Saveetha School of Engineering

Saveetha Institute of Medical and Technical Sciences

Department of Computer Science Engineering

Engineer to Excel

Assignment on topic: Unit 5

CO2: Develop object-oriented programs using User Interfaces and Event Handling.

How would you design a simple media player system class and implement the described functionality? Include the code and expected output based on the given scenario.

Scenario:

You are tasked with designing and implementing a simple media player system that can play, pause, stop, and display the details of media files. The system should allow users to manage their media library and control playback.

The system will allow media player to:

- **Play Media Files: The system will allow users to play various types of media files, including audio and video formats such as MP3, WAV, MP4, AVI, etc.**
- **Pause, Resume, and Stop Playback: Users will have the ability to pause the media playback, resume from where it left off, or completely stop the playback.**
- **Volume Control: The system will provide users with the ability to adjust the volume of the media being played, either through a slider or predefined controls.**
- **Media Navigation: The system will allow users to skip forward or backward within a media file, enabling easy navigation to specific parts of a song or video.**
- **Playlist Management: Users can create, manage, and organize playlists of media files. They can add, remove, and reorder files in a playlist.**
- **File Compatibility: The media player will support a range of common media file formats and ensure smooth playback without crashes or errors for supported file types.**
- **Media File Information: The system will display basic information about the media file currently playing, such as file name, duration, artist (for music), or video resolution (for video files).**
- **Secure File Handling: The system will ensure that only accessible, non-corrupted media files are played. It will provide error messages or avoid playing unsupported or damaged files.**
- **User Interface Customization: The media player will allow users to customize the interface, such as switching between light and dark modes, resizing windows, or changing the color scheme.**
- **Background Playback: The system will support background playback, enabling users to play media while using other apps or while the media player window is minimized.**
- **Media File Metadata: The player will read and display metadata associated with media files (like album art, artist name, year, genre) without tampering with the media content.**
- **Update and Compatibility Checks: The system will regularly check for updates to ensure the media player can handle the latest file formats and stay compatible with operating system upgrades.**

Tasks:

1. **Define the MediaPlayer class**:
    - Create attributes to store the currentTrack, volume, isPlaying, isPaused, trackList and mediaThread.

2. **Implement Methods in the MediaPlayer Class:**

- `play(String track):` Starts the playback of the selected track.
- `pause():` Pauses the current media playback.
- `stop():` Stops the media playback.
- `changeTrack(String track):` Switches to a new track.
- `adjustVolume(int level):` Adjusts the volume of the media player.

2. **Simulate the media playing process**:
    - **Create Objects for Different songs**: Create two instances of the `MediaPlayer` class for different songs, artist, album, volume.
    - **Display the song Details**: Display the information of the song, displaying the song status.
    - **Attempt to play the next song**: Try to play the next song by manual or it plays randomly or with the line up.

Steps:

1. Define the MediaFile class with appropriate attributes and a constructor.

2. Define the SimpleMediaPlayer class with a list to store media files and implement the required methods.

Test Scenarios: To validate the system, create two instances of media player:

1. **Song A, (volume)75, (Album)hits, playing.**
2. **Song B, (volume)80, (Album)hits, paused.**

Deliverables:

Adding media files.

Removing media files.

Playing, pausing, and stopping media files.

Displaying the media library.

Grading Rubrics

Criterion	Needs Improvement (0-2)	Satisfactory(3-5)	Good(6-8)	Excellent(9-10)
Class Design	- Missing class definition. - Class lacks required attributes..	- Basic class structure with some attributes but may be incomplete or inaccurate.	- Class structure is mostly correct, but may miss one or two attributes or minor issues in naming conventions.	-Complete class with all required attributes - Proper naming conventions and encapsulation.
Constructor Implementation	- Constructor is missing or incomplete. - Fails to initialize attributes properly.	- Constructor is present but may miss initializing some attributes correctly.	- Constructor initializes most attributes correctly, minor issues like missing default values.	- Properly initializes all attributes in the constructor. - Clear and logical parameterized constructor.
Object Creation and Testing	- Missing or incorrect object creation. - Fails to create and test ticket instances as required.	- Objects are created, but testing may be incomplete or incorrect	- Objects are created and tested but may miss edge cases	- Objects are correctly created and thoroughly tested
Code Quality and Testing	Poor code quality with many errors or lack of testing. Doesn't follow naming conventions or Java best practices.	Code works but lacks consistency Some parts of the code are untested or don't handle all edge cases.	Code is clean and follows Java best practices Code is tested, but some edge cases are missed.	High-quality code that adheres to Java best practices, well-documented, and fully tested. Handles edge cases and input validation thoroughly.
Submission Deadline	Frequently misses deadlines, often requiring significant extensions	Submits work within a short grace period after deadlines, with some minor delays	Deadlines are met in most cases, with only occasional minor delays	Always meets or beats submission deadlines with no delays

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSA09 – PROGRAMMING IN JAVA

ASSIGNMENT REPORT

REGISTER NUMBER: 192373014

NAME: Sowmiyan.I

SUBMISSION DATE: 20-11-2024

Simple Media Player System

1. Objective

The objective of this project is to design and implement a simple media player system that can play audio and video files, allow users to control the media playback (play, pause, stop), and manage multiple media files concurrently. The system should use multithreading to handle various tasks like playback, volume control, and track switching without affecting the user experience.

2. Design and Implementation

The solution involves designing a **MediaPlayer** class to represent the media player, with methods for playing, pausing, stopping, and switching tracks. We will use **multithreading** to simulate multiple media tasks running simultaneously, such as adjusting the volume, changing tracks, or playing media files. The threads will ensure that these operations can run concurrently without blocking the main media playback.

Class Design

MediaPlayer Class:

- This class will represent the media player and include attributes and methods for controlling media playback.

Attributes include:

- `currentTrack:` Stores the currently playing track.
- `volume:` The current volume level.
- `isPlaying:` A flag indicating whether the media is playing.
- `isPaused:` A flag indicating whether the media is paused.
- `trackList:` A list of available tracks in the media player.
- `mediaThread:` A thread to handle media playback.

Methods Implemented

- `play(String track):` Starts the playback of the selected track.
- `pause():` Pauses the current media playback.
- `stop():` Stops the media playback.
- `changeTrack(String track):` Switches to a new track.
- `adjustVolume(int level):` Adjusts the volume of the media player.

The program performs the following actions:

1. `TrackNotFoundException`: Handles invalid track selection (when the user selects a non-existent track).
2. `InvalidVolumeException`: Handles invalid volume adjustments (e.g., volume out of range).

3. CODING

```
import java.util.*;

// Custom exception for invalid track selection
class TrackNotFoundException extends Exception {
    public TrackNotFoundException(String message) {
        super(message);
    }
}

// Custom exception for invalid volume level
class InvalidVolumeException extends Exception {
    public InvalidVolumeException(String message) {
        super(message);
    }
}

// MediaPlayer class to handle media playback
class MediaPlayer {
    private String currentTrack;
    private int volume;
    private boolean isPlaying;
    private boolean isPaused;
    private List<String> trackList;
    private Thread mediaThread;

    // Constructor to initialize the media player
    public MediaPlayer(List<String> tracks) {
        this.trackList = tracks;
        this.isPlaying = false;
        this.isPaused = false;
        this.volume = 50; // Default volume level
    }
```

```
// Method to start playing a track
public synchronized void play(String track) throws TrackNotFoundException {
    if (!trackList.contains(track)) {
        throw new TrackNotFoundException("Track " + track + " not found.");
    }
    currentTrack = track;
    isPlaying = true;
    isPaused = false;

    mediaThread = new Thread(() -> {
        try {
            System.out.println("Playing " + currentTrack + "...");
            Thread.sleep(5000); // Simulate track playing for 5 seconds
            if (!isPaused) {
                System.out.println("Finished playing " + currentTrack);
            }
        } catch (InterruptedException e) {
            System.out.println("Playback interrupted.");
        }
    });
    mediaThread.start();
}

// Method to pause the current track
public synchronized void pause() {
    if (isPlaying && !isPaused) {
        isPaused = true;
        System.out.println("Paused " + currentTrack);
    } else {
        System.out.println("No media is currently playing.");
    }
}

// Method to stop the current track
public synchronized void stop() {
```

```
    if (isPlaying) {
        isPlaying = false;
        isPaused = false;
        System.out.println("Stopped " + currentTrack);
    } else {
        System.out.println("No media is currently playing.");
    }
}

// Method to change to another track
public synchronized void changeTrack(String newTrack) throws TrackNotFoundException {
    if (!trackList.contains(newTrack)) {
        throw new TrackNotFoundException("Track " + newTrack + " not found.");
    }
    currentTrack = newTrack;
    if (isPlaying) {
        stop(); // Stop current track
        play(newTrack); // Start new track
    } else {
        System.out.println("Track changed to " + newTrack);
    }
}

// Method to adjust the volume level
public synchronized void adjustVolume(int level) throws InvalidVolumeException {
    if (level < 0 || level > 100) {
        throw new InvalidVolumeException("Invalid volume level. Must be between 0 and 100.");
    }
    volume = level;
    System.out.println("Volume set to " + volume);
}

// Method to display current media details
public void displayMediaInfo() {
    System.out.println("Currently playing: " + currentTrack);
```

```
        System.out.println("Volume: " + volume);
    }
}

// Main Class to simulate the media player with multithreading
public class SimpleMediaPlayerSystem {
    public static void main(String[] args) {
        // List of available tracks
        List<String> tracks = Arrays.asList("Song A", "Song B", "Song C");

        // Create media player object
        MediaPlayer mediaPlayer = new MediaPlayer(tracks);

        // Simulate media player actions with multiple threads
        Thread playThread = new Thread(() -> {
            try {
                mediaPlayer.play("Song A");
            } catch (TrackNotFoundException e) {
                System.out.println(e.getMessage());
            }
        });

        Thread pauseThread = new Thread(() -> {
            try {
                Thread.sleep(2000); // Wait 2 seconds before pausing
                mediaPlayer.pause();
            } catch (InterruptedException e) {
                System.out.println("Error in pausing.");
            }
        });

        Thread stopThread = new Thread(() -> {
            try {
                Thread.sleep(4000); // Wait 4 seconds before stopping
                mediaPlayer.stop();
```

```
            } catch (InterruptedException e) {
                System.out.println("Error in stopping.");
            }
        });

        Thread volumeThread = new Thread(() -> {
            try {
                Thread.sleep(1000); // Wait 1 second before adjusting volume
                mediaPlayer.adjustVolume(75);
            } catch (InvalidVolumeException | InterruptedException e) {
                System.out.println(e.getMessage());
            }
        });

        // Start the threads
        playThread.start();
        pauseThread.start();
        stopThread.start();
        volumeThread.start();

        // Wait for threads to finish
        try {
            playThread.join();
            pauseThread.join();
            stopThread.join();
            volumeThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Display current media info
        mediaPlayer.displayMediaInfo();
    }
}
```

4. OUTPUT:

Playing Song A...
Volume set to 75
Paused Song A
Stopped Song A
Currently playing: Song A
Volume: 75

Conclusion

The Simple Media Player System was successfully designed and implemented using multithreading and synchronization. The system allows concurrent operations like media playback, pausing, volume adjustment, and track switching without interfering with each other. The use of custom exceptions ensures proper error handling for invalid tracks and volume levels. This project can be extended further with features like playlist management, media file formats, and user authentication.