# Reprodutor de Músicas

TCP - Etapa 3

Augusto Flach (314134), Tiago Flach (275896)

## 1ª Parte: Requisitos

### Requisitos do sistema:

- O sistema deve possuir uma entrada de texto;
- O sistema deve possuir uma entrada de texto através de arquivo;
- O sistema deve reproduzir músicas mapeadas do texto de entrada;
- O sistema deve exportar arquivos sonoros (MIDI) com base nas músicas mapeadas através do texto de entrada;
- O sistema deve permitir o controle de volume, BPM e instrumento musical através do texto de entrada;

### Requisitos não-funcionais:

- O sistema deve possuir uma interface simples e objetiva visando facilitar o seu uso;
- O sistema possuir ao menos 2 instrumentos musicais;
- O botão para reprodução de músicas deve estar destacado, de preferência com uma cor diferente dos demais botões;

# 2ª Parte: Projeto

O projeto de revisão das classes através de um modelo MVC (Model-View-Controller). As classes de “View” foram utilizadas para interação com o usuário. As classes de “Model” foram utilizadas para tratamento do texto de entrada, assim como a execução e reprodução do som através da biblioteca JFugue. As classes de “Controller” foram usadas para gerenciar a interação entre as classes de modelo e a interface com o usuário.

## Model:

A classe “MusicPlayer” recebe comandos para gerenciamento de música (alterar note, aumento de volume, troca de instrumento, etc) e faz a sua execução através da biblioteca JFugue.

A classe “TextReader” recebe o texto de entrada, e o analisa token a token. Para a classe, um token é definido através de padrões de entrada reconhecidos por Regex. Também foi criada um módulo “Enum” que fica responsável por definir quais tipos de comandos de entrada existentes.

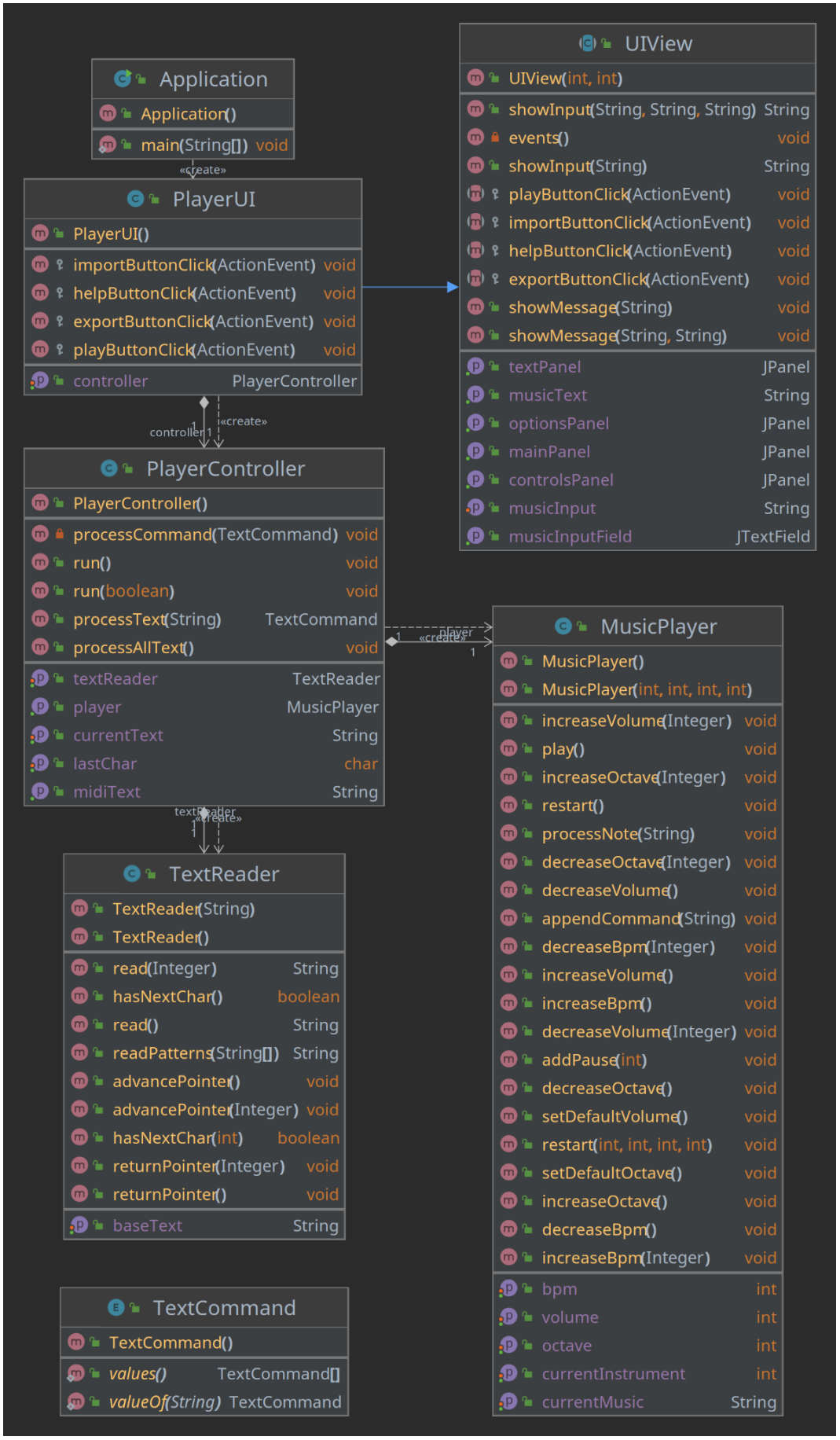
## View:

A classe “UIView” foi criada para definir a estrutura e disposição dos elementos na tela, através do uso da biblioteca Java Swing.

## Controller:

A classe “PlayerController” recebe um texto de entrada e gerencia interações entre as classes “TextReader” e “MusicPlayer” para gerar músicas a partir das especificações definidas no escopo do trabalho.

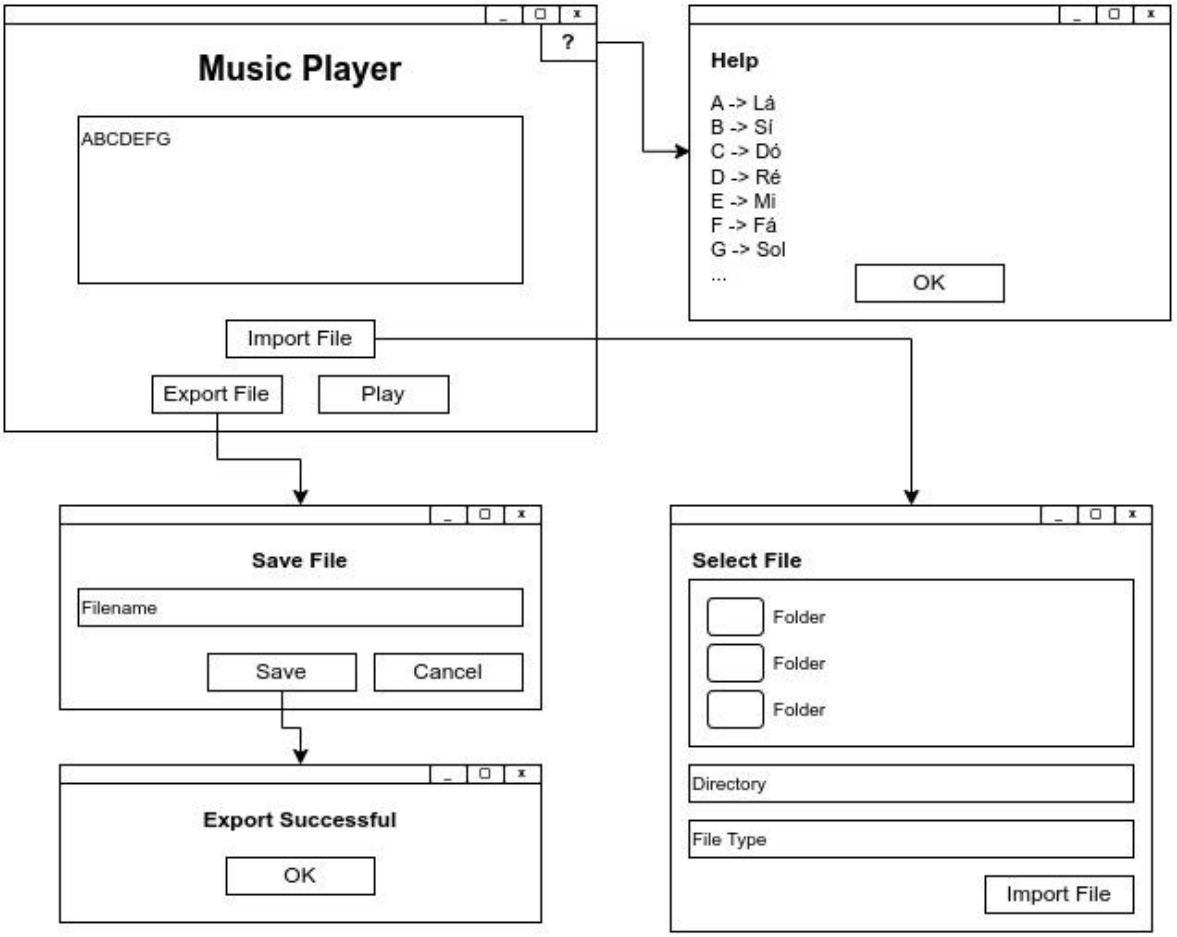
A classe “PlayerUI” gerencia a comunicação entre a interface “UIView” e a classe “PlayerController” para repassar as entradas do usuário para o restante do programa.


Application
Application()
main(String[]) void
«create»
PlayerUI
PlayerUI()
importButtonClick(ActionEvent) void
helpButtonClick(ActionEvent) void
exportButtonClick(ActionEvent) void
playButtonClick(ActionEvent) void
controller PlayerController
controller
1
1
«create»
UIView
UIView(int, int)
showInput(String, String, String) String
events() void
showInput(String) String
playButtonClick(ActionEvent) void
importButtonClick(ActionEvent) void
helpButtonClick(ActionEvent) void
exportButtonClick(ActionEvent) void
showMessage(String) void
showMessage(String, String) void
textPanel JPanel
musicText String
optionsPanel JPanel
mainPanel JPanel
controlsPanel JPanel
musicInput String
musicInputField JTextField
PlayerController
PlayerController()
processCommand(TextCommand) void
run() void
run(boolean) void
processText(String) TextCommand
processAllText() void
textReader TextReader
player MusicPlayer
currentText String
lastChar char
midiText String
player
«create»
1
1
MusicPlayer
MusicPlayer()
MusicPlayer(int, int, int, int)
increaseVolume(Integer) void
play() void
increaseOctave(Integer) void
restart() void
processNote(String) void
decreaseOctave(Integer) void
decreaseVolume() void
appendCommand(String) void
decreaseBpm(Integer) void
increaseVolume() void
increaseBpm() void
decreaseVolume(Integer) void
addPause(int) void
decreaseOctave() void
setDefaultVolume() void
restart(int, int, int, int) void
setDefaultOctave() void
increaseOctave() void
decreaseBpm() void
increaseBpm(Integer) void
bpm int
volume int
octave int
currentInstrument int
currentMusic String
textReader
«create»
1
TextReader
TextReader(String)
TextReader()
read(Integer) String
hasNextChar() boolean
read() String
readPatterns(String[]) String
advancePointer() void
advancePointer(Integer) void
hasNextChar(int) boolean
returnPointer(Integer) void
returnPointer() void
baseText String
TextCommand
TextCommand()
values() TextCommand[]
valueOf(String) TextCommand

# 3ª Parte: Interface

A interface é composta de uma tela principal e de outras telas secundárias informativas. A tela principal possui um título (“Music Player”), um campo de texto e quatro botões. O campo de texto serve para o input de música a partir do usuário (segundo o padrão especificado no trabalho). Ao clicar no botão “Play”, o programa reproduz a música inserida no campo de texto. O botão “?” Apresenta uma lista detalhada da função de cada comando de texto em uma janela separada (popup).

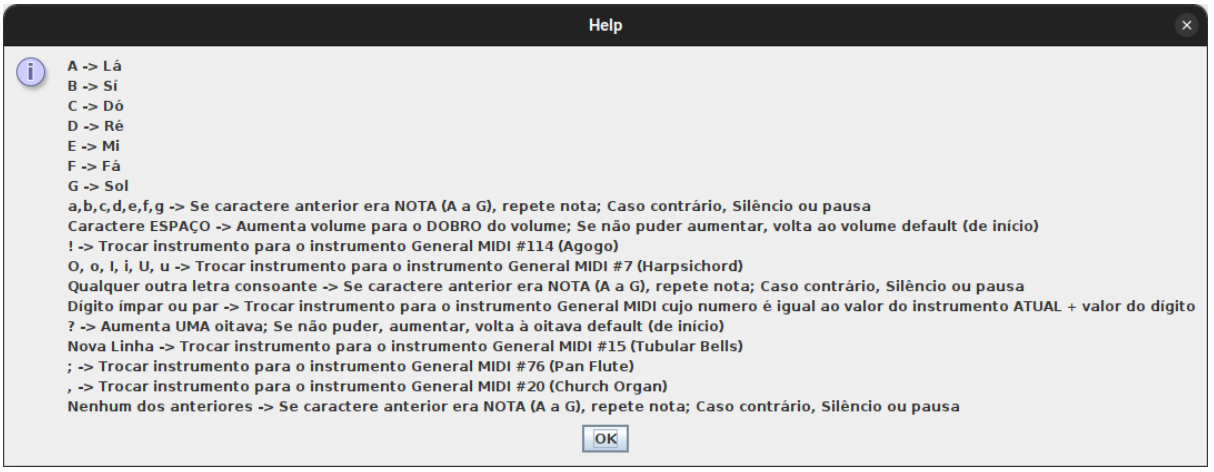
O botão “Import File” apresenta a tela de seleção de arquivo padrão do sistema operacional. Caso o usuário selecione um arquivo de texto válido, seu conteúdo é inserido no campo de input de texto, e o usuário é notificado de que tudo correu corretamente. O botão “Export File” abre uma janela para que o usuário escolha o nome do arquivo de destino. Após digitar o nome (ex: “filename”) e selecionar “OK”, é gerado um arquivo MIDI chamado “filename.mid” na raiz do diretório de execução do programa.



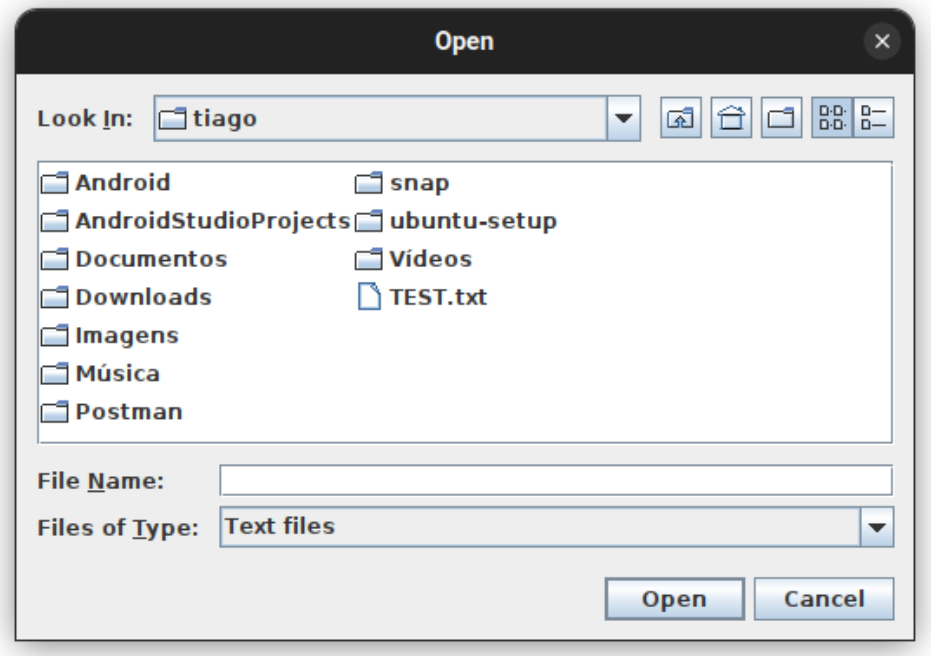
# 4ª Parte: Implementação

Tela principal:

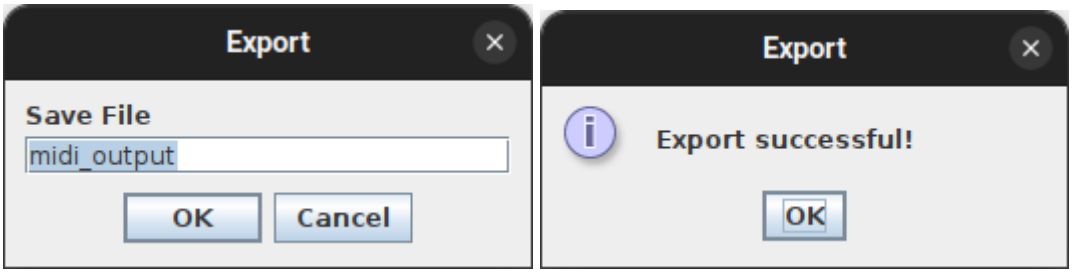

Tela de Ajuda:



Importação de Arquivos:




Exportação de Arquivos:



## Testes nas classes MusicPlayer e TextReader:

Foram gerados alguns testes para garantir que as funcionalidades principais das classes estejam funcionando como o esperado. Para isto, foram gerados testes unitários com através da biblioteca JUnit.