nerDJ True Star

Gerador de Música

Técnicas de Construção de Programas

Trabalho Prático - FASE 2

Nicolle Pimentel Favero

1. Descrição do Programa

    a) Objetivo do programa
    O programa tem por objetivo receber uma String e tocar uma música de acordo com os caracteres dela. Além disso, será possível pausar e retomar a música e passar alguns parâmetros adicionais como a oitava, o volume e o ritmo iniciais da música.

    b) Elementos principais e seus objetivos

    Main: A classe Main é responsável por instanciar um novo objeto da classe WindowHandler para carregar a GUI.

    WindowHandler: A classe WindowHandler é responsável por configurar e inicializar a GUI.

    Controller: A classe Controller é responsável por indicar qual pedaço da GUI pertence a qual pedaço de código. É ela quem conecta (com o auxílio de um arquivo .fxml) uma caixa de texto na GUI, por exemplo, a uma variável no código do programa. Nessa classe que vai estar o método ligado ao botão "Play" da GUI, que irá chamar a classe responsável por traduzir a String recebida em uma String que a biblioteca JFugue seja capaz de interpretar e passá-la ao player do JFugue.

    Parser: A classe Parser será responsável por traduzir a String recebida para a sintaxe do JFugue.

    MusicHandler: Essa classe também será chamada no Controller, logo após a classe Parser retornar a String traduzida. A classe MusicHandler é responsável por transformar a String em um Pattern e tocá-lo através de uma instância da classe Player.

2. Avaliação da modularidade do programa

Avaliação pelos Critérios da Modularidade:

- Decomposability: O objetivo do software foi decomposto em objetivos menores, como: carregar a interface (Main), configurar a interface (WindowHandler), traduzir o input (Parser), tocar a música (MusicHandler) e controlar o fluxo do programa manipulando os inputs e chamando outras classes (Controller).
- Composability: Todas as classes são auto suficientes.
- Understandability: Os módulos do programa estão fáceis de entender, pois cada um executa apenas a sua função sem ter vínculos

desnecessários com outros módulos. Uma vantagem de usar Java, é que os métodos têm input/output explícito.
- Continuity: Por serem auto suficientes, certas mudanças em qualquer das classes não se perpetuam pelo resto do código. Desde que os métodos retornem/recebam o mesmo tipo de variável declarado inicialmente, é possível modificá-lo ou expandi-lo sem afetar quem lhes passa o parâmetro ou quem recebe seu retorno.
- Protection: No Java, a maioria das exceções são explícitas. Além disso, no programa, o MusicHandler toca somente Patterns válidos, o Parser segue uma lógica que desconsidera todo input não especificado, o Controller valida o formato dos inputs.

Avaliação pelas Regras da Modularidade:

- Direct Mapping: O software foi dividido conforme os seus problemas foram divididos: parsing, interface, player, main e controle de fluxo e de inputs.
- Few and Small Interfaces: O módulo Main apenas se comunica com o módulo WindowHandler e o Controller apenas chama Parser antes de passar o input ao MusicHandler.
- Explicit Interfaces: Toda a comunicação entre os módulos é bem explícita no programa.
- Information Hiding: As configurações da interface são manipuladas apenas na classe WindowHandler, a lógica de tradução do input fica guardada na classe Parser e as classes do JFugue são chamadas exclusivamente na classe MusicHandler.

Avaliação pelos Principios da Modularidade:

- Linguistic Modular Units: Os módulos do programa são divididos de acordo com as classes do programa em Java.
- Self-Documentation: Os módulos do programa são escritos de modo a poderem ser entendidos olhando apenas para o módulo.
- Uniform Access: Um exemplo no programa do trabalho seria a classe WindowHandler. Para instanciar e inicializar a interface, a Main sempre chama o método *load()* da classe WindowHandler. No programa, por enquanto, não há nenhum exemplo com getters.
- Open-Closed: A interface pode ser expandida, assim como o MusicHandler. Expandir a classe Parser já seria um pouco mais difícil, pois, ao mudar a sintaxe do input, teria que mudar o código da classe Parser.
- Single Choice: O único módulo que controlaria o fluxo do programa se houvesse mais de uma alternativa seria o Controller.

OBS: os trechos de código serão colocados a seguir e não diretamente em cada tópico, pois, várias vezes, um mesmo trecho de código se refere a mais de um critério, regra ou princípio da modularidade. A classe Parser ainda não está bem definida, por isso não foi incluída.

```java
import nerdj.WindowHandler;

public class Main {

    public static void main(String[] args) {
        new WindowHandler().load();
    }
}
```

```java
public class WindowHandler extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource( name: "nerdj.fxml"));
        primaryStage.setTitle("Nerdj");
        primaryStage.setScene(new Scene(root,  width: 1000,  height: 800));
        primaryStage.show();
    }

    public void load() {
        launch();
    }
}
```

```java
package nerdj;

import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import nerdj.parser.Parser;

public class Controller {

    @FXML
    TextArea musicString;
    MusicHandler musicHandler = new MusicHandler();
    Parser parser = new Parser();

    public void run(){
        this.musicHandler.playSong(this.parser.parser(this.musicString.getText()));
    }

}
```

```java
package nerdj;

import org.jfugue.pattern.Pattern;
import org.jfugue.player.Player;

public class MusicHandler {

    Player player;

    public MusicHandler() { this.player = new Player(); }

    public void playSong(String string) {
        Pattern pattern = new Pattern(string);
        this.player.play(pattern);
    }
}
```