School of Computing

Semester 2, 2023/24

Problem Solving Document

Data Structures and Algorithms

Lecturer: Emer Thornbury

Continuous Assessment 1

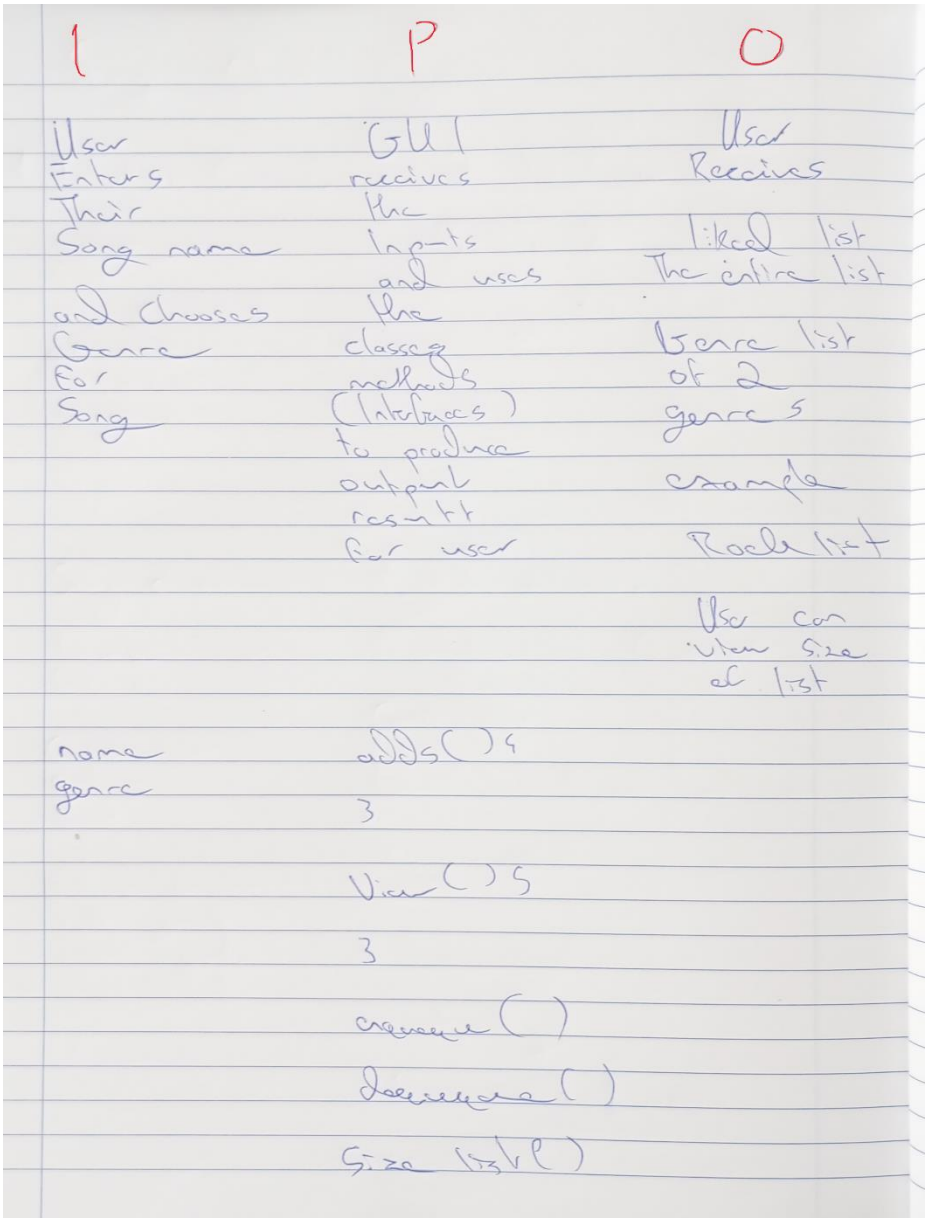
Created by: Alex Garbalyauskas: Student number: x22440482

Summary

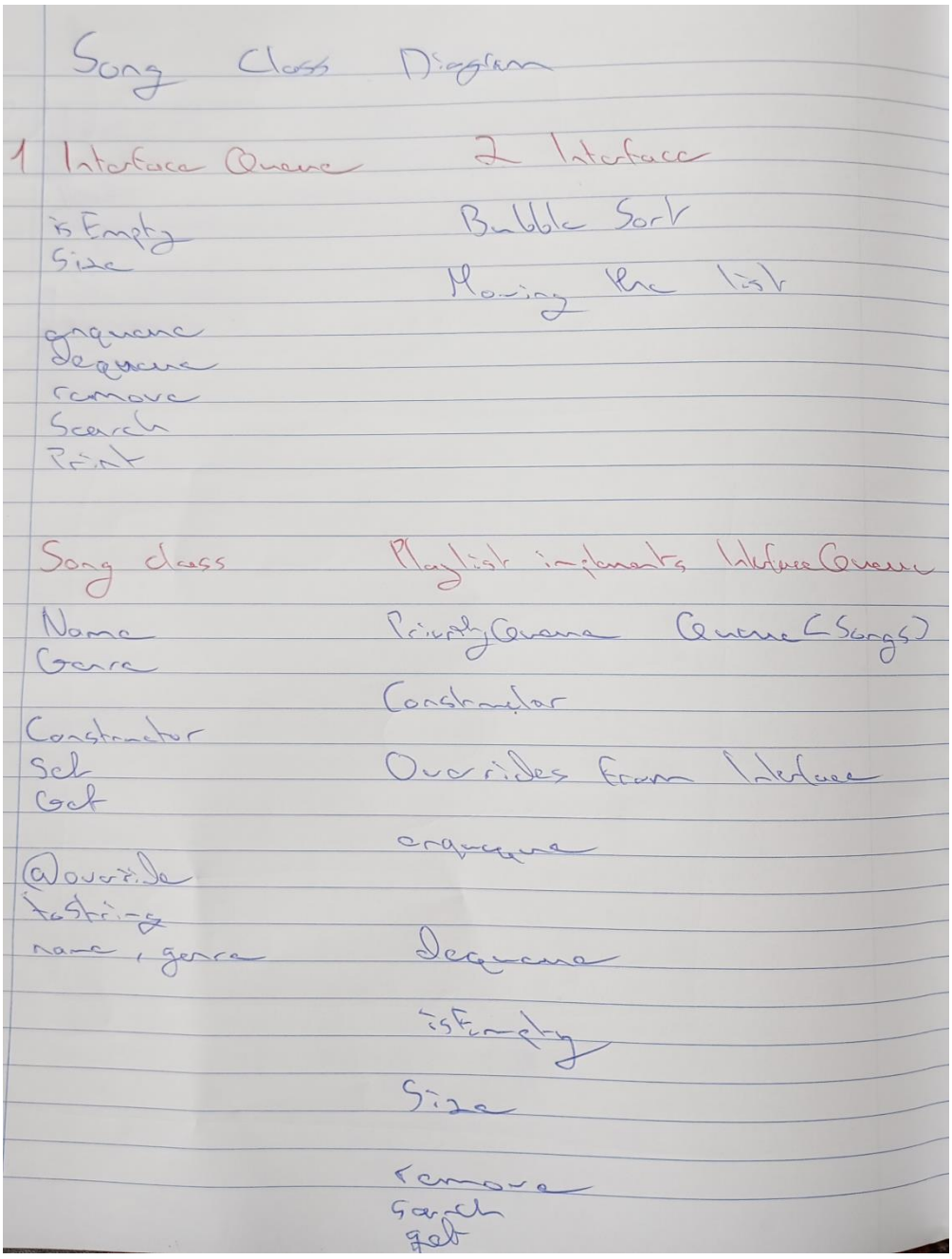
Documentation of the problem solving done for the continuous assessment, showcasing starting sketches, class diagram, and summary of the interfaces, classes, methods, GUI and app used.

Initial Sketch Diagrams

Input Process Output sketch outlining initial brainstorming ideas at approaching the assessment.

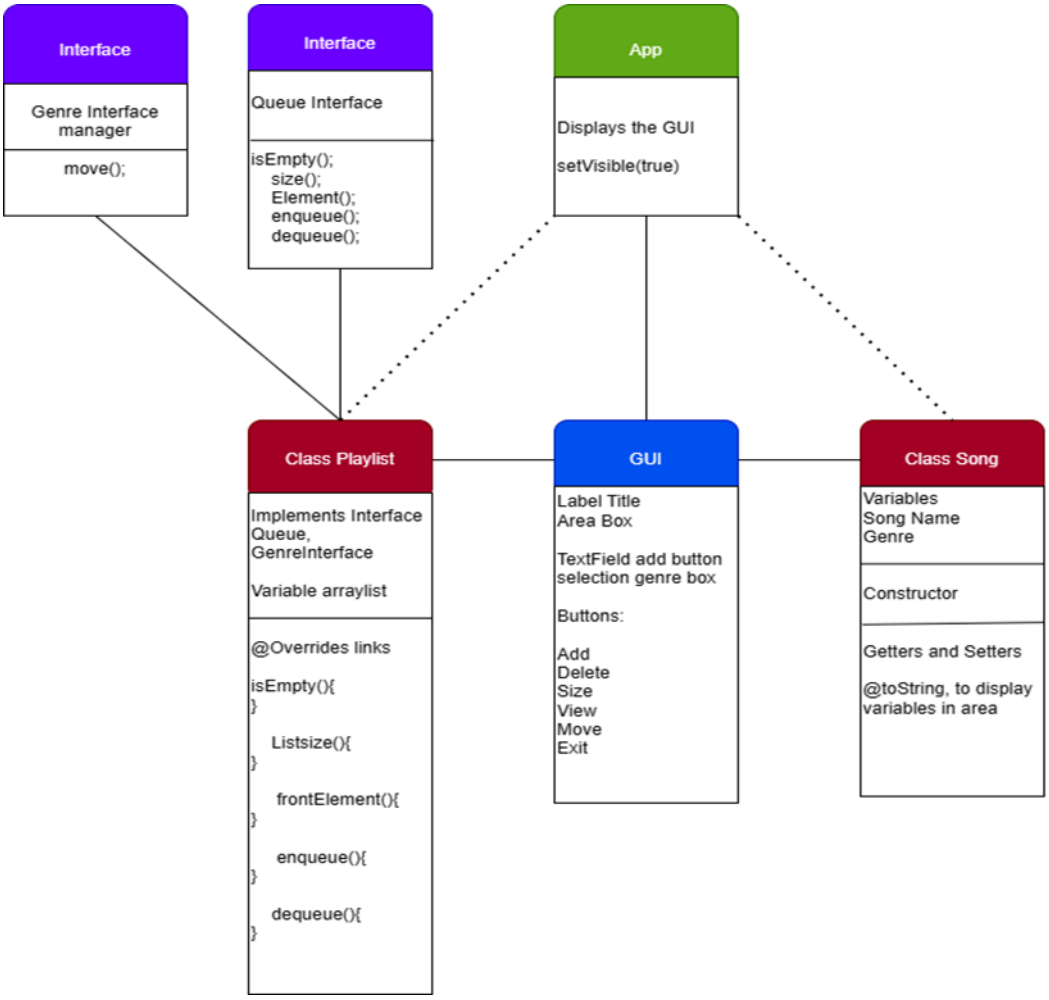


Simple Class sketch showcasing structure idea for the java application.



Class Diagram

Showcasing interfaces, classes, GUI, and app of application.



Summary Design

Abstract Data Types: Queue.

Interfaces:

Interface one is an interface that uses queue data managing methods for the playlist class that includes checking if the list is empty or not, the size of the list, adding the first element song to the list, adding song to the rear and front of the queue.

Interface two sets up move genre method, sorting the list.

Classes:

Song class initializes the song variables.

```
 *
 * @author Alex
 */
public class SongsClass {
    private String name, genre;
```

The variables allow user to input them within the graphic user interface. The class sets up the name and genre of the song sets them up, allows them to be retrieved and displayed using toString() method into the text area in GUI.

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getGenre() {
    return genre;
}

public void setGenre(String genre) {
    this.genre = genre;
}

 @Override
public String toString() {
    return "Name: " + name + ", Genre: " + genre;
}
```

PlaylistQueue class implements both of the interfaces within itself. It sets up multiple array lists for each song list viewing option, the licked list and two of the genre lists.

```
public class PlaylistQueue implements QueueInterface, GenreMoveInterface{
    private final ArrayList<SongsClass> theQueue;
    private final ArrayList<SongsClass> popList;
    private final ArrayList<SongsClass> rockList;
```

Since it implements the interfaces the managing methods have to be brought over using @Override. In this class the interface methods are coded in for functionality. isEmpty() method checks if the list is empty,

```
@Override
public boolean isEmpty(){
```

List size() returns the size of the licked list,

```
@Override
public int listsize(){
```

Front element() receives the first song at the start,

```
@Override
public Object frontElement(){
    if (!theQueue.isEmpty()){
        return theQueue.get(0);
    }else {
        return null;
    }
```

enqueue() adds a new song to the end of queue ,

```
@Override
public void enqueue(Object element){
    theQueue.add((SongsClass)element);
```

dequeue() removes and returns the song at the front,

```
@Override
public Object dequeue(){

    if (theQueue.size() > 0 ) {
        return theQueue.remove(0);
    }else{
        return null;
    }
```

move() method moves the selected genre to its own list visible inside the text area,

```
// Moves the songs from the liked list to the genre list
@Override
public void moveSong(String genre){
    //moves either to pop or rock list
    ArrayList<SongsClass> genreList = (genre.equals("Pop")) ?
    //goes through queue and moves song to right genre
    while (!theQueue.isEmpty()) {
        SongsClass song = (SongsClass) theQueue.remove(0);
        genreList.add(song);
    }
    //builds string text to display within the text area
    StringBuilder sb = new StringBuilder();

    sb.append(genre).append(" List\n");
    for (SongsClass song : genreList) {
        sb.append("Name:" + song.getName()).append("\n");
        sb.append("Genre:" + song.getGenre()).append("\n");
    }
```

Not linked to the interfaces the display method helps displays the text of the inputted songs into the text area.

```
public String display() {
    StringBuilder sb = new StringBuilder();
    sb.append("Liked List\n");
    for (SongsClass song : theQueue){
        sb.append("Name:" + song.getName()).append("\n");
        sb.append("Genre:" + song.getGenre()).append("\n");
    }
```

Search method.

```
//a search method , looks for the given name, returns the 1st
public String search(String songName){
    for (SongsClass song : theQueue){
        //checks if the name of the current song is equal to
        if (song.getName().equalsIgnoreCase(songName)){
            return song.toString();
        }
```

Has current song method makes sure the same song name not added.

```
//true false checker, checks if the same name was added alreac
public boolean hasSong(String songName){
    for (SongsClass song : theQueue){
        if (song.getName().equalsIgnoreCase(songName)){
            return true;
```

App:

Initializes the graphic user interface to be viewed by the user.

```
public class SongPlaylistApp {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        PlaylistGUI mygui = new PlaylistGUI();
        mygui.setVisible(true);
    }
```

GUI:

Graphic User Interface Playlist links the playlist class by creating an instance variable.

```
public class PlaylistGUI extends javax.swing.JFrame {
    private PlaylistQueue playlist;
```

the playlist class helps manage the GUI button functions such as adding, viewing, searching, delete, size, and displaying the outputs inside the text area or as a JOption pop up message.

Add button gets inputs and tells user its added, no empty input or same names.

```
    //Action listerners for the buttons
    private void addBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        //gets the textfield and selection box
        String songName = addTF.getText();
        String genre = genreBox.getSelectedItem().toString();

        //checks if empty
        if(songName.isEmpty()){
            JOptionPane.showMessageDialog(null,"Please enter a song name.");
            //if song is the same this appears
        } else if (playlist.hasSong(songName)) {
            JOptionPane.showMessageDialog(null, "That song already exists in the playlist enter another song");
        } else {
            SongsClass song = new SongsClass(songName, genre);
            //adds song
            playlist.enqueue(song);
            JOptionPane.showMessageDialog(null, "Song added successfully.");
        }
```

View button.

```
    private void viewBtnActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        //Clears the playlist content
        area.setText("");
        //then displays by using playlist class method display
        area.setText(playlist.display());
```

Exit button.

```
private void exitBtnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    //systems exists while saying goodbye
    JOptionPane.showMessageDialog(null, "Good Bye");
    System.exit(0);
```

Search button.

```
  private void searchBtnActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:
      String songName = JOptionPane.showInputDialog(null, "Enter the song name to search:");

      //Check if the search is empty
      if (songName != null && !songName.isEmpty()) {

          //Performs the search linked method from playlist class
          String searchResult = playlist.search(songName);
```

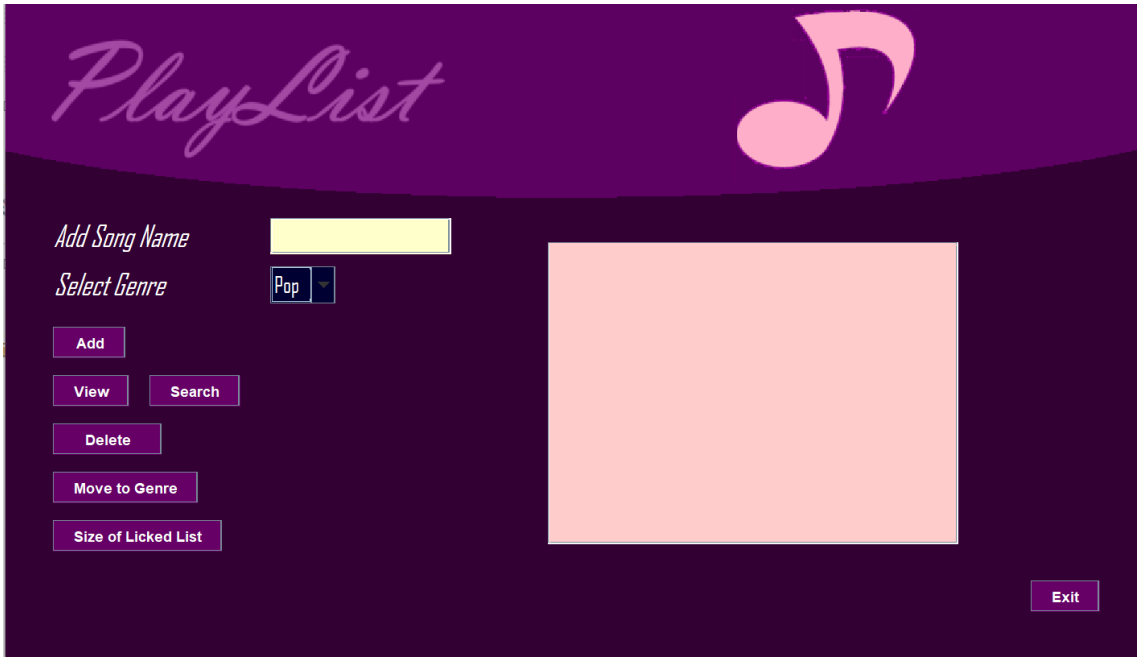
Delete button.

```
private void deleteBtnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    //if empty then enter name
    if (!playlist.isEmpty()) {
        JOptionPane.showInputDialog(null, "Enter name of Song");
        //removes from list using dequeue
        SongsClass deleteSong = (SongsClass) playlist.dequeue();
        JOptionPane.showMessageDialog(null, "Removed: " + deleteSong.toString());
    }else  {
        JOptionPane.showMessageDialog(null, "No songs in the queue.");
    }
```

Size display button.

```
  private void sizeBtnActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:

      //gives you size of main list
      int size = playlist.listsize();
      JOptionPane.showMessageDialog(this, "Number of songs in the liked list: " + size);
```

The GUI is designed appropriately to make it user friendly by using styling options of background change and icon images.



Github.com link

https://github.com/NCIAlexGar/Data-Structures-Algorithms-CA2/tree/main.