Criterion C: Development

Introduction:

I have used the java development environment Netbeans to develop my program and have used the JTable, JComboBox, JTextfield, alongside other functions to make my Graphical User Interface. My program allows for the user to insert in their song with a chart to display the chords used. The main unique feature is the ability to transpose those chords inserted in the programs to different keys, which would be helpful to use in a live setting.

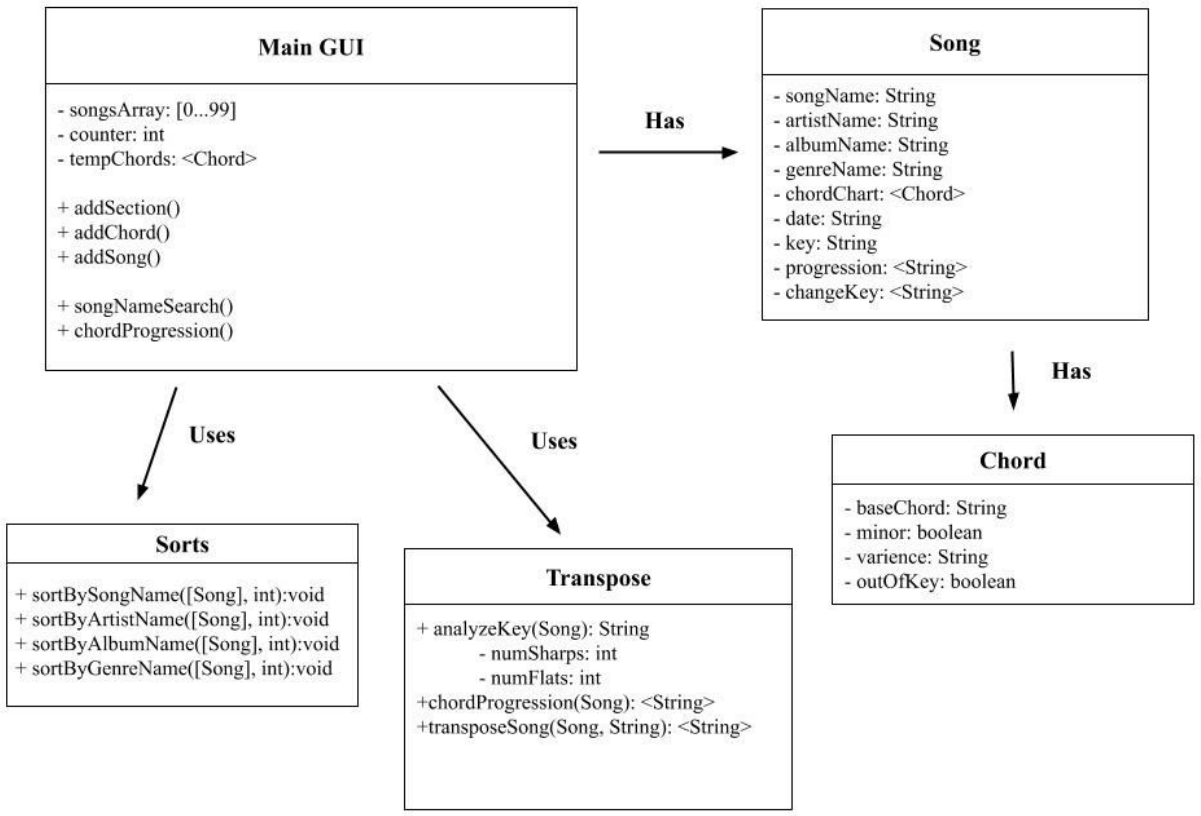
Summary List of All Techniques

- “For” loops
- ArrayList
- Arrays
- Methods (return and void)
- User defined objects made from an OOP “template” class
- Encapsulation
- Empty and overloaded constructors
- If/else statement
- Searching (Linear search)
- Sorting
- Error handling
- Java GUI techniques: TextArea.setText(), ComboBox.getSelectedItem(), etc.
- Get and set methods
- Use of logic gates
- Parameter passing
- Substring

Structure of the Program

<u>What:</u>
My program uses OOP to create, display, and transposes an array of songs. There are a total of five classes in the application: “Main.java”, “Sorts.java”, “Song.java”, “Chord.java”, and “Transpose.java”. The program is mainly based around the Song template class, where in some way each class will have interactions with it.

The Main class is a GUI class, allowing for it to interact with the user. It functions the program by adding instances of Songs into the database, displaying them in a display table, and allows users to Transpose songs into different keys. The Song class has an ArrayList of Chords as one of the private attributes, allowing for the Transpose class to transpose the Chords within the song. Lastly, the Sorts class helps sort Songs on the display table of the GUI.



Why:

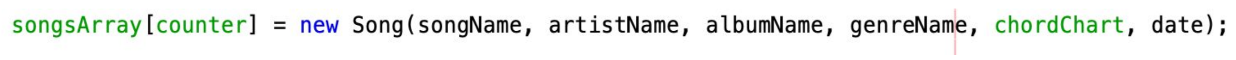
As the program uses the OOP (Object Oriented Programming) model, it has the characteristics of abstraction, polymorphism, and encapsulation making the program more effective and efficient. The use of multiple classes allow for abstraction, where each class would have their own function, and makes debugging and management of the program easier. The private attributes within the template classes can be returned or changed from public "get" and "set" methods, showing the use of encapsulation providing security when accessing or changing data. Furthermore, each constructor method uses polymorphism and allows for the use of overloaded constructors.

Data Structures Used

<u>What:</u>

- Array of objects

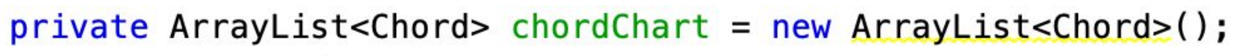
Sample:



```
songsArray[counter] = new Song(songName, artistName, albumName, genreName, chordChart, date);
```

- ArrayList of objects

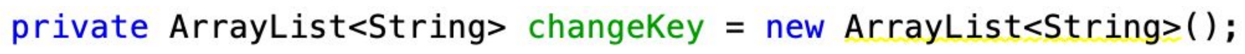
Sample:



```
private ArrayList<Chord> chordChart = new ArrayList<Chord>();
```

- ArrayList

Sample:



```
private ArrayList<String> changeKey = new ArrayList<String>();
```

<u>Why:</u>

Arrays of objects were used primarily due to it being directly accessible, allowing for the program to search and sort effectively. Therefore, it was used to store an Array of songs, which needed to be sorted in the display tab or searched in the transpose tab. ArrayLists of objects were used due to its dynamic nature, efficiently storing data. Thus, it was used to take in an ArrayList of chords which were used in the song, as each song differs in length. Likewise, an ArrayList of strings was used to take in the ArrayList of Chords and change into progressions or different keys.

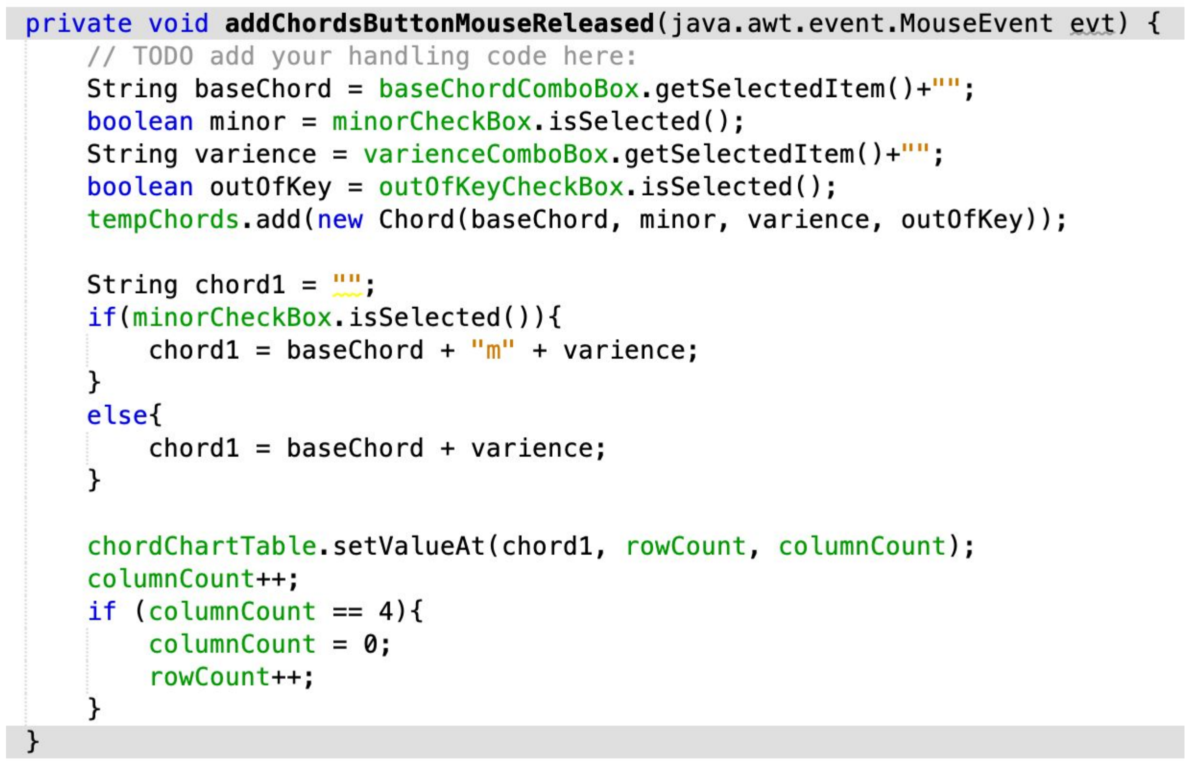
Main Unique Algorithms

A unique algorithm of this program is taking in the chord to form an ArrayList, which is then displayed to the user on the table of chords. The ArrayList is then stored as the private attribute "chordChart" of a song instance. The chords are taken based on the inputs from the GUI, where the attributes align with that of the chord class.

/* Four attributes are taken from the GUI to match the private attributes from the Chord class.

- The String Chord1 is created to be displayed in the inputted chord in the GUI table
- If the minor is selected in the check box:
    - The table displays the base chord with a small 'm' and variance.
- If the minor is not selected in the check box:
    - The table displays the base chord with the variance only.
- Last is the function to add Chord1 into the GUI table, setting it in the proper row and column.
- As chords are inputted, the column counter rises.
- If the column hits four, the column counter is reset to zero and the row counter increases.

*/



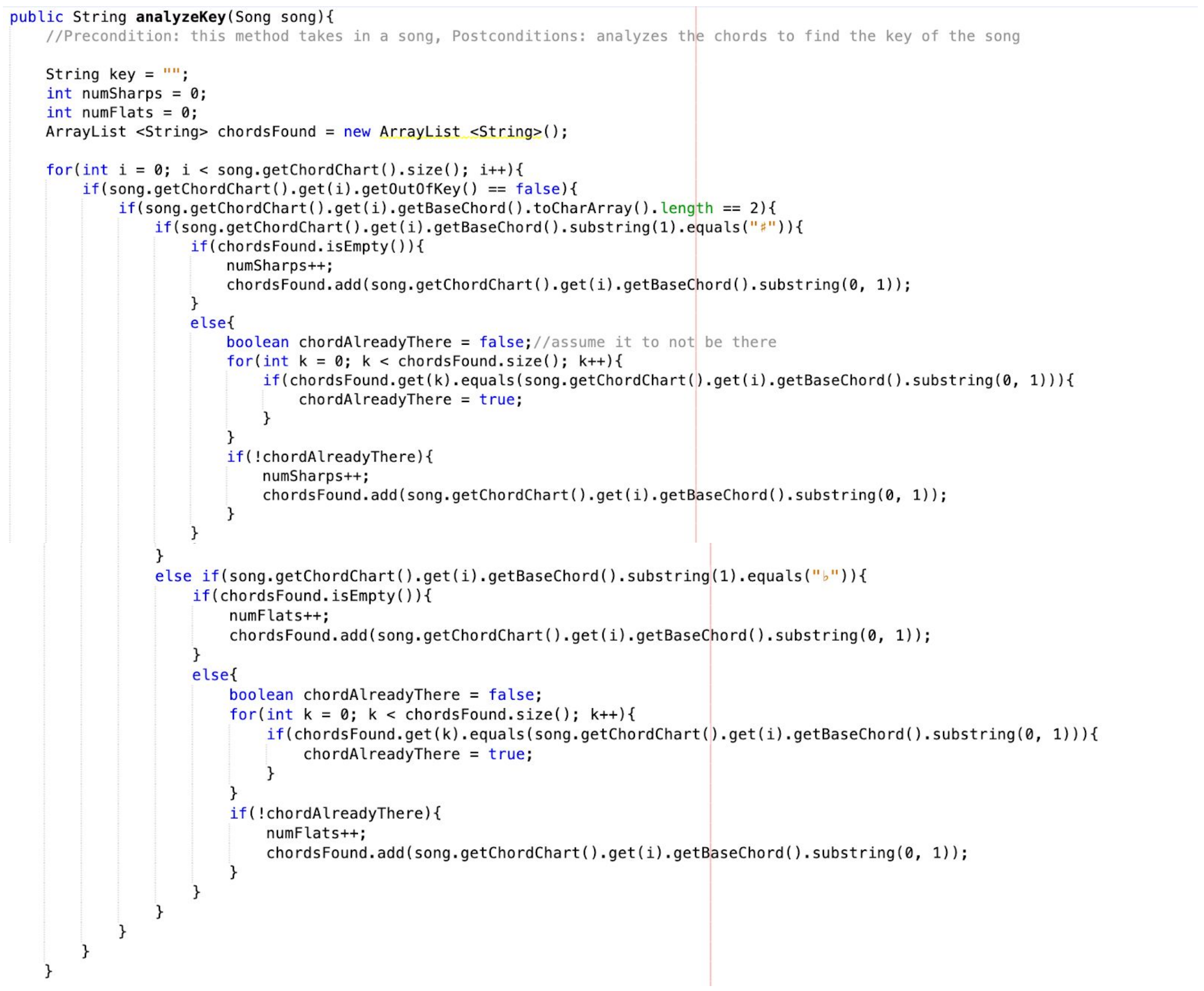
```
private void addChordsButtonMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    String baseChord = baseChordComboBox.getSelectedItem()+"";
    boolean minor = minorCheckBox.isSelected();
    String varience = varienceComboBox.getSelectedItem()+"";
    boolean outOfKey = outOfKeyCheckBox.isSelected();
    tempChords.add(new Chord(baseChord, minor, varience, outOfKey));

    String chord1 = "";
    if(minorCheckBox.isSelected()){
        chord1 = baseChord + "m" + varience;
    }
    else{
        chord1 = baseChord + varience;
    }

    chordChartTable.setValueAt(chord1, rowCount, columnCount);
    columnCount++;
    if (columnCount == 4){
        columnCount = 0;
        rowCount++;
    }
}
```

Another unique algorithm of this program is calculating the key of a song the user has inputted. It does this by counting the unique numbers of sharps or flats within the chords of a song, and then assigning it to the proper key. This key is then added to the instance of the song as it's private attribute "key", which is then used for sorting on the GUI's display tab and transposing on the GUI's transpose tab.

/* Precondition: this method takes in a song.
Postconditions: analyzes the chords to find the key of the song.

- The first for loop is needed to run through all the chords in the song.
- The first condition is to find whether or not the chord is out of key
    - The chords should be only analyzed if they are in the key.
- The second condition is whether or not there are 2 characters in the base chord
    - meaning that it has an accidental (# or ♭).
- The next condition separates the two, as a song cannot have both a sharp and a flat.
- The next condition evaluates whether or not the chord has already been found
    - uses a method of adding newly found chords to the ArrayList chordsFound.
- If the chords have not been found it is added to the chordsFound ArrayList
    - the numSharps or numFlats increase by one respectively.

*/

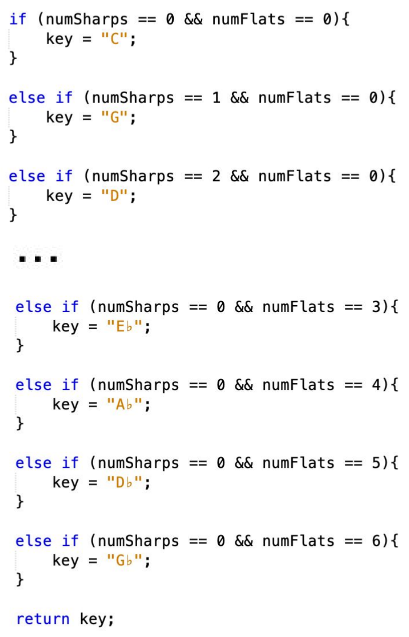


```java
public String analyzeKey(Song song){
    //Precondition: this method takes in a song, Postconditions: analyzes the chords to find the key of the song

    String key = "";
    int numSharps = 0;
    int numFlats = 0;
    ArrayList <String> chordsFound = new ArrayList <String>();

    for(int i = 0; i < song.getChordChart().size(); i++){
        if(song.getChordChart().get(i).getOutOfKey() == false){
            if(song.getChordChart().get(i).getBaseChord().toCharArray().length == 2){
                if(song.getChordChart().get(i).getBaseChord().substring(1).equals("♯")){
                    if(chordsFound.isEmpty()){
                        numSharps++;
                        chordsFound.add(song.getChordChart().get(i).getBaseChord().substring(0, 1));
                    }
                    else{
                        boolean chordAlreadyThere = false;//assume it to not be there
                        for(int k = 0; k < chordsFound.size(); k++){
                            if(chordsFound.get(k).equals(song.getChordChart().get(i).getBaseChord().substring(0, 1))){
                                chordAlreadyThere = true;
                            }
                        }
                        if(!chordAlreadyThere){
                            numSharps++;
                            chordsFound.add(song.getChordChart().get(i).getBaseChord().substring(0, 1));
                        }
                    }
                }
                else if(song.getChordChart().get(i).getBaseChord().substring(1).equals("♭")){
                    if(chordsFound.isEmpty()){
                        numFlats++;
                        chordsFound.add(song.getChordChart().get(i).getBaseChord().substring(0, 1));
                    }
                    else{
                        boolean chordAlreadyThere = false;
                        for(int k = 0; k < chordsFound.size(); k++){
                            if(chordsFound.get(k).equals(song.getChordChart().get(i).getBaseChord().substring(0, 1))){
                                chordAlreadyThere = true;
                            }
                        }
                        if(!chordAlreadyThere){
                            numFlats++;
                            chordsFound.add(song.getChordChart().get(i).getBaseChord().substring(0, 1));
                        }
                    }
                }
            }
        }
    }
```

/*

- Next the numSharps and numFlats are evaluated
- The proper key is then assigned according to it

*/



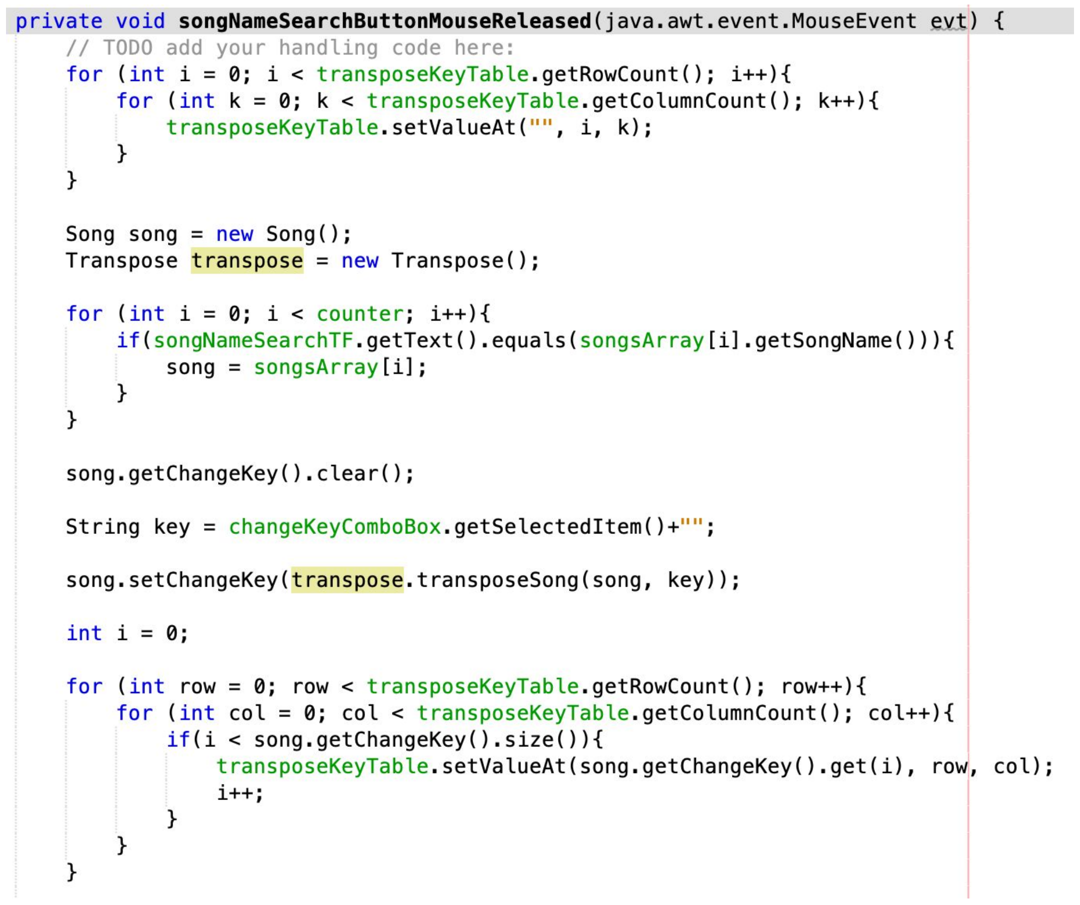
```java
if (numSharps == 0 && numFlats == 0){
    key = "C";
}

else if (numSharps == 1 && numFlats == 0){
    key = "G";
}

else if (numSharps == 2 && numFlats == 0){
    key = "D";
}

...

else if (numSharps == 0 && numFlats == 3){
    key = "E♭";
}

else if (numSharps == 0 && numFlats == 4){
    key = "A♭";
}

else if (numSharps == 0 && numFlats == 5){
    key = "D♭";
}

else if (numSharps == 0 && numFlats == 6){
    key = "G♭";
}

return key;
```

The last unique algorithm is transposing the chords of the original song into chord progressions or into another key which the user has selected. Evaluating the song's progression uses the songs key, which would equate a chord with a progression number. Likewise, when the song is transposed, the user selects a key and according to that key, the progression gets changed to match a chord.

/* Precondition: The song is searched for in the Transpose tab and the chord progressions are created when the song is added.
Postconditions: The GUI table shows either chord progressions or a transposed chord according to the key which the user choses.

- The user searches for the song via linear search.
- According to the key selected in the combo box, the song is transposed and displayed in the GUI table

*/



```java
private void songNameSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    for (int i = 0; i < transposeKeyTable.getRowCount(); i++){
        for (int k = 0; k < transposeKeyTable.getColumnCount(); k++){
            transposeKeyTable.setValueAt("", i, k);
        }
    }

    Song song = new Song();
    Transpose transpose = new Transpose();

    for (int i = 0; i < counter; i++){
        if(songNameSearchTF.getText().equals(songsArray[i].getSongName())){
            song = songsArray[i];
        }
    }

    song.getChangeKey().clear();

    String key = changeKeyComboBox.getSelectedItem()+"";

    song.setChangeKey(transpose.transposeSong(song, key));

    int i = 0;

    for (int row = 0; row < transposeKeyTable.getRowCount(); row++){
        for (int col = 0; col < transposeKeyTable.getColumnCount(); col++){
            if(i < song.getChangeKey().size()){
                transposeKeyTable.setValueAt(song.getChangeKey().get(i), row, col);
                i++;
            }
        }
    }
```

/*
Preconditions: The chordProgression method takes in a song
Postconditions: The progression gets returned, and added as a private song attribute

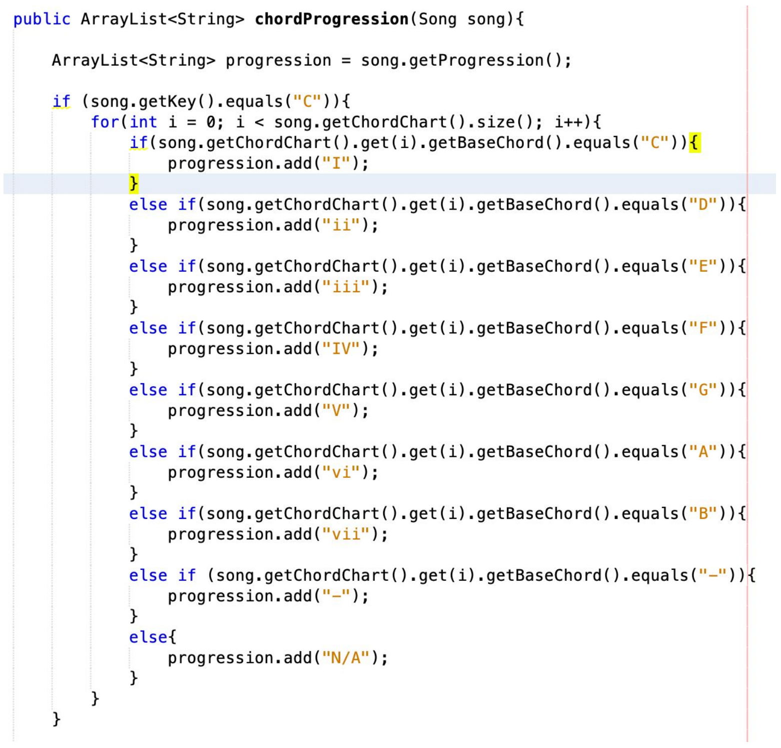
- Firstly ArrayList progression is declared
- The for loop runs through all the chords in the song which has been taken in
- According to the key of the song:
    - Each chord gets evaluated and the correct progressions are added to the ArrayList
- The ArrayList progression is then returned

The chord progression is then added to the song as a private attribute ready for transposing.

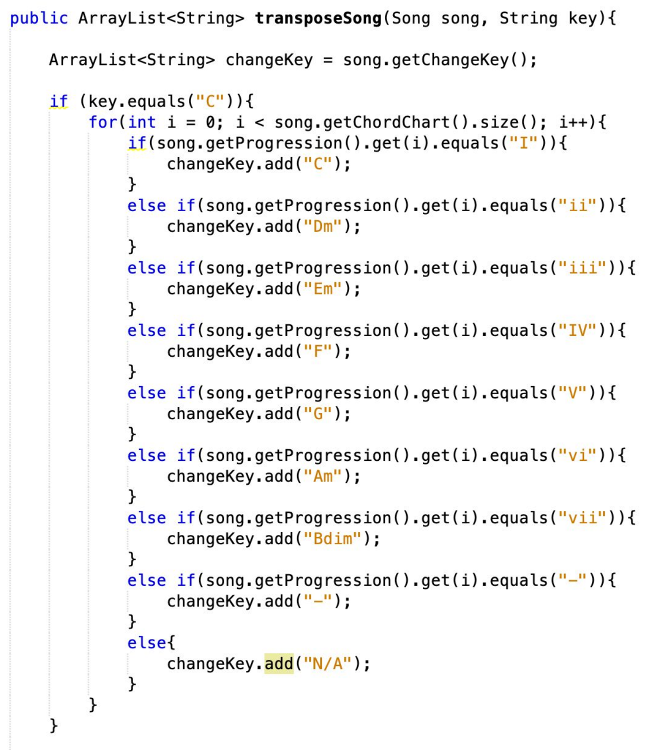
Preconditions: transposeSong method takes in a song and a key
Postconditions: The new chords of a different key are returned

- Firstly ArrayList changeKey is declared
- The first condition is to find out which key the user wants to transpose to
- The for loop runs through all the chords in the song which has been taken in
- According to the progression:
    - Each progression gets evaluated and the correct chord is added to the ArrayList
- The ArrayList changeKey is then returned

*/



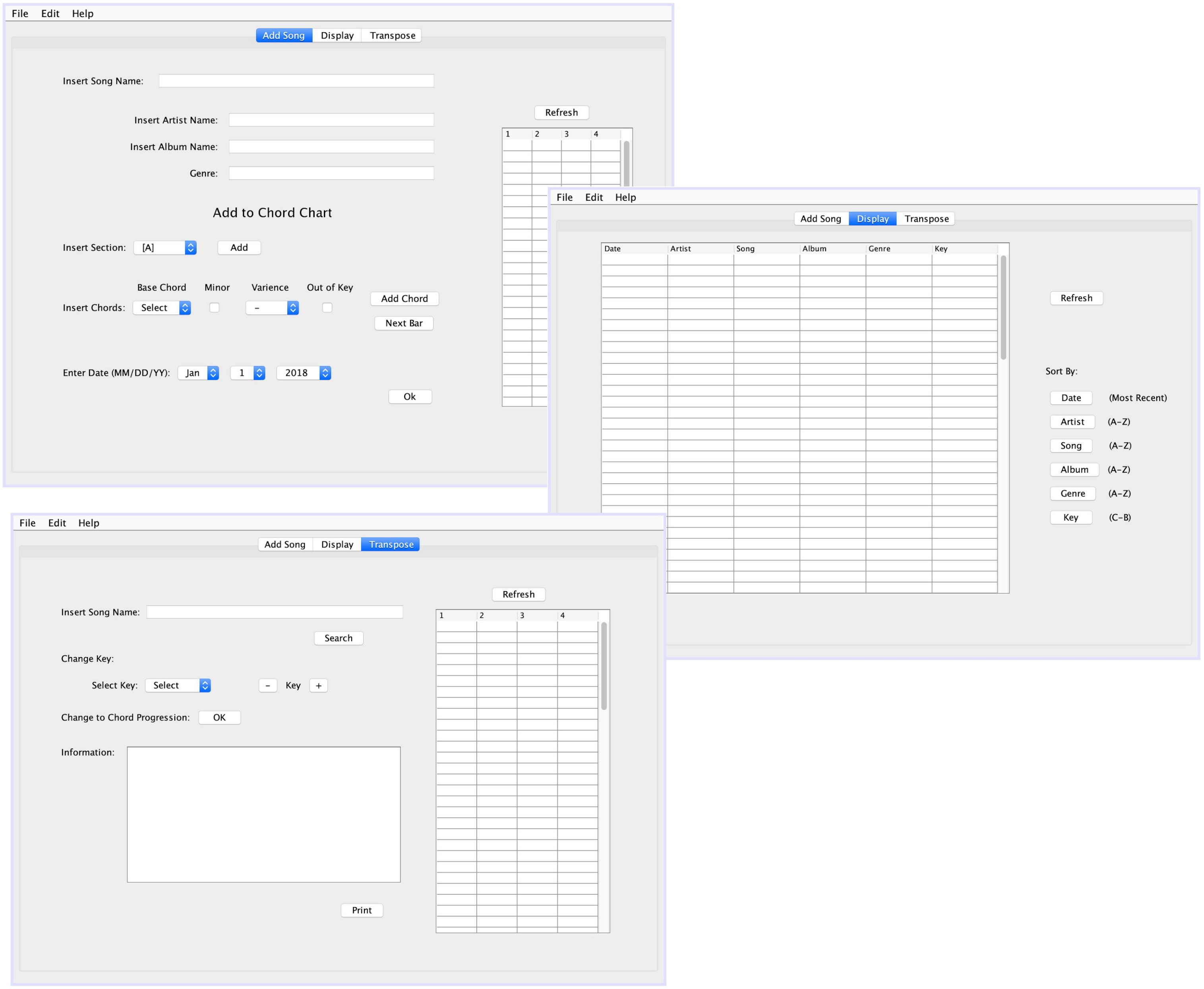
```java
public ArrayList<String> chordProgression(Song song){

    ArrayList<String> progression = song.getProgression();

    if (song.getKey().equals("C")){
        for(int i = 0; i < song.getChordChart().size(); i++){
            if(song.getChordChart().get(i).getBaseChord().equals("C")){
                progression.add("I");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("D")){
                progression.add("ii");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("E")){
                progression.add("iii");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("F")){
                progression.add("IV");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("G")){
                progression.add("V");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("A")){
                progression.add("vi");
            }
            else if(song.getChordChart().get(i).getBaseChord().equals("B")){
                progression.add("vii");
            }
            else if (song.getChordChart().get(i).getBaseChord().equals("-")){
                progression.add("-");
            }
            else{
                progression.add("N/A");
            }
        }
    }
```



```java
public ArrayList<String> transposeSong(Song song, String key){

    ArrayList<String> changeKey = song.getChangeKey();

    if (key.equals("C")){
        for(int i = 0; i < song.getChordChart().size(); i++){
            if(song.getProgression().get(i).equals("I")){
                changeKey.add("C");
            }
            else if(song.getProgression().get(i).equals("ii")){
                changeKey.add("Dm");
            }
            else if(song.getProgression().get(i).equals("iii")){
                changeKey.add("Em");
            }
            else if(song.getProgression().get(i).equals("IV")){
                changeKey.add("F");
            }
            else if(song.getProgression().get(i).equals("V")){
                changeKey.add("G");
            }
            else if(song.getProgression().get(i).equals("vi")){
                changeKey.add("Am");
            }
            else if(song.getProgression().get(i).equals("vii")){
                changeKey.add("Bdim");
            }
            else if(song.getProgression().get(i).equals("-")){
                changeKey.add("-");
            }
            else{
                changeKey.add("N/A");
            }
        }
    }
```

User Interface/GUI Work

My GUI allows users to store songs using different inputs (such as the chords used in the song), which can be searched for and transposed into chord progressions or different keys. My “Add Song” tab allows users to add a song, with inputs such as the song name, artist name, chords used in the song, etc. My “Display” tab allows the user to see the songs which have been added to the database, with the ability to sort by different attributes. The “Transpose” tab allows users to transpose a saved song into chord progressions or into a key of their selection.



Software Tools Used

The software used to create this program was the Netbeans IDE 8.2. The software allows for the use of an OOP model, creating a reliable and effective program. Additionally, This software has premade GUI components, allowing programmers to easily create an effective GUI creating an easy to use program for users.

Word Count: 817