Programação III

Licenciatura em Engenharia Informática Redes e Telecomunicações

3º Trabalho Prático

Grupo 5

47593 – José Jorge

44621 – Daniel Pardal

Docente: Jorge Martins

1º Semestre letivo 2021/2022

Índice

Introdução

Este documento surge como relatório do terceiro trabalho prático desenvolvido no âmbito da unidade curricular de Programação III, do curso de Licenciatura em Engenharia Informática, Redes e Telecomunicações.

Adiante são explicados o conceito e funcionalidades do projeto desenvolvido durante o trabalho.

Resumo

O terceiro trabalho prático da disciplina de PG3, consiste no desenvolvimento parcial de uma estrutura pré-existente com o objetivo de servir uma aplicação de reprodução de ficheiros áudio, vulgo, um leitor de ficheiros MP3.

As principais funcionalidades da aplicação desenvolvida são as de:

- Carregamento a partir de um diretório ou hierarquia de diretórios, de ficheiros musicais, sejam eles músicas, álbuns de músicas ou listas de reprodução (playlists) numa base de dados;
- Listagem e apresentação de ficheiros musicais e da informação associada aos mesmos, quer na integra, quer apresentando resultados de filtros multicritério (intérprete, género musical, etc);
- Operações básicas sobre esses mesmos ficheiros musicais: a sua reprodução, pausa e paragem.

Desenvolvimento

Componentes principais da aplicação

Mp3PlayerFrame

A classe MP3PlayerFrame consiste na interface gráfica, esta classe irá utilizar uma instância de MenuItemCheckList para que os géneros selecionados permitem selecionar determinados tipos de música.

Esta classe irá também utilizar os getters da classe MusicItemFinder para que possamos adicionar faixas, álbuns, diretórios e playlists, e por último esta classe utiliza o método setItems do ItemsViewer para visualizar determinadas músicas.

MusicDB

A classe MusicDB implementa a base de dados e tem métodos para a adição de faixas e álbuns, para a construção de playlists e para a obtenção de coleções selecionadas de músicas como por exemplo: todos os álbuns, faixas musicais de determinado género, ou artistas.

Mp3Player

A classe Mp3Player contém os métodos que permitem reproduzir uma música ou uma sequência de músicas.

Esta classe permite também registar listeners para os seguintes eventos:

- Faixa começou a reprodução
- Terminou a reprodução das faixas
- Erro de reprodução

MusicItemFinder

A classe MusicItemFinder disponibiliza métodos para obter:

- uma faixa, dado o nome do ficheiro
- um álbum, dada a diretoria;
- todas as músicas (faixas e álbuns) que estejam contidas numa diretoria e suas subdiretorias.

Listeners

ItemsViewer

Os listeners da classe ItemsViewer disparam um evento de seleção quando um item é selecionado ou um evento de zoom quando, sobre o evento selecionado, for premido o botão direito do rato.

```
private void fireZoomItem(Object item) {
    if (zoomListener != null)
        zoomListener.onZoomItem(item);
}
```

Figura 1 - FireZoomItem

```
public ItemsViewer() {
    itemsPanel = new JTable(new DefaultTableModel( new String[] {"list of ..."}, rowCount: 0)) {
        @Override
        public boolean isCellEditable(int rowIndex, int columnIndex) { return false; }
    };

    final MouseListener mouse = (MouseAdapter) mouseClicked(e) -> {
            Object selected = getSelectedItem();
            if (selected == null ) return;
            if (SwingUtilities.isRightMouseButton(e))
                fireZoomItem( selected );
            else
                fireSelectedItem( selected );
    };
```

Figura 2 - Demonstração do evento

Mp3Player

Como referido anteriormente, a classe Mp3Player contém os métodos que permitem reproduzir uma música ou uma sequência de músicas e esta classe contém listeners para os seguintes eventos:

- Faixa começou a reprodução
- Terminou a reprodução das faixas
- Erro de reprodução

```
private void fireStartMusicEvent(String title) {
    if (startListener != null) {
        SwingUtilities.invokeLater(
            () -> startListener.onStartMusic(title)
        );
    }
}
```

Figura 3 - Evento para a faixa começar a reprodução

```
private void fireCompletedEvent() {
    if (completedListener != null) {
        SwingUtilities.invokeLater(
            () -> completedListener.onComplete()
        );
    }
}
```

Figura 4 - Evento para terminar a reprodução das faixas

```
private void fireErrorEvent(Exception  e) {
    if (errorListener != null) {
        SwingUtilities.invokeLater(
            () -> errorListener.onError(e)
        );
    }
}
```

Figura 5 - Evento de erro de reprodução

Elementos adicionados ou modificados

Na classe Mp3PlayerFrame foi adicionado ao MenuItemCheckList Genres os JBoxMenuItem através de um cliclo for. Este ciclo ia percorrer o arrays de strings que continham os nomes que iam fichar nos JBoxMenuItem e ia adicionar os respetivos ActionsEvents a cada um. Foi adicionado também o JMenu Artists e os respetivos JMenuItems e ActionEvents.

```
private void buildMenus() {
    // Only for test
    JMenuBar menuBar = new JMenuBar();

    MenuItemCheckList genres = new MenuItemCheckList( title: "Genres");
    menuBar.add(genres);
    String[] genreStrings = {"Jazz", "Folk", "Rock", "Pop", "Metal", "R&B", "Soul", "Blues"};

    for (String str : genreStrings ){
        JCheckBoxMenuItem mi = new JCheckBoxMenuItem( str );
        genres.add( mi );
        mi.addActionListener( this::showSongItemByGenre );
    }

    JMenu artists = new JMenu( s: "Artists");
    menuBar.add(artists);
    String[] artistStrings = {"Al Jarreau", "Al Di Meola", "Eric Clapton", "Bee Gees"};


    for (String str : artistStrings ){
        JMenuItem mi = new JMenuItem( str );
        artists.add( mi );
        mi.addActionListener( this::showSongItemByArtist );
    }
```

Figura 6 - MenuItemCheckLists e respetivos actionevents

As figuras 7 e 8 remetem ao adicionamento dos JMenu Add e Show. Para o JMenu Add foram adicionados os JMenuItem Album, Song, Folder e Playlist e os respetivos ActionEvents.

Para o JMenu Show foram adicionados os JMenuItem Albums, Songs, Playlists, Genres e Artists e os respetivos ActionEvents.

```
JMenu add = new JMenu( s: "Add");
menuBar.add(add);
JMenuItem addAlbum = new JMenuItem( text: "Album");
JMenuItem addSong = new JMenuItem( text: "Song");
JMenuItem addFolder = new JMenuItem( text: "Folder");
JMenuItem addPlaylist = new JMenuItem( text: "Playlist");
add.add(addAlbum);
add.add(addSong);
add.add(addFolder);
add.add(addPlaylist);

JMenu show = new JMenu( s: "Show");
menuBar.add(show);
JMenuItem showAlbum = new JMenuItem( text: "Albums");
JMenuItem showSong = new JMenuItem( text: "Songs");
JMenuItem showPlaylist = new JMenuItem( text: "Playlist");
JMenuItem showGenres = new JMenuItem( text: "Genres");
JMenuItem showArtists = new JMenuItem( text: "Artists");
show.add(showAlbum);
show.add(showSong);
show.add(showPlaylist);
show.add(showGenres);
show.add(showArtists);
```

Figura 7 – JMenu Add e Show e os seus respetivos JMenuItems

```
//Action Events

//-------- Genres ------------
//ActionListener adicionado no ciclo For

//-------- Add ----------------
addSong.addActionListener(this::addSongItem);
addAlbum.addActionListener(this::addAlbumItem);
addPlaylist.addActionListener(this::addPlaylistItem);
addFolder.addActionListener(this::addPlaylistItem);


//-------Show-------------
showSong.addActionListener(this::showSongItem);
showAlbum.addActionListener(this::showAlbumItem);
showPlaylist.addActionListener(this::showPlaylistItem);
showGenres.addActionListener(this::showGenresItem);
showArtists.addActionListener(this::showArtistsItem);


setJMenuBar( menuBar );
```

Figura 8 – ActionEvents

Estruturas de dados implementadas para armazenar as músicas

Na classe MusicDB foram implementadas estruturas de dados para armazenar músicas.

Foram realizadas algumas funções, dentre as quais se destacam a função addRandomPlaylist e os diversos getters existentes.

```
public Playlist addRandomPlayList(String name, String[] genres,
                                  int totalSongs, long maxDuration) {

    Playlist pl = new Playlist(name);
    Random rndm = new Random();
    int random;
    // até n tempo ou n musicas ou n géneros
    for ( int i = 0; i < totalSongs && pl.getDuration() < maxDuration && i < genres.length; ++i ){
        if ( pl.getGenres().containsAll( Arrays.asList( genres ) ) ){    // se ainda nao tiver todos os generos
            TreeSet<Song> it = ( TreeSet<Song> ) this.getSongs( genres[i] );
            random = rndm.nextInt( it.size() );
            Song[] array = it.toArray( it.toArray( new Song[ it.size() ] ) );
            pl.addSong( array[ random ] );
        }
        else{
            // se ainda nao tiver todos, mas já tem este
            if ( pl.getGenres().contains( genres[i] ) ){ break; }
            // se ainda nao tiver todos, mas ainda nao tem este
            else{
                TreeSet<Song> it = ( TreeSet<Song> ) this.getSongs( genres[i] );
                random = rndm.nextInt( it.size() );
                Song[] array = it.toArray( it.toArray( new Song[ it.size() ] ) );
                pl.addSong( array[ random ] );
            }
        }
    }
    return pl;
}
```

Figura 9 - Função addRandomPlaylist

A função addRandomPlaylist recebe como parâmetros uma string com o nome da playlist, um array de strings com os géneros das músicas, o número de músicas existentes na playlist e a máxima duração da playlist e irá retornar uma nova playlist.

Para a realização desta função começámos, primeiramente, por criar uma nova playlist com o nome passado por parâmetro, seguidamente foi realizado um ciclo for de modo a adicionar as musicas à Playlist, inicialmente criada, para depois ser retornada no final da função.

Na classe MusicDB foram também implementados os seguintes getters:

- getSongs
- getAlgums
- getPlaylists
- getArtists
- getGenres

```
public Iterable<Song> getSongs() {
    ArrayList<Song> songs = new ArrayList<>();
    table.forEach( (key, value) -> {
        if (value instanceof Song) songs.add( (Song) value );
    } );
    return songs;
}
public Iterable<Album> getAlbums() {
    ArrayList<Album> albums = new ArrayList<>();
    table.forEach( (key, value) -> {
        if (value instanceof Album) albums.add( (Album) value );
    } );
    return albums;
}
public Iterable<Playlist> getPlaylists() {
    ArrayList<Playlist> playlists = new ArrayList<>();
    table.forEach( (key, value) -> {
        if (value instanceof Playlist) playlists.add( (Playlist) value );
    } );
    return playlists;
}
public Iterable<String> getArtists() {
    Set<String> artists = new TreeSet<>();
    table.forEach( (key, value) -> {
        for( String artist : value.getArtists() ){
            artists.add( artist );
        }
    } );
    return artists;
}
```

Figura 10 - Getters de Songs, Albuns, Playlists e de Artists

```
public Iterable<String> getGenres() {
    Set<String> genres = new TreeSet<>();
    table.forEach( (key, value) -> {
        for( String genre : value.getGenres() ){
            genres.add( genre );
        }
    } );
    return genres;
}
```

Figura 11 - Getter de Genres

Abordagem dos temas estudados ao longo do semestre

Herança e Polimorfismo

O Polimorfismo, Encapsulamento, e Herança, são as principais características da Programação Orientada por Objetos (Object Oriented Programming).

Herança

A herança permite que a partir da definição de uma classe geral se definam classes mais especializadas simplesmente por adição de novos detalhes à definição da classe geral. A classe mais especializada herda as características da classe geral, por isso só as novas características necessitam de ser programadas.

Polimorfismo

A palavra polimorfismo deriva do grego e significa "muitas formas."

O mesmo programa adapta-se a diferentes meios e age conforme os diferentes contextos como por exemplo, o mesmo nome de método, usado numa instrução, produz resultados que dependem do tipo do objeto sobre o qual foi chamado o método.

Polimorfismo é um conceito que exprime a ideia de "o objeto faz algo", ou seja, objetos diferentes possuem métodos com a mesma assinatura que quando chamados executam tarefas de maneiras diferentes.

O polimorfismo em Java envolve os dois mecanismos:

- Sobreposição (overriding)
- Ligação dinâmica (dynamic binding)

A sobreposição consiste na possibilidade de, numa relação de herança entre classes, as classes derivadas poderem sobrepor os métodos herdados, mantendo a respetiva assinatura.
A ligação dinâmica consiste, na seleção, em tempo de execução, de um método consoante o tipo do objeto sobre o qual um método é invocado.

Estruturas de dados

Uma estrutura de dados (data structure) é usada para organizar os dados de forma especifica.

Existem dois tipos de estruturas de dados, as estruturas de dados estáticas e as estruturas de dados dinâmicas.

As estruturas de dados estáticas (static data structure) são estruturas que não crescem nem diminuem o seu tamanho enquanto o programa estiver em execução, como por exemplo um array.

Por outro lado, as estruturas de dados dinâmicas (dynamic data structures) podem crescer e diminuir enquanto o programa está em execução, como por exemplo os ArrayList e LinkedList.

Interfaces funcionais

O Java 8 acrescentou o package java.util.function com interfaces funcionais. As quatro interfaces funcionais mais conhecidas são:

- Predicate - permite verificar se o valor do parâmetro de entrada tem determinada característica.
- Function - permite transformar o parâmetro de entrada.
- Consumer - é usada para realizar uma ação sobre o argumento.
- Supplier - fornece um valor.

As interfaces funcionais BiPredicate, BiFunction e BiConsumer têm a mesma funcionalidade que as interfaces Predicate, Function e Consumer respetivamente, só que têm dois parâmetros de entrada.

Streams

Em Java a leitura e escrita em ficheiros, bem como a leitura do keyboard e a escrita no ecrã é feita através de streams.

Um stream é uma sequência de dados (caracteres, números, etc.).

Um stream é um objeto. Os dados escritos num output stream são entregues num destino que pode ser um ficheiro ou o standard output.

Os dados que o programa lê de um input stream são recebidos de uma fonte que pode ser um ficheiro ou o standard input (keyboard).

Os objetos da classe Scanner, usados para a leitura do keyboard, leem os dados de um input stream.

Nos BINARY OUTPUT STREAMS todos os dados são armazenados num ficheiro em binário.

A classe PrintWriter é utilizada para escrita em ficheiros de texto. Os métodos print e println da classe PrintWriter têm comportamento idêntico aos métodos: System.out.print e System.out.println respetivamente.

Um ficheiro é aberto para escrita usando:

- new FileWriter("out.txt")

Esta instrução cria um stream, que é um objeto do tipo Writer, e liga-o a um ficheiro vazio. Se já existe um ficheiro com o nome passado no argumento, todo o seu conteúdo é perdido, caso o nome do ficheiro não exista, é criado um novo ficheiro com o nome passado como argumento. O construtor de FileWriter, pode lançar (throw) uma exceção IOException, que significa que o ficheiro não foi criado.

Programação event driven

As aplicações que usam GUI usam eventos e tratamento de eventos (event handlers).

Um evento é um objeto que representa qualquer ação tal como o click ou o arrasto do rato, o premir de uma tecla. Quando um componente gera um evento, diz-se que dispara (fire) o evento. Um componente que "dispara" um evento pode ter um ou mais objetos listener, especificados pelo programador para tratar os eventos.

O tratamento do evento é definido pelo programador registando no componente que dispara o evento um ou mais objetos listener.

Um componente pode disparar um ou mais eventos e ter um ou mais objetos listener por evento. O programador escolhe quais são os eventos disparados pelos componentes que devem ter listeners registados.

Na programação event-driven, os eventos produzidos por ações do utilizador é que determinam a ordem das atividades.

Em geral, é impossível predizer a sequência de eventos. Quando escrevemos um GUI, existem métodos que não se chamam diretamente, no entanto, esses métodos são chamados para responder aos eventos. As classes que definimos geralmente implementam interfaces ou estendem de classes das API Swing ou AWT.

Conclusão

Em suma, podemos concluir que os objetivos impostos no trabalho foram alcançados uma vez que a implementação do Mp3 foi concluída.
Ao decorrer do desenvolvimento do trabalho foram encontradas algumas dificuldades como por exemplo na função addRandomPlaylist da classe MusicDB, em que não nos era especificado o que era o maxDuration e o totalsongs. Apesar dessas dificuldades, conseguimos ultrapassá-las e concluir o trabalho.