Middlesex University

CST2550 Coursework 2

M00655420

Abstract

Work

I have been asked to create “Karaoke” application. Idea is that the user would be able to search songs in the library which is already in the player, play songs, add them to playlist in order or delete them from it. Each song has video which can be seen in the player. Application needs to open up and work on our CST2550 module environment which is Linux, Fedora.

Results

The “Karaoke” player that I have created can accomplish every function that specifications of the work had asked. User can play or pause songs, search and add them to the playlist. User can delete it from the playlist as well and see video of the song in the player. I did stick to my plan and did everything before the deadline. GUI is looking pretty simple but does it’s work and is very easy to use and understand for a new user.

Conclusion

I have started this coursework way before deadline and spent most of the time learning about “javafx” and about its event handlers, how they are working. Doing work step by step by provided detailed description helped me not to miss anything on the specifications and helped me to stick to the structure of the work. Testing whole application after each function is added to the code was helpful as well as it did not let me leave any mistakes behind and let me keep moving forward knowing no errors will appear in the future because of the previous functions. The last thing I did was Graphic User Interface. As I mentioned before I spent quite a lot of time learning it because it was kind of new thing for me. Of course, because it had to execute previously implemented functions it made even harder but once you start understanding how layouts and event handlers are working, it is achievable.

Introduction

This whole report will be about my second coursework of module CST2550, how I approached it, what I used to complete it, what issues I faced, deeper insights of the work than user will see.

In the following sections of this report I will talk about design, testing, conclusion, will show appendices, references that I have used. All of this includes analysis of time complexity of solution, wireframe diagrams, testing approaches and results, limitations of my approach and critical reflection of my work by me, what I would have done differently, summary of the work and more.

Design

Pseudo codes –

Add function

```
add.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event){
                if(!(mlist.contains(getres.get(0))))
                mlist.add(getres.get(0));
        }
});
```

Search function

```
search.setOnKeyPressed(new EventHandler<KeyEvent>(){
        @Override
        public void handle(KeyEvent ke){
                if (ke.getCode().equals(KeyCode.ENTER)){
                        try{
                                if(!(getres.isEmpty())) getres.remove(0);
                                String arr[] = read(search.getText());
                                String[] ts = arr[2].split("");
                                getres.add(new song(arr[0],arr[1],ts[0]+":"+ts[1]+ts[2],arr[3]));
                                search.clear();
                        }
                        catch(Exception e){}
                }
        }
});
```

Delete function

```
del.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event){
                song s = mview.getSelectionModel().getSelectedItem();
                mlist.remove(mlist.indexOf(mview.getSelectionModel().getSelectedItem()));
        }
});
```

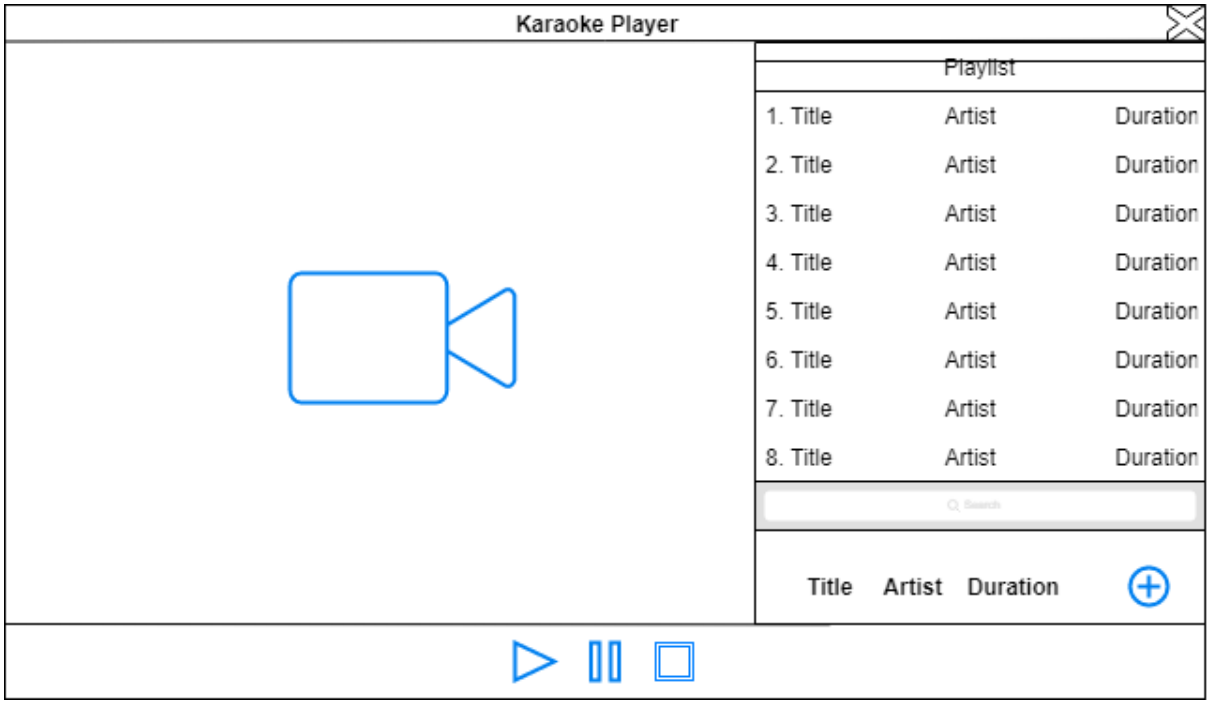
Analysis of time complexity of solution – all of those functions are O(1). The total time is found by adding the times for all statements. You can see an explanation below:

```
                                                        Cost    Times
add.setOnAction(new eventHandler<ActionEvent>() {       c1      1
        public void handle(ActionEvent event){
                if(!(mlist.contains(getres.get(0))))    c2      1
                mlist.add(getres.get(0));               c3      from 0 to 1
        }
});
```

```
T(n) = c1 + c2 + c3
each statement is constant and the total time is also constant: O(1)

The same is with the other functions (delete, search) because they do not have any complex statements,
for loops or nested loops.
```

Wireframe diagrams – Where is the biggest square in the player it is the place where video will be played. On the right there is a playlist and below it, user can search for a song by entering its title and pressing “Enter”. If there is a song in the library by the title user entered, then it appears and you can add it to playlist by pressing “+” button. In the middle there a play, pause and stop buttons.



Testing

Because of the previous experience I know that tests are very helpful and is useful to do them. Therefore, I did few of them to make sure everything is working and that program would be more like professional and ready to use publicly. All of the testing methods I used and how I approached them are explained below in a table.

Testing method	Explanation of approach	Result
Unit Testing	I used "Unit Testing" for separate units in my code, for e.g. after creating each function I tried it if it is working. Those functions include: Add song's to library, search library by title, add song to playlist, play next song, view playlist, delete song from playlist	These tests gave me the answers to be sure that those functions are working separate and I can enable them later for required functionality with GUI.
Integration Testing	Integration testing was used when a first version of player was created using "javafx", so all of those functions like add, delete and etc. were checked if they are working together.	Using this test method showed me that some of the functions were not working when they got connected to GUI and that let me to fix them.
User Testing BETA	When a simple version of GUI was created, I let my friend to try to use my Karaoke application. I told her that its only a beta version of it.	I got a feedback from the user what extra's should be added to application to make it more professional, what to change in UI to make it easier and more understandable to use.
Automated Testing	I used automated testing in functions where input is required and you expecting specific output. So, I used it on Search function of the application.	In search field user enters song title and if there is a song in the library with the exact same title then as an output the song title, artist and duration appear in the list below.
System Testing	I used more than one approach to system testing. One of them was to verify thorough testing of every input to check for desired outputs. Another one was to run application on another machine.	First step of the testing went well. Application testing in another machine was a bit complicated. I tried to run this program on Windows 10 instead of Linux. I got some errors and what was working on Linux was not working on windows 10, but what was working on windows 10 was working on Linux. Unfortunately, I could not solve this problem and left it as it is.
Acceptance Testing	Last but not least was acceptance testing where user (not developer) was asked to try out the application and hear the feedback from his perspective.	User was satisfied with the product but said it would be easier if playlist would not be in a separate window.

Conclusion

Summary of work done

I created a "Karaoke" singing application. Application has a big library of different songs, so user can search for a song by entering song title in the search field and add the song to his own playlist without downloading it. In the same easy way user can delete song from playlist just by choosing the song and pressing "delete" button. In the player itself you can see a video with lyrics of the song.

Player has a nice and simple intuitive user interface which executes functionality without any problems. It is very easy to use and understand.

Limitations of my approach and critical reflection of this work

In talk of functionality side, I think I have done everything correctly. Application has no problems, every function does it's work and implementation meets specifications. I could have spent more time on GUI by making it more attractive or adding playlist in the same window as a player. But that do not affect application efficiency so I am pretty happy with my work.

How I would approach a similar project differently in the future

By not making the same mistakes as I did in previous coursework, I am pretty happy how I approached this project. But still there are some improvements I could do. For example, during this project I started very early, did some work and then left it for 3 or 4 weeks. That mistake cost me a bit of time just because when I got back to it, I had to remind myself what I was doing, how to do it and things like that.

References

None

Appendices

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.stage.Screen;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.paint.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import java.io.*;
import javafx.beans.binding.Bindings;
import javafx.beans.property.DoubleProperty;
import javafx.geometry.*;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.beans.value.*;
import javafx.event.*;
import javafx.scene.input.*;
import java.util.*;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.control.cell.PropertyValueFactory;
public class dummy extends Application{
```

```
// main class
	List<song> slist;
	String[] read(String title) throws Exception{
		File file = new File("sample-song.txt");
		BufferedReader br = new BufferedReader(new FileReader(file));
		String st;
		String arr[];
		// reading file
		while((st = br.readLine()) != null){
			arr = st.split("    ");
			if(arr[0].equalsIgnoreCase(title.trim())){
				return(arr);
			}
		}
		return null;
	}

	public static class song{
	  // Properties for a track, Url is for location
		private final SimpleStringProperty title;
		private final SimpleStringProperty artist;
		private final SimpleStringProperty duration;
		private final SimpleStringProperty url;

	  // Constructor of the class which sets the values passed when the class call is made
		private song(String titlename, String artistname, String len, String loc) {
			this.title = new SimpleStringProperty(titlename);
			this.artist = new SimpleStringProperty(artistname);
			this.duration = new SimpleStringProperty(len);
			this.url = new SimpleStringProperty(loc);
		}

	  // methods to get values or set the values of the current object
		public String getTitle() {
			return title.get();
		}
		public void setTitle(String titlename) {
			title.set(titlename);
		}

		public String getArtist() {
			return artist.get();
		}
		public void setArtist(String artistname) {
			artist.set(artistname);
		}

		public String getDuration() {
```

```
                return duration.get();
            }
            public void setDuration(String len) {
                duration.set(len);
            }

            public String getURL() {
                return url.get();
            }
            public void setURL(String loc) {
                url.set(loc);
            }

        }

        final ObservableList<song> mlist = FXCollections.observableArrayList(new song("Gorgeous","Property Prophets","2:62","test.mp4"));

        void playlistwindow(Stage stage){
          // search box
            TextField search = new TextField("search song");
            search.setPromptText("Search Songs");

            // Observable list for the search result
            final ObservableList<song> getres = FXCollections.observableArrayList();
            final TableView<song> result = new TableView<>();

            // Setting style of the columns and table.
            result.setEditable(false);
            result.setPrefHeight(100);

            TableColumn titlecol = new TableColumn("Title");
            titlecol.setPrefWidth(400);
            titlecol.setResizable(false);
            titlecol.setCellValueFactory(new PropertyValueFactory<song,String>("title"));

            TableColumn artistcol = new TableColumn("Artist");
            artistcol.setPrefWidth(300);
            artistcol.setResizable(false);
            artistcol.setCellValueFactory(new PropertyValueFactory<song,String>("artist"));

            TableColumn durationcol = new TableColumn("Duration");
            durationcol.setPrefWidth(100);
            durationcol.setResizable(false);
            durationcol.setCellValueFactory(new PropertyValueFactory<song,String>("duration"));

            result.setItems(getres);
```

```
result.getColumns().addAll(titlecol,artistcol,durationcol);

/* Function triggers when enter is pressed in search box. If track exists it returns 4 strings back and then its added to the Observable list. If nothing is found, null is returned. It also clears old results */
search.setOnKeyPressed(new EventHandler<KeyEvent>(){
	@Override
	public void handle(KeyEvent ke){
		if (ke.getCode().equals(KeyCode.ENTER)){
			try{
				if(!(getres.isEmpty())) getres.remove(0);
				String arr[] = read(search.getText());
				String[] ts = arr[2].split("");
				getres.add(new song(arr[0],arr[1],ts[0]+":"+ts[1]+ts[2],arr[3]));
				search.clear();
			}
			catch(Exception e){
				System.out.println("Something went wrong in search function");}
		}
	}
});


Button add = new Button("add");

// table for actual playlist and its properties like the search one
final TableView<song> mview = new TableView<>();

mview.setEditable(false);
mview.setPrefHeight(500);

TableColumn mtitlecol = new TableColumn("Title");
mtitlecol.setPrefWidth(400);
mtitlecol.setResizable(false);
mtitlecol.setCellValueFactory(new PropertyValueFactory<song,String>("title"));

TableColumn martistcol = new TableColumn("Artist");
martistcol.setPrefWidth(300);
martistcol.setResizable(false);
martistcol.setCellValueFactory(new PropertyValueFactory<song,String>("artist"));

TableColumn mdurationcol = new TableColumn("Duration");
mdurationcol.setPrefWidth(100);
mdurationcol.setResizable(false);
mdurationcol.setCellValueFactory(new PropertyValueFactory<song,String>("duration"));
```

```
		mview.setItems(mlist);
		mview.getColumns().addAll(mtitlecol,martistcol,mdurationcol);

		// When add button is clicked it add the result from the search result to the playlist
table.
		add.setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event){
				if(!(mlist.contains(getres.get(0))))
				mlist.add(getres.get(0));
			}
		});

		Button del = new Button("delete");

		// Delete button, when its pressed and song is selected in playlist it will be deleted
from it
		del.setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event){
				song s = mview.getSelectionModel().getSelectedItem();

	mlist.remove(mlist.indexOf(mview.getSelectionModel().getSelectedItem()));
			}
		});


		Button done = new Button("done");

		// a layout for grid pane
		GridPane gp = new GridPane();
		gp.addRow(0,search);
		gp.addRow(1,result);
		gp.addRow(2,add);
		gp.addRow(3,mview);
		gp.addRow(4,del);
		gp.addRow(5,done);

		gp.setVgap(10);
		gp.setAlignment(Pos.CENTER);
		VBox vb = new VBox();
		vb.getChildren().addAll(gp);
		Scene scene2 = new Scene(vb, 1000, 800);
		Stage playlist = new Stage();
		playlist.setTitle("Playlist");
		playlist.setScene(scene2);
		playlist.setResizable(false);
```

```
                playlist.setX(stage.getX() + 200);
                playlist.setY(stage.getY() + 50);
                playlist.show();

                // When you press "done" it gets mlist(ObservableArrayList of playlist and converts it to a list
                done.setOnAction(new EventHandler<ActionEvent>() {
                        @Override
                        public void handle(ActionEvent event){
                                slist = mlist;
                                itr = slist.iterator();
                                playlist.close();
                        }
                });

        }

   final MediaView mv = new MediaView(); // its global because in start function bindings use mv
        Button nf = new Button("Next");

        Button pp = new Button("Pause");
   Iterator<song> itr = mlist.iterator(); // iterator initially listening to mlist

    void playsong(Stage stage){ // This stage is passed as args because we need the original stage to set the window title everytime the song changes

        // its init with mlist(observablearraylist) but after choosing the playlist its assigned to slist.
                song s = itr.next();
                stage.setTitle(s.getTitle() + " - " + s.getArtist()); // setting title
                File f = new File(s.getURL()); // getting file location from struct getURL method
                Media media = new Media(f.toURI().toString());
      MediaPlayer player = new MediaPlayer(media);
      mv.setMediaPlayer(player); // mv is globally initialised
      player.play(); // plays the track
        // when track ends and the list has next value then it calls the same playsong function with the same stage.
      player.setOnEndOfMedia(new Runnable() {
        @Override
        public void run() {
                                player.stop();
          if(itr.hasNext()) {
            playsong(stage);
                          }
          return;
        }
      });

        // Simple check when pressed play or pause.
```

```
                pp.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event){
          if(player.getStatus().equals(player.getStatus().PLAYING)){
                                player.pause();
                                pp.setText("Play");
                            }
                            else{
                                player.play();
                                pp.setText("Pause");
                            }
        }
      });

                // "next" track button. it stops the player, cheks the lists, if it has next value it calls itself again
                nf.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event){
                            player.stop();
                             if(itr.hasNext()) {
            playsong(stage);
          }
                        }
      });

   }
          @Override
          // main stage
          public void start(Stage stage) throws Exception{
                Button lst = new Button("List");

                playsong(stage);
                // below is just simple properties and panes
                HBox hb = new HBox(10);
                hb.setPadding(new Insets(10));
                hb.getChildren().addAll(pp,nf,lst);
                hb.setBackground(new Background(new BackgroundFill(Color.WHITE,CornerRadii.EMPTY, Insets.EMPTY)));

                BorderPane bp = new BorderPane(mv,null,null,hb,null);
                hb.setAlignment(Pos.CENTER);
                bp.setAlignment(mv,Pos.CENTER);
                bp.setBackground(new Background(new BackgroundFill(Color.BLACK,CornerRadii.EMPTY, Insets.EMPTY)));

                Scene scene = new Scene(bp);
                stage.setMaximized(true);
```

```
                final DoubleProperty width = mv.fitWidthProperty();
                final DoubleProperty height = mv.fitHeightProperty();
                width.bind(Bindings.selectDouble(mv.sceneProperty(), "width"));
                height.bind(Bindings.selectDouble(mv.sceneProperty(), "height").subtract(100));
                // Action event which triggers the playlist window
                lst.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event){
                                        playlistwindow(stage);
                                }
        });
        stage.setScene(scene);
        stage.setMinHeight(480);
        stage.setMinWidth(720);
                stage.show();
            }

            public static void main(String args[]){
                launch(args);
            }
}
```