

## **Introduction**

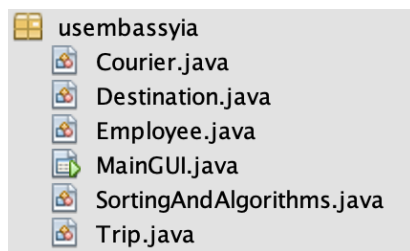
My program was developed in Apache Netbeans IDE in Java. The program uses a database to keep track of trip history with my clients' subordinates. With this information the program can run an algorithm that calculates the best courier for the next trip. I used Netbeans to take advantage of the Java swing elements to construct a GUI so usability of the program would be better.

Word Count: 66

## **Summary of Techniques**

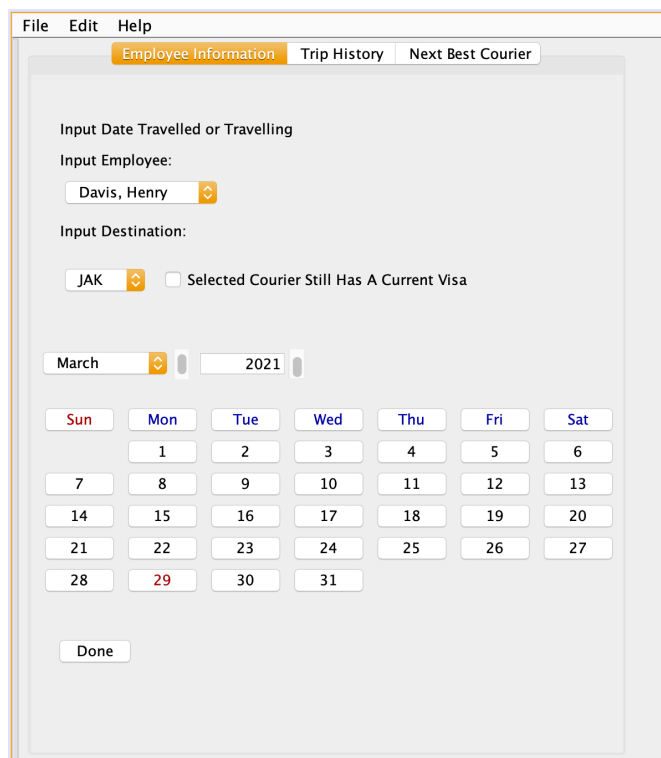
- nested loops
- method returning a value
- arrays, 2d arrays
- user defined objects made from an OOP "template" class
- encapsulation of private methods that work on public attribute of a "template" class
- Array of objects
- sorting (bubble sort etc.), and in particular sorting an array of objects based on one key attribute
- Graphical User Interface
- inheritance between a super class and a sub class
- use of some other specialized library you imported
- Use of ArrayList

## Structure of the Program



The structure of the program uses many different classes to reach its ultimate goals. The program contains 6 Classes, 1 to execute and display information to the user (MainGUI), 4 others to serve as template classes, and the last one to serve as a sorting and algorithm executioner. I used these classes to take an object oriented approach to this program. Using this approach allows me to take advantage of OOP's Polymorphism, Inheritance, and Encapsulation. The Courier, Destination and Employee classes contain overloaded constructors and thus explains why Polymorphism is used. The Courier class is an Inheritance from the Employee Class. This was done due to the fact that not every employee may be a Courier for my client's firm. An example of encapsulation is the Trip class where a Courier and Destination are contained and can be used as a logger to keep track of where and when a courier travelled.

### GUI Structure:



The First structure of the GUI is the first tab, which is a logger which creates a Trip and with each input, the respective courier's destination travelled counter is added by one. This

allows for the calculation of the “Next Best Courier” to occur. In the Second tab, “Trip History”  
The ArrayList of inputted Trips is displayed

The screenshot shows a Java Swing window with a menu bar (File, Edit, Help) and three tabs: "Employee Information", "Trip History" (selected), and "Next Best Courier". The "Trip History" tab contains a table with three columns: "Name", "Destination", and "Date". The table is currently empty. A "Refresh" button is located at the bottom left of the table area.

In the 3rd Tab the Next Best Courier Is calculated based on the information inputted from the Employee Information tab and their Name is displayed.

The screenshot shows the same Java Swing window with the "Next Best Courier" tab selected. The tab contains two sections. The first section, "Choose Destination:", has a list box with the following items: JAK, BKK, LEN, and BAN. The second section, "Best Courier For The Job:", has a text field containing "Last, First". Below this, the "Frequency of post service:" section has a text field containing "Frequency". A "Calculate" button is located at the bottom left of the tab area.

Word Count: 236

## Data Structures Used

### Object arrays and 2D Arrays

I used various abstract data types such as: Object Arrays, ArrayLists, and 2d arrays in my program. They are abstract because they do not exist primitively within Java such as an int or String. I used Object arrays and 2d arrays conjunctively and the reason is the same. As arrays are static in size, I used it to apply to the destinations and couriers.

```
29     Courier[] courierArray = new Courier[10];
30     Destination[] destinationArray = new Destination[4];

14     public class Courier extends Employee {
15         boolean fitToFly;
16         public Destination[] allDestinations = new Destination[4];
```

The core 2d array is structured where an array of Couriers where each Courier contains an Array of Destinations. I used the 2d array to keep track of the amount of times a particular courier went to a specific destination. They are arrays because destinations and employees are unlikely to change very quickly. The size of the courier and destination arrays are predictable.

### ArrayLists

```
27     ArrayList<Trip> tripsList = new ArrayList<>();
28     ArrayList<Courier> courierArrayList = new ArrayList<>();
```

The program uses ArrayLists in addition to object arrays, the ArrayLists exist to support a more volatile amount of data. For example the amount of Trips that the user logs will always be changing in a short amount of time, therefore there is no way the program can know how many Trips will be made and therefore cannot set a static array for that size. So In this case the ArrayList is used to compensate for the volatility of the amount of Trips.

The Courier ArrayList exists as a way of sorting the existing courierArray from the boolean hasVisa and only contains couriers who have a valid visa for the specified destination. This ArrayList is used in the SortingAndAlgorithms class to carry out calculations. The reason the ArrayList is used is similar to the one stated for the Trips ArrayList and that is the amount of couriers who hasVisa is unknown and cannot create a new array with the set size.

Word Count: 289

## Main Algorithms

The `sortByHasVisa` method takes in the Courier array the user specified destination. A temporary `ArrayList` is declared and the courier array is looped through once to check the `hasVisa` boolean for each courier at that destination. If a courier hasVisa, then it is added to the `ArrayList`. At the end of the loop, the `ArrayList` is returned.

```
35 public ArrayList<Courier> sortByHasVisa(Courier [] courierArray, int destinationIndex){
36     ArrayList<Courier> temp = new ArrayList<>();
37
38     for(int i=0; i < courierArray.length-1; i++){
39         if(courierArray[i].allDestinations[destinationIndex].getHasVisa()){
40             temp.add(courierArray[i]);
41         }
42     }
43
44     return temp;
45 }
46 }
```

The `tripDeterminer` method takes in the returned Courier `ArrayList` and the specified destination. The `ArrayList` is first sorted by the couriers' counters. Then if the size of the `ArrayList` is greater than or equal to 2, the date tenured of the last 2 couriers are compared. If the last courier's date tenured is longer than the second to last courier, the last courier is returned. Else the second to last courier is returned. For all other situations, the last courier is returned.

```
48 public int tripDeterminer(ArrayList<Courier> courierArray, int destinationIndex){
49
50     sortByCourierDestinationCounter(courierArray, destinationIndex);
51     if(courierArray.size() >=2){
52         if(courierArray.get(courierArray.size()-1).allDestinations[destinationIndex].getCounter() ==
53            courierArray.get(courierArray.size()-2).allDestinations[destinationIndex].getCounter()){
54             if(courierArray.get(courierArray.size()-1).getDateTenured() <
55                courierArray.get(courierArray.size()-2).getDateTenured() ){
56                 return courierArray.size()-1;
57             }
58             else{
59                 return courierArray.size()-2;
60             }
61         }
62         else{
63             return courierArray.size()-1;
64         }
65     }
66     else{
67         return courierArray.size()-1;
68     }
69 }
70 }
71 }
72 }
73 }
```

The submitCourierMouseButtonReleased method is executed when the user has finished inputting the trip history in the Employee Information tab. The courier is defined by the user specified item in its combo box. The Destination is declared along with the hasVisa. The counter for the Courier at the specified destination is incremented by 1 and the hasVisa value for the courier at the destination is also set. The Date of the trip is taken in by lines 586-589. In line 593, the Courier, Destination and Date are inputted to a new trip and added to the ArrayList in 595. Lines 596 and 597 reset the combo boxes.

```

579 private void submitCourierMouseButtonReleased(java.awt.event.MouseEvent evt) {
580     // TODO add your handling code here:
581     int employeeIndex = employeeCB.getSelectedIndex();
582     int destinationIndex = destinationCB.getSelectedIndex();
583     boolean hasVisa = hasVisaCB.isSelected();
584     courierArray[employeeIndex].allDestinations[destinationIndex].counter++;
585     courierArray[employeeIndex].allDestinations[destinationIndex].setHasVisa(hasVisa);
586     String month = String.valueOf(jMonthChooser1.getMonth());
587     String day = String.valueOf(jDayChooser1.getDay());
588     String year = String.valueOf(jYearChooser2.getYear());
589     String inputtedDate = new String(month + "/" + day + "/" + year);
590
591
592
593     Trip newTrip = new Trip(courierArray[employeeIndex], destinationArray[destinationIndex],inputtedDate);
594
595     tripsList.add(newTrip);
596     employeeCB.setSelectedIndex(0);
597     destinationCB.setSelectedIndex(0);
598
599
600
601
602 }

```

The refreshButtonMouseReleased method is executed when the Refresh button is pressed in the "Trip History" tab, when it is pressed, the Trips ArrayList gets displayed in the displayTable. This is done using the for loop which loops through the whole array list.

```

626 private void refreshButtonMouseReleased(java.awt.event.MouseEvent evt) {
627
628     for (int row = 0; row < tripsList.size(); row++) {
629         displayTable.setValueAt(tripsList.get(row).getTripCourier().getName(), row, 0);
630         displayTable.setValueAt(tripsList.get(row).getTripDestination().getName(), row, 1);
631         displayTable.setValueAt(tripsList.get(row).getTripDate(), row, 2);
632     }
633     // TODO add your handling code here:
634 }

```

**Word Count: 287**

**Software Tools Used:**

The main development software I used was Netbeans IDE. I used it because of its Drag and drop style GUI builder. This made it very easy to construct and prototype the program for my client. On top of that the software is object oriented friendly and allows for the addition of foreign classes i.e jCalendar.

**Word Count: 56**

**Total Word Count: 934**

grade