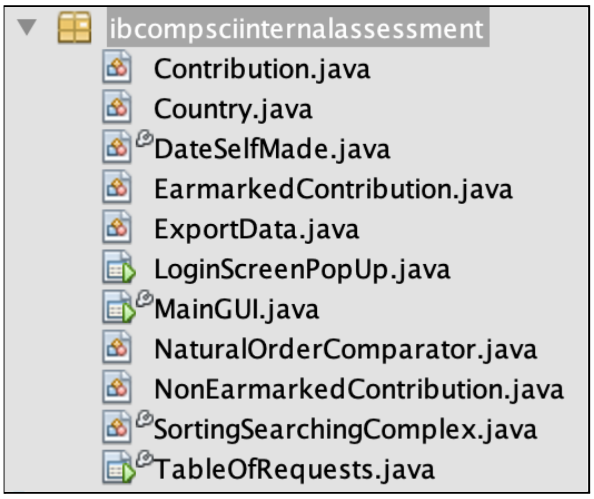
```
import java.util.Comparator;

/**
 * A sorting comparator to sort strings numerically,
 * ie [1, 2, 10], as opposed to [1, 10, 2].
 */
public final class NaturalOrderComparator<T> implements  Comparator<T> {
```

Another imported class is the ExportData class which allows my program to save a .xlsx type file which can be opened in Excel. This functionality uses apache.poi libraries and creates a data table with headers and rows for each unique contribution. This allows my client to save data in such a way that they can send it to coworkers easily.



```
// CONSTANTS
/*------------------------------------------------------------*/
private static int counter = 1;
static final String fileName = "ContributionForms";
static final String fileType = ".xlsx";
/*------------------------------------------------------------*/

public void exportToExcel(ArrayList<Contribution> contributions) throws IOException{
    // creating workbook
    XSSFWorkbook workbook = new XSSFWorkbook();
    // creating sheet with name "Report" in workbook
    XSSFSheet sheet = workbook.createSheet("Report");
    // this method creates header for our table
    createHeader(sheet, workbook);
```

Structure of the Program *words: 177*

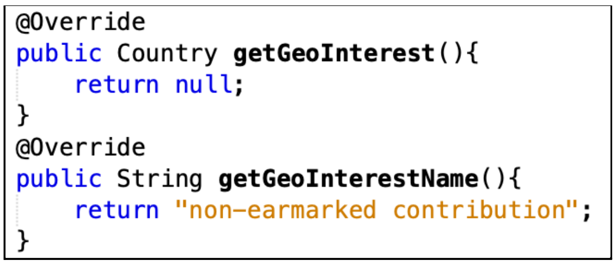


```
ibcompsciinternalassessment
    Contribution.java
    Country.java
    DateSelfMade.java
    EarmarkedContribution.java
    ExportData.java
    LoginScreenPopUp.java
    MainGUI.java
    NaturalOrderComparator.java
    NonEarmarkedContribution.java
    SortingSearchingComplex.java
    TableOfRequests.java
```

- The main class has instances of the jFrames *LoginScreenPopUp* and *TableOfRequests*.
- The classes *NonEarmarkedContribution* and *EarmarkedContribution* are both subclasses of the *Contribution* class
- The Country class takes in instances of objects of type Country and DateSelfMade.
- Instances of the SortingSearchingComplex, NaturalOrderComparator, and ExportData are used to modify and interact with the data in the TableOfRequest class.

I decided to separate my program into these different classes because it allowed me to keep track of the functionality for each class. It also allowed me to use encapsulation to remove the access the client would have to some of the variables and methods. This increases the security and hides the data in the classes from non-authorized users or methods.

The subclass structures allow polymorphism in the arrayList. By creating the ArrayList of Contribution objects I could then specify the object type as one of the subclasses and have different functionalities depending on the item type. By overriding methods I was able to handle contribution forms both when they are assigned to a particular country and when they are not.

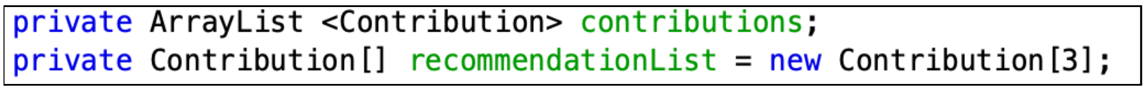


```java
@Override
public Country getGeoInterest(){
    return null;
}
@Override
public String getGeoInterestName(){
    return "non-earmarked contribution";
}
```

Data Structures Used *words: 137*

I used an Array of objects, 2d Arrays, and ArrayLists to structure my data. Both the 2d Arrays and ArrayLists are not fundamental data structures in Java, however their additional functionalities were the most appropriate for my program. The dynamic nature of ArrayLists helped ensure that my client will be able to add an unrestricted amount of new contributions, without setting aside a potentially excessive amount of memory space. The 2d arrays were used as tables to present the contributions in a visually intuitive manner, while the array was used to store the recommended contributions. This made sense because there will only ever be at most 3 contributions that need to be in the array at any given moment.

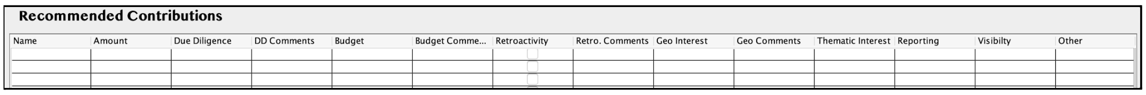
Most of the data structures used, employed attributes of the user defined object Contribution and its 2 subclasses.



```java
private ArrayList <Contribution> contributions;
private Contribution[] recommendationList = new Contribution[3];
```

Unique Algorithms *words: 275*

A more complex part of my program is the recommendation table. While all the contributions are shown in the main table, 3 specific contributions show up in a smaller table. This is to help my client prioritize working on these particular contributions which are sorted and selected based on a few criteria.



Recommended Contributions

Name	Amount	Due Diligence	DD Comments	Budget	Budget Comme...	Retroactivity	Retro. Comments	Geo Interest	Geo Comments	Thematic Interest	Reporting	Visibilty	Other
						☐							
						☐							
						☐							

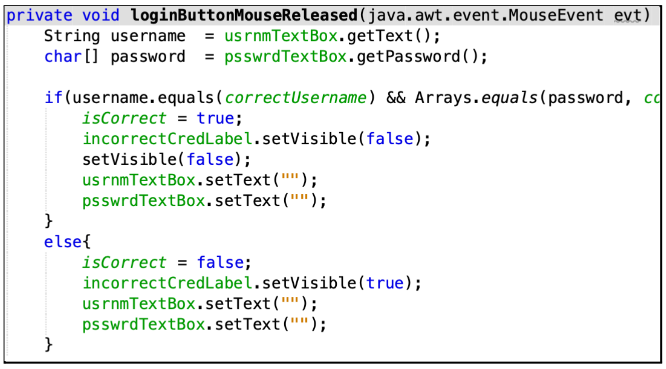
In this recommendation table there is one example of a non earmarked contribution (meaning that it has no geographical interest), one Earmarked contribution that does not have all essential attributes such as donor name or amount estimated filled in, and a final contribution that has all important data fields completed.

```java
try{
    for(int index = 0; index<contributions.size(); index++){
        String country = contributions.get(index).getGeoInterestName();
        //if contribution is not earmarked and there is space in the genArray left
        if (country.equals("non-earmarked contribution") && genIndex<2) {
            System.out.println("nonE Added");
            genArray[genIndex] = contributions.get(index);
            genIndex++;
        }//if contribution is earmarked, key info like donor name/amount is blank, and there is space in the ncArray left
        else if((contributions.get(index).getDonorName().equals("") || contributions.get(index).getAmountEstimated().equals("")) && ncIndex<2){
            System.out.println("nc Added");
            ncArray[ncIndex] = contributions.get(index);
            ncIndex++;
        }//otherwise and space in cArray
        else if(cIndex <2){
            System.out.println("c Added");
            cArray[cIndex] = contributions.get(index);
            cIndex++;
        }
        else
            System.out.print("Out of space in the arrays");
    }
```

(...more code can be found in my program..)

```java
}
catch(NullPointerException error){
    System.out.println("OH NO! fatal " + error);
}
```

From the large list of all contributions the recommended options are selected based on the severity of the crisis in a particular country (this severity rating is set based on the refugee crisis in a set of countries) and on the amount estimated for the donation. I chose this design because it seemed important to have an easy to access set of contributions that my client might want to focus on. The slight random selection of two high priority contributions in each of the three categories is an efficient way to cycle through important contributions and increase the likelihood of highlighting a contribution my client may want to focus on.



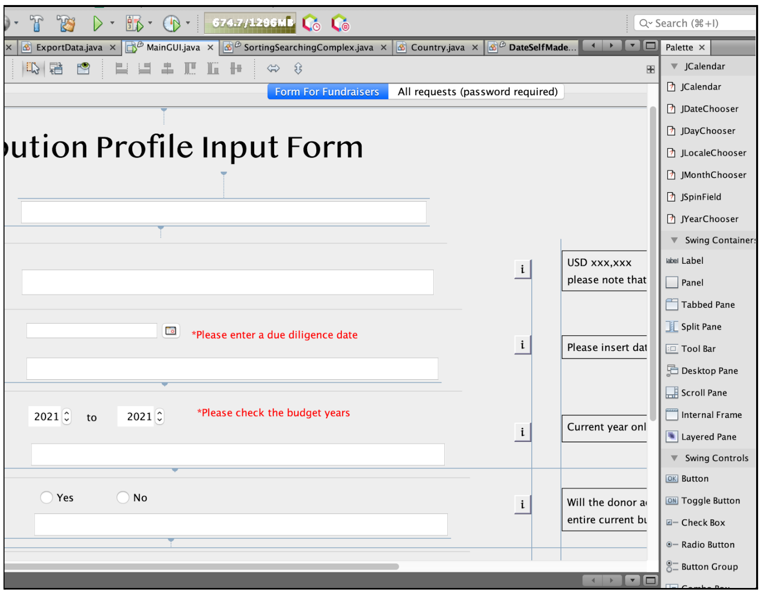
```
private void loginButtonMouseReleased(java.awt.event.MouseEvent evt)
    String username  = usrnmTextBox.getText();
    char[] password  = psswrdTextBox.getPassword();

    if(username.equals(correctUsername) && Arrays.equals(password, co
        isCorrect = true;
        incorrectCredLabel.setVisible(false);
        setVisible(false);
        usrnmTextBox.setText("");
        psswrdTextBox.setText("");
    }
    else{
        isCorrect = false;
        incorrectCredLabel.setVisible(true);
        usrnmTextBox.setText("");
        psswrdTextBox.setText("");
    }
```

I also separated the profile input form from the list of contributions by restricting access through a login screen. This is important because the donors should not have access to the other contributions. By having a login screen pop up and a boolean value linked to the username and password text fields, only the allowed client can make modifications to the contribution list.

Software Tools Used and User Interface/GUI Work *words: 122*

I used the Netbeans IDE - which stands for Integrated Development Environment - to design my code. I chose this IDE because it has features such as integrated spellchecker, debugger, and can run the program from inside the IDE. It is especially user friendly because of its intuitive interface and vibrant color scheme.

Netbeans also allows the creation of a GUI interface. I was able to use the netbeans features to design GUI components to improve the accessibility of my program. I used Combo Boxes, Radio Buttons, Text Fields, Buttons, and Tables to ensure an ease of use for my client. Netbeans made the implementation of these java Swing elements simpler as each item can be dragged/dropped into place and used very intuitively.



Form For Fundraisers | All requests (password required)

ution Profile Input Form

*Please enter a due diligence date

2021 to 2021 *Please check the budget years

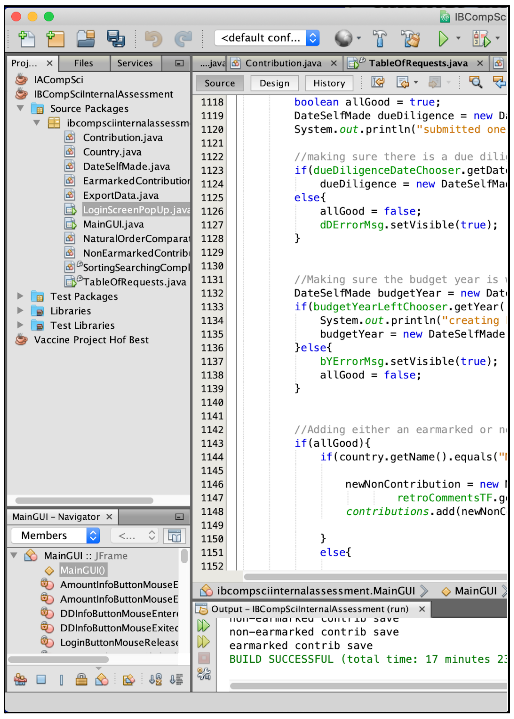
Yes No

USD xxx,xxx please note that

Please insert dat

Current year onl

Will the donor a entire current bu



```
        boolean allGood = true;
        DateSelfMade dueDiligence = new Da
        System.out.println("submitted one

        //making sure there is a due dili
        if(dueDiligenceDateChooser.getDate
            dueDiligence = new DateSelfMad
        else{
            allGood = false;
            dDErrorMsg.setVisible(true);
        }


        //Making sure the budget year is
        DateSelfMade budgetYear = new Date
        if(budgetYearLeftChooser.getYear(
            System.out.println("creating
            budgetYear = new DateSelfMade
        }else{
            bYErrorMsg.setVisible(true);
            allGood = false;
        }


        //Adding either an earmarked or no
        if(allGood){
            if(country.getName().equals("

                newNonContribution = new N
                        retroCommentsTF.ge
                contributions.add(newNonC

            }
            else{
```

```
non-earmarked contrib save
non-earmarked contrib save
earmarked contrib save
BUILD SUCCESSFUL (total time: 17 minutes 23
```