**PROJECT REPORT**

**INDEX**

1. Problem Statement
2. Software Used
3. Functions / Modules
4. List of Errors Encountered while coding the project
5. Key or challenging logic in the project
6. Prerequisites
7. Project Code
8. Elaborate Sample Input and Output Screenshots

**PROBLEM STATEMENT**

To make a basic music player using the knowledge of various Data structures learnt as part of course CSE2003

Data structures used include a doubly linked list, stacks and queues.

File handling to read and write songs has also been used.

**SOFTWARE REQUIREMENT**

CODE BLOCKS 13.12

or

JAVA COMPILER

**FUNCTIONS**

1. tofile() –

Function to work on playslist.txt.

2. add\_node() –

Function that adds songs to the list using a linked list.

3. add\_node\_file() –

Function that adds songs to the playlist to linked list from the data passed in addplaylist() function.

4. delete\_file() –

Function to delete song from text file playlist.txt.

5. del\_node() –

Function that deletes the last song from the input linked list.

6. printlist() –

Function that displays the input songs of the playlist.

7. count\_nodes() –

Function that tracks the number of inputs in the linked list.

8. del\_pos() –

Function that deletes songs from the linked list using the position of that song.

9. search1() –

Function that takes song input and linearly searches through the linked list and finds matching case.

10. push() –

Function that pushes the last played track of play() function into a stack to store and create a recently played list.

11. display() -

Function to display the stack generated in push() function.

12. play() -

Function to search input song and show if it can be played. It then passes the song to push() function to be added to recently played list.

13. recent() -

Function that calls display() function.

14. topelement() –

Function that displays the last played song.

15. addplaylist() –

Function that opens text file playlist.txt and passes data to add\_node\_file() function.

16. del\_search() –

Function to search input song and delete it from the list.

17. deletemenu() –

Function to invoke del\_search() or del\_pos() functions depending on user input.

18. main() –

Function that invokes all other functions of the project based on user defined input.

**LIST OF ERRORS**

**KEY OR CHALLENGING LOGIC**

The major part of the challenge included making a doubly linked list that would accept string data and store it efficiently. Involving the usage of file handling was another challenge. To retrieve songs from a pre-made list and add it to the linked list. It also required us to be able to write the new input songs to the file. All changes in the program required to be reflected on the text file.

**PREQUISITES**

To use the File handling part of the program a text file of name

“playlist.txt” with certain data elements is required.

Sample data elements-

Animals

Spaceman

Apollo

Care

Fragile

Melody

**PROJECT CODE**

import java.io.\*;

import java.util.\*;

class Node {

String song;

Node next;

Node prev;

public Node(String song) {

this.song = song;

this.next = null;

this.prev = null;

}

}

public class PlaylistManager {

private Node top;

private Node start;

public PlaylistManager() {

top = null;

start = null;

}

private void writeToFile(String song) {

try (FileWriter fw = new FileWriter("playlist.txt", true)) {

fw.write(song + "\n");

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

}

}

private void deleteFromFile(String song) {

File inputFile = new File("playlist.txt");

File tempFile = new File("temp.txt");

boolean songDeleted = false;

try (BufferedReader reader = new BufferedReader(new FileReader(inputFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile))) {

String line;

while ((line = reader.readLine()) != null) {

if (!line.trim().equals(song)) {

writer.write(line + "\n");

} else {

songDeleted = true;

}

}

} catch (IOException e) {

System.out.println("Error processing file: " + e.getMessage());

}

if (songDeleted) {

if (inputFile.delete() && tempFile.renameTo(inputFile)) {

System.out.println("Song deleted.");

} else {

System.out.println("Error updating file.");

}

} else {

System.out.println("Song not found in the playlist.");

}

}

public void addSong(String songName) {

Node newNode = new Node(songName);

if (start == null) {

start = newNode;

} else {

Node temp = start;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

newNode.prev = temp;

}

writeToFile(songName);

}

public void deleteSong(String songName) {

if (start == null) {

System.out.println("Playlist is empty.");

return;

}

Node temp = start;

while (temp != null) {

if (temp.song.equals(songName)) {

if (temp.prev != null) {

temp.prev.next = temp.next;

} else {

start = temp.next;

}

if (temp.next != null) {

temp.next.prev = temp.prev;

}

deleteFromFile(songName);

System.out.println("Song deleted: " + songName);

return;

}

temp = temp.next;

}

System.out.println("Song not found.");

}

public void displayPlaylist() {

if (start == null) {

System.out.println("Playlist is empty.");

return;

}

Node temp = start;

System.out.println("Playlist:");

while (temp != null) {

System.out.println(temp.song);

temp = temp.next;

}

}

public void playSong(String songName) {

Node temp = start;

while (temp != null) {

if (temp.song.equals(songName)) {

System.out.println("Now playing: " + songName);

pushToRecent(songName);

return;

}

temp = temp.next;

}

System.out.println("Song not found in playlist.");

}

private void pushToRecent(String song) {

Node newNode = new Node(song);

newNode.next = top;

top = newNode;

}

public void displayRecent() {

if (top == null) {

System.out.println("No recently played tracks.");

return;

}

System.out.println("Recently played tracks:");

Node temp = top;

while (temp != null) {

System.out.println(temp.song);

temp = temp.next;

}

}

public void loadPlaylistFromFile() {

try (BufferedReader reader = new BufferedReader(new FileReader("playlist.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

addSong(line);

}

System.out.println("Playlist loaded from file.");

} catch (IOException e) {

System.out.println("Error loading playlist: " + e.getMessage());

}

}

public static void main(String[] args) {

PlaylistManager playlist = new PlaylistManager();

Scanner scanner = new Scanner(System.in);

System.out.println("\*\*WELCOME\*\*\n\*\*Please use '\_' for spaces.\*\*");

while (true) {

System.out.println("\nMenu:\n1. Add Song\n2. Delete Song\n3. Display Playlist\n4. Play Song\n5. Recently Played\n6. Load Playlist from File\n7. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print("Enter song name: ");

String song = scanner.nextLine();

playlist.addSong(song);

break;

case 2:

System.out.print("Enter song name to delete: ");

song = scanner.nextLine();

playlist.deleteSong(song);

break;

case 3:

playlist.displayPlaylist();

break;

case 4:

System.out.print("Enter song name to play: ");

song = scanner.nextLine();

playlist.playSong(song);

break;

case 5:

playlist.displayRecent();

break;

case 6:

playlist.loadPlaylistFromFile();

break;

case 7:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Try again.");

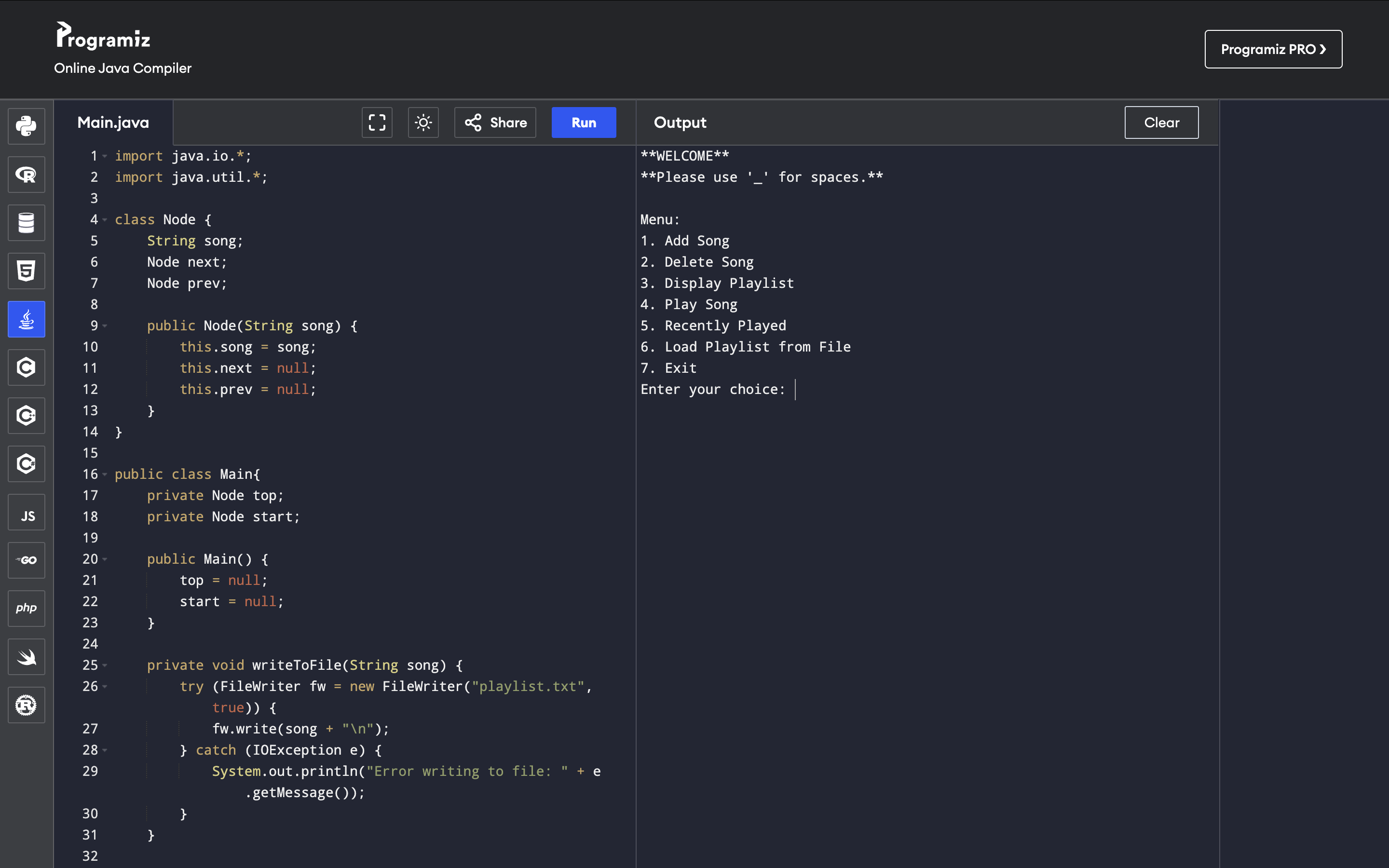
}

}

}

}

**SAMPLE INPUT AND OUTPUT**

****