AMANDA NEVES DO CARMO

BIBLIOTECAS SWING E JAVAFX PARA SOFTWARES JAVA EM AMBIENTE DESKTOP – UMA ANÁLISE DE USABILIDADE E CÓDIGO-FONTE

BACHARELADO
EM
CIÊNCIA DA COMPUTAÇÃO

CARATINGA - MG
2017

BIBLIOTECAS SWING E JAVAFX PARA SOFTWARES JAVA EM AMBIENTE DESKTOP – UMA ANÁLISE DE USABILIDADE E CÓDIGO-FONTE

Monografia apresentada à banca examinadora da Faculdade de Ciência da Computação das Faculdades Doctum de Caratinga como exigência parcial para obtenção do grau de bacharel em Ciência da Computação, sob orientação do professor Esp. Maicon Vinícius Ribeiro.

CARATINGA – MG

2017

FACULDADES DOCTUM DE CARATINGA

FOLHA DE APROVAÇÃO

O Trabalho de Conclusão de Curso intitulado: BIBLIOTECAS SWING E JAVAFX PARA SOFTWARES JAVA EM AMBIENTE DESKTOP – UMA ANÁLISE DE USABILIDADE E CÓDIGO-FONTE, elaborado pela aluna AMANDA NEVES DO CARMO foi aprovado por todos os membros da Banca Examinadora e aceita pelo curso de Ciência da Computação das Faculdades Doctum de Caratinga, como requisito parcial para a obtenção do título de:

BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

Caratinga, 13 de dezembro 2017

Maicon Vinícius Ribeiro

Fabrícia Pires Souza

Wagner Aquino Zeferino

AGRADECIMENTOS

Agradeço muito a Deus, por ele ter me capacitado e me dado forças para chegar até aqui.

Agradeço ao apoio da minha família, principalmente dos meus pais, Angélica e José Félix que me incentivaram até o fim. Agradeço ao meu namorado João Paulo pelo incentivo, compreensão e força.

Agradeço também a todos professores, amigos e colegas, em especial ao meu orientador Maicon Ribeiro, pelo conhecimento compartilhado, pela paciência e também por ter o privilégio de conhecer e conviver com todos.

RESUMO

Hoje em dia, os *softwares* são uma necessidade na sociedade moderna, a partir disso o mercado nesta área se torna cada vez mais dinâmico, competitivo e inovador. Neste ambiente, a necessidade de ter *softwares* com mais qualidade no mercado é bem grande, e para os engenheiros e desenvolvedores, realizar o que o mercado pede para a satisfação de usuário e clientes pode não ser uma tarefa fácil quando se trata de *softwares* em ambiente desktop.

Para esta tarefa, é importante a escolha de boas bibliotecas, que proporcione qualidade externa para os usuários operarem no *software*, e qualidade interna para a manutenção do mesmo. Muitos profissionais, tem muitas dúvidas em relação em qual biblioteca escolher, podendo causar sérios prejuízos de custos no projeto de *software*. Visualizando este cenário, este trabalho teve por objetivo analisar a usabilidade e o código-fonte em projetos usando duas bibliotecas de interface gráfica do Java, Swing e JavaFX. Para isso foi aplicado uma pesquisa através de um questionário, desenvolvido tendo como base para a avaliação a norma SQuaRE ISO/IEC 25010 e as heurísticas de Nielsen.

Foram coletadas respostas de 51 profissionais e estudantes da área de Tecnologia da Informação, e os resultados demonstraram que os participantes têm ciência da importância da usabilidade em um *software* e ambas as bibliotecas proporcionam interfaces gráficas que atendem às métricas e heurísticas de usabilidade, mas de acordo com a análise de código-fonte foi visto que internamente são bem diferentes uma da outra, podendo influenciar no desenvolvimento e manutenção de um *software*.

Palavras-chave: Análise de usabilidade, análise de código-fonte, Java, Swing, JavaFX, heurísticas de Nielsen, ISO/IEC 25010.

ABSTRACT

Nowadays, softwares are a necessity in modern society, from there the market in this area becomes more and more dynamic, competitive and innovative. In this environment, the need to have higher quality software in the market is quite large, and for engineers and developers, accomplishing what the market asks for user and customer satisfaction may not be an easy task when it comes to environmentally friendly software desktop.

For this task, it is important to choose good libraries, to provide external quality for users to operate on the software, and internal quality to maintain the software. Many professionals, have many doubts regarding which library to choose, and can cause serious cost damages in software design. Looking at this scenario, this work aimed to analyze the usability and the source code in projects using two graphic interface libraries of Java, Swing and JavaFX. For that, a research was applied through a questionnaire, developed based on the evaluation of ISO / IEC 25010 and Nielsen heuristics.

Responses were collected from 51 professionals and students in the area of Information Technology, and the results demonstrated that the participants are aware of the importance of usability in a software and both libraries provide graphical interfaces that meet usability metrics and heuristics, but according to with the analysis of source code was seen that internally they are very different from each other, being able to influence in the development and maintenance of a software.

Keywords: Analysis of usability, analysis of source code, Java, Swing, JavaFX, Nielsen heuristics, ISO / IEC 25010.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ACC	Conexões aferentes (*Afferent Connections per Class*).
ACCM	Média da complexidade ciclomática dos métodos (*Average Cyclomatic Complexity per Method*).
AMLOC	Média de linhas de código por método (*Average Method Lines Of Code*).
ANPM	Média do número de parâmetros por método (*Average Number of Parameters per Method*).
API	Interface de programação de aplicativos (*Application Programming Interface*).
AWT	Kit de ferramentas de janelas (*Abstract Window Toolkit*).
CBO	Acoplamento entre objetos (*Coupling Between Objects*).
COF	Fator de acoplamento (*COupling Factor*).
CSS	Linguagem de estilo em cascata (*Cascading Style Sheets*).
CSV	Valores separados por vírgula, formato de arquivo (*Comma-Separated Values*).
DIT	Profundidade na árvore de herança (*Depth of Inheritance Tree*).
DTO	Objeto de transferência de dados (*Data Transfer Object*).
FXML	Linguagem de marcação extensível do JavaFX (*FX Markup Language).*
GUI	Interface gráfica do usuário (*Graphical User Interface*).
IDE	Ambiente integrado para desenvolvimento de software (*Integrated Development Environment*).
IEC	Comissão Eletrotécnica Internacional (*International Electrotechnical Commission*).
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos.
ISO	Organização Internacional de Normalização (*International Organization for Standardization*).
JAR	Arquivo de formato Java (*Java Archive*).
JDK	Kit de desenvolvimento do Java (*Java Development Kit*).
JVM	Máquina virtual do Java (*Java Virtual Machine*).
JRE	Ambiente de Execução do Java (*Java Runtime Environment*).
LCOM4	Falta de coesão entre métodos, versão 4 (*Lack of COhesion between Methods*).
LOC	Linhas de código (*Lines Of Code*).
MB	*MegaByte.*

NOM	Número de métodos (*Number Of Methods*).
NOP	Número de parâmetros (*Number Of Parameters*).
NOC	Número de subclasses (*Number Of Children*).
NPA	Número de atributos públicos (*Number of Public Attributes*).
Ph.D	Doutor da Filosofia (*Philosophiæ Doctor*).
RFC	Resposta para uma classe (*Response For a Class*).
SC	Complexidade estrutural (*Structural Complexity*).
SRC	Fontes (Abreviatura de *sources*).
RAM	Memória de acesso aleatório (*Random-access memory*).
XML	Linguagem de marcação extensível (*Extensible Markup Language*).
SQuaRE	Requisitos e Avaliação da Qualidade de *Software* (*Software Quality Requirements and Evaluation*).

SUMÁRIO

INTRODUÇÃO

Hoje em dia, os *softwares* são uma necessidade para grande parte da sociedade, seja para automatizar o estoque de uma grande empresa ou simplesmente se comunicar com uma pessoa que está distante. Em consequência disso e da concorrência de mercado, surgiu a preocupação de otimizar a qualidade destes *softwares* com o menor custo possível. Para isso ser alcançado, é necessário aplicar métodos de Engenharia de *Software*, práticas de gestões sólidas e controle de qualidade. Além disso é preciso escolher ferramentas e tecnologias que são adequadas aos requisitos do projeto de *software* e que também sejam de qualidade, pois uma má escolha destas, podem gerar falhas, inconsistências no *software*, insatisfação do usuário, entre outras frustrantes situações.

As aplicações Desktop são antigas no mercado de *softwares* e até hoje são bem utilizadas, pois têm como principais características não depender do uso da internet para serem disponibilizadas ao usuário e permitem o acesso a vários usuários simultâneos em um ambiente de rede.

A linguagem de programação Java (criada em 1991), possibilita a criação de aplicações multiplataforma, tem um vasto conjunto de bibliotecas com recursos de rede, comunicação com Hardware, interface Gráfica e entre outras. Segundo uma pesquisa feita pela revista IEEE *Spectrum* (publicada no mês de julho de 2016) a linguagem Java é uma das mais populares no mundo, estando em segundo lugar no ranking. Como o Java é uma linguagem que está a bastante tempo no mercado, muitas bibliotecas foram criadas, ficando um pouco difícil para os desenvolvedores escolherem a mais adequada para seus projetos.

Em meio a tantas bibliotecas, existem duas principais para criação de interface gráfica no Java que são o Swing e o JavaFX. Em uma busca feita em um site de perguntas e repostas bem acessado entre desenvolvedores, *Stackoverflow*, usando o termo “Java Swing” foram retornados 935 resultados, já usando o termo “JavaFX” foram retornados 262 resultados. O Swing é mais antigo que o JavaFX e segundo Deitel, P. e Deitel, H. (2016) ele ainda é amplamente utilizado entre os desenvolvedores. O JavaFX trouxe muitas novidades que muitos desconhecem, e há muitas dúvidas e discussões entre os desenvolvedores em qual biblioteca utilizar.

Neste contexto, o propósito deste estudo é analisar a qualidade de usabilidade e código-fonte que as duas bibliotecas, Swing e JavaFX, podem proporcionar para uma

aplicação desktop, e com os resultados desta análise, realizar uma comparação entre ambas. Para realizar a análise de usabilidade foram criados dois *softwares*. Para cada um, foi utilizado uma biblioteca diferente. A partir disso, foi criada uma pesquisa através de um questionário para coletar o ponto de vista dos profissionais e estudantes da área. Esta pesquisa foi elaborada a partir das métricas de usabilidade da norma SQuaRE ISO/IEC 25010 e das heurísticas de Nielsen.

Os resultados da aplicação do questionário demonstraram que as duas bibliotecas proporcionam interfaces gráficas que atendem às métricas e as heurísticas, e apresentam ainda que podem ficar aparentemente parecidas uma com a outra. Mas de acordo com a análise de código-fonte, internamente, são bem diferentes.

Este estudo pode determinar fatores importantes para garantir a qualidade de usabilidade e código-fonte em *softwares* Java Desktop. Os resultados podem servir como base para desenvolvedores poderem analisar e determinar qual será a melhor biblioteca de interface gráfica a ser utilizada e que atende aos seus requisitos de *software*.

Esse trabalho está dividido em quatro capítulos, sendo eles Referencial Teórico onde são apresentados todos os conceitos teóricos utilizados neste estudo, Metodologia onde é apresentado como se deu a condução deste trabalho para alcançar os resultados, Resultados onde são discutidos os dados obtidos e Conclusão.

1 REFERENCIAL TEÓRICO

Nesta seção serão introduzidos os conceitos necessários para o entendimento do trabalho, serão explicadas todas as tecnologias que foram utilizadas para o desenvolvimento do mesmo, detalhando a importância da qualidade de *software*, análise de código-fonte, usabilidade de *software* e sua influência para o desenvolvimento de um. Além de apresentar a evolução da linguagem Java em ambiente Desktop e de suas bibliotecas de interface gráfica.

1.1 JAVA

A tecnologia Java é uma plataforma que permite o desenvolvimento e execução de aplicações criadas com linguagem de programação também chamada Java. Atualmente pertence a Oracle (empresa multinacional de tecnologia da informação). Esta tecnologia se encontra no mercado desde o ano de 1995 e é muito popular, pois aplicações desenvolvidas por meio desta podem ser executadas em vários sistemas operacionais e dispositivos diferentes.

A linguagem de programação Java encontra-se em sua versão 9, na qual além de suportar três paradigmas de programação (programação procedural, programação orientada a objetos e programação genérica) acrescentou a programação funcional. Neste trabalho será utilizado a versão 8 do Java.

Segundo Deitel, P. e Deitel, H. (2016) a linguagem Java é uma das mais utilizadas no mundo. Para muitas organizações, é a linguagem preferida a fim de atender às necessidades de programação corporativa, ela também é amplamente utilizada para implementar aplicativos e *softwares* baseados na internet para dispositivos que se comunicam por uma rede.

A plataforma Java é constituída por outras plataformas que se correlacionam. Todas elas, fornecem uma máquina virtual JVM (*Java Virtual Machine*, em português, Máquina Virtual do Java) na qual as aplicações são emuladas, não dependendo de um sistema operacional ou hardware específicos para serem executadas.

As plataformas são:

- Java SE (*Standard Edition*) - é a base da plataforma Java. Inclui o ambiente de

execução e as bibliotecas comuns de interface gráfica de usuário, rede, entre outras. De acordo com a documentação da Oracle esta plataforma contém os recursos necessários para desenvolver aplicativos de desktop e servidor.

- Java EE (*Enterprise Edition*) - De acordo com a documentação da Oracle a *Enterprise Edition* é adequada para desenvolver aplicativos em rede distribuída e em grande escala e também aplicativos baseados na web.
- Java ME (*Micro Edition*) - De acordo com a documentação da Oracle a *Micro Edition* é um subconjunto do Java SE e fornece bibliotecas voltadas para o desenvolvimento de aplicações para dispositivos móveis e embarcados, como *smartphones*, *MP3 players*, entre muitos outros.

Cada plataforma tem suas peculiaridades, existem bibliotecas que funcionam para todas, e existem também, as específicas para cada ambiente. No caso deste trabalho, será utilizado a plataforma Java SE para ambiente Desktop.

1.1.1 Bibliotecas

Em computação, uma biblioteca é um conjunto de funções genéricas armazenadas em um arquivo, com o intuito de auxiliar e facilitar no desenvolvimento de um *software*, permitindo o aumento de reutilização de códigos, facilitar a manutenção, tendo potencial a reduzir custos. As bibliotecas não se prendem a uma aplicação específica, ou seja, em qualquer aplicação, elas podem ser usadas.

Muitas linguagens disponibilizam várias bibliotecas padrões. Na linguagem C por exemplo, contém a biblioteca *math.h*, na qual contém várias funções que permitem realizar operações matemáticas mais complexas como logaritmos, potências, funções trigonométricas, entre outras. Na linguagem Java, são disponibilizadas várias bibliotecas também, para manipulação de *strings*, data e hora, comunicação com banco de dados, interface gráfica, entre várias outras.

Além das bibliotecas que já são padrões e nativas da linguagem, existem também muitas que são criadas por outros programadores, e disponibilizadas na internet, muitas são bem populares. No Java por exemplo, existe a biblioteca *XStream* para manipular arquivos XML, a biblioteca *Gson* para manipular arquivos JSON, entre outras.

Este trabalho, foi focado, as bibliotecas de interface gráfica do Java, ao todo, são 3

nativas. A primeira lançada foi a AWT (*Abstract Window Toolkit*, em português, Kit de ferramentas de janelas). Após um tempo, foi lançado a biblioteca Swing, com melhorias e mais escolha de componentes. E no ano de 2012 o JavaFX foi lançado como uma biblioteca nativa também e com várias novidades. Nas seções seguintes serão explicadas com mais detalhes as bibliotecas que foram escolhidas para o desenvolvimento deste trabalho.

1.1.1.1 AWT

A biblioteca AWT foi a primeira de interface gráfica do Java, de acordo com Campos e Nunes (2006) os componentes da AWT se pareciam com os componentes GUI (*Graphical User Interface*, em português, Interface Gráfica do Usuário) nativos da plataforma em que um programa Java executava. Por exemplo, se o sistema operacional do computador for Windows, as janelas e componentes eram semelhantes do mesmo e assim acontecia em outros sistemas operacionais, às vezes a forma de interação de um componente poderia mudar de um sistema operacional para o outro.

A estrutura e alguns componentes da biblioteca AWT são mostradas na Figura 1.

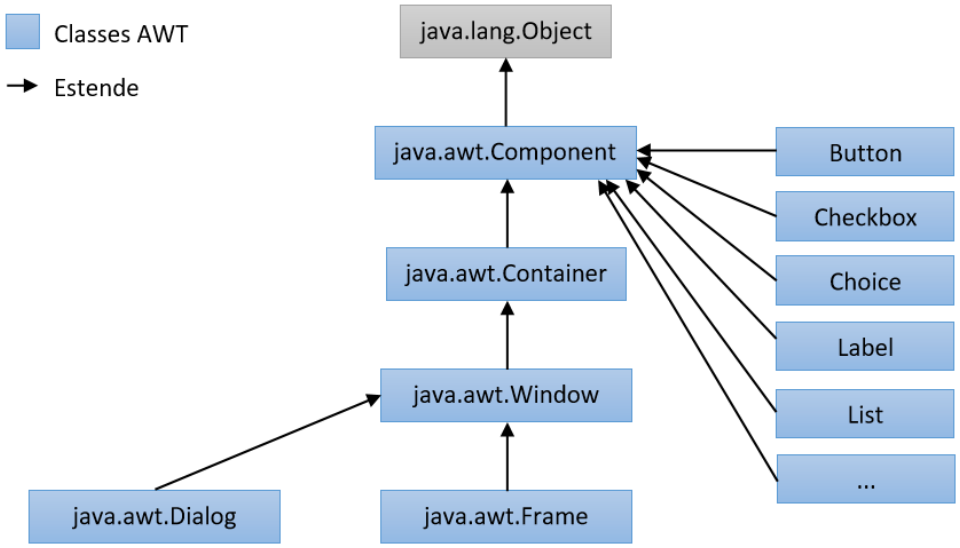


Figura 1 - Estrutura e alguns componentes da biblioteca AWT.
Fonte: Próprio autor.

A Figura 1 foi criada a partir da documentação do Java SE 8 disponível no portal da Oracle, no qual ao acessar a documentação de cada classe, é mostrado toda sua hierarquia. Nesta figura, é mostrado resumidamente alguns componentes da biblioteca AWT e de quais

classes são estendidas. As setas indicam qual é a superclasse de uma classe, por exemplo, as classes *Button*, *Checkbox*, *Choice*, *Label* e *List* estendem de *java.awt.Component* na qual se estende de *java.lang.Object*. Algumas classes de *java.awt.Component* mostradas na Figura 1 tem as seguintes funções em uma interface:

- *Button* - Dispara um evento ao clique do mouse.
- *Checkbox* - Especifica uma opção que pode ou não ser selecionada.
- *Choice* - Apresenta um menu para escolhas.
- *Label* - Exibe textos não editáveis.
- *List* - Exibe uma lista de itens, na qual um item pode ser selecionado.

Segundo Deitel, P. e Deitel, H. (2016) os componentes AWT são pesados, porque eles contam com sistema de janelas do sistema operacional para determinar sua funcionalidade, aparência e comportamento. Apesar de parecer uma boa solução, a biblioteca AWT tinha muitas limitações pois dependia muito do sistema operacional no qual era executado. Estas limitações incentivou a criação de uma outra biblioteca que será explicada na próxima seção.

1.1.1.2 Swing

A biblioteca Swing foi adicionada à plataforma na versão do Java SE 1.2, com o objetivo de melhorar recursos e componentes para criar interfaces gráficas no Java.

Uma das melhorias que o Swing proporcionou, é que vários IDE's (*Integrated Development Environment*, em português, Ambiente de Desenvolvimento Integrado) começaram a fornecer ferramentas de design de interfaces gráficas para arrastar e soltar componentes em um layout e geram o código automaticamente com intuito de facilitar para o desenvolvedor a criar as telas de um *software*.

O Swing é compatível com o AWT, sendo possível usar componentes e recursos das duas bibliotecas no mesmo projeto.

Uma interface gráfica criada pelo Swing é única em qualquer sistema operacional que a executa, seja Windows, Linux ou qualquer outro, permanece o mesmo layout e estilos. Grande parte da complexidade da biblioteca Swing foi pelo fato da preocupação de portabilidade que a AWT não tinha. Um exemplo desta portabilidade, além da característica de ser único em qualquer sistema operacional é a questão de posicionamento de componentes na tela independente da resolução da tela.

A estrutura e alguns componentes da biblioteca Swing são mostradas na Figura 2.

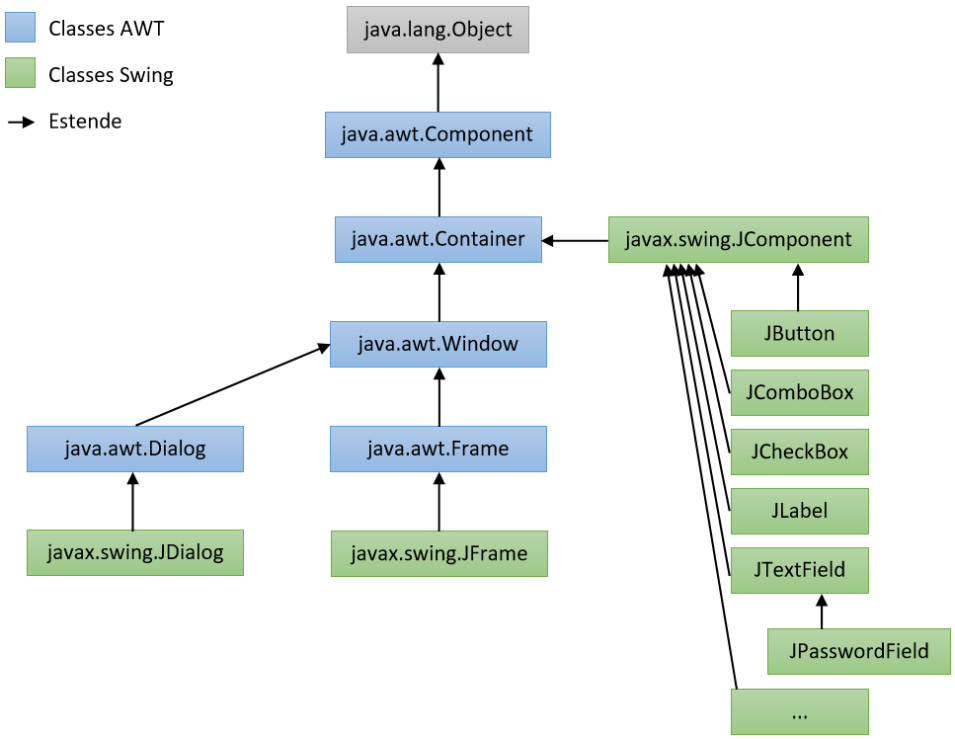


Figura 2 - Estrutura e alguns componentes da biblioteca Swing.
Fonte: Próprio autor.

A Figura 2, assim como a Figura 1, também foi criada a partir da documentação do Java SE 8 disponível no portal da Oracle. Esta figura mostra a estrutura e alguns componentes da biblioteca Swing, na qual é possível ver que a base da biblioteca Swing é uma parte da biblioteca AWT, pois os componentes de interface gráfica Swing estendem da superclasse *javax.swing.JComponent*, e esta superclasse estende de *java.awt.Container*, na qual pertence à biblioteca AWT. Algumas classes de *java.swing.JComponent* mostradas na Figura 2 tem as seguintes funções em uma interface:

- *JButton* - Dispara um evento ao clique do mouse.
- *JComboBox* - Um componente que combina um botão ou um campo editável e uma lista suspensa.
- *JCheckbox* - Especifica uma opção que pode ou não ser selecionada.
- *JLabel* - Exibe texto e/ou ícones não editáveis.
- *JTextField* - Recebe uma entrada do usuário.
- *JPasswordField* – Recebe uma entrada do usuário, mas não mostra os caracteres originais. Geralmente é usado para digitar senhas.

De acordo com Deitel, P. e Deitel, H. (2016) a maioria dos componentes Swing são

componentes leves, pois eles são escritos, manipulados e exibidos completamente no JVM, com isto, a biblioteca Swing requer um pouco mais desempenho do processador, em compensação, possui uma estética de layout mais agradável. Mas como a base do Swing é o AWT, ele também tem alguns componentes pesados (que depende do sistema operacional para determinar sua funcionalidade, aparência e comportamento).

Campos e Nunes (2006) dizem que desde quando a biblioteca Swing foi adicionada à plataforma, a mesma foi mantida como a principal tecnologia de interfaces gráficas no Java. O Swing agora está no modo de manutenção, pois a Oracle parou de desenvolver para esta biblioteca e fornecerá apenas correções de bugs daqui para a frente, no entanto, continuará a ser parte do Java e ainda é amplamente utilizada.

1.1.1.3 JavaFX

O JavaFX foi criado com o intuito de agregar mais valor às interfaces de aplicações Java e facilitar a forma de desenvolver as mesmas. Ela foi anunciada em maio de 2007, na época, como uma plataforma a parte, tendo sua primeira versão lançada no dia 4 de dezembro de 2008.

Em suas versões iniciais, era usada uma linguagem chamada JavaFX *Script* para o desenvolvimento de aplicações. Após a versão 2.0, a tecnologia passou a ser uma biblioteca inclusa no Java SE, podendo usufruir do código nativo para o desenvolvimento das aplicações, e a partir da versão 7 *Update* 6 do Java, passou a ser distribuída nas instalações da linguagem, deixando de ser necessário baixar e instalar como uma solução externa.

De acordo com Oliveira (2014) ela fornece uma API de elementos gráficos e multimídia (imagens, animação, áudio e vídeo), além de permitir estilizar a aplicação usando folha de estilos com CSS (*Cascading Style Sheets*, em português, Folhas de Estilo em Cascata), permitindo um completo controle sobre a aparência e comportamento da interface.

A estrutura e alguns componentes da biblioteca JavaFX são mostradas na Figura 3.

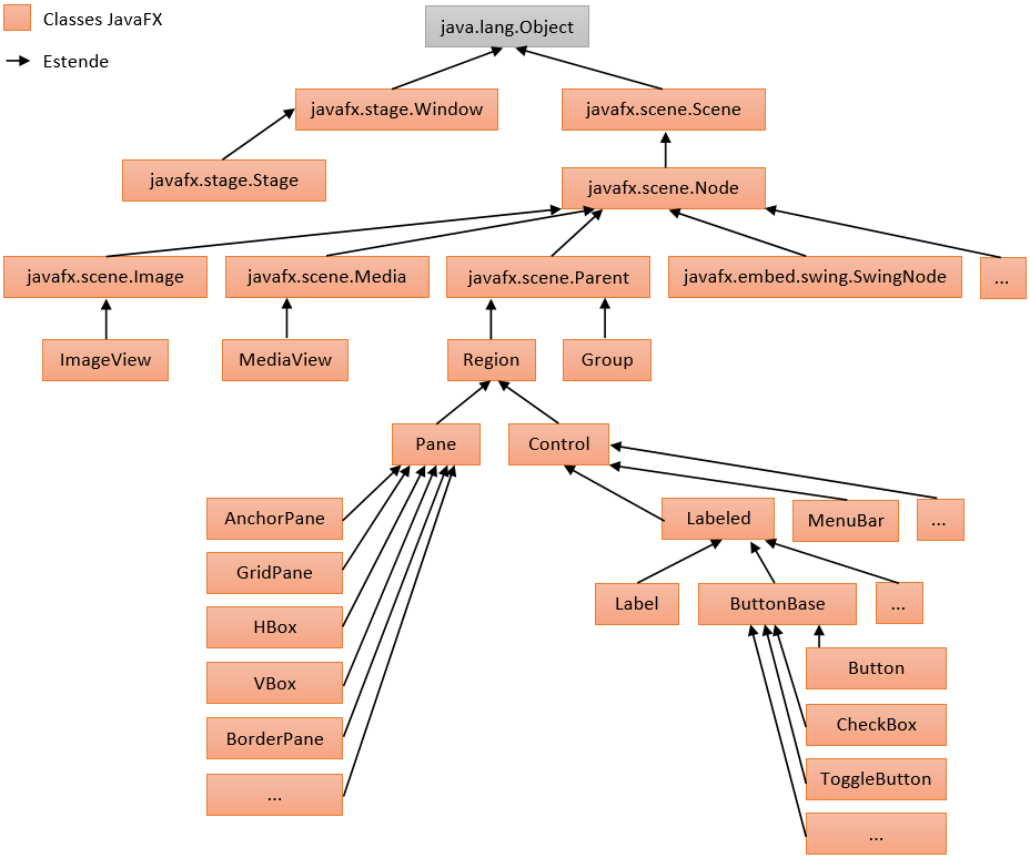


Figura 3 - Estrutura e alguns componentes da biblioteca JavaFX.
Fonte: Próprio autor.

A Figura 3, assim como as Figuras 1 e 2, também foi criada a partir da documentação do Java SE 8 disponível no portal da Oracle. Semelhante às bibliotecas mostradas nas seções anteriores, a biblioteca JavaFX também se estende da classe do Java *java.lang.Object*, mas sua estrutura é totalmente diferente das demais.

A janela de uma interface gráfica no JavaFX é representada como uma *Stage* (*javafx.stage.Stage*), a cena mostrada dentro da janela é representada como uma *Scene* (*javafx.scene.Scene*) e todos componentes dentro de uma cena são representados como *Node* (*javafx.scene.Node*), como é mostrado na Figura 4.

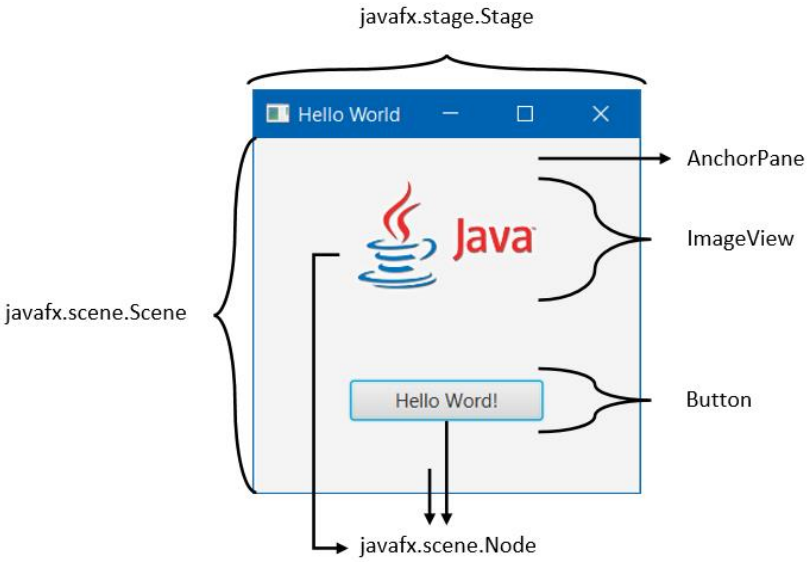


Figura 4 - Componentes de uma janela pela biblioteca JavaFX.
Fonte: Próprio autor.

A Figura 4 foi criada para complementar a Figura 3. Nesta Figura é mostrado que a *Scene* (em português, Cena) é chamada dentro de uma *Stage* (em português, Etapa)

Dentro da cena vários *Nodes* (em português, Nós) podem ser chamados. Como visto na Figura 3, os nós podem ser vários tipos de componentes no JavaFX, no caso da Figura 4 os nós são: *AnchorPane*, *ImageView* e um *Button*.

Na Figura 3 mostra que existem várias classes que estendem de *Nodes*, algumas delas são:

- *ImageView* - permite o uso de imagens na interface.
- *MediaView* - permite o uso de vídeos e áudios na interface.
- *Parent* - é a superclasse de grande parte dos componentes do JavaFX, a partir dela existem as classes de layout como *AnchorPane*, GridPane, *Vbox*, *Hbox* entre várias outras. Existem também as classes de controle, que permitem o uso de ações e eventos na interface, algumas delas são: *Label*, *Button*, *MenuBar*, *ComboBox*, *ListView* e *TableView*.
- *SwingNode* - permite o uso de componentes da biblioteca *Swing* na interface.

A bilblioteca JavaFX possui mais componentes do que os destacados aqui, e de acordo com Oliveira (2014), ela permite a organização de código, manutenção rápida e qualidade gráfica.

1.1.1.3.1 Scene Builder

O Scene Builder é uma ferramenta que foi feita em JavaFX e que permite a criação de interfaces gráficas para *softwares* JavaFX multiplataforma, sem precisar de codificação. Foi criada pela Oracle, é gratuita e de código aberto. Hoje, a companhia Gluon disponibiliza em seu portal a versão mais recente e todo o suporte necessário.

A ferramenta JavaFX Scene Builder pode ser usada com vários IDE's Java, incluindo NetBeans, Eclipse e IntelliJ IDEA.

De acordo com Oliveira (2014) no JavaFX Scene Builder cria-se uma interface gráfica arrastando e soltando componentes da biblioteca para uma área de design e, então, modificando e modelando a mesma sem escrever nenhuma linha de código. Os recursos de edição e visualização dinâmicas do JavaFX Scene Builder permitem que se tenha uma visualização da interface gráfica à medida que a cria e modifica, sem compilar e executar o aplicativo. Além de permitir usar CSS também, para alterar toda a aparência e o comportamento da interface.

Durante a criação de uma interface no Scene Builder, a medida em que se adiciona, exclui ou edita um componente na interface, a ferramenta gera automaticamente um código XML (*Extensible Markup Language*, em português, Linguagem de Marcação Extensível) específico do JavaFX chamado FXML (*FX Markup Language*), no qual é responsável por apresentar toda a parte de interface do *software* separada da parte de lógica (código Java), permitindo uma organização na codificação do projeto de *software*. Como o código é gerado automaticamente, e todas as funcionalidades e configurações da interface pode-se fazer pela ferramenta, não há necessidade do desenvolvedor ou designer conhecer a linguagem XML.

A ferramenta JavaFX Scene Builder permite também fazer toda a configuração de componentes da interface para ser usada na parte de lógica, como nome de identificação e vários eventos que podem ser acionados ao clique do mouse, ao apertar uma tecla, ou um toque no caso de aplicações mobile. Neste trabalho a ferramenta foi usada apenas para interfaces Desktop.

1.2 QUALIDADE DE SOFTWARE

De acordo com Pressman (2011) a necessidade da qualidade de *software* começou quando os mesmos passaram a se tornar cada vez mais integrados em várias atividades na vida das pessoas. Pressman (2011) destaca também que várias empresas reconheciam que bilhões de dólares por ano estavam sendo desperdiçados em *softwares* que não apresentavam as características e funcionalidades prometidas.

Um relatório chamado *Software Fail Watch* feito recentemente por uma empresa internacional que fornece soluções de garantia de qualidade de *software*, a Tricentis, revelou que em 548 *softwares* analisados, cerca de 4,4 bilhões de pessoas foram afetadas por falhas de softwares em 2016, gerando perdas de quase 1,1 trilhão de dólares em todos envolvidos. Nesta pesquisa, foram identificados 3 tipos principais de falhas de *software*:

- Falhas de *software*: quando algumas funcionalidades do *software* não funcionam como projetado.
- Falhas de usabilidade: falhas de design que diminui a usabilidade do *software*.
- Vulnerabilidade de segurança: falhas de segurança que atacantes podem explorar para alterar o comportamento de um *software*.

A revista ComputerWorld também publicou:

> Software de má qualidade está em praticamente todas as organizações que usam computadores, provocando horas de trabalho perdidas durante o tempo em que a máquina fica parada, dados perdidos ou corrompidos, oportunidades de vendas perdidas, custos de suporte e manutenção de TI elevados e baixa satisfação do cliente.
> (Hildreth, 2005)

Segundo Pressman (2011) a definição de qualidade de *software* não é tão simples quanto se imagina. São vários fatores que levam um *software* a ser de qualidade, como o projeto de desenvolvimento de *software*, as ferramentas de desenvolvimento, motivação da equipe e tempo disponibilizado para o desenvolvimento do projeto. O ideal é que o *software* atenda às necessidades do cliente, execute de forma precisa e confiável e gere valor para todos os envolvidos.

> Qualidade de projeto refere-se às características que os projetistas especificam para um produto. A qualidade dos materiais, as tolerâncias e as especificações de desempenho, todos são fatores que contribuem para a qualidade de um projeto.
> (Pressman, 2011, p. 359)

Muitos problemas e imprevistos podem ocorrer em um projeto de *software*, por isso é muito importante gerenciar riscos que podem afetar o cronograma ou a qualidade do projeto. Pressman (2011) montou um *checklist* com 7 tipos de riscos considerados genéricos que são importantes serem verificados em um projeto de desenvolvimento de *software*:

- Tamanho do produto: riscos associados ao tamanho geral do *software* a ser criado e modificado.
- Impacto de negócio: riscos associados a restrições impostas pela gerência ou pelo mercado.
- Características do cliente: são riscos associados à sofisticação dos clientes e habilidade do desenvolvedor em se comunicar com os interessados a tempo.
- Definição do processo: riscos associados ao grau em que a gestão de qualidade foi definida e é seguida pela organização de desenvolvimento.
- Ambiente de desenvolvimento: riscos associados à disponibilidade e qualidade das ferramentas a serem usadas para criar o produto.
- Tecnologia a ser criada: riscos associados à complexidade do sistema a ser criado e com a "novidade" da tecnologia que está embutida no sistema.
- Quantidade de pessoas e experiência: riscos associados à experiência técnica em geral e de projeto dos engenheiros de *software* que farão o trabalho.

Os riscos de ambiente de desenvolvimento estão diretamente ligados à importância da escolha das ferramentas para o desenvolvimento de *software*, pois as mesmas influenciam em grande parte da estrutura do *software*, desde a qualidade do código-fonte à interface gráfica do mesmo. Isso demostra a importância de realizar escolhas conscientes para que não prejudique a qualidade do software e não gere prejuízos.

1.2.1 Usabilidade de software

No início da era dos computadores, na maioria dos casos os usuários eram os próprios desenvolvedores e não havia a preocupação de uma boa usabilidade nos *softwares*, até que surgiram os computadores pessoais no final da década de 70 e as pessoas queriam aprender a usar os computadores, a partir daí, surgiu também a necessidade de criar *softwares* que seriam fáceis de usar, intuitivos, eficientes e bonitos. E foi a partir desta necessidade que começou a surgir o termo usabilidade.

A interface gráfica de um *software* está diretamente ligada a usabilidade, pois é a parte em que é responsável pela interação do usuário com o *software*.

Segundo Sommerville (2011), as propriedades emergentes de um sistema são as características do sistema como um todo, e não de seus componentes. Elas incluem propriedades como desempenho, confiabilidade, usabilidade, segurança e proteção. O sucesso ou fracasso de um sistema é, muitas vezes, dependente dessas propriedades emergentes.

> A interface do usuário é discutivelmente o elemento mais importante de um produto ou sistema computacional. Se a interface for mal projetada, a capacidade de o usuário aproveitar todo o poder computacional e conteúdo de informações de uma aplicação pode ser seriamente afetada. Na realidade, uma interface fraca pode fazer com que uma aplicação, em outros aspectos bem projetada e solidamente implementada, falhe. (PRESSMAN, 2011, p. 313).

De acordo com Nielsen (1993) a usabilidade tem como objetivo elaborar interfaces capazes de permitir uma interação fácil, agradável, com eficácia e eficiência ao usuário. Deve-se permitir ao usuário pleno controle do ambiente sem se tornar um obstáculo durante a interação.

> Fornecer aos diferentes aplicativos componentes de interface com o usuário consistentes e intuitivos permite que os usuários se familiarizem com um novo aplicativo, para que possam aprendê-lo e utilizá-lo mais rápida e produtivamente. (DEITEL, P. e DEITEL, H. 2016, p. 374).

Também de acordo com Nielsen (1993), a usabilidade pode ser dividida em cinco critérios:

- Intuitividade: o sistema deve apresentar facilidade de uso permitindo que, mesmo um usuário sem experiência, seja capaz de produzir algum trabalho satisfatoriamente.
- Eficiência: o sistema deve ser eficiente em seu desempenho apresentando um alto nível de produtividade.
- Memorização: suas telas devem apresentar facilidade de memorização permitindo que usuários ocasionais consigam utilizá-lo mesmo depois de um longo intervalo de tempo.
- Erro: a quantidade de erros apresentados pelo sistema deve ser o mais reduzido possível, além disso, eles devem apresentar soluções simples e rápidas mesmo para usuários iniciantes. Erros graves ou sem solução não podem ocorrer.
- Satisfação: o sistema deve agradar ao usuário, sejam eles iniciantes ou avançados, permitindo uma interação agradável.

1.2.1.1 Heurísticas de Nielsen

Considerado como o "Rei da Usabilidade" pela revista Internet, Jakob Nielsen é um cientista da computação com Ph.D. em IHC (Interação humano-computador) que criou as heurísticas de Nielsen em 1994, com o intuito de padronizar e nortear a forma de análise de usabilidade em um *software* para melhoria da qualidade do mesmo.

Jakob Nielsen fazia estudos em análise heurística desde 1990 ao lado de Rolf Molich. Após análises práticas e mais estudos, ele revisou as heurísticas e publicou sua versão original em 1994. Depois desta, as heurísticas têm sido alteradas e expandidas, para cobrir novas tecnologias e ambientes computacionais. Ao todo, são 10 heurísticas:

1. Visibilidade do estado do sistema: o sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de feedback adequado dentro de um prazo razoável.
2. Correspondência entre o sistema e o mundo real: o sistema deve falar o idioma dos usuários, com palavras, frases e conceitos familiares para o usuário, em vez de termos orientados para o sistema. Todas as nomenclaturas devem ser contextualizadas e coerentes com o modelo mental do usuário. Isso também é aplicado à ícones e

imagens ilustrativas.

3. Controle e liberdade do usuário: Permitir liberdade ao usuário das decisões e ações que podem ser tomadas na aplicação, exceto a regras de negócio ou que interfira outra funcionalidade. Permitir também desfazer ou refazer alguma ação no sistema e retornar ao ponto anterior quando o usuário estiver perdido ou em situações inesperadas. Não forçar o usuário a fazer algo e também não tomar a decisão por ele.
4. Consistência e padrões: Manter uma consistência e padrão visual (texto, cor, desenho do elemento, som e etc). Nunca identificar uma mesma ação com ícones ou fontes diferentes. Tratar coisas similares da mesma maneira, facilitando a identificação do usuário para usar o sistema.
5. Prevenção de erros: Não deixar o usuário errar sem explicar previamente o motivo do erro. Ter interfaces com características que não permite o usuário errar, como máscaras (data, telefone, CEP, entre outras) nos campos de cadastro e mensagens explicativas caso o usuário tenta fazer algo que está contra as regras de negócio da aplicação.
6. Reconhecimento em vez de memorização: Minimizar a carga de memória do usuário, tornando objetos, ações e opções visíveis. O usuário não deve ter que lembrar as informações de uma parte do diálogo para outra. As instruções para o uso do sistema devem ser visíveis ou facilmente recuperáveis sempre que apropriado.
7. Flexibilidade e eficiência de uso: O *software* deve ser ágil para usuários avançados e fácil de utilizar para usuários leigos. O uso de atalhos de teclados, preenchimento automático a partir de dados anteriores e máscaras de campos são exemplos de itens que aprimoram a eficiência do sistema com flexibilidade.
8. Design estético e minimalista: O *software* não deve usar desnecessariamente excessos de cores e elementos visuais que podem confundir o usuário. Deve-se “dialogar” de forma simples e direta, com um layout mais limpo, com diálogos naturais, de fácil entendimento e aparecerem só em momentos necessários. Deve ter uma interface que permite uma interação agradável e satisfatória para o usuário.
9. Recuperação de erros: O *software* deve ter mensagens de erro claras, com textos simples e diretos, não intimidar o usuário e sim ajudá-lo a reconhecer, diagnosticar e conduzir à possíveis soluções do erro.
10. Ajuda e documentação: O *software* deve ser intuitivo e claro para o usuário no qual, quase não seja necessário o uso de documentações complementares. Mesmo assim é

preciso fornecer documentações (manuais e explicações sobre o uso) ao alcance do usuário.

As heurísticas são fatores importantes que os desenvolvedores devem levar em conta no desenvolvimento de um *software*, pois um atendendo a todas elas, poderá causar grande impacto para aceitação do mesmo.

1.2.2 Norma de qualidade de software

De acordo com Guerra e Colombo (2009) a qualidade de *software* constitui uma área cuja demanda está crescendo significativamente, pois os usuários exigem cada vez mais eficiência, eficácia, dentre outras características de qualidade importantes para um produto tão especial como o *software*. Paralelamente à demanda do mercado, existe um movimento nacional e internacional, no sentido de estabelecer normas na área de Engenharia de *Software*, como é o caso da ISO (*International Organization for Standardization*, em português, Organização Internacional de Normalização), no Subcomitê de Engenharia de *Software*, e da ABNT (Associação Brasileira de Normas Técnicas).

A ISO é uma organização mundial não governamental fundada em 1947 que tem como principal atividade a elaboração de padrões para especificações e métodos de trabalho em diversas áreas. De acordo com Guerra e Colombo (2009) o principal objetivo da ISO é o desenvolvimento de padrões mundiais, com vistas a facilitar o intercâmbio internacional de produtos e serviços e a criar uma cooperação intelectual, científica, econômica e técnica, tendo participação de mais de 130 países.

A IEC (*International Electrotechnical Commission*, em português, Comissão Eletrotécnica Internacional) também é uma organização mundial fundada em 1906, que de acordo com Guerra e Colombo (2009) publica normas internacionais relacionadas a eletricidade, eletrônica e áreas semelhantes, tendo participação de mais de 50 países.

Segundo Guerra e Colombo (2009) a ISO, juntamente com a IEC, elaborou um conjunto de normas que tratam, especificamente, da atual padronização mundial para a qualidade dos produtos de *software*.

As normas recomendam modelos de qualidade e processos de medição de qualidade que são importantes serem seguidos para criação e avaliação de um *software*. Segundo Guerra

e Colombo (2009) podem ser utilizadas por desenvolvedores, adquirentes e avaliadores. O uso destes modelos e processos influenciam muito para garantir a qualidade de um *software*, no qual também deve-se levar em conta a escolha de boas ferramentas para atender os requisitos do mesmo.

Os processos de medição de qualidade estão ligados às métricas de qualidade de *software* que de acordo com a ISO/IEC 9126-1 (2001), uma métrica é a composição de procedimentos para a definição de escalas e métodos para medidas. Neste trabalho serão utilizadas métricas para análise de código-fonte e análise de usabilidade, que serão explicadas nas seções posteriores.

1.2.2.1 Norma ISO/IEC 25010

A norma ISO/IEC 25010, publicada em 2011, foi desenvolvida pelo projeto SQuaRE (*Software Quality Requirements and Evaluation*, em português, Requisitos e Avaliação da Qualidade de *Software*) com o intuito de reestruturar a norma ISO/IEC 9126 que também se tratava de qualidade de *software*.

De acordo com Guerra e Colombo (2009) a necessidade da reestruturação foi verificada quando especialistas do mundo todo concordaram que faltava clareza na utilização nas normas de qualidade de produto. Assim, a série 9126 levou a uma lista de melhorias que foram implementadas na nova série 25010.

As características definidas nesta norma, podem ser aplicadas em qualquer tipo de produto de *software*, nas quais são mostradas na figura a seguir.

Figura 5 - Características e subcaracterísticas de qualidade de produto de *software*.
Fonte: Adaptação SQuaRE ISO/IEC 25010.

A Figura 5 mostra a estrutura da norma SQuaRE ISO/IEC 25010 de qualidade de produto de *software* divididas em 8 características, as quais são por sua vez subdivididas em subcaracterísticas. De acordo com esta norma, as características são:

- Adequação funcional - Capacidade do produto de *software* de prover funções que atendam às necessidades explícitas e implícitas, quando o *software* estiver sendo utilizado sob condições especificadas.
- Eficiência de desempenho - Representa o desempenho em relação à quantidade de recursos utilizados sob condições especificadas.
- Compatibilidade - Capacidade de dois ou mais sistemas ou componentes para trocar informações e/ou executar suas funções necessárias quando compartilham o mesmo ambiente de hardware ou *software*.
- Usabilidade - Capacidade do produto de *software* a ser entendido, aprendido, usado e atraente para o usuário, quando usado sob condições especificadas.

- Confiabilidade - Capacidade de um sistema ou componente para executar as funções especificadas, quando é usado em determinadas condições e período de tempo.
- Segurança - Capacidade de proteger informações e dados para que pessoas ou sistemas não autorizados não possam ler ou modificá-los.
- Manutenibilidade - Capacidade do produto de *software* a ser modificado efetivamente e de forma eficiente, devido a necessidades evolutivas, corretivas ou perfeitas.
- Portabilidade - Capacidade do produto ou componente a ser transferido de forma eficaz e eficiente de um hardware, *software*, operacional ou ambiente de utilização para outro.

Guerra e Colombo (2009) dizem que, as características de qualidade e suas medidas associadas podem ser utilizadas tanto para avaliar um produto de *software* quanto para definir requisitos de qualidade. A nova série (25010) estabelece critérios para a especificação dos requisitos de qualidade de produto de *software*, para medição e avaliação.

1.2.2.1.1 Métricas de usabilidade

A primeira norma que se tratou de métricas de usabilidade foi a ISO/IEC 9126, no ano de 2003, na qual se tratava de qualidade de produto de *software*. Como foi mostrado na seção 1.2.2.1 em 2011 foi criada a norma SQuaRE ISO/IEC 25010 que atualmente substitui a norma ISO/IEC 9126.

A parte de usabilidade nesta norma mais recente contém 6 métricas que são mostradas a seguir:

- Capacidade de reconhecer sua adequação - capacidade de produto que permite ao usuário entender se o software é adequado para suas necessidades.
- Capacidade de aprendizagem - capacidade de produto que permite ao usuário aprender sua aplicação.
- Capacidade de usar - capacidade de produto que permite aos usuários operar e facilmente controlada.
- Proteção contra erros do usuário - capacidade do sistema para proteger os usuários de cometer erros.
- Estética da Interface de usuário - capacidade de agradar e satisfazer interação do

usuário.

- Acessibilidade - capacidade de produto que permite que ele seja usado por usuários com determinadas características e deficiências.

1.2.3 Análise de código-fonte

A análise de código-fonte tem relação direta com a qualidade interna de *software*, pois de acordo com Meirelles (2013) dentre as inúmeras características que fazem um bom *software*, várias delas podem ser percebidas no código-fonte, e algumas são exclusivas dele. Por exemplo, código compilado pode ser analisado, mas características como organização e legibilidade são perdidas. Mesmo uma bateria de testes com ótima cobertura, só apresenta informação sobre o funcionamento atual, não refletindo manutenibilidade, modularidade, flexibilidade e simplicidade. Nesse contexto, as métricas de código-fonte complementam as outras abordagens de monitoramento da qualidade do *software*.

Segundo a ISO/IEC 25023 (2011) as métricas de código-fonte são conhecidas como métricas internas, nas quais aferem a qualidade interna do *software* por meio da avaliação de estruturas internas que compõem o *software* em estágio de desenvolvimento.

Os valores de qualidade interna podem ser utilizados para prever os valores de qualidade externa que o produto vai apresentar. É importante perceber essa relação entre qualidade interna e externa para perceber que qualidade interna impacta fortemente na qualidade externa de um produto, e então observar a importância de começar esforços de medição e acompanhamento da qualidade interna desde o início do projeto. A avaliação de qualidade de código-fonte no início do desenvolvimento pode ser de grande valia para auxiliar equipes inexperientes durante as etapas seguintes do desenvolvimento de *software* (MEIRELLES, 2013).

1.2.3.1 Métricas de código-fonte

As métricas de código-fonte foram propostas por alguns autores desde a década de 70 quando os primeiros conceitos de engenharia de *software* surgiram, algumas foram

melhoradas por outros autores, e outras são usadas até nos dias de hoje sem nenhuma mudança ou melhoria.

Existem vários tipos de métricas de código-fonte, nas quais são separadas por categorias, as principais são: métricas de tamanho, complexidade, acoplamento e coesão.

As métricas de tamanho medem o tamanho do *software*, nas quais são:

- LOC (*Lines of Code* – Número de linhas de código) foi uma das primeiras métricas utilizadas para medir o tamanho de um *software*, na qual são contadas apenas linhas executáveis, ou seja, são excluídos linhas em branco e comentários. Para efetuar comparações entre sistemas usando LOC, é necessário que ambos tenham sido feitos na mesma linguagem de programação e que o estilo esteja normalizado (JONES, 1991).
- AMLOC (*Average Method LOC* – Média do número de linhas de código por método) indica se o código está bem distribuído entre os métodos. Quanto maior, mais pesados são os métodos. É preferível ter muitas operações pequenas e de fácil entendimento do que poucas operações grandes e complexas (MEIRELLES, 2013).
- MMLOC (*Max Method LOC* – Número de linhas de código do maior método da classe) assim como o AMLOC, este também é derivado do LOC.

Características como manutenibilidade, flexibilidade, compreensão e qualidade do código-fonte, em geral, deve-se levar em consideração não só as métricas de tamanho descritas, mas também métricas de complexidade, que são explicadas a seguir:

- NOA (*Number of Attributes* – Número de atributos) calcula o número de atributos de uma classe. Meirelles (2013) diz que uma classe com muitos atributos pode indicar que ela tem muitas responsabilidades e apresentar pouca coesão, pode também estar tratando de vários assuntos diferentes.
- NOM (*Number of Methods* – Número de métodos) é usado para medir o tamanho das classes em termos das suas operações implementadas. Essa métrica é usada para ajudar a identificar o potencial de reúso de uma classe. Em geral, as classes com um grande número de métodos são mais difíceis de serem reutilizadas, pois elas são propensas a serem menos coesas (LORENZ E KIDD, 1994).
- NPA (*Number of Public Attributes* – Número de atributos públicos) mede o encapsulamento. Os atributos de uma classe devem servir apenas às funcionalidades da própria classe. Portanto, boas práticas de programação recomendam que os

atributos de uma classe devem ser manipulados através dos métodos de acesso (BECK, 1997).

- NPM (*Number of Public Methods* – Número de métodos públicos) representa o tamanho da "interface" de uma classe (MEIRELLES, 2013). Métodos estão diretamente relacionados às operações previstas na respectiva classe. Altos valores para essa métrica indicam que uma classe tem muitos métodos e, provavelmente, muitas responsabilidades (BECK, 1997).
- ANPM (*Average Number of Parameters per Method* – Média do Número de Parâmetros por Método) calcula a média de parâmetros dos métodos de uma classe. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado, mas um número alto de parâmetros pode indicar que um método pode ter mais responsabilidades, ou seja, mais de uma função (BANSIYA E DAVI, 1997).
- DIT (*Depth of Inheritance Tree* – Profundidade da Árvore de Herança) é o número de superclasses ou classes ancestrais da classe sendo analisada. São contabilizadas apenas as superclasses do sistema, ou seja, as classes de bibliotecas não são contabilizadas. Nos casos onde herança múltipla é permitida, considera-se o maior caminho da classe até uma das raízes da hierarquia. Quanto maior for o valor DIT, maior é o número de atributos e métodos herdados, portanto maior é a complexidade (MEIRELLES, 2013).
- NOC (*Number of Children* – Número de filhos) número total de filhos de uma classe (ROSENBERG E HYATT, 1997).
- RFC (*Response for a Class* – Respostas para uma Classe) número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe (SHARBLE; COHEN, 1993). De acordo com Pereira Júnior (2015) este valor é calculado pelo somatório de todos os métodos daquela classe, e todos os métodos chamados diretamente por essa classe. Um valor alto de RFC pode indicar baixa coesão e alto acoplamento.
- ACCM (*Average Cyclomatic Complexity per Method* – Média da Complexidade Ciclomática por método) mede a complexidade de um método ou programa (MCCABE, 1976). Essa métrica pode ser representada através de um grafo de fluxo de controle, onde os nós representam uma ou mais instruções sequenciais e os arcos orientados indicam o sentido do fluxo de controle entre várias instruções. Quanto maior o valor para esta métrica, mais complexo é o método ou classe analisado (a).

  Acoplamento é uma medida de como uma classe está ligada a outras classes no

software. Altos valores de acoplamento indicam uma maior dificuldade para alterar uma classe do sistema, pois uma mudança em uma classe pode ter um impacto em todas as outras classes que são acopladas a ela. Em outras palavras, se o acoplamento é alto, o *software* tende a ser menos flexível, mais difícil de se adaptar, modificar e entender (MEIRELLES, 2013). Métricas que medem o acoplamento são descritas a seguir:

- ACC (*Afferent Connections per Class* – É o número total de classes externas de um pacote que dependem de classes de dentro deste pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-in* da classe, medindo o número de classes das quais a classe é derivada e, assim, valores elevados indicam uso excessivo de herança múltipla (MCCABE; DREYER; WATSON, 1994).
- CBO (*Coupling Between Objects* – Acoplamento entre objetos) é o número total de classes dentro de um pacote que dependem de classes externas ao pacote. Quando calculada no nível da classe, essa medida também é conhecida como *Fan-out* da classe (CHIDAMBER; KEMERER, 1994).
- COF (*Coupling Factor* – Fator de acoplamento) indica o quão interconectado é o *software*. Um *software* fortemente conectado apresenta maior COF, indicando um baixo grau de independência entre os módulos, alta complexidade e difíceis entendimento e manutenção (MEIRELLES, 2013).

Coesão mede a diversidade de responsabilidades que uma classe implementa. De acordo com Meirelles (2013) altos valores de coesão indicam se o "foco" de uma classe está em um único aspecto do sistema. Enquanto uma baixa coesão indica que a classe trata de diferentes aspectos. Assim, uma classe deve ser coesa. As métricas comumente usadas para análise do grau de coesão são:

- LCOM4 (*Lack of Cohesion in Methods* – Ausência de coesão em métodos) foi originalmente proposta por Chidamber e Kemerer (1994) como LCOM, mas não teve muita aceitabilidade. Após críticas e sugestões a métrica foi revisada por Hitz e Montazeri (1995). Para calcular LCOM4 de um módulo, é necessário construir um grafo não-orientado em que os nós são os métodos e atributos de uma classe. Para cada método, deve haver uma aresta entre ele e um outro método ou variável que ele usa. O valor da LCOM4 é o número de componentes fracamente conectados nesse grafo. Valores grandes para LCOM indicam uma baixa coesão, enquanto valores baixos indicam uma alta coesão (MEIRELLES, 2013).

- SC (*Structural Complexity* – Complexidade estrutural) quanto mais complexo for um *software*, mais difícil será alterá-lo e evoluí-lo. Meirelles (2013) considerou esta métrica como o produto de acoplamento (CBO) e coesão (LCOM4), pois de acordo com seus estudos é aconselhável manter baixo acoplamento e uma alta coesão das classes, além de que estas duas métricas são indicadores essenciais de complexidade estrutural.

Estas métricas podem ser usadas para analisar o código-fonte de um *software* em qualquer estado de desenvolvimento. Existem algumas ferramentas que auxiliam no cálculo destas métricas e visualização de resultado das mesmas. Uma delas é a Analizo, que será explicada na próxima seção.

1.2.3.2 Analizo

A Analizo é um conjunto de ferramentas de código livre, multi-linguagem e extensível para análise e visualização de código-fonte. Foi criado por um grupo de brasileiros com o intuito de analisar e visualizar projetos de *softwares* livres. A Analizo faz a coleta e análise das métricas estáticas de código-fonte compilável e não compilável (com erros de sintaxe e conter bibliotecas que não estão mais disponíveis).

De acordo com Meirelles (2013) atualmente a Analizo realiza a análise de código-fonte escrito em C, C++ e Java. Ela calcula métricas tanto em nível de projeto, que são calculadas a partir de um valor que agrega dados de todos os arquivos do projeto, quanto métricas no nível de classe, que são calculados individualmente para cada classe. Grande parte das métricas calculadas pela Analizo foram descritas na seção 1.2.3.1.

Segundo Oliveira Filho (2013) a Analizo possui um comando para o processamento em lote de vários projetos, produzindo um arquivo CSV (*Comma-separated values*, em português, Valores separados por vírgulas) com os dados das métricas para cada projeto, bem como um resumo geral das métricas no nível de projeto. Estes arquivos podem ser facilmente importados em ferramentas de análise estatística ou em planilhas. A Analizo também pode processar repositórios *Git* e *Subversion*, gerando um arquivo CSV com valores de métricas para cada revisão em que o código-fonte foi alterado.

2 METODOLOGIA

O presente trabalho teve como objetivo analisar a usabilidade e código-fonte em projetos de *software*, usando as bibliotecas Swing e JavaFX do Java em ambiente desktop. O intuito desta análise é verificar qual das bibliotecas traz mais qualidade de usabilidade e código-fonte a um projeto de *software*. Com esta finalidade, foram realizadas pesquisas bibliográficas para conhecer as bibliotecas e como fazer uma análise sobre elas.

Para a análise de código-fonte, foi utilizada uma ferramenta que auxiliou muito para a coleta e análise das métricas de código-fonte.

Para realizar a análise de usabilidade, foi feita uma pesquisa com profissionais relacionados ao desenvolvimento de *software* e estudantes de computação, a fim de conhecer suas experiências e opiniões sobre o assunto, a partir de um questionário (Apêndice 1).

Nesta seção será explicado em detalhes como foram desenvolvidos todos os passos para a realização deste trabalho, incluindo o desenvolvimento dos *softwares*, questionário e a forma de como foi coletado todos os dados para fazer as análises.

2.1 ESCOLHA DAS BIBLIOTECAS

Para conhecer detalhadamente as bibliotecas de interface gráfica do Java, foram feitas pesquisas no portal da Oracle, na qual encontram-se as documentações das mesmas. Nos livros de Deitel, P. e Deitel, H. e de Oliveira contém informações relevantes também.

Como explicado na seção 1.1.1, o Java tem 3 bibliotecas padrões de interface gráfica. A escolha destas para este trabalho foi realizada por uma ferramenta chamada Google Trends (em português, Google Tendências) que mostra os termos de pesquisa mais buscados no site de busca do Google. Nela é possível comparar alguns termos de pesquisa para analisar qual é o mais popular. Através dela, pôde-se ver a popularidade de pesquisa das bibliotecas ao longo do tempo.

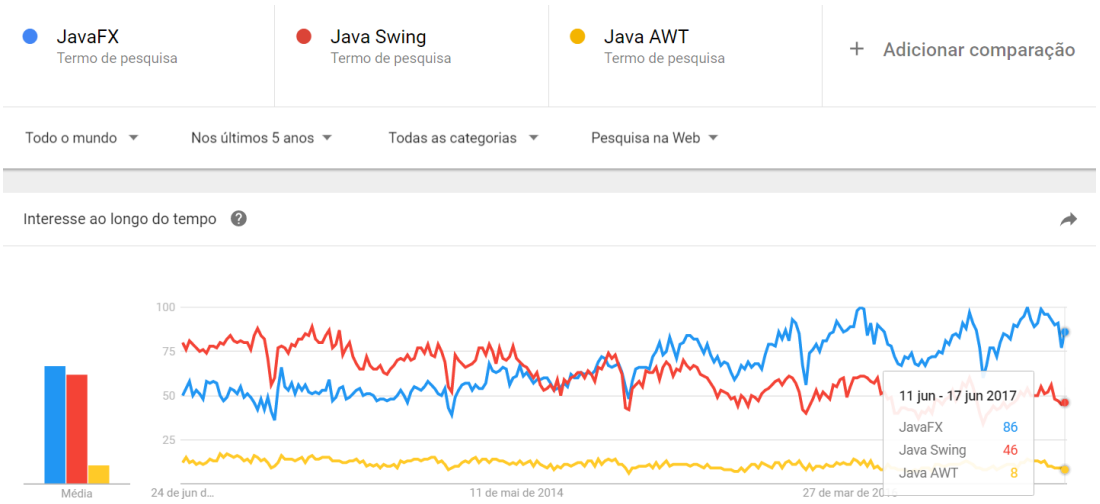


Gráfico 1 - Pesquisas feitas sobre Bibliotecas de Interface Gráfica do Java.
Fonte: Google Trends (2017).

O Gráfico 1 mostra dados coletados no site do Google Trends no dia 18 de junho de 2017, referente a popularidade de pesquisa das bibliotecas de interface gráfica do Java. Foram comparados 3 termos de pesquisa: JavaFX, Java Swing e Java AWT. A partir do Gráfico 1 é possível ver que as duas mais populares é a Swing e JavaFX, nas quais foram escolhidas para este trabalho.

2.2 ESCOLHA DA NORMA, HEURÍSTICAS E MÉTRICAS

A norma SQuaRE ISO/IEC 25010 foi escolhida por ser a mais recente e completa na área de qualidade de *software*, como foi mostrado na seção 1.2.2.1. As métricas escolhidas desta norma foram a de usabilidade que são mostradas na seção 1.2.2.1.1. Além destas métricas foi escolhido também as heurísticas de Nielsen, mostradas na seção 1.2.1.1, para enriquecer ainda mais a análise de usabilidade deste trabalho.

Para a análise de código-fonte foram escolhidas as métricas de tamanho, complexidade, acoplamento e coesão, mostradas na seção 1.2.3.1. Estas métricas foram criadas por diversos autores, muitas, desde a década de 70 e são amplamente utilizadas até hoje para este tipo de análise.

A análise de usabilidade foi feita pois as bibliotecas Swing e JavaFX tem o intuito de fornecer a interface gráfica para o usuário, e está associada com a usabilidade. Já a análise de código-fonte foi feita pois o mesmo causa grande impacto para a qualidade externa de um

software, e apesar das duas bibliotecas serem da linguagem Java, ambas podem apresentar grandes diferenças no desenvolvimento de *software*, nas quais podem ser mostradas através desta análise.

2.3 SOFTWARES PARA ANÁLISE

Para realizar as análises de usabilidade e código-fonte das duas bibliotecas, foram desenvolvidos dois *softwares* Java Desktop para serem avaliados, ambos têm as mesmas funcionalidades, componentes de interface e objetivos, mas foram desenvolvidos de forma diferente, pois em um foi utilizada a biblioteca de interface gráfica Swing e o outro o JavaFX.

Os *softwares* têm os mesmos componentes de interface, por exemplo, no mesmo lugar que tem um botão de cadastro em uma janela de um, no outro conteve o botão também, da mesma forma serão os formulários, campos de cadastro e os processos para realizar um. Mas como dito anteriormente, foram utilizadas bibliotecas diferentes.

Os *softwares* não têm comunicação com banco de dados, sendo assim, quando o usuário encerrar os mesmos, todos os dados serão apagados, pois serão disponíveis só enquanto estiverem em execução.

Os *softwares* são simuladores de aluguel de filmes, nos quais permitem o usuário escolher vários filmes, se cadastrar e alugar filmes. O nome dos mesmos foi definido como Locflix.

Os dados dos filmes presentes nos *softwares* foram coletados a partir de um portal brasileiro de filmes e séries chamado Adoro Cinema, no qual foi possível pegar vários dados confiáveis de filmes para serem adicionados nos *softwares*.

Para o desenvolvimento dos *softwares* foi definida uma estrutura de interface gráfica que foi usada como base para ambos, que será mostrada na próxima seção.

2.3.1 Definição da estrutura

A estrutura dos *softwares* foi pensada para ser chamativa e de fácil interação para os usuários. Na definição das funcionalidades e componentes, foi levado em consideração que a

interface gráfica pudesse atender todas as características de usabilidade contidas na norma ISO/IEC 25010 mencionadas na seção 1.2.2.1.1 e nas heurísticas de Nielsen mencionadas na seção 1.2.1.1, para que seja possível realizar a avaliação completa de usabilidade.

Para definir a estrutura dos *softwares*, foram feitos *mockups* das telas para ter noção de todas funcionalidades e componentes que os mesmos teriam. Um *mockup* é a representação visual de um *software*, como se fosse um rascunho da interface, na qual mostra as funcionalidades e estrutura de forma estática.

Para criar estes *mockups* foi utilizado uma ferramenta chamada *Balsamiq Mockups*, que proporciona todos componentes necessários para a criação de *mockups* em ambientes Desktop, Web e *Mobile*. Neste trabalho foi utilizada a versão 3 desta ferramenta, com licença gratuita por 30 dias.

Para a criação do Locflix, foram definidas 4 telas, cada uma teve um *mockup* desenhado usando a ferramenta *Balsamiq Mockups*, nos quais são mostrados a seguir:

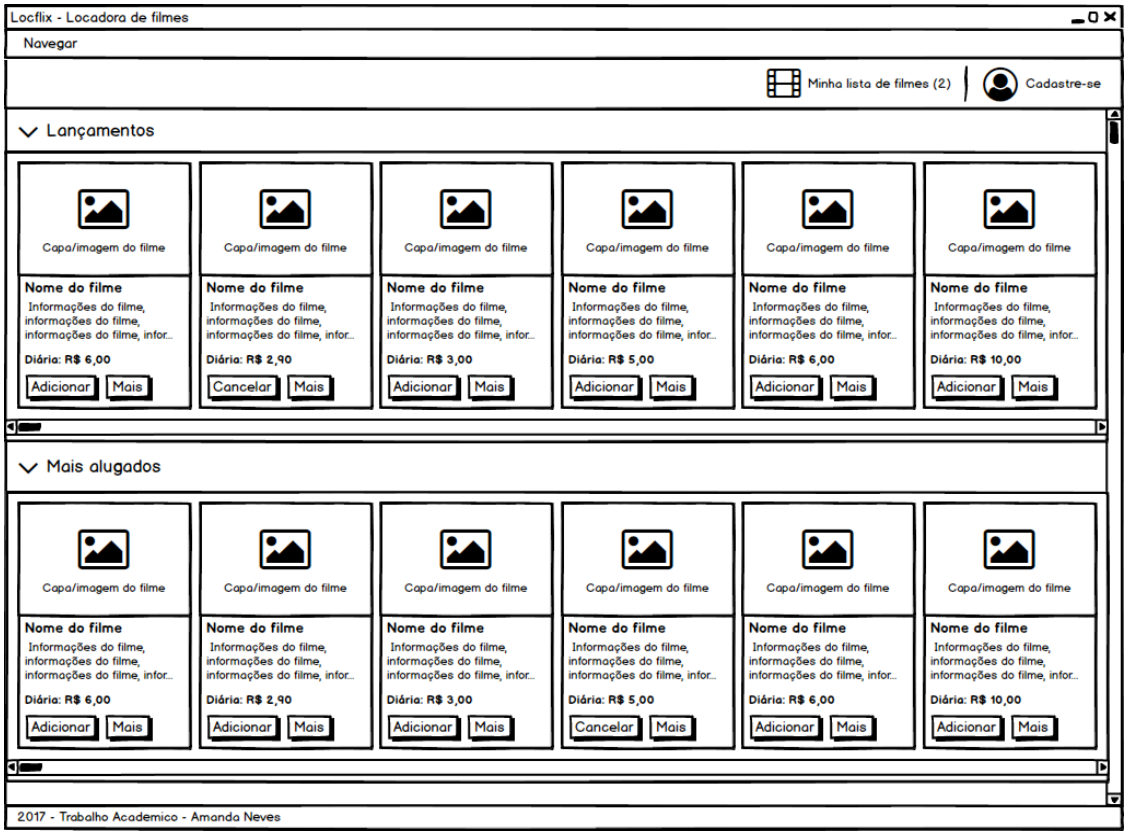


Figura 6 - *Mockup* da tela inicial e principal do Locflix.
Fonte: Próprio autor.

A Figura 6 mostra o *mockup* da tela inicial. Esta é a principal tela do Locflix, pois a partir dela é possível ter uma visão de todas as funcionalidades, através dos botões ‘Minha

lista de filmes' e 'Cadastre-se'. É possível também, ver todos os filmes disponíveis separados por categorias e permite a interação com o usuário através de teclas de atalho, botões e menus.

Cada filme mostrado na interface tem uma opção de adicionar ou cancelar um filme à uma lista através do botão 'Adicionar' que ao clicar neste, sua legenda é alterada para 'Cancelar' caso o usuário deseja desfazer a ação.

Ao lado do botão 'Adicionar/Cancelar' fica o botão 'Mais', no qual ao clicar neste, abre uma tela com várias informações sobre o filme, na qual é mostrada através da Figura 7.

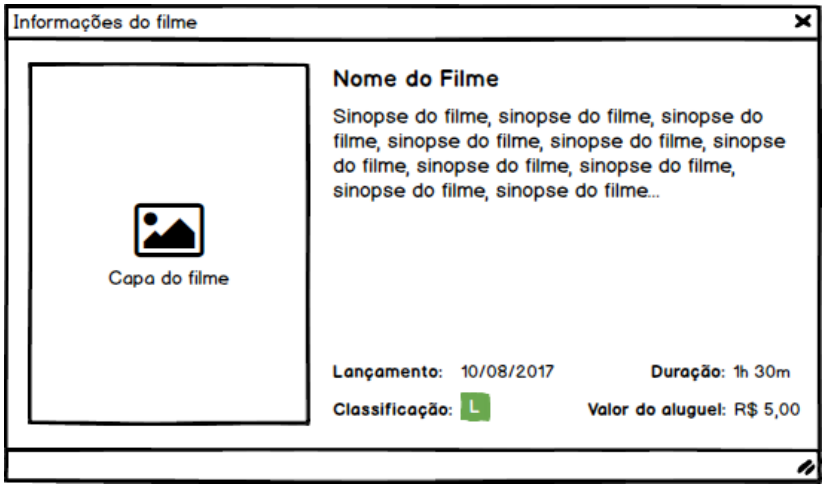


Figura 7 - *Mockup* da tela de informações do filme do Locflix.
Fonte: Próprio autor.

A Figura 7 mostra o *mockup* da tela Informações do filme, que é responsável por mostrar mais detalhes sobre o filme, como capa, sinopse completa, data de lançamento, classificação, duração e valor do aluguel que é um valor simbólico.

Para verificar os filmes que forem escolhidos, existe um botão na tela inicial chamado 'Minha lista de filmes' que ao clicar neste, abrirá uma tela com uma lista de todos os filmes escolhidos, mostrada na Figura 8.

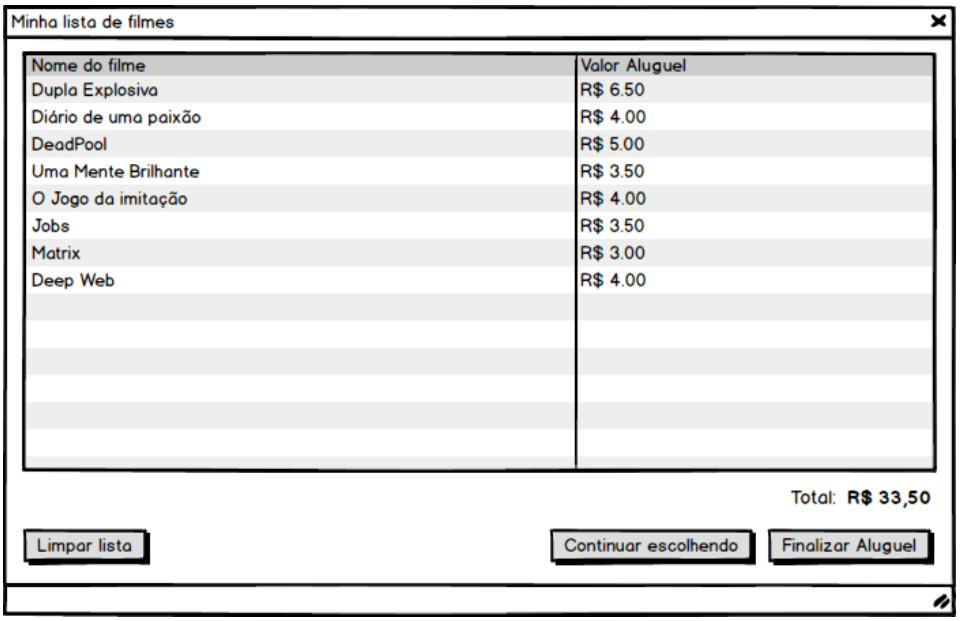


Figura 8 - *Mockup* da tela de lista de filmes escolhidos do Locflix.
Fonte: Próprio autor.

Na Figura 8 mostra o *mockup* da tela de lista de filmes, que é responsável por mostrar todos filmes escolhidos pelo usuário e o valor total dos mesmos. As opções disponíveis nesta tela é limpar a lista, continuar escolhendo e finalizar aluguel através dos botões.

Para finalizar um aluguel, é preciso que o usuário se cadastre no Locflix. Para se cadastrar existe um botão na tela inicial chamado ‘Cadastre-se’, ao clicar neste, abrirá uma tela com um formulário para ser preenchido, mostrado na Figura 9.

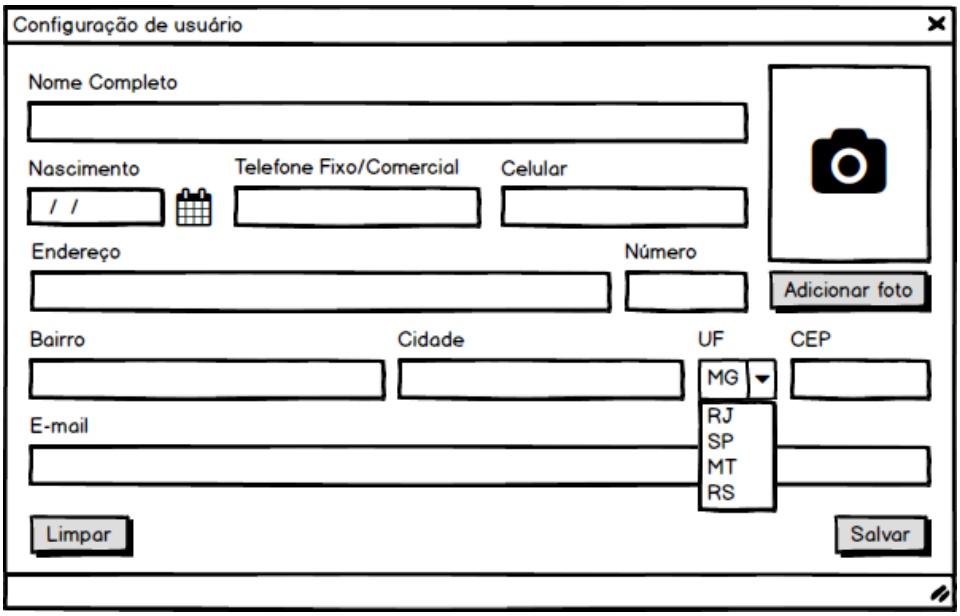


Figura 9 - *Mockup* da tela de configuração de usuário do Locflix.
Fonte: Próprio autor.

Na Figura 9 mostra o *mockup* da tela de configuração de usuário do Locflix, na qual tem vários campos para o usuário preencher, alguns sendo obrigatórios e outros não. Nesta tela têm as opções de limpar os dados e salvar os dados.

Em algumas situações, durante a interação do usuário, os *softwares* retornam mensagens para o usuário, por exemplo: caso o usuário deixar um campo obrigatório em branco, e tentar salvar os dados, aparecerá uma mensagem e indicar qual campo está faltando a ser preenchido. Outra situação também é quando o usuário tentar finalizar o aluguel de filmes sem se cadastrar no Locflix, aparecerá uma mensagem explicando ao usuário que só pode ser confirmado o aluguel ao se cadastrar, e indica o botão no qual ele pode clicar e se cadastrar.

A cada mensagem de erro ou alerta no Locflix, a primeira frase indica o que aconteceu para o usuário e logo em seguida uma possível solução para que o mesmo esteja ciente do que aconteceu e saber o que fazer para terminar uma ação.

Para mais detalhes sobre a estrutura dos *softwares,* foram criados e disponibilizados os manuais dos mesmos, que estão em Apêndice 2 e Apêndice 3.

2.3.2 Desenvolvimento

O desenvolvimento dos *softwares* foi baseado na estrutura que foi explicada na seção 2.3.1 que apesar de ser a mesma estrutura para fazer com as duas bibliotecas de interface gráfica Swing e JavaFX, o desenvolvimento de cada um foi diferente, pois as mesmas fornecem componentes e ferramentas diferentes para o mesmo.

Para o desenvolvimento foram utilizadas várias ferramentas que serão descritas na seção a seguir.

2.3.2.1 Ambiente de desenvolvimento

No decorrer do desenvolvimento dos *softwares*, foi preciso utilizar algumas ferramentas para auxiliar no mesmo:

- JDK (*Java Development Kit*, em português, Kit de Desenvolvimento do Java) –

ferramenta gratuita disponibilizada pela Oracle, que contém todos os pacotes necessários para desenvolver e executar aplicações na linguagem Java. Nela contém todas as bibliotecas padrões, o compilador da linguagem, o JRE (*Java Runtime Environment*, em português, Ambiente de Execução do Java) que permite a execução de aplicações Java e contém o JVM (*Java Virtual Machine*, em português, Máquina Virtual do Java) na qual as aplicações são emuladas. Foi instalada a versão 8 *Update* 131.

- *IntelliJ Idea Ultimate* – IDE (*Integrated Development Environment*, em português, Ambiente de Desenvolvimento Integrado) que foi utilizado para desenvolver o Locflix usando a biblioteca JavaFX. Foi instalada a versão 2017.1.4 sob licença de estudante da *JetBrains* (empresa criadora do *IntelliJ Idea*).
- *Netbeans* – IDE criada pela Oracle, foi utilizado para desenvolver o Locflix usando a biblioteca Swing. Foi instalada a versão 8.2, é uma ferramenta gratuita.
- *Scene Builder* – como foi explicada na seção 1.1.1.3.1 é uma ferramenta para criar interfaces gráficas com o JavaFX de maneira rápida e conveniente, que pode ser usada separadamente ou com qualquer um dos IDEs Java. Foi instalada a versão 8.3.0, é uma ferramenta gratuita.

Para a instalação destas ferramentas e desenvolvimento dos *softwares* foi utilizado um notebook com as seguintes especificações técnicas:

- Modelo Samsung Expert X51.
- Processador Intel core i7 7500U.
- 8 GB de memória RAM.
- Sistema Operacional Windows 10 Home de 64 bits.

Este notebook atende aos pré-requisitos das ferramentas a serem instaladas.

2.3.2.2 Como os *softwares* foram desenvolvidos

Para o desenvolvimento dos *softwares* foram utilizados 2 IDE's diferentes pois, o Locflix com a biblioteca JavaFX foi o primeiro a ser desenvolvido através do IDE *IntelliJ Idea*. Ao começar o desenvolvimento do Locflix com a biblioteca Swing foi percebido a

necessidade de utilizar o IDE *Netbeans*, pois este proporciona uma paleta de componentes do Swing mais completa e amigável que o *IntelliJ Idea*, o que facilitou para o desenvolvimento do Locflix Swing.

Os *softwares* foram desenvolvidos cuidadosamente com o intuito de trazer facilidade de uso para o usuário. Ao desenhar e programar todo o layout e estrutura, foram implantadas teclas de atalho e menus para acesso rápido à todas as funcionalidades, máscaras para digitação de telefone, data e CEP na configuração de usuário e mensagens para o usuário sempre que for preciso.

Após toda estrutura pronta dos dois *softwares*, foram definidas todas categorias de filmes que os mesmos iriam ter, nas quais são:

- Lançamentos (alguns filmes que esteve em cartaz nos cinemas no período entre agosto e setembro de 2017)
- Mais alugados (suposições)
- Ação
- Comédia
- Documentários
- Drama
- Ficção científica
- Romance
- Terror

Para cada categoria foram selecionados 10 filmes, que foram pesquisados pelo portal brasileiro de filmes e séries Adoro Cinema. Através deste portal foi possível adquirir todas informações e imagens necessárias para serem adicionados aos *softwares*. Todos os filmes foram armazenados através de listas estáticas internamente no código-fonte dos *softwares*, pois os mesmos não têm comunicação com nenhum banco de dados. Ao todo, foram adicionados 90 filmes.

Nas seções 2.3.2.2.1 e 2.3.2.2.2 será explicado com mais detalhes como foi o desenvolvimento em cada biblioteca.

2.3.2.2.1 JavaFX

O Locflix JavaFX foi o primeiro *software* a ser desenvolvido. Primeiramente foi criada toda a parte de interface gráfica do *software* com o Scene Builder. Para isso, foi criado um arquivo com extensão FXML para cada tela, no qual a partir do Scene Builder foi formado o layout e adicionado e posicionado todos componentes necessários para cada uma. Ainda no Scene Builder foram configurados identificadores para cada componente e eventos para os botões e menus.

Após a criação da interface gráfica pelo Scene Builder, foi programada a parte dos controladores de cada tela no *IntelliJ Idea* com classes Java, e também foi feita uma classe somente para a inserção dos filmes no *software*.

Na Figura 11 pode-se ver como ficou a tela inicial do Locflix JavaFX.

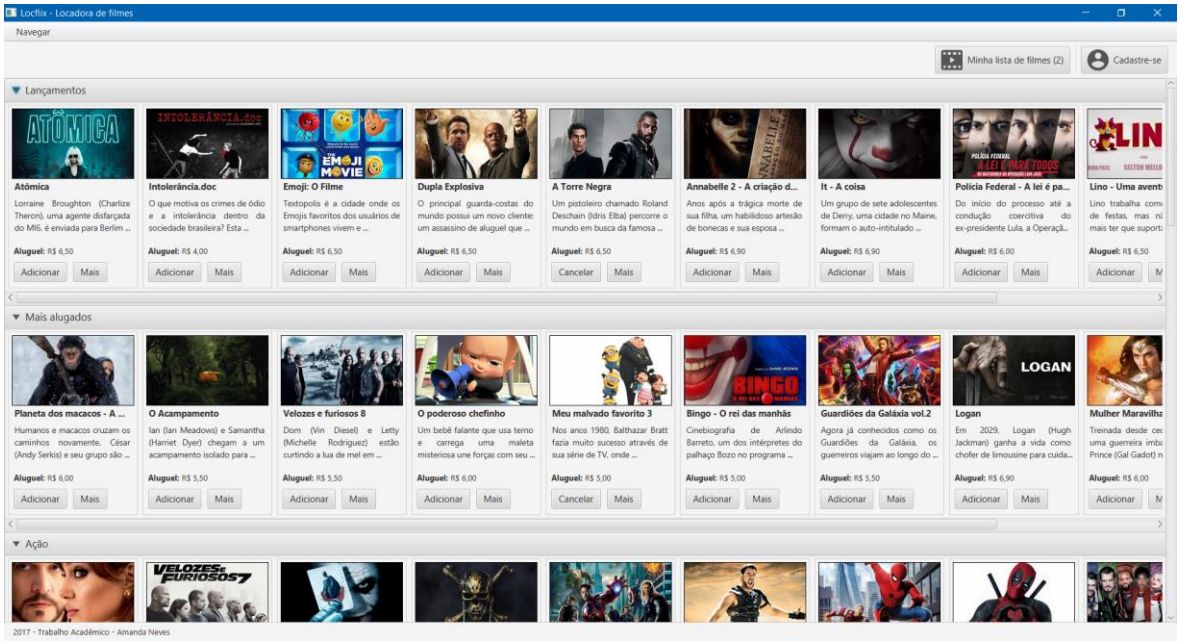


Figura 10 - Tela inicial do Locflix JavaFX.
Fonte: Próprio autor.

Neste *software* foi usado o design original da biblioteca JavaFX, não foi criado nenhum estilo CSS que a mesma suporta.

2.3.2.2.2 Swing

O Locflix Swing foi desenvolvido após o Locflix JavaFX. Assim como no JavaFX o primeiro passo foi criar toda a parte de interface gráfica do *software*. Com a biblioteca Swing tudo foi feito pelo *Netbeans*. Ao criar uma classe de controle no *Netbeans*, automaticamente é

criado um arquivo de extensão *form* com um formato de XML contendo a estrutura de todos os componentes de interface gráfica e uma classe Java com a parte de controlador dos componentes contendo os identificadores e posicionamentos dos mesmos.

Muitos componentes na biblioteca Swing não são posicionados diretamente no editor de interface gráfica como no Scene Builder, muitos posicionamentos são programados nos códigos-fontes nas classes Java. Um exemplo são os painéis de rolagem, nos quais todos foram inseridos e posicionados pelo código-fonte diretamente na classe Java e não no *form* que contém a estrutura dos componentes de interface gráfica.

Na Figura 10 pode-se ver como ficou a tela inicial do Locflix Swing.

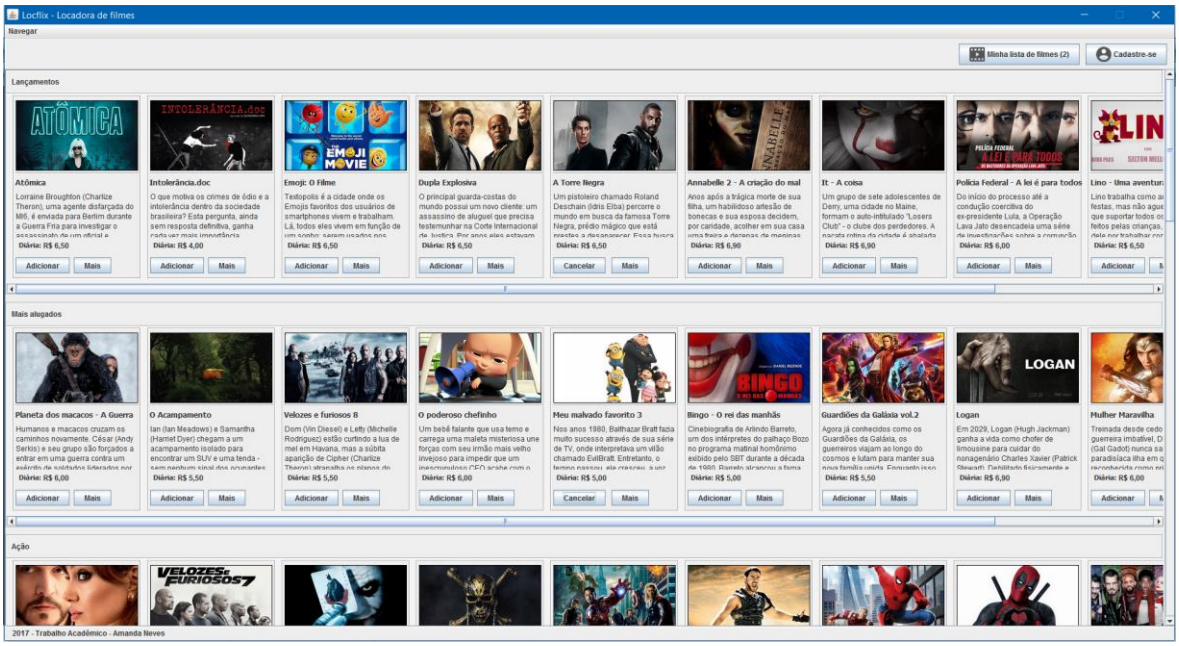


Figura 11 - Tela inicial do Locflix Swing.
Fonte: Próprio autor.

A biblioteca Swing tem alguns tipos de design para interface chamados: *Metal*, *Nimbus*, *Windows* e *Windows Classic*. Neste *software* foi usado o design que é sugerido pelo *Netbeans*, o *Metal*.

Para avaliar a usabilidade deste *software* (Locflix Swing) e do Locflix JavaFX foi feito um questionário que será explicado com mais detalhes na seção 2.4.

2.4 QUESTIONÁRIO

O questionário foi criado com o intuito de fazer uma pesquisa de usabilidade dos *softwares* criados com as bibliotecas de interface gráfica Swing e JavaFX, e fazer uma análise a partir dos resultados desta.

As perguntas do questionário foram divididas em dois grupos, um com perguntas para conhecer o perfil do participante, e o outro com perguntas para avaliar a usabilidade dos *softwares*.

Nas seções seguintes será explicado com mais detalhes sobre a criação e divulgação do questionário.

2.4.1 Público-alvo e divulgação

Foi definido que o público-alvo para responder o questionário e avaliar os dois *softwares* foram estudantes da área de Tecnologia da Informação e profissionais da área de desenvolvimento de *software*, pois desta forma, a probabilidade de se obter resultados mais confiáveis sobre a usabilidade de *softwares* é maior.

Para divulgar o questionário, foram coletados *e-mails* e contatos de alunos (a partir do 6º período, no qual tem a disciplina de Engenharia de *Software* tem o conceito de usabilidade em sua ementa) e ex-alunos do curso de Ciência da Computação das Faculdades Integradas de Caratinga. Foram coletados também *e-mails* e contatos de profissionais da área de desenvolvimento de *softwares* na cidade de Caratinga. Desta forma, quanto mais participantes especialistas avaliar a usabilidade dos *softwares*, mais qualidade terão os dados coletados para a análise.

Após a coleta de *e-mails* e contatos, foram compartilhados para todos, via correio eletrônico e mensagens por redes sociais, o link do questionário que contém todas as informações necessárias para acessar os *softwares* e fazer a avaliação dos mesmos.

O questionário também foi divulgado em um grupo brasileiro da rede social *Facebook* chamado Java, no qual se trata sobre a linguagem Java e contém um grande número de participantes ativos.

2.4.2 Disponibilização dos *softwares*

Para a disponibilização dos *softwares* (arquivos executáveis ou JARs) para o público-alvo fazer as avaliações, foi utilizada uma ferramenta do Google chamada Google *Drive*, que faz o armazenamento de arquivos em nuvem e é gratuita (até 15 GB).

No Google *Drive*, foi criada uma pasta, na qual foram colocados os dois *softwares*, Locflix Swing e Locflix JavaFX e os manuais dos mesmos. Nesta pasta também foi adicionado um arquivo explicativo, contendo os pré-requisitos para executar os *softwares* e uma breve explicação de como obter o JRE (*Java Runtime Environment*, em português, Ambiente de Execução do Java), caso o participante não tivesse este instalado em seu computador.

Os manuais dos *softwares* foram criados para auxiliar o participante no uso dos mesmos. Nestes, contém toda a estrutura dos *softwares* e o passo a passo de todas funcionalidades dos mesmos. É importante ressaltar que os manuais não são objetivo de estudo neste trabalho e foram criados para servir como ferramentas auxiliares e satisfazer a 10ª heurística de Nielsen, eles estão no Apêndice 2 e Apêndice 3 deste trabalho.

Para permitir que qualquer pessoa pudesse acessar a pasta, foi configurado no Google *Drive* o compartilhamento por link, no qual foi liberado um link, que a partir deste pôde acessar a pasta em qualquer local com acesso à internet. Ao acessar a pasta o participante poderia visualizar e fazer o download de todo o conteúdo da mesma.

Após a organização dos *softwares* para avaliação na pasta, foi criado o questionário que será explicado com mais detalhes na próxima seção.

2.4.3 Elaboração do questionário

Para a criação e disponibilização do questionário foi necessário o uso de uma ferramenta gratuita do Google chamada Google *Forms* (em Português, Google Formulários) que permite a criação de formulários e concede os resultados dos mesmos para análise. Além disso, esta ferramenta permite o acesso dos formulários de qualquer dispositivo que tenha acesso à internet.

Na primeira página do questionário contém uma mensagem explicativa, contendo o objetivo do mesmo, como responder, como acessar os *softwares* para avaliação e o link da pasta compartilhada no Google *Drive*.

Foram elaboradas 21 perguntas para o questionário, que foram divididas em dois grupos, um para coletar os dados sobre o perfil do participante e outro para coletar os dados da avaliação de usabilidade dos *softwares*.

Para o grupo perfil do participante foram elaboradas 6 perguntas, dentre elas são *e-mail*, nível de formação acadêmica, tempo de experiência de uso de dispositivos relacionados à tecnologia (*smartphones* ou computadores), tempo de experiência de trabalho na área e se o participante teve alguma experiência com a linguagem Java.

Para o grupo das perguntas de avaliação dos *softwares* foram elaboradas 15 perguntas, que teve como base as métricas de usabilidade da ISO/IEC 25010 citadas na seção 1.2.2.1.1 e as heurísticas de Nielsen citadas na seção 1.2.1.1. Algumas métricas da ISO/IEC 25010 se tratam das mesmas características de algumas heurísticas de Nielsen, estas foram unificadas para uma mesma pergunta. Um exemplo é a métrica de Aprendizagem da ISO/IEC 25010 que tem o mesmo objetivo de avaliação que a heurística ‘Reconhecimento em vez de memorização’ de Nielsen.

Para cada pergunta de avaliação de usabilidade foi usada uma escala, para especificar o nível de concordância do respondente a uma afirmação apresentada na questão, cada pergunta teve duas escalas, uma referindo-se ao *software* usando a biblioteca Swing e outra referindo-se ao *software* usando a biblioteca JavaFX, ou seja, cada pergunta teve 2 respostas.

As perguntas do questionário não foram enumeradas e a maioria delas foram configuradas para serem obrigatórias, no que só teria como o participante enviar as respostas caso estas (perguntas obrigatórias) estiverem respondidas. Foi configurado também para que cada participante enviasse somente uma resposta, para evitar duplicidade de respostas.

Após o participante responder e enviar todas as respostas, foi configurado para retornar uma mensagem de agradecimento e confirmação.

2.4.4 Coleta e tratamento de dados

O questionário ficou disponível a partir do dia 12 de outubro de 2017 até o dia 15 de novembro de 2017. Neste período foram coletadas 51 respostas.

As respostas coletadas pelo formulário foram armazenadas automaticamente em uma planilha criada pelo Google *Sheets* (em português, planilhas), na qual foi possível visualizar cada resposta de cada participante separadamente.

Para a tabulação dos dados a planilha foi exportada para edição no Microsoft Excel, onde os dados foram organizados de acordo com o perfil do participante, possibilitando a análise das respostas e a exibição dos resultados através de gráficos com a porcentagem das respostas, para proporcionar uma melhor compreensão dos resultados que serão apresentados na seção 3.1.

2.5 ANÁLISE DO CÓDIGO-FONTE

A análise do código-fonte foi feita com o intuito de fazer uma avaliação da qualidade do código-fonte dos *softwares* criados com as bibliotecas de interface gráfica Swing e JavaFX.

Para realizar esta análise, foram coletados os resultados das métricas de tamanho, complexidade, acoplamento e coesão, mostradas na seção 1.2.3.1, a partir da ferramenta Analizo.

Nas seções seguintes será explicado com mais detalhes como foi feita a análise de código-fonte.

2.5.1 Ambiente de análises

A análise do código-fonte foi feita por meio da ferramenta Analizo, explicada na seção 1.2.3.2, que funciona por meio de comandos, ela pode fazer a análise de uma classe somente ou de um projeto inteiro de *software*.

A ferramenta Analizo ainda não funciona no sistema operacional Windows. Em consequência disso, ela foi instalada em um computador com sistema operacional Debian, versão 9.2.1.

As pastas dos projetos Locflix Swing e Locflix JavaFX foram copiadas para este computador, e a partir disso, vários testes e análises foram feitas no código-fonte com os

comandos da Analizo. Os detalhes de como foram coletadas as métricas do código-fonte dos projetos Locflix Swing e Locflix JavaFX são mencionados na seção 2.5.2.

2.5.2 Coleta dos resultados das métricas de código-fonte

A análise de código-fonte do Locflix Swing e Locflix JavaFX foram feitas separadamente.

Mesmo sido criados em IDEs diferentes, os projetos Locflix Swing e Locflix JavaFX têm uma pasta em comum, chamada SRC (abreviatura de *sources*, em português, fontes), nas quais ficam todas as classes Java e os arquivos de interface gráfica (arquivos de extensão *form* no caso da biblioteca Swing e arquivos *fxml* no caso da biblioteca JavaFX) dos *softwares*. As análises foram feitas dentro da pasta SRC de cada projeto, na qual a Analizo só realiza a análise em classes Java, ignorando todos os outros arquivos.

A coleta de todas as métricas é feita a partir de um comando. Após a Analizo realizar a análise de todas as classes Java, é retornado uma lista com todas as métricas na tela. Nesta lista retornada pela Analizo, contém os resultados das métricas do projeto inteiro (pasta SRC onde contém as fontes) e também os resultados das métricas separadas por classe. Além da lista de métricas aparecerem na tela, também é possível exportá-la em um arquivo de formato yaml, no qual pode ser acessado posteriormente.

A partir disso pôde-se ter uma lista com todos os resultados das métricas do projeto Locflix Swing e outra lista com os resultados das métricas do Locflix JavaFX.

Após adquirir os resultados das métricas, foi criada uma planilha no Microsoft Excel, na qual foi possível separar as métricas por categorias e projetos (Locflix Swing e Locflix JavaFX) e a partir desta planilha, foram criados gráficos para proporcionar uma melhor compreensão dos dados e serem apresentados na seção 3.2.

3 RESULTADOS

Nesta sessão serão apresentados todos resultados obtidos durante o desenvolvimento da metodologia deste trabalho. Os resultados serão divididos em duas sessões: análise de usabilidade e análise de código-fonte, nas quais serão explicadas com detalhes como foram obtidos e analisados.

3.1 ANÁLISE DE USABILIDADE

A análise de usabilidade foi concluída por meio das respostas obtidas pelo questionário, um total de 51 pessoas responderam o mesmo.

As respostas do questionário serão apresentadas em 2 grupos, da mesma forma como no questionário, em seções, sendo elas: Perfil do participante e Perguntas específicas de usabilidade das bibliotecas.

3.1.1 Perfil do participante

Através das respostas sobre o perfil do participante, foi possível conhecer um pouco sobre as experiências dos respondentes em relação ao uso de dispositivos relacionados à tecnologia, atividades exercidas na área da computação e desenvolvimento de *software* com a linguagem Java e suas opiniões sobre a usabilidade em um *software*.

A 1ª pergunta teve como objetivo identificar qual o nível de formação acadêmica dos entrevistados. As respostas são apresentadas no gráfico a seguir.

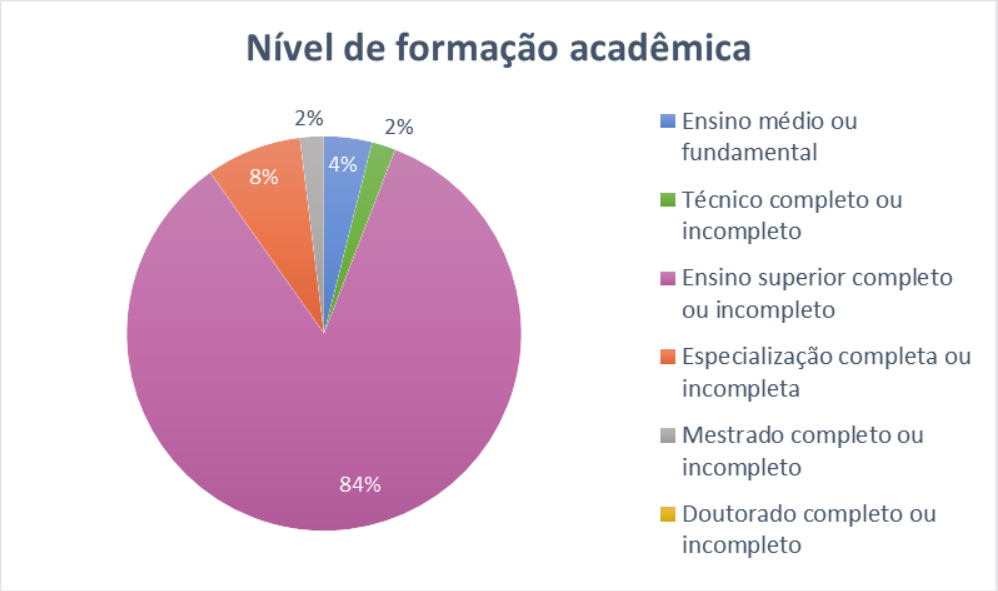


Gráfico 2 - Nível de formação acadêmica dos entrevistados.
Fonte: Próprio autor.

A partir do Gráfico 2, é possível ver que a maioria dos entrevistados fazem ou já fizeram um curso superior, sendo 87% deles. 8% dos entrevistados são de nível especialista ou fazem alguma especialização. 4% fizeram ou fazem o ensino médio, e os 4% restantes ficaram divididos em nível Técnico e de Mestrado.

A 2ª pergunta indica o tempo de experiência que o entrevistado tem com atividades relacionada à computação. As respostas são mostradas no gráfico a seguir.

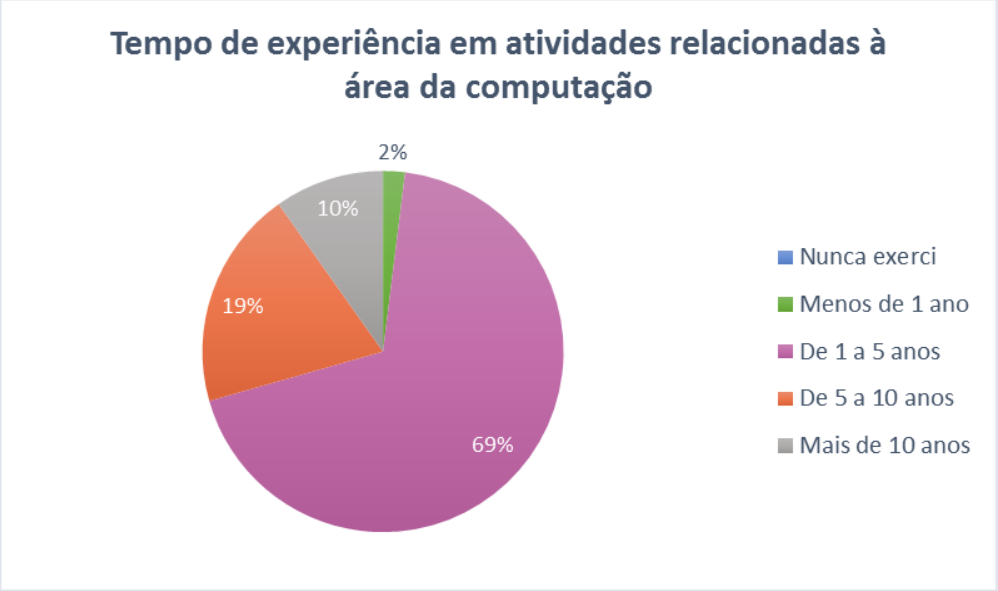


Gráfico 3 - Tempo de experiência em atividades relacionadas à área da computação dos entrevistados.
Fonte: Próprio autor.

O Gráfico 3, mostra que todos entrevistados tinham pelo menos 1 ano de experiência em atividades relacionadas à área da computação. A maioria deles (69%) demostraram ter 1 a 5 anos de experiência, 19% com 5 a 10 anos de experiência, 10% com mais de 10 anos de experiência e os outros 2% com menos de 1 ano de experiência.

A 3ª pergunta foi sobre o tempo que os entrevistados utilizam dispositivos relacionados à tecnologia. No gráfico seguinte é mostrado o resultado desta.

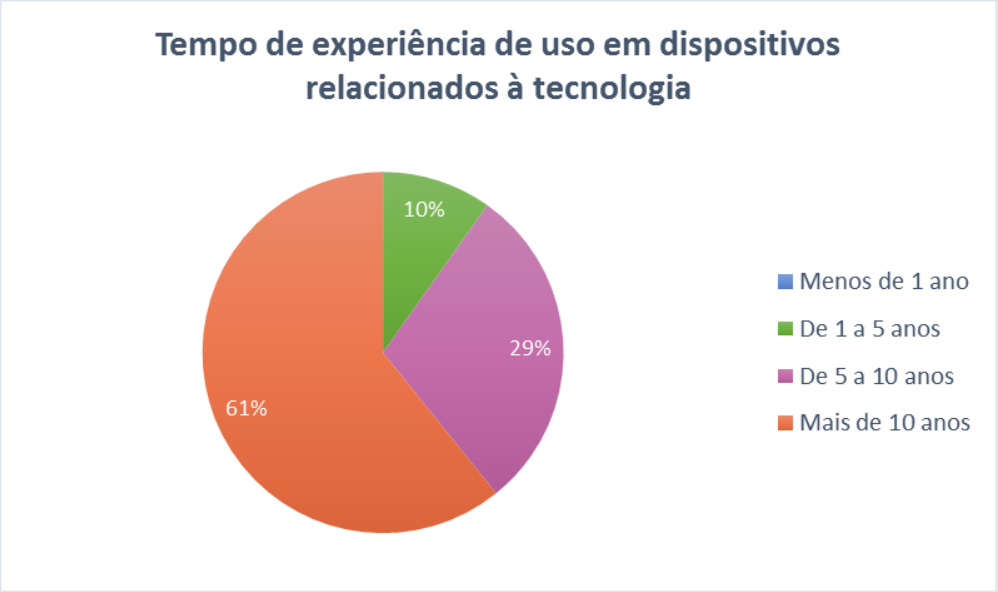


Gráfico 4 - Tempo de experiência de uso em dispositivos relacionados à tecnologia dos entrevistados.
Fonte: Próprio autor.

É mostrado no Gráfico 4 que mais de 60% dos entrevistados tem mais de 10 anos de experiência de uso em dispositivos relacionados à tecnologia, 29% tem de 5 a 10 anos de experiência e 10% tem de 1 a 5 anos de experiência.

Isso mostra que, grande parte deles já são adeptos ao uso de interfaces gráficas de *softwares*, seja em *smartphones*, computadores ou videogames, o que pode trazer uma certa relevância nas respostas de usabilidade das bibliotecas.

A 4ª pergunta tem o intuito de verificar se o respondente tem alguma experiência de desenvolvimento de *software* com a linguagem de programação Java. As respostas desta são apresentadas do Gráfico 5.

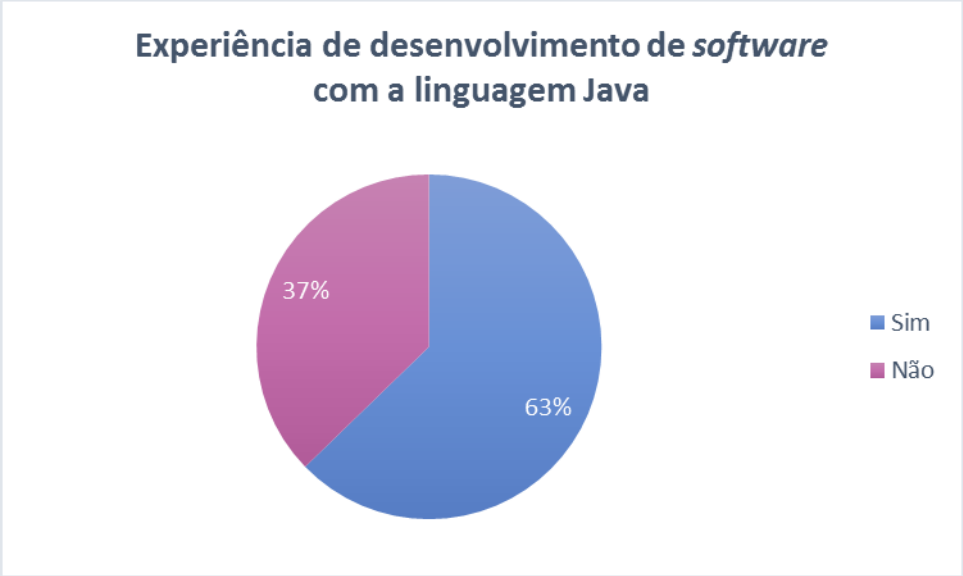


Gráfico 5 - Experiência de desenvolvimento de *software* com a linguagem Java dos entrevistados.
Fonte: Próprio autor.

O Gráfico 5 revela que a maioria dos entrevistados, cerca de 63% tem experiência de desenvolvimento de *software* com a linguagem Java e 37% não tem experiência.

A 5ª e última pergunta desta sessão, mensurou numa escala de 1 a 5, a importância da usabilidade em um *software* de acordo com a opinião dos entrevistados. O resultado desta é apresentado no gráfico a seguir.

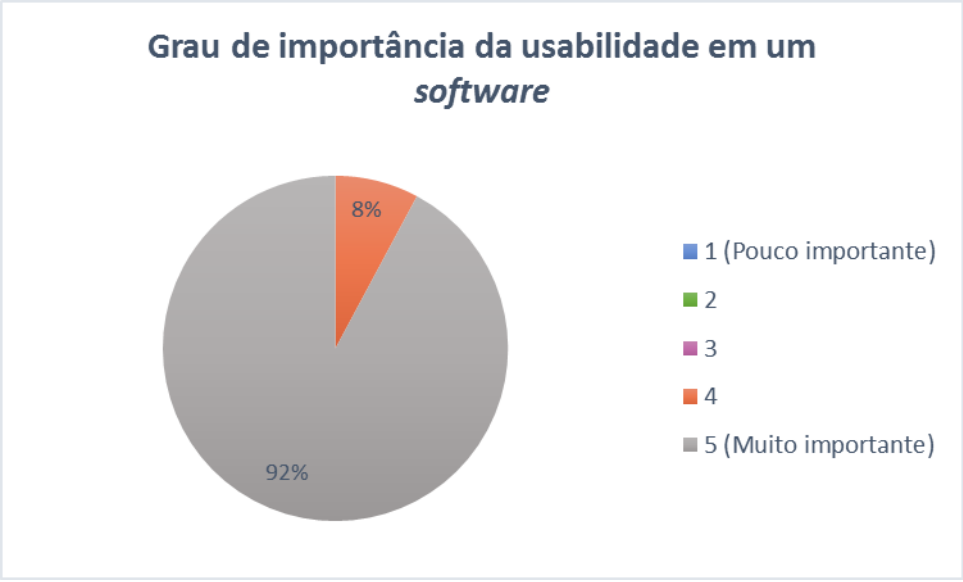


Gráfico 6 - Grau de importância da usabilidade em um *software*.
Fonte: Próprio autor.

No Gráfico 6 é exibido que 92% dos entrevistados classificaram a usabilidade em um *software* em maior número da escala (5), considerando como muito importante e 8%

classificaram como 4 na escala, considerando como importante para usabilidade em um *software*.

Os resultados obtidos do perfil do participante foram bem positivos para esta pesquisa, pois grande parte dos participantes apresentaram em média 5 a 10 anos de experiência de uso em dispositivos relacionados à tecnologia e 1 a 5 anos de experiência em atividades relacionadas à área da computação. Além de que grande parte deles também apresentaram ter experiência em desenvolvimento de *software* com a linguagem Java e demostraram que é importante a usabilidade em um. A partir disso, pode-se dizer que a maioria dos participantes tem conhecimento do assunto apresentado nesta pesquisa, que tem o intuito de analisar a usabilidade entre as bibliotecas de interface gráfica do Java, JavaFX e Swing.

A sessão seguinte mostrará com detalhes como foram os resultados obtidos pelas perguntas específicas de usabilidade das bibliotecas.

3.1.2 Perguntas específicas de usabilidade das bibliotecas

Através das respostas das perguntas específicas de usabilidade das bibliotecas, foi possível avaliar qual das bibliotecas (Swing ou JavaFX) traz mais qualidade de usabilidade para um *software* Java desktop.

O questionário coletou o nível de concordância do respondente às afirmações apresentadas, nas quais foram baseadas nas heurísticas de Nielsen e as métricas de usabilidade da ISO/IEC 25010. As afirmações foram baseadas também nos *softwares* disponibilizados Locflix Swing e Locflix JavaFX que auxiliou para o entendimento de cada métrica e heurística. Cada questão teve 2 escalas, uma se referindo a interface gráfica proporcionada pela biblioteca JavaFX e outra pela biblioteca Swing.

A 1ª pergunta desta seção, se baseou na 1ª heurística de Nielsen: ‘Visibilidade do estado do sistema’, na qual propõe que o *software* deve sempre manter os usuários informados sobre o que está acontecendo, através de feedbacks adequados dentro de um prazo razoável. As respostas desta pergunta podem ser vistas através do gráfico a seguir.

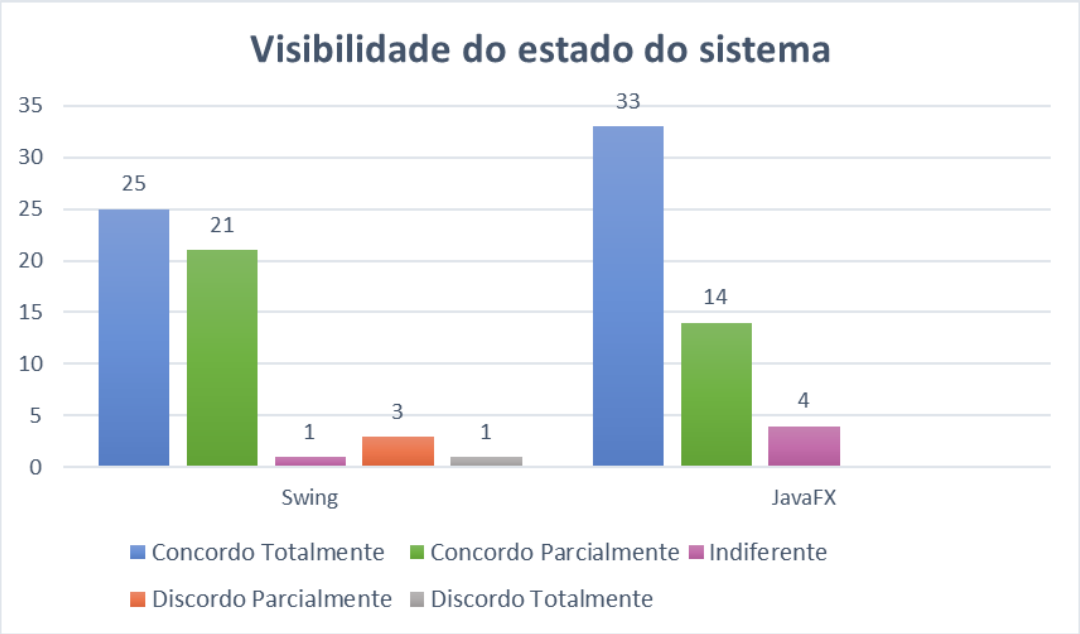


Gráfico 7 - Heurística de Nielsen 'Visibilidade do estado do sistema' para softwares com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

No Gráfico 7 é possível visualizar que a maioria dos respondentes concordaram totalmente com a aplicação desta heurística nas interfaces gráficas proporcionadas pelas bibliotecas, sendo 25 para o Swing e 33 para o JavaFX. No caso do Swing, 21 respondentes concordaram parcialmente e 3 discordaram parcialmente. No caso do JavaFX, 14 respondentes concordaram parcialmente e 4 consideraram como indiferente, não houve nenhuma resposta negativa.

A 2ª pergunta desta seção, foi baseada na 2ª heurística de Nielsen: 'Correspondência entre o sistema e o mundo real', na qual sugere que o *software* tenha nomenclaturas, ícones e imagens ilustrativas contextualizadas e coerentes com o modelo mental do usuário. Para apresentar as respostas desta pergunta, foi criado o Gráfico 8, mostrado a seguir.

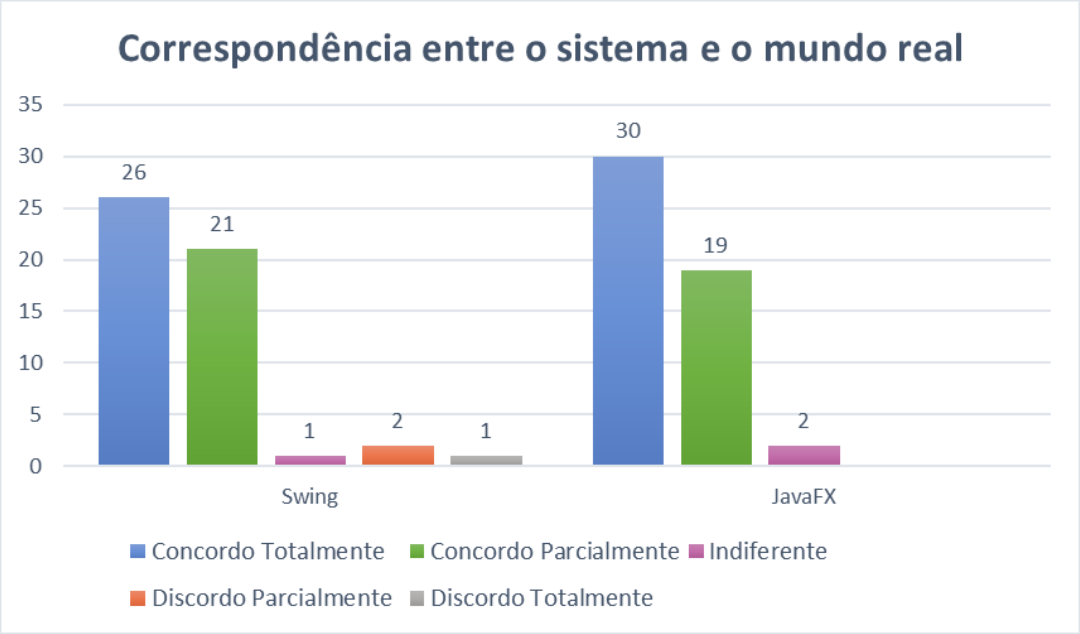


Gráfico 8 - Heurística de Nielsen 'Correspondência entre o sistema e o mundo real' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

A partir do Gráfico 8, pode-se ver que a maioria dos respondentes concordaram totalmente com esta heurística aplicada nas interfaces gráficas proporcionadas pelas bibliotecas, sendo 26 para o Swing e 30 para o JavaFX. Muitos dos respondentes concordaram parcialmente na aplicação desta heurística, com 21 para o Swing e 19 para o JavaFX. Para o Swing teve 2 respondentes que discordaram parcialmente e 1 que discordou totalmente, já para o JavaFX não teve nenhuma resposta negativa para esta heurística.

A 3ª pergunta desta seção, foi criada de acordo com a 3ª heurística de Nielsen: 'Controle e liberdade do usuário', na qual propõe que o *software* permite máxima liberdade ao usuário das decisões e ações que podem ser tomadas no mesmo, exceto a regras de negócio ou que interfira outra funcionalidade. As respostas desta pergunta, são apresentadas no gráfico a seguir.

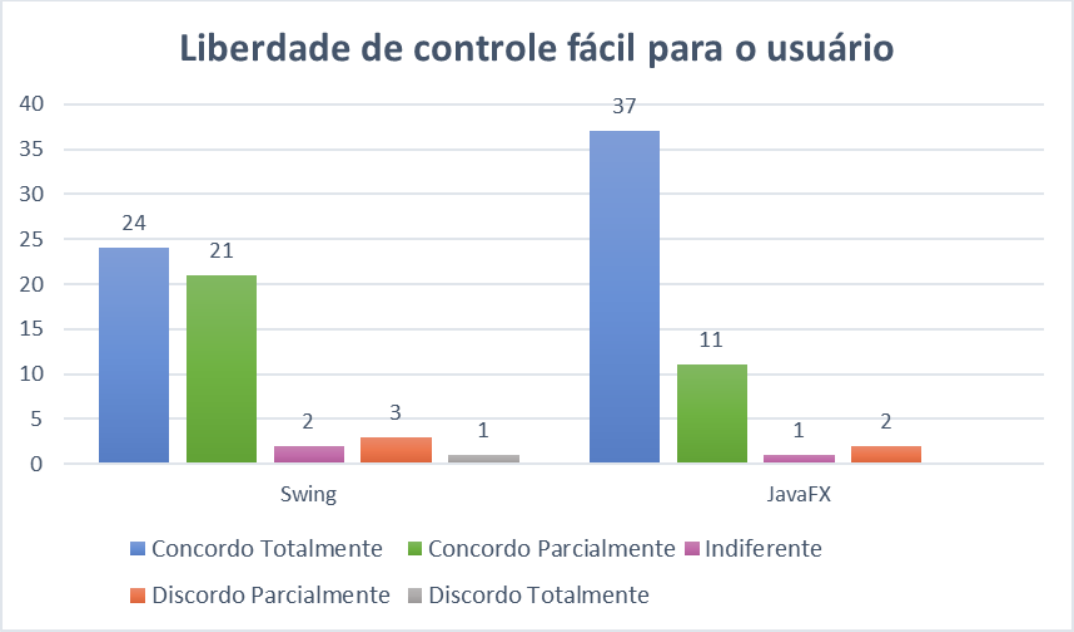


Gráfico 9 - Heurística de Nielsen 'Liberdade de controle fácil para o usuário' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

No Gráfico 9, pode-se ver uma maior diferença das respostas entre as bibliotecas. Semelhante às respostas anteriores, a maioria dos respondentes concordaram totalmente com a aplicação desta heurística nas interfaces gráficas proporcionadas pelas bibliotecas, visto que 37 concordaram totalmente na interface da biblioteca JavaFX e 24 na interface da biblioteca Swing. Alguns respondentes concordaram parcialmente, sendo 21 para a biblioteca Swing e somente 11 para a biblioteca JavaFX. Alguns respondentes discordaram parcialmente, sendo 3 para o Swing e 2 para o JavaFX. E somente 1 respondente discordou totalmente para aplicação desta heurística na interface da biblioteca Swing.

A 4ª pergunta desta seção se baseou na 4ª heurística de Nielsen: 'Consistência e padrões' que recomenda manter uma consistência e padrão visual (texto, cor, desenho do elemento, som) no *software*, facilitando a identificação do usuário para usar o mesmo. As respostas desta pergunta são mostradas no Gráfico 10.

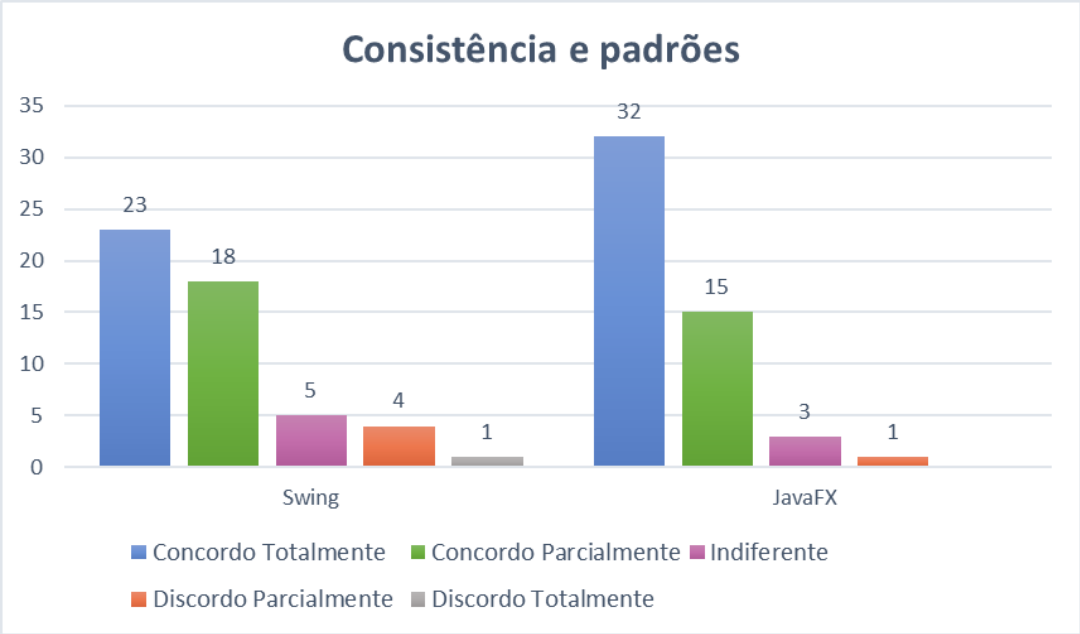


Gráfico 10 - Heurística de Nielsen 'Consistência e padrões' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

Para a heurística 'Consistência e Padrões', 32 dos respondentes concordaram totalmente com a aplicação desta (heurística) na interface proporcionada pela biblioteca JavaFX, 15 concordaram parcialmente, 3 avaliaram como indiferente e apenas 1 discordou parcialmente. Já na interface proporcionada pela biblioteca Swing, houve uma maior divisão entre os níveis de concordância da aplicação desta heurística, sendo 23 respondentes concordaram totalmente, 18 concordaram parcialmente, 5 avaliaram como indiferente, 4 discordaram parcialmente e apenas 1 discordou totalmente.

A 5ª pergunta desta seção, além de ter sido criada como base na 5ª heurística de Nielsen: 'Prevenção de erros', ela também foi baseada em uma das métricas de usabilidade da ISO/IEC 25010: 'Proteção contra erros do usuário', pois ambas possuem o mesmo objetivo. Elas recomendam não deixar o usuário errar sem explicar previamente o motivo do erro no *software*. Recomendam também, proporcionar interfaces com características que não permite o usuário errar, como máscaras (data, telefone, CEP, entre outras) nos campos e fornecer mensagens explicativas caso o usuário tentar fazer algo que está contra as regras de negócio do *software*. Para apresentar os resultados desta pergunta, foi criado um gráfico, que é mostrado a seguir.

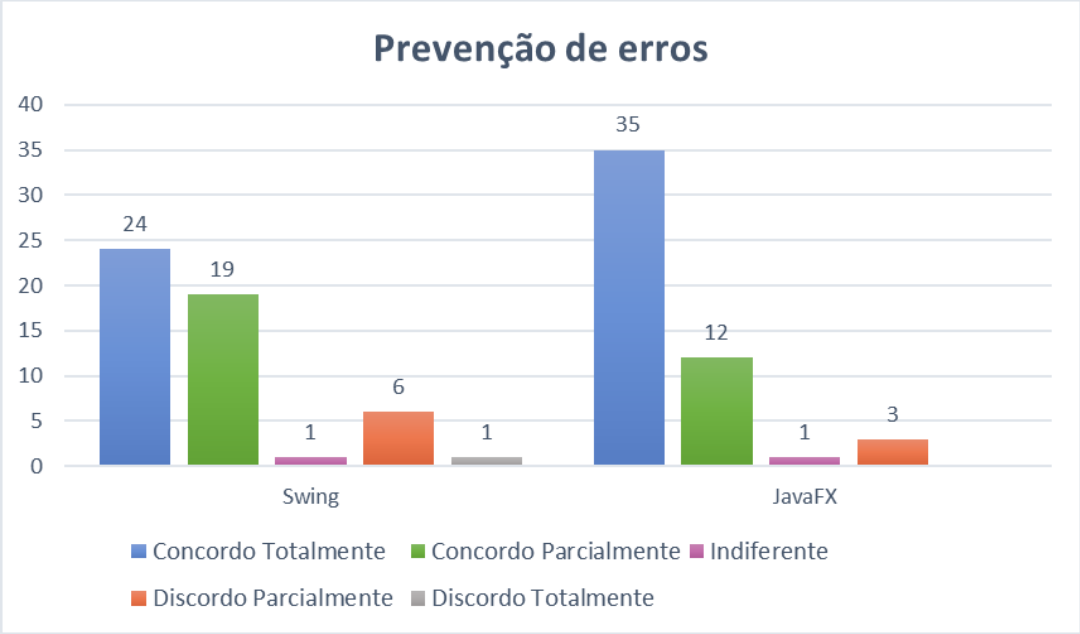


Gráfico 11 - Heurística de Nielsen 'Prevenção de erros' e métrica da ISO/IEC 25010 'Proteção contra erros do usuário' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

De acordo com o Gráfico 11, pode-se ver que a aplicação desta heurística e métrica na interface gráfica proporcionada pela biblioteca JavaFX obteve 35 respondentes que concordaram totalmente, 12 concordaram parcialmente e 3 discordaram parcialmente. Já na interface gráfica proporcionada pela biblioteca Swing, a aplicação desta métrica e heurística, 24 respondentes concordaram totalmente, 19 concordaram parcialmente, 6 discordaram parcialmente e 1 discordou totalmente. Para ambas as bibliotecas 1 respondente em cada uma avaliou como indiferente.

A 6ª pergunta desta seção, assim como a 5ª pergunta, também se baseou em uma heurística de Nielsen e uma métrica da ISO/IEC 25010. Neste caso foi utilizada a 6ª heurística de Nielsen: 'Reconhecimento em vez de memorização' e a métrica de usabilidade 'Aprendizagem', nas quais têm objetivos semelhantes, sendo eles minimizar a carga de memória do usuário, tornando objetos, ações e opções visíveis, capacitando o *software* a ser de fácil aprendizado para o mesmo (usuário). As respostas desta pergunta são apresentadas no Gráfico 12.

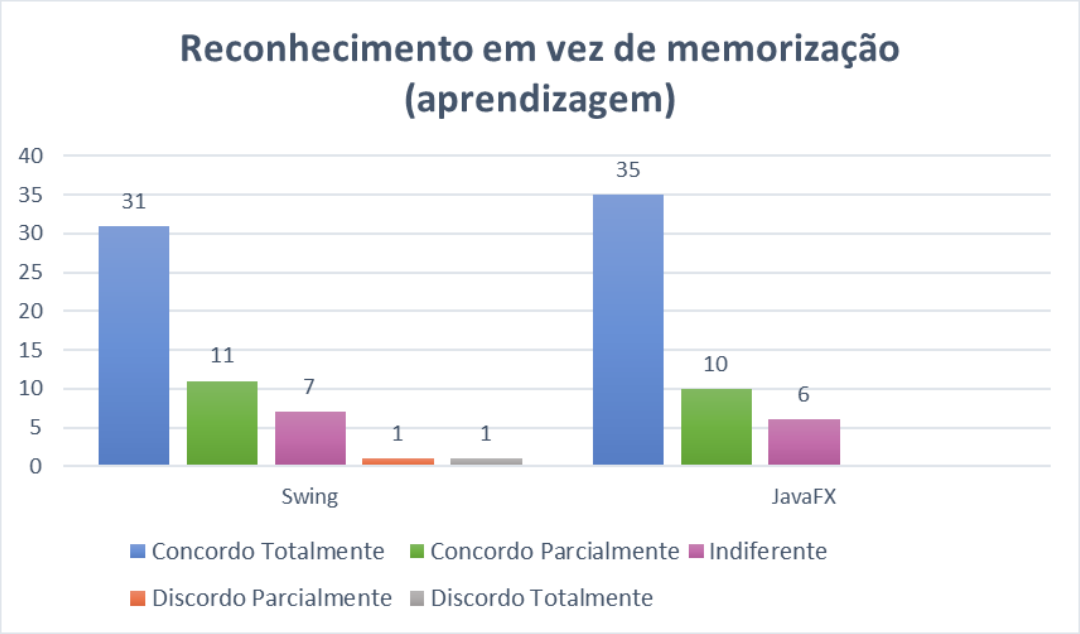


Gráfico 12 - Heurística de Nielsen 'Reconhecimento em vez de memorização' e métrica da ISO/IEC 25010 'Capacidade de aprendizagem' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

No Gráfico 12, é visto que, os resultados para cada biblioteca com o uso desta heurística e métrica estão parecidos um com o outro. A aplicação desta heurística e métrica na interface gráfica proporcionada pela biblioteca Swing, 31 respondentes concordaram totalmente, 11 concordaram parcialmente, 7 consideraram com indiferente, e os dois restantes ficaram divididos, 1 discordou parcialmente e outro, discordou totalmente. Já para a biblioteca JavaFX, 35 respondentes concordaram totalmente, 10 concordaram parcialmente, 6 consideraram com indiferente e não houve resultados negativos.

A 7ª pergunta desta seção teve como base a heurística de Nielsen 'Flexibilidade e eficiência de uso' e a métrica da ISO/IEC 25010 'Capacidade de usar' ou 'Operabilidade', nas quais ambas propõem características semelhantes para o *software*, que é a capacidade de um (*software*) permite aos usuários operar e controlar facilmente, ele deve ser ágil para usuários avançados e fácil de utilizar para usuários leigos. Esta pergunta teve os resultados obtidos mostrados no Gráfico 13.

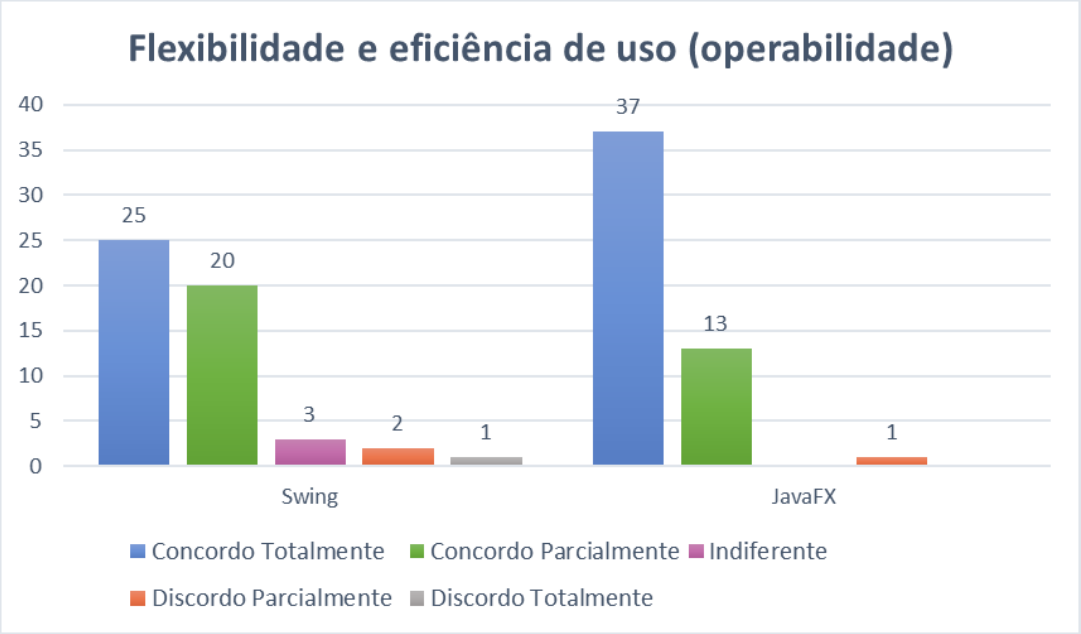


Gráfico 13 - Heurística de Nielsen 'Flexibilidade e eficiência de uso' e métrica da ISO/IEC 25010 'Capacidade de usar' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

Para a heurística 'Flexibilidade e eficiência de uso' e métrica 'Capacidade de usar', 37 respondentes concordaram totalmente com o uso destas na interface gráfica proporcionada pela biblioteca JavaFX, 13 deles concordaram parcialmente e apenas 1 respondente discordou parcialmente. Já no uso da heurística e métrica na interface gráfica proporcionada pela interface gráfica Swing, 25 respondentes concordaram totalmente, 20 concordaram parcialmente, 3 consideraram como indiferente, 2 discordaram parcialmente e apenas 1 discordou totalmente.

Para a 8ª pergunta desta seção, foram utilizadas a 8ª heurística de Nielsen 'Estética e design minimalista' e a métrica de usabilidade da norma ISO/IEC 25010 'Estética da interface do usuário', que tem os mesmos objetivos para avaliar um *software*, nos quais visam a agradar e satisfazer a interação do usuário através de interfaces com layout limpo, sem usar desnecessariamente excessos de cores e elementos visuais que podem confundir o usuário. Para visualizar as respostas desta pergunta, foi criado um gráfico que é mostrado a seguir.

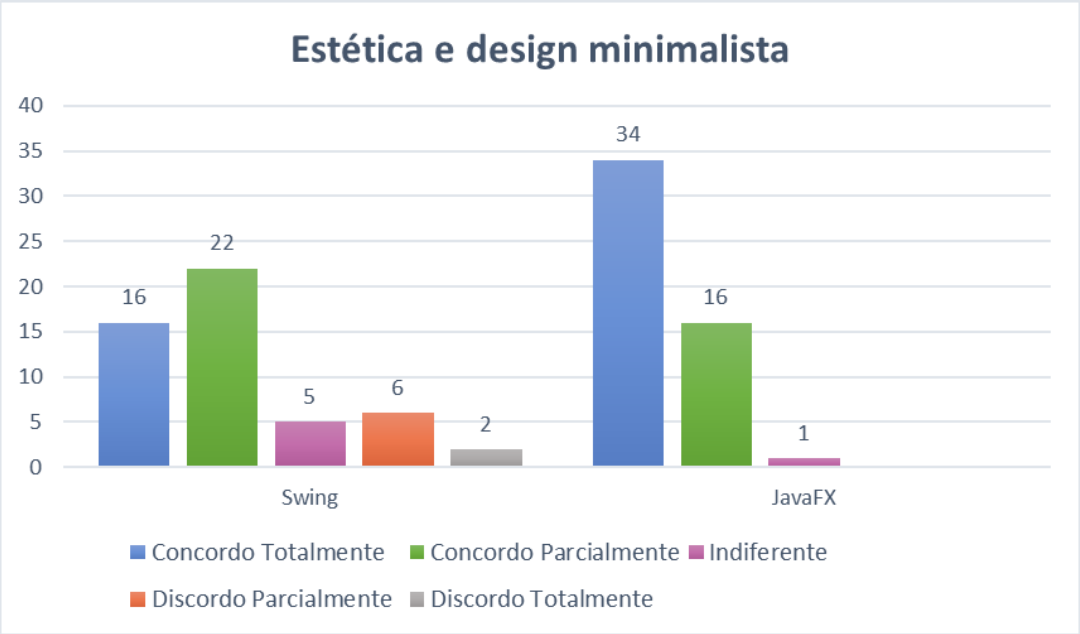


Gráfico 14 - Heurística de Nielsen 'Estética e design minimalista' e métrica da ISO/IEC 25010 'Estética da Interface de usuário' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

No Gráfico 14, pode-se notar uma maior diferença nos resultados de uma biblioteca para outra. A interface gráfica proporcionada pela biblioteca JavaFX aplicada a essa heurística e métrica, teve bastante resultados positivos, sendo 34 respondentes concordaram totalmente, 16 concordaram parcialmente e apenas 1 considerou como indiferente. Já a interface gráfica proporcionada pelo Swing aplicada a essa heurística e métrica obteve uma divergência entre o nível de concordância dos respondentes em comparação as respostas das outras perguntas, mas ainda sim a maioria deles ficaram entre concordo totalmente (16) e parcialmente (22), 5 avaliaram como indiferente, 6 discordaram parcialmente e 2 discordaram totalmente.

Na 9ª pergunta desta seção, foi utilizada a 9ª heurística de Nielsen: 'Recuperação de erros', na qual recomenda a um *software* ter mensagens de erro claras, com textos simples e diretos, não intimidar o usuário e sim ajudá-lo a reconhecer, diagnosticar e conduzir à possíveis soluções do erro. Esta pergunta teve suas respostas apresentadas pelo gráfico a seguir.

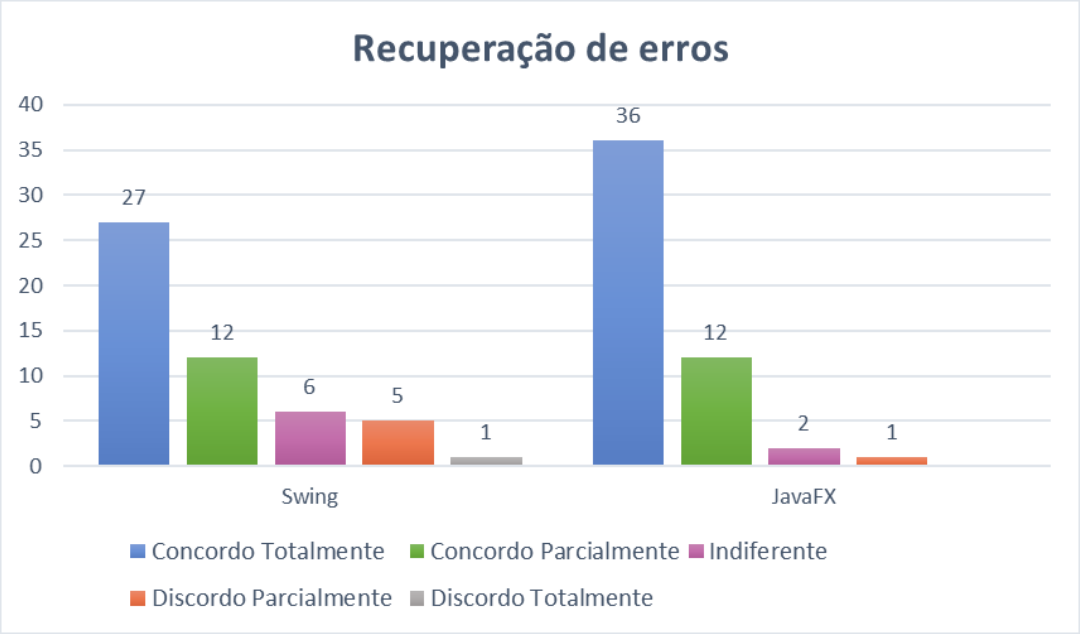


Gráfico 15 - Heurística de Nielsen 'Recuperação de erros' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

Na interface gráfica proporcionada pela biblioteca Swing, aplicada a esta heurística de Nielsen, 27 respondentes concordaram totalmente, 12 concordaram parcialmente, 6 consideraram indiferente, 5 discordaram parcialmente e apenas 1 discordou totalmente. Já na interface gráfica proporcionada pela biblioteca JavaFX, aplica a esta heurística, 36 respondentes concordaram totalmente, 12 concordaram parcialmente, 2 avaliaram como indiferente e somente 1 respondente discordou parcialmente.

Na 10ª pergunta desta seção, foi fundamentada a métrica de usabilidade da norma ISO/IEC 25010: 'Reconhecimento de adequação', na qual indica se o usuário considera que o *software* é adequado para suas necessidades, em outras palavras, se o *software* proporciona ao usuário uma certa facilidade de identificar se a interface gráfica atende a todas suas necessidades. As respostas desta pergunta são apresentadas no Gráfico 16.

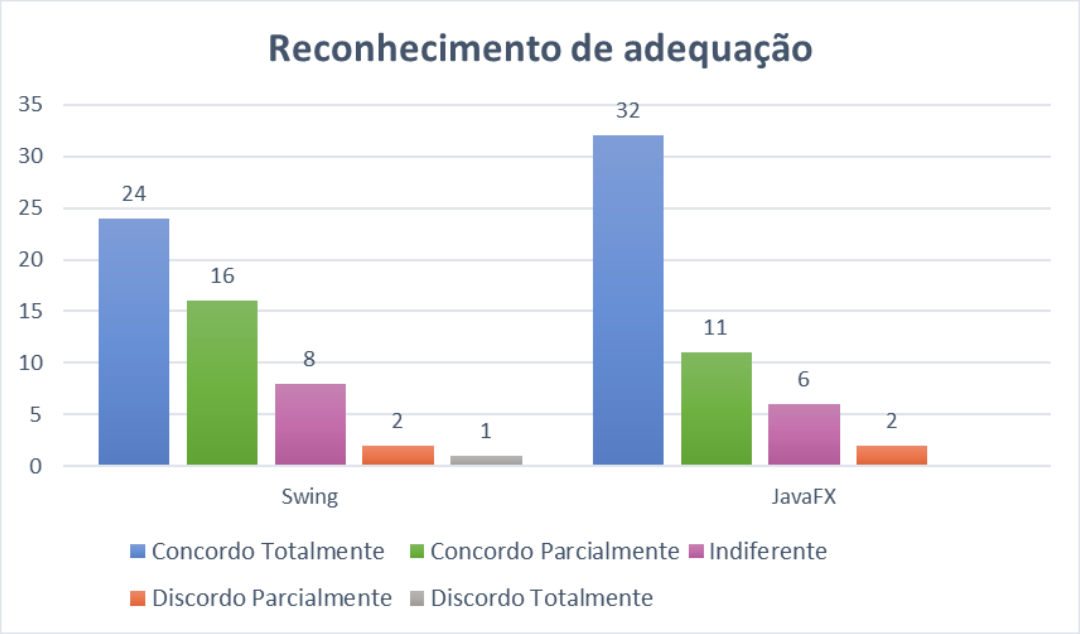


Gráfico 16 - Métrica ISO/IEC 'Reconhecimento de adequação' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

No Gráfico 16, pode-se ver que a interface gráfica proporcionada pela biblioteca Swing aplicada a esta métrica, 24 respondentes concordaram totalmente, 16 concordaram parcialmente, 8 consideraram como indiferente, 2 discordaram parcialmente e apenas 1 participante discordou totalmente. Já na interface gráfica proporcionada pela biblioteca JavaFX aplicada a esta métrica, 32 respondentes concordaram totalmente, 11 concordaram parcialmente, 6 avaliaram como indiferente e apenas 2 respondentes discordaram parcialmente.

A 11ª pergunta desta seção, foi baseada na última métrica de usabilidade da norma ISO/IEC 25010: 'Acessibilidade', na qual visa a capacidade do *software* proporcionar uma interface gráfica para usuários com determinadas características e deficiências. As respostas desta pergunta são exibidas no gráfico a seguir.

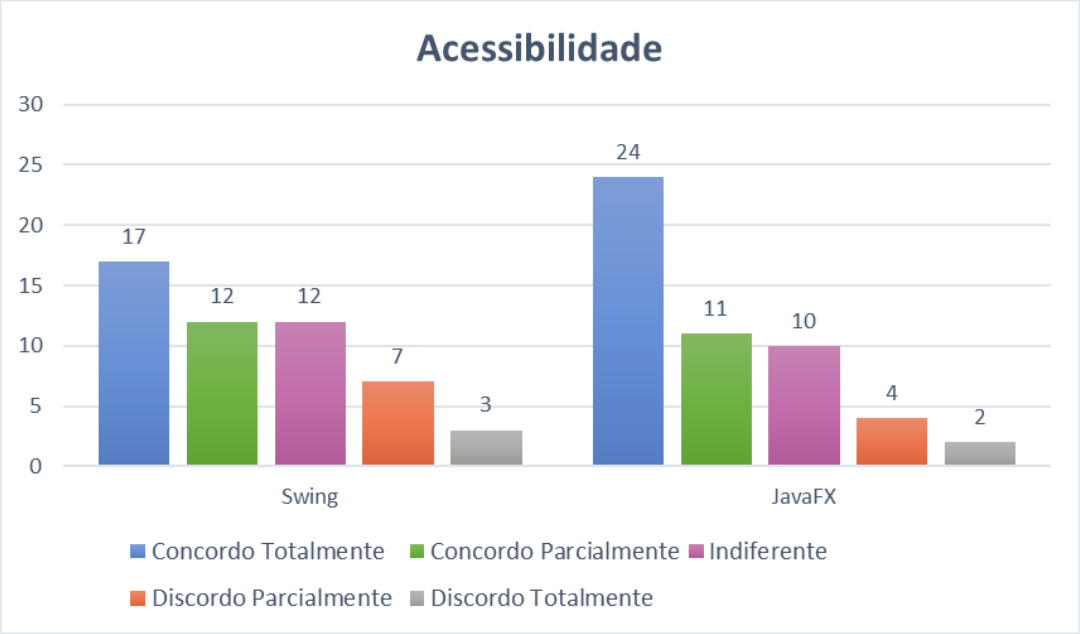


Gráfico 17 - Métrica ISO/IEC ‘Acessibilidade’ para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

A partir do Gráfico 17, pode-se ver que a opção indiferente foi bem escolhida pelos respondentes, um pouco mais que nas respostas das perguntas anteriores. Na interface gráfica proporcionada pela biblioteca Swing aplicada a esta métrica, 17 respondentes concordaram totalmente, 12 concordaram parcialmente, 12 avaliaram como indiferente, 7 discordaram parcialmente e apenas 3 discordaram totalmente. Já na interface gráfica proporcionada pela biblioteca JavaFX aplicada a esta métrica, 24 respondentes concordaram totalmente, 11 concordaram parcialmente, 10 avaliaram como indiferente, 4 discordaram parcialmente e apenas 2 discordaram totalmente.

A 12ª pergunta desta seção, teve como base a 10ª e última heurística de Nielsen: ‘Ajuda e documentação’, na qual recomenda que o *software* deve ser intuitivo e claro para o usuário no qual, quase não seja necessário o uso de documentações complementares. Mas mesmo assim é preciso fornecer documentações (manuais e explicações sobre o uso) ao alcance do usuário. No próximo gráfico é apresentado as respostas obtidas por esta pergunta.

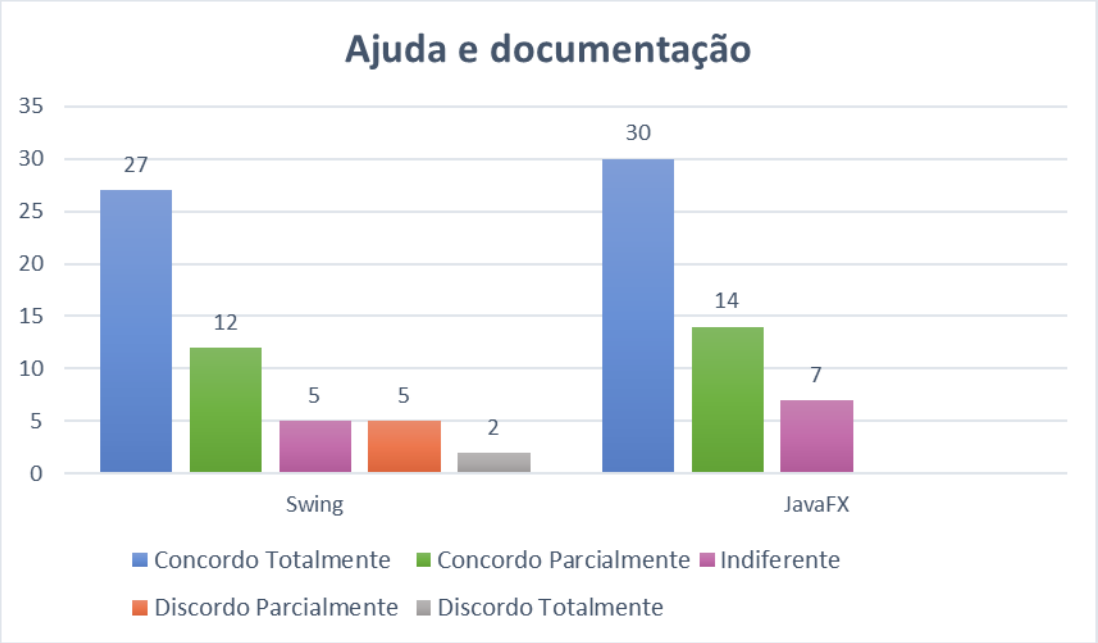


Gráfico 18 - Heurística de Nielsen 'Ajuda e documentação' para *softwares* com as bibliotecas Swing e JavaFX.
Fonte: Próprio autor.

A partir do Gráfico 18, é possível ver que a interface gráfica proporcionada pela biblioteca Swing aplicada a esta métrica, 27 respondentes concordaram totalmente, 12 concordaram parcialmente, 5 consideraram com indiferente, 5 discordaram parcialmente e 2 discordaram totalmente. Já na interface gráfica proporcionada pela biblioteca JavaFX aplicada a esta métrica, 30 respondentes concordaram totalmente, 14 concordaram parcialmente e 7 avaliaram como indiferente.

A 13ª pergunta desta seção foi para avaliar a opinião final do usuário em relação as bibliotecas, depois de analisar os *softwares*, as heurísticas e métricas de usabilidade. Os resultados desta poderão ser vistos a partir do próximo gráfico.

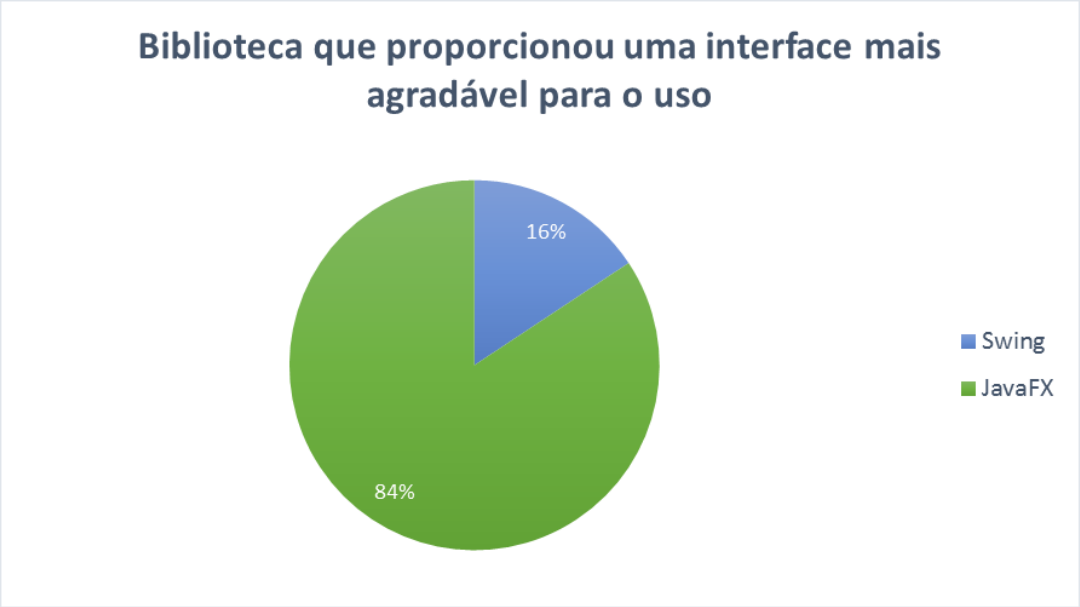


Gráfico 19 - Biblioteca que proporcionou uma interface mais agradável para o uso.
Fonte: Próprio autor.

No Gráfico 19, é possível ver a opinião dos respondentes em relação à escolha de uma biblioteca que proporcionou uma interface mais agradável para o uso, sendo 84% escolheram a biblioteca JavaFX e 16% a biblioteca Swing.

As duas últimas perguntas desta seção, foram discursivas, mas não foram obrigatórias. Ambas foram criadas com o intuito de coletar opiniões e considerações dos participantes em relação a cada biblioteca.

A 1ª pergunta discursiva foi em relação a biblioteca JavaFX, na qual se obteve 13 respostas. Dentre estas, 10 respondentes elogiaram a interface gráfica do JavaFX, justificando ser mais intuitiva, agradável, parecer mais atual e ter uma qualidade superior em design. Nestas 10 respostas, 5 respondentes informaram que são desenvolvedores Java e alguns destacam que o JavaFX é mais fácil para o desenvolvimento e possui mais recursos em termos gráficos. 1 respondente elogiou o trabalho, e outros 2, responderam que não tinham considerações a fazer.

A 2ª pergunta discursiva foi em relação a biblioteca Swing, na qual se obteve 11 respostas. Dentre elas, vários respondentes também se identificaram como programadores Java. 7 respondentes elogiaram a interface, justificando ser bem funcional e se parecer bem com o JavaFX em alguns detalhes, mas 3 destes acharam que em outros detalhes lembram *softwares* antigos e preferem o JavaFX por ter um design mais moderno em relação ao Swing. 1 respondente destacou que gostou, mas que linguagem de programação ou bibliotecas não se discutem e o importante é que atenda a necessidade e satisfação do cliente ou usuário. 1

respondente destacou que o desenvolvimento em Swing é mais complicado. Outro destacou que em sua opinião, foram pequenos detalhes que diferenciaram o Swing e JavaFX, pois ambos ficaram bem estruturados. 2 responderam que não tinham considerações a fazer.

3.1.3 Correlações entre o perfil do participante e escolha da biblioteca

A partir dos resultados das perguntas das duas seções do questionário: perfil do participante e perguntas específicas de usabilidade das bibliotecas, foram feitas correlações entre os níveis de experiências dos respondentes e a biblioteca escolhida por eles (respostas coletadas pela 13ª pergunta da segunda seção do questionário), mostradas nos gráficos a seguir.

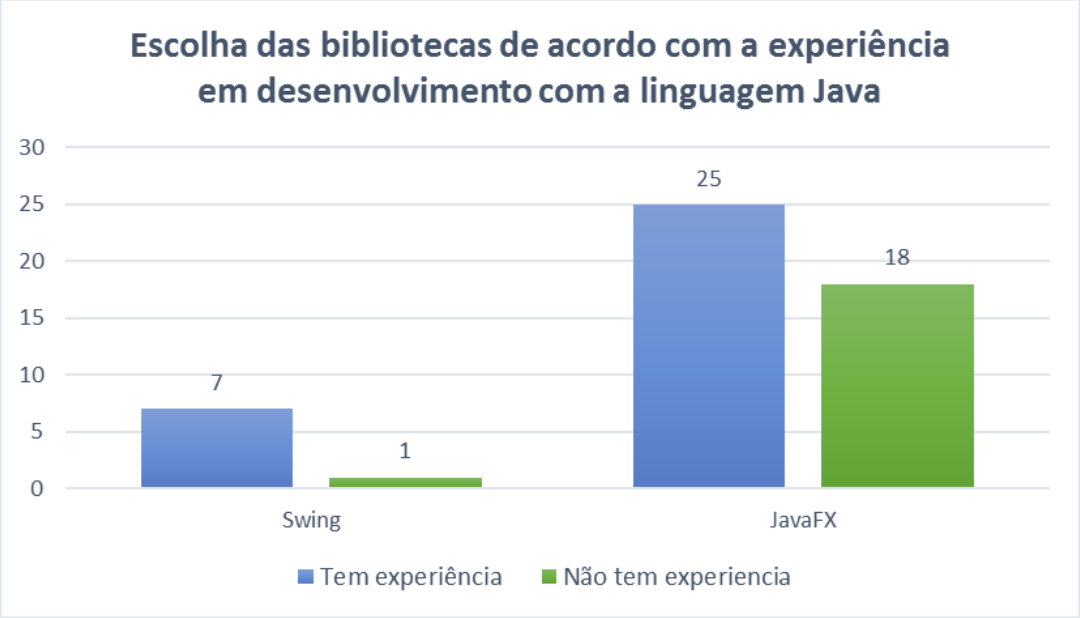


Gráfico 20 - Escolha das bibliotecas de acordo com a experiência em desenvolvimento com a linguagem Java.
Fonte: Próprio autor.

No Gráfico 20 foi feita uma correlação entre a experiência dos participantes de desenvolvimento com a linguagem Java, coletada através das respostas obtidas pela 4ª pergunta da seção perfil do participante e a escolha das bibliotecas. Através desta correlação, 7 respondentes com experiência e apenas 1 sem experiência com desenvolvimento Java escolheram o Swing, a maior parte dos respondentes, independente da experiência com desenvolvimento Java escolheram a biblioteca JavaFX.

Além de analisar a experiência em desenvolvimento de *software* com a linguagem Java dos participantes, foram analisados também o tempo que os mesmos usam dispositivos relacionados à tecnologia.

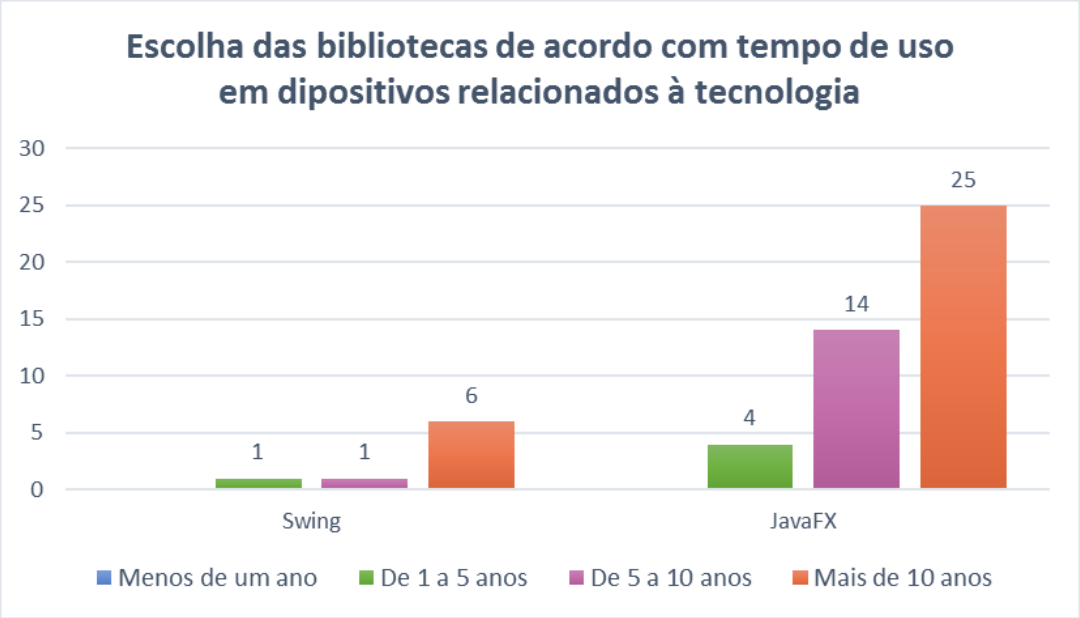


Gráfico 21 - Escolha das bibliotecas de acordo com tempo de uso em dispositivos relacionados à tecnologia.
Fonte: Próprio autor.

No Gráfico 21 foi feita uma correlação entre o tempo que os participantes utilizam dispositivos relacionados à tecnologia da informação, coletado através das respostas obtidas pela 3ª pergunta da seção perfil do participante e a escolha das bibliotecas.

Nesta correlação, é visto que 6 participantes que utilizam dispositivos há mais de 10 anos escolheram a biblioteca Swing. Mas, independentemente do tempo em que os participantes já utilizam dispositivos relacionados à tecnologia da informação, também é visível que a maior parte deles escolheram a biblioteca JavaFX.

Após a análise de experiência em desenvolvimento de *software* com a linguagem Java dos participantes e tempo que os mesmos utilizam dispositivos relacionados à tecnologia da informação, foi feito também uma análise do tempo de experiência em atividades relacionadas à computação.

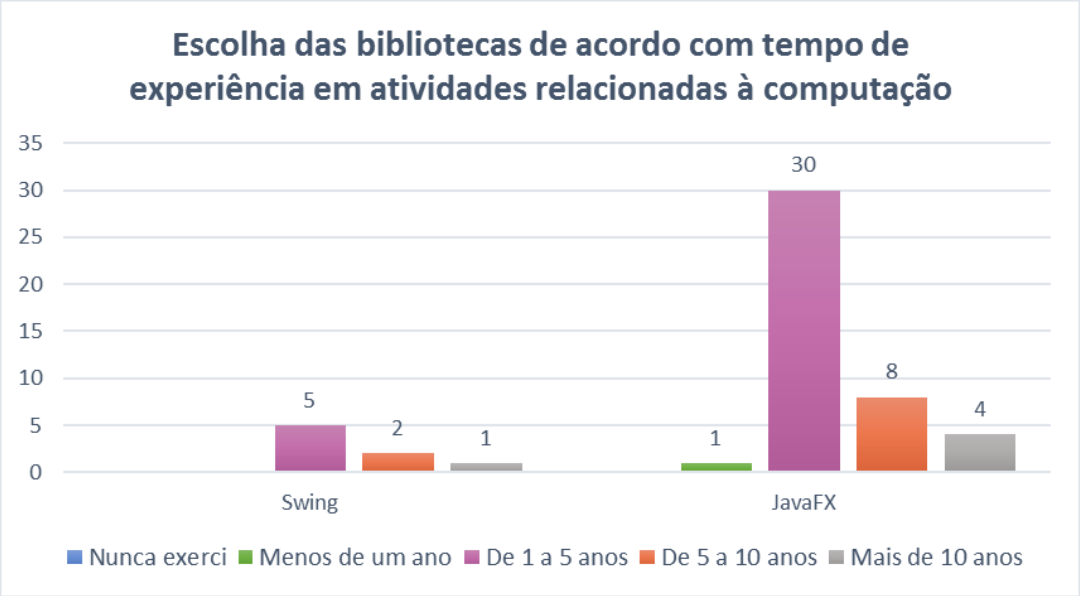


Gráfico 22 - Escolha das bibliotecas de acordo com o tempo de experiência em atividades relacionadas à computação.
Fonte: Próprio autor.

No Gráfico 22 também foi feita uma correlação, entre a experiência dos participantes em atividades relacionados à computação, coletada através das respostas obtidas pela 2ª pergunta da seção perfil do participante e a escolha das bibliotecas. Através deste (gráfico), 5 respondentes com 1 a 5 anos de experiência escolheram a biblioteca Swing, mas a maior parte dos respondentes, independente da experiência em alguma atividade na área da computação escolheram a biblioteca JavaFX.

3.1.4 Discussão dos resultados

Com os resultados obtidos na segunda sessão do questionário, foi possível perceber que ambas as bibliotecas tiveram respostas positivas que são ‘Concordo totalmente’ e ‘Concordo parcialmente’ em maior número do que respostas negativas que são ‘Discordo totalmente’ e ‘Discordo parcialmente’ em todas as heurísticas de Nielsen e métricas de usabilidade da norma ISO/IEC 25010. Com isso é comprovado que ambas podem proporcionar qualidade de usabilidade em um *software*. Apesar de que a biblioteca JavaFX teve um peso maior de respostas positivas e menor em respostas negativas que a biblioteca Swing.

Entretanto, é preciso considerar que a forma com que os *softwares* foram desenvolvidos e o objetivo dos mesmos, influenciaram nas respostas dos participantes. Pois

apesar de serem desenvolvidos com o cuidado de satisfazer a todas as métricas e heurísticas de usabilidade para mostrar o potencial de cada biblioteca, existem várias formas de se criar uma interface gráfica em um *software* com elas.

3.2 ANÁLISE DE CÓDIGO-FONTE

A análise de código-fonte foi concluída por meio dos resultados das métricas obtidas através da ferramenta Analizo.

Para que os resultados do código-fonte serem mostrados de forma justa e objetiva para cada biblioteca, todos os dados foram organizados. Essa organização será detalhada na próxima sessão.

3.2.1 Organização dos resultados das métricas de código-fonte

A ferramenta Analizo retorna uma lista com o resultado de todas as métricas, com a análise do código-fonte do projeto inteiro e também das análises do código-fonte divididas por classes.

Nos projetos dos *softwares* Locflix Swing e o Locflix JavaFX foram criados uma quantidade diferente de classes, pois algumas são particulares de cada projeto e outras são bem semelhantes.

As classes Java particulares do Locflix Swing são:

- Categoria - classe modelo na qual contém os atributos de uma categoria de filmes.
- CategoriaCTR - classe responsável por mostrar a interface gráfica de uma categoria na tela principal/inicial.

As classes Java particulares do Locflix JavaFX são:

- Locflix - classe responsável por fazer a chamada da tela principal.
- FilmeTBV - classe responsável para a configuração de um componente *TableView*, que é uma tabela do JavaFX. É usada na tela de Lista de Filmes.

- Istage - é uma interface (orientado a objetos), contendo alguns métodos para controle de eventos em uma janela no JavaFX.
- Jstage - é uma classe que é implementada pela Istage, na qual é responsável para controlar os eventos de janelas dos controladores. Todas classes controladoras de telas no projeto como tela Principal/Inicial, Informações do Filme, Configurações de usuário e Lista de filmes estendem desta classe.

As classes Java que os dois *softwares* têm em comum são:

- Filme - classe modelo na qual contém os atributos de um filme.
- Usuario - classe modelo na qual contém os atributos do usuário.
- Dados - classe responsável por alimentar os *softwares* com todos os dados dos 90 filmes em listas estáticas.
- PrincipalCTR - classe responsável pelo controle de componentes na tela principal/inicial.
- FilmeCTR - classe responsável pelo controle de componentes de cada filme na tela principal/inicial.
- InformacoesFilmeCTR - classe responsável pelo controle de componentes da tela de Informações do filme.
- ListaFilmesCTR - classe responsável pelo controle de componentes da tela de Lista de filmes.
- UsuarioCTR - classe responsável pelo controle de componentes da tela de Configuração de usuário.

Serão analisadas as métricas das classes em comum para os 2 *softwares* exceto as classes Filme e Usuario, pois são inteiramente iguais nos 2 *softwares*, e por causa disso, tiveram resultados iguais.

Os resultados das métricas serão divididos em 4 seções: métricas de tamanho, complexidade, acoplamento e coesão, que são as seções seguintes.

As 2 primeiras seções (métricas de tamanho e métricas de complexidade) conterão 7 gráficos, nos quais 6 serão mostrados os resultados das métricas por classe em comum dos dois *softwares*, e 1 gráfico será mostrado os resultados das métricas do projeto inteiro incluindo as classes particulares de cada *software*.

Já nas outras 2 seções (métricas de acoplamento e métricas de coesão) conterá 1 gráfico com os resultados das métricas do projeto inteiro incluindo as classes particulares de cada *software*.

3.2.2 Métricas de tamanho

Os resultados das métricas de tamanho LOC (número de linhas de código), AMLOC (média do número de linhas de código por método) e MMLOC (número de linhas de código por método) serão separados por 7 gráficos como explicado na seção anterior.

Os resultados das métricas de tamanho dos projetos Locflix Swing e Locflix JavaFX inteiros são mostradas no Gráfico 23.

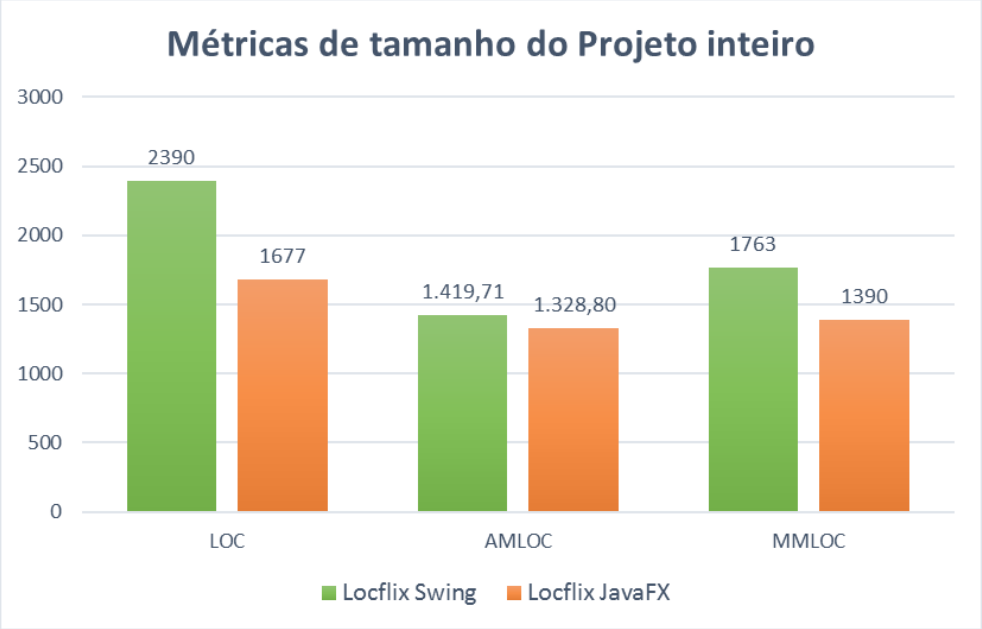


Gráfico 23 - Métricas de tamanho do Projeto inteiro.
Fonte: Próprio autor.

A quantidade de linhas para os 2 projetos ficou em um número alto pois, os 90 filmes dos *softwares* foram incluídos dentro de uma classe no código-fonte de cada um, o que influenciou muito para este resultado.

Nesta métrica são considerados somente as linhas executáveis do código-fonte, ignorando todos comentários e linhas em branco. A métrica LOC do projeto Locflix Swing ficou em 2390 linhas e do Locflix JavaFX ficou em 1677 linhas, tendo 730 linhas de

diferença. Lembrando que neste gráfico contém os resultados de classes em comum e classes particulares de cada projeto. O Locflix Swing possui 10 classes e o Locflix JavaFX possui 12 classes.

Os resultados das métricas de tamanho para a classe Dados de cada *software* são apresentados no gráfico a seguir.

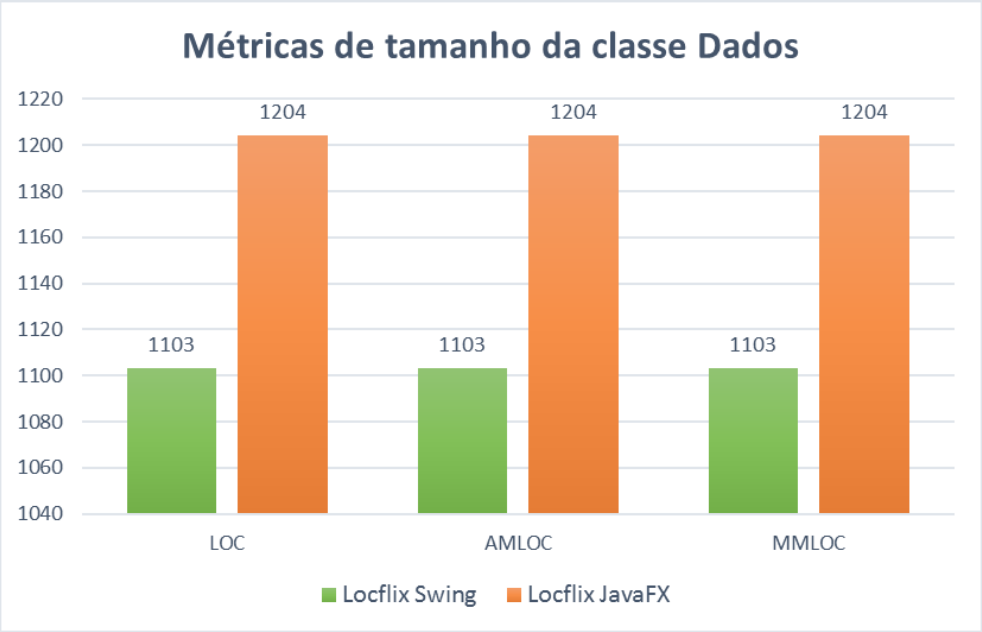


Gráfico 24 - Métricas de tamanho da classe Dados.
Fonte: Próprio autor.

No Locflix Swing, a classe Dados ficou com a métrica LOC em 1103 linhas e no Locflix JavaFX em 1204 linhas, tendo uma diferença de 101 linhas. As métricas AMLOC e MMLOC ficaram com resultados iguais pois esta classe só tem um método que tem por objetivo preencher as listas estáticas com dados de 90 filmes.

Os resultados das métricas de tamanho para a classe controladora de Filme de cada *software* são mostradas no Gráfico 25.

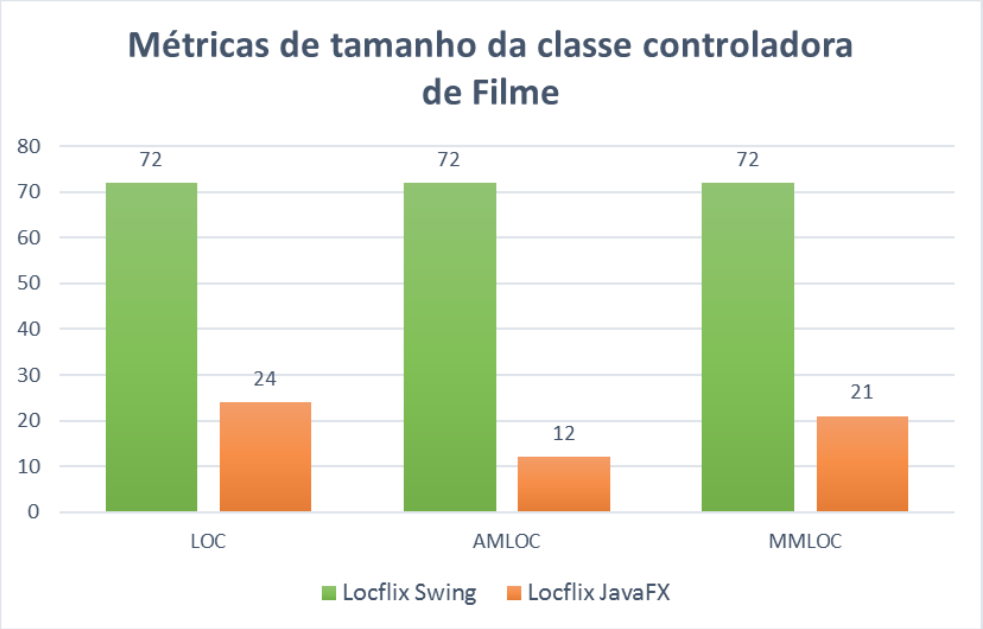


Gráfico 25 - Métricas de tamanho da classe controladora de Filme.
Fonte: Próprio autor.

No Locflix Swing, a classe Controladora de Filme ficou com a métrica LOC, AMLOC e MMLOC em 72 linhas, pois nesta só tem um método. No Locflix JavaFX, a métrica LOC ficou em 24 linhas, tendo uma diferença de 48 linhas para o resultado de LOC do Locflix Swing, a métrica AMLOC ficou em 12 linhas e a MMLOC 21 linhas.

Os resultados das métricas de tamanho para a classe controladora da tela Informações do Filme de cada *software* são apresentados no gráfico a seguir.

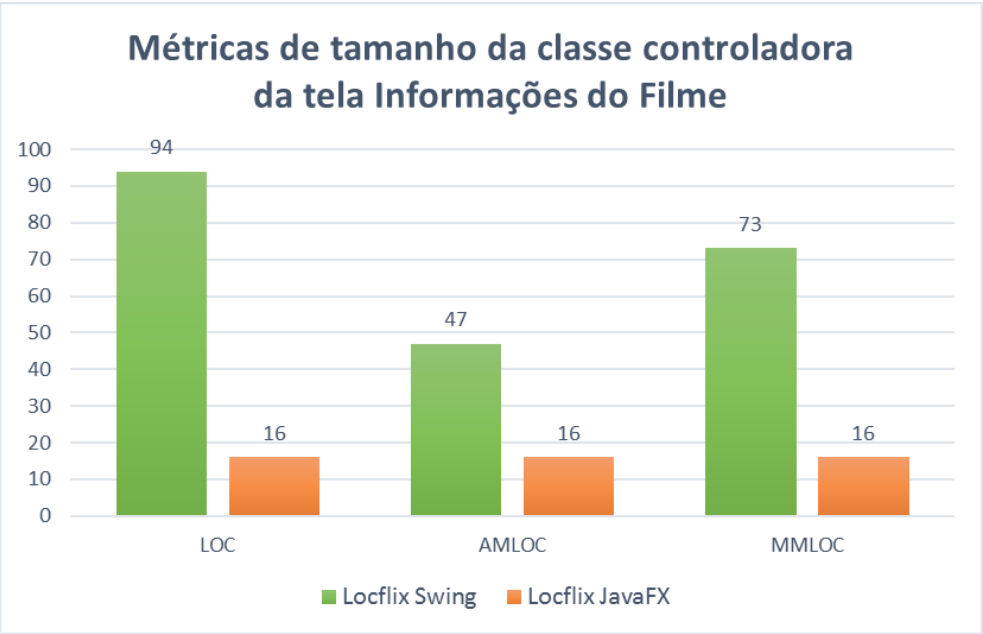


Gráfico 26 - Métricas de tamanho da classe controladora da tela Informações do Filme.
Fonte: Próprio autor.

A partir do Gráfico 26, pode-se notar que os resultados de LOC, AMLOC e MMLOC do Locflix JavaFX ficaram iguais, em 16 linhas, pois nesta classe contém somente um método. Para o Locflix Swing o resultado de LOC ficou em 94 linhas tendo uma diferença de 78 linhas ao do Locflix JavaFX, o resultado de AMLOC ficou em 47 linhas e o de MMLOC ficou em 73 linhas.

Os resultados das métricas de tamanho para a classe controladora da tela Lista de Filmes de cada *software* são mostrados no gráfico a seguir.

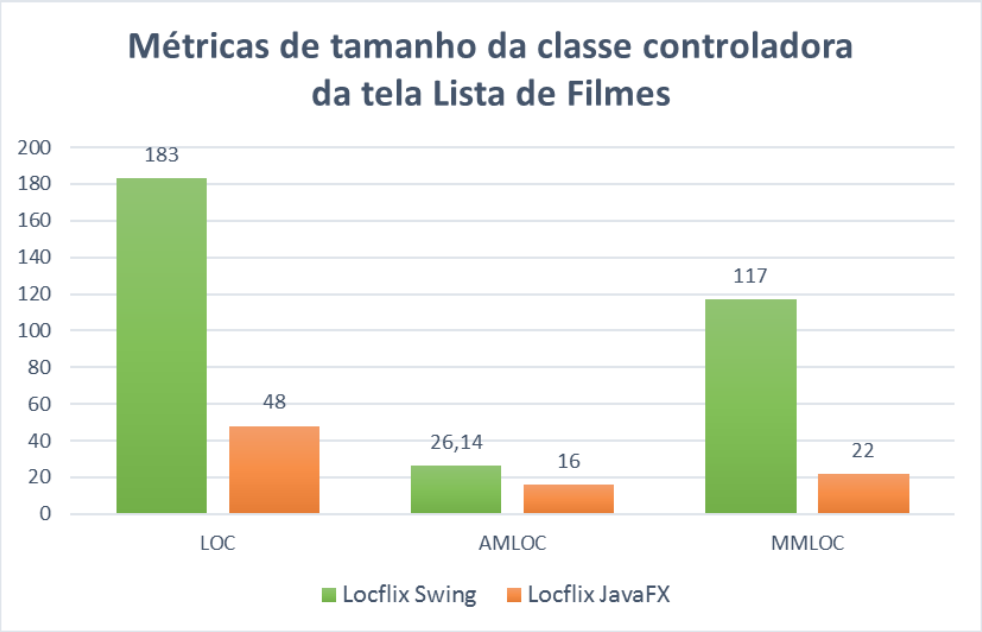


Gráfico 27 - Métricas de tamanho da classe controladora da tela Lista de Filmes.
Fonte: Próprio autor.

O resultado de LOC da classe no Locflix Swing ficou em 183 linhas e no Locflix JavaFX ficou em 48 linhas, tendo uma diferença de 135 linhas. O AMLOC da classe no Locflix Swing ficou em 26,14 e no Locflix JavaFX ficou em 16. E por último o MMLOC da classe no Locflix Swing ficou em 117 e no Locflix JavaFX ficou em 22.

Os resultados das métricas de tamanho para a classe controladora da tela Inicial/Principal de cada *software* são apresentados no gráfico a seguir.

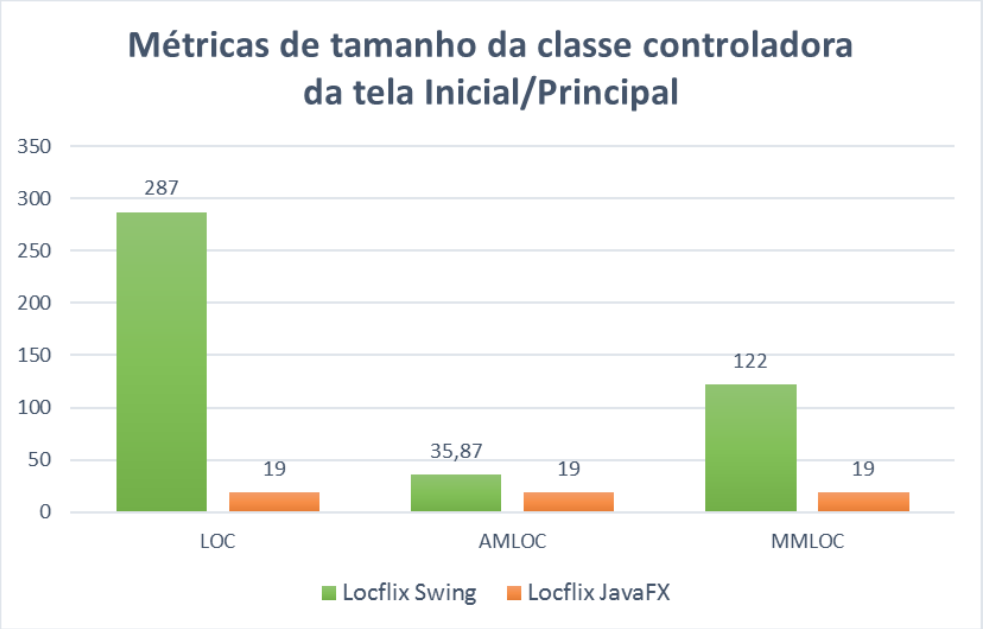


Gráfico 28 - Métricas de tamanho da classe controladora da tela Inicial/Principal.
Fonte: Próprio autor.

O Gráfico 28 mostra que os resultados de LOC nesta classe para os 2 *softwares* tiveram uma grande diferença, pois o LOC desta classe no Locflix Swing ficou em 287 linhas e no JavaFX ficou em apenas 19 linhas, tendo uma diferença de 268 linhas. Esta classe no Locflix JavaFX tem apenas um método, fazendo com que os resultados de AMLOC e MMLOC sejam iguais ao LOC. O AMLOC resultou em 35,87 para o Locflix Swing o MMLOC ficou em 122 linhas.

Os resultados das métricas de tamanho para a classe controladora da tela Configurações de Usuário de cada *software* são apresentados no gráfico a seguir.

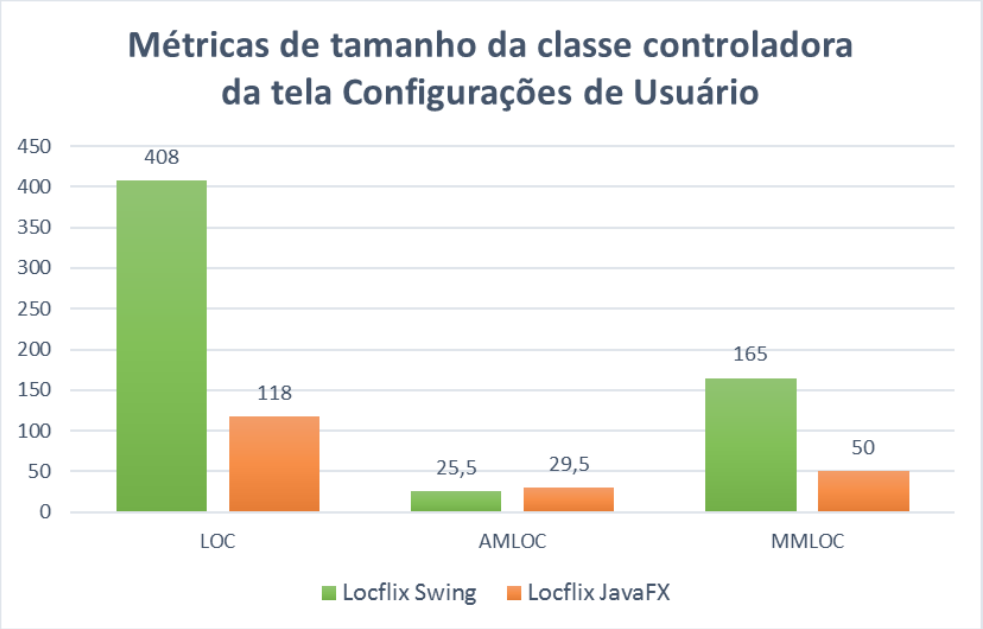


Gráfico 29 - Métricas de tamanho da classe controladora da tela de Configurações de Usuário.
Fonte: Próprio autor.

No Gráfico 29 também é possível ver uma grande diferença de resultados da métrica LOC nesta classe para os *softwares*, pois o resultado desta no Locflix Swing ficou em 408 linhas, e no Locflix JavaFX ficou em 118 linhas, tendo uma diferença de 290 linhas. O resultado da métrica AMLOC desta classe ficou 25,5 no Locflix Swing e 29,5 no Locflix JavaFX. E por fim o resultado da métrica MMLOC no Locflix Swing ficou em 165 linhas e no Locflix JavaFX ficou 50 linhas.

Nos resultados obtidos pelas métricas de tamanho pôde-se observar que no *software* Locflix JavaFX foram utilizados menos linhas de código-fonte que no *software* Locflix Swing para desenvolver as mesmas funcionalidades.

A única classe na qual o Locflix JavaFX teve maior quantidade de linhas ao Locflix Swing foi na classe Dados, na qual os resultados da métrica LOC tiveram uma diferença de 101 linhas.

As diferenças de quantidade de linhas das demais classes em comum do Locflix Swing e Locflix JavaFX variam entre 48 e 290 linhas, sendo menos linhas no projeto Locflix JavaFX.

3.2.3 Métricas de complexidade

Os resultados das métricas de complexidade NOA (número de atributos), NOM (número de métodos), NPA (número de atributos públicos), NPM (número de métodos públicos), ANPM (média do número de parâmetros por método), DIT (profundidade da árvore de herança), NOC (número de filhos), RFC (respostas para uma classe) e ACCM (média de complexidade ciclomática por método) serão separados por 7 gráficos como foi explicado na seção 3.2.1.

Para avaliar cada métrica Meirelles (2013) e Pereira Júnior (2015) sugeriram alguns intervalos e recomendações para interpretar os valores:

- Para a métrica NOA é recomendado que uma classe não tenha um número excessivo de atributos, pois pode indicar que ela tem muitas responsabilidades e apresentar pouca coesão, além de poder estar tratando de vários assuntos diferentes.
- Para a métrica NOM é recomendado que uma classe não tenha um número excessivo de métodos, pois pode ficar mais difícil de ser reutilizada, e a ficar menos coesa.
- Para a métrica NPA os intervalos sugeridos para Java são: até 1 (bom); entre 1 e 9 (regular); de 9 em diante (ruim).
- Para a métrica NPM os intervalos sugeridos para Java são: até 10 (bom); entre 10 e 40 (regular); de 40 em diante (ruim).
- Para a métrica ANPM não é recomendado que um método tenha um número alto de parâmetros pois pode indicar que o mesmo tenha muitas responsabilidades.
- Para a métrica DIT os intervalos sugeridos são: até 2 (bom); entre 2 e 4 (regular); de 4 em diante (ruim). Entretanto, valores baixos para DIT indicam pouco reúso de código via herança.
- Para a métrica NOC, quando seu valor é alto, significa que uma mudança na classe pode ter consequências graves, pois seus métodos são utilizados em muitos filhos.
- Para a métrica RFC, quando seu valor é alto, pode ser uma classe com um número muito grande de métodos, e/ou uma classe bastante dependente de outra(s) classe(s). Com isso pode indicar baixa coesão e alto acoplamento.
- Para a métrica ACCM, quanto maior o valor, mais complexo é o método ou classe analisado (a).

Os resultados das métricas de complexidade dos projetos Locflix Swing e Locflix JavaFX inteiros são apresentados no Gráfico 30.

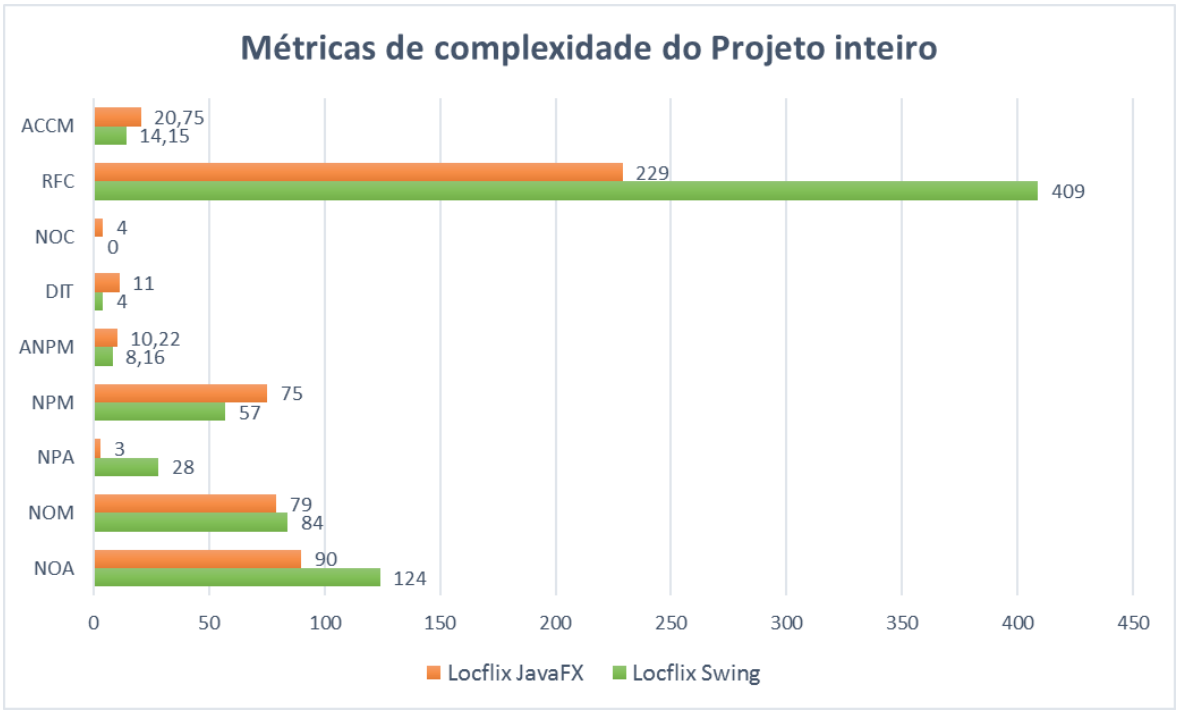


Gráfico 30 - Métricas de complexidade do Projeto inteiro.
Fonte: Próprio autor.

Os resultados obtidos da métrica NOA (número de atributos) dos projetos ficaram 124 no Locflix Swing e 90 no Locflix JavaFX. Já para NOM (número de métodos) ficaram 84 no projeto Locflix Swing e 79 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) foram 28 no projeto Locflix Swing e 3 no Locflix JavaFX. Já para NPM (número de métodos públicos) ficaram 57 no Locflix Swing e 75 no Locflix JavaFX.

Para a métrica ANPM (média do número de parâmetros por método) os resultados foram 8,16 no projeto Locflix Swing e 10,22 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 4 no Locflix Swing e 11 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) que está ligada a métrica DIT, ficaram 0 no Locflix Swing e 4 no Locflix JavaFX.

Para a métrica RFC (respostas para uma classe) resultou 408 no Locflix Swing e 229 no Locflix JavaFX.

Por fim, a métrica ACCM (média de complexidade ciclomática por método) resultou em 14,5 no projeto Locflix Swing e 20,75 no Locflix JavaFX.

Neste resultado dos projetos, foram retornados números altos destas métricas pela ferramenta Analizo, pois foram analisados os projetos inteiros, incluindo classes particulares de cada um e classes em comum de ambos. Várias métricas são calculadas por método, sendo o resultado delas somadas e apresentadas nestes resultados.

No gráfico a seguir, são apresentados os resultados das métricas de complexidade, referente a classe Dados de cada *software*.

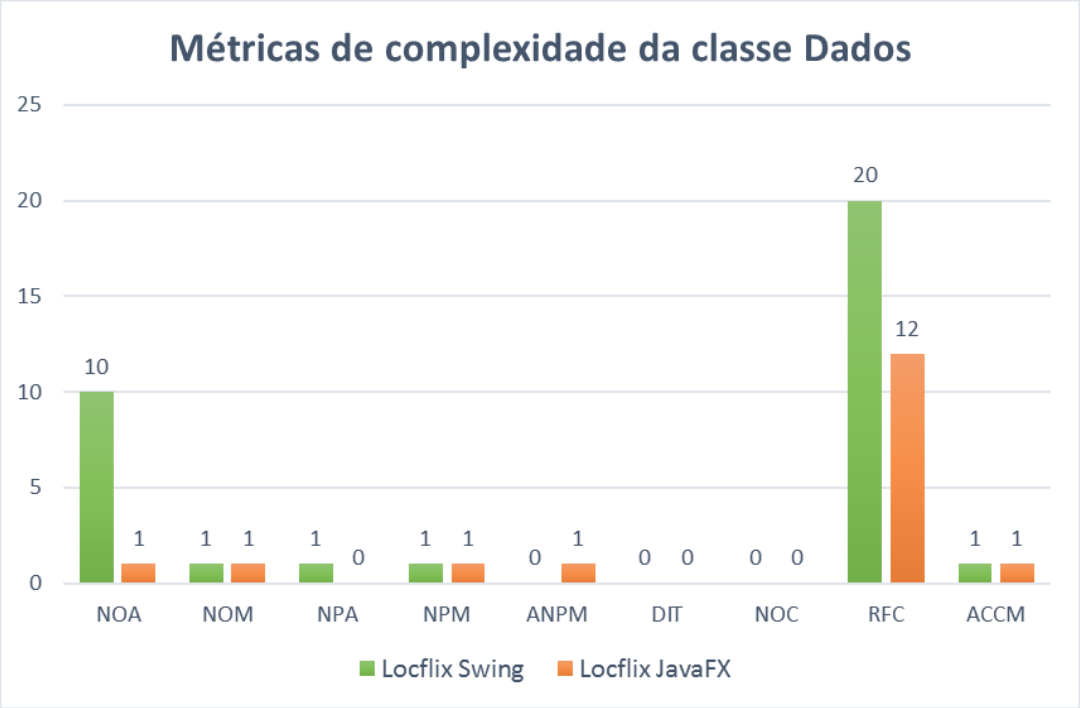


Gráfico 31 - Métricas de complexidade da classe Dados.
Fonte: Próprio autor.

Os resultados da métrica NOA (número de atributos) desta classe ficaram 10 no projeto Locflix Swing e 1 no Locflix JavaFX. Já a métrica NOM (número de métodos) ficou 1 em ambos os projetos.

Os resultados de NPA (número de atributos públicos) foram 1 no projeto Locflix Swing e 0 no Locflix JavaFX. Já para a métrica NPM (número de métodos públicos) desta classe resultou 1 em ambos os projetos.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 0 no Locflix Swing e 1 no Locflix JavaFX.

Os resultados das métricas DIT (profundidade da árvore de herança) e NOC (número de filhos) para a classe Dados de cada projeto, ficaram 0 em ambos projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 20 no Locflix Swing e 12 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 1 em ambos projetos.

Os resultados das métricas de complexidade para a classe controladora de Filme de cada *software* são mostrados no gráfico a seguir.

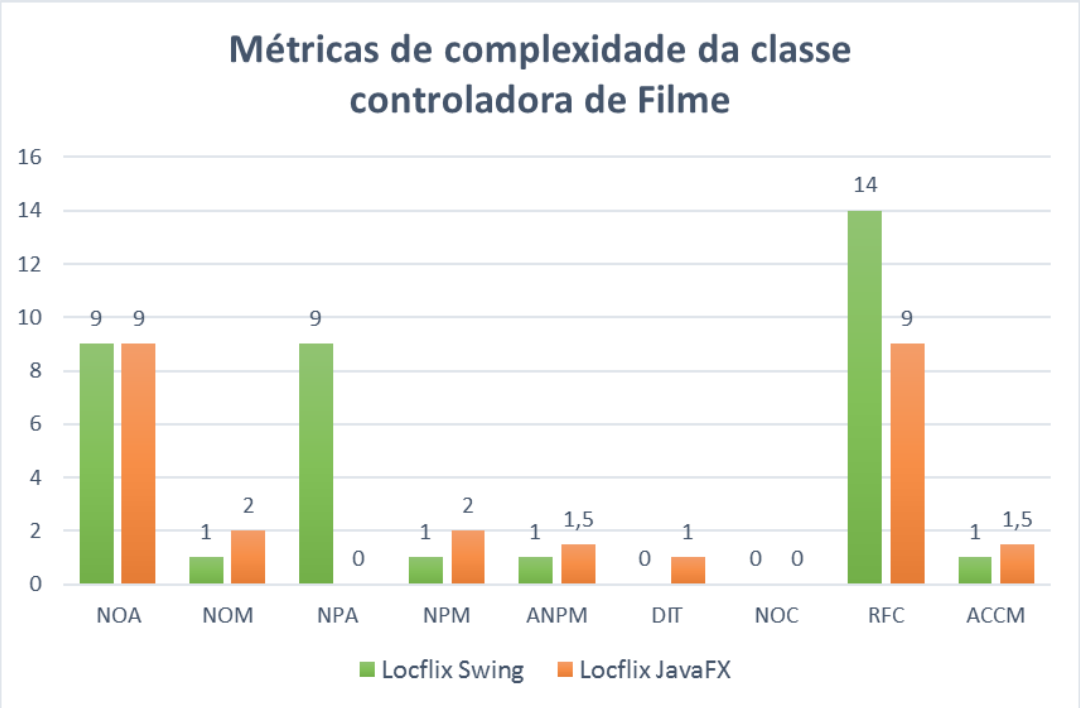


Gráfico 32 - Métricas de complexidade da classe controladora de Filme.
Fonte: Próprio autor.

Os resultados da métrica NOA (número de atributos) desta classe ficaram 9 em ambos projetos. Já a métrica NOM (número de métodos) ficaram 1 no Locflix Swing e 2 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) foram 9 no Locflix Swing e 0 no Locflix JavaFX. Já para a métrica NPM (número de métodos públicos) desta classe resultou 1 no Locflix Swing e 2 no Locflix JavaFX.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 1 no Locflix Swing e 1,5 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 0 no Locflix Swing e 1 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) que está ligada a DIT, ficaram 0 em ambos os projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 14 no projeto Locflix Swing e 9 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 1 no Locflix Swing e 1,5 no Locflix JavaFX.

Os resultados das métricas de complexidade para a classe controladora da tela Informações do Filme de cada *software* são apresentados no gráfico a seguir.

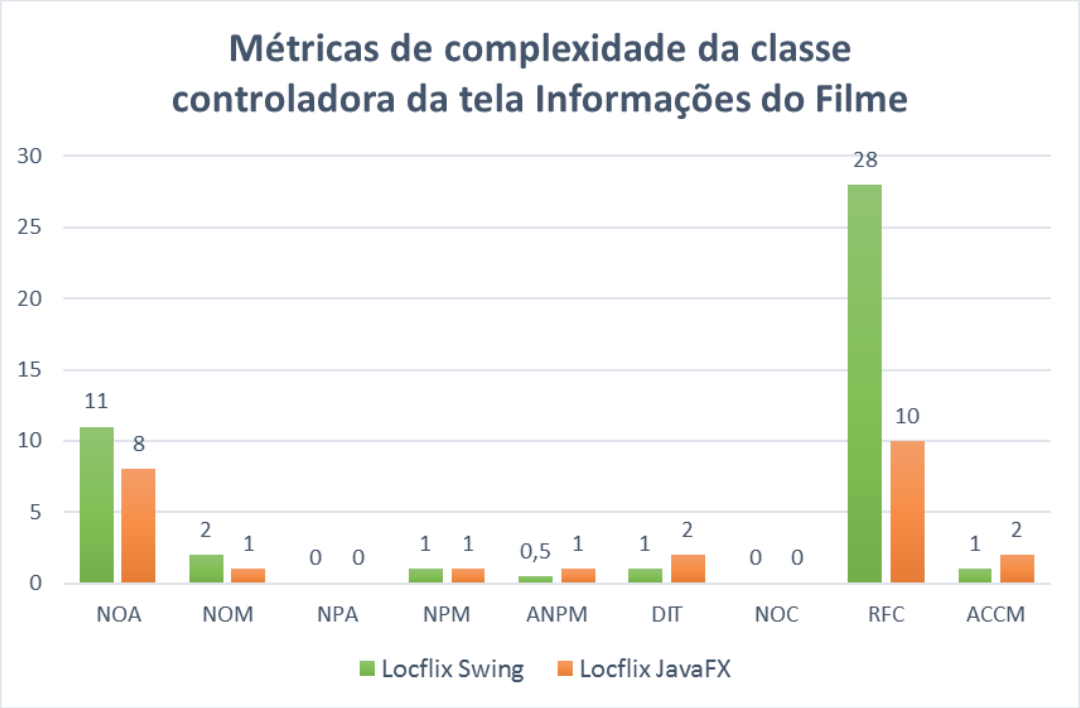


Gráfico 33 - Métricas de complexidade da classe controladora da tela Informações do Filme.
Fonte: Próprio autor.

Os resultados obtidos da métrica NOA (número de atributos) desta classe ficaram 11 no projeto Locflix Swing e 8 no Locflix JavaFX. Já a métrica NOM (número de métodos) ficaram 2 no Locflix Swing e 2 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) ficaram 0 em ambos projetos. A métrica NPM (número de métodos públicos) desta classe resultou 1 em ambos projetos também.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 0,5 no Locflix Swing e 1 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 1 no projeto Locflix Swing e 2 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) ficaram 0 em ambos projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 28 no projeto Locflix Swing e 10 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 1 no Locflix Swing e 2 no Locflix JavaFX.

Os resultados das métricas de complexidade para a classe controladora da tela Lista de Filmes de cada *software* são mostrados no gráfico a seguir.

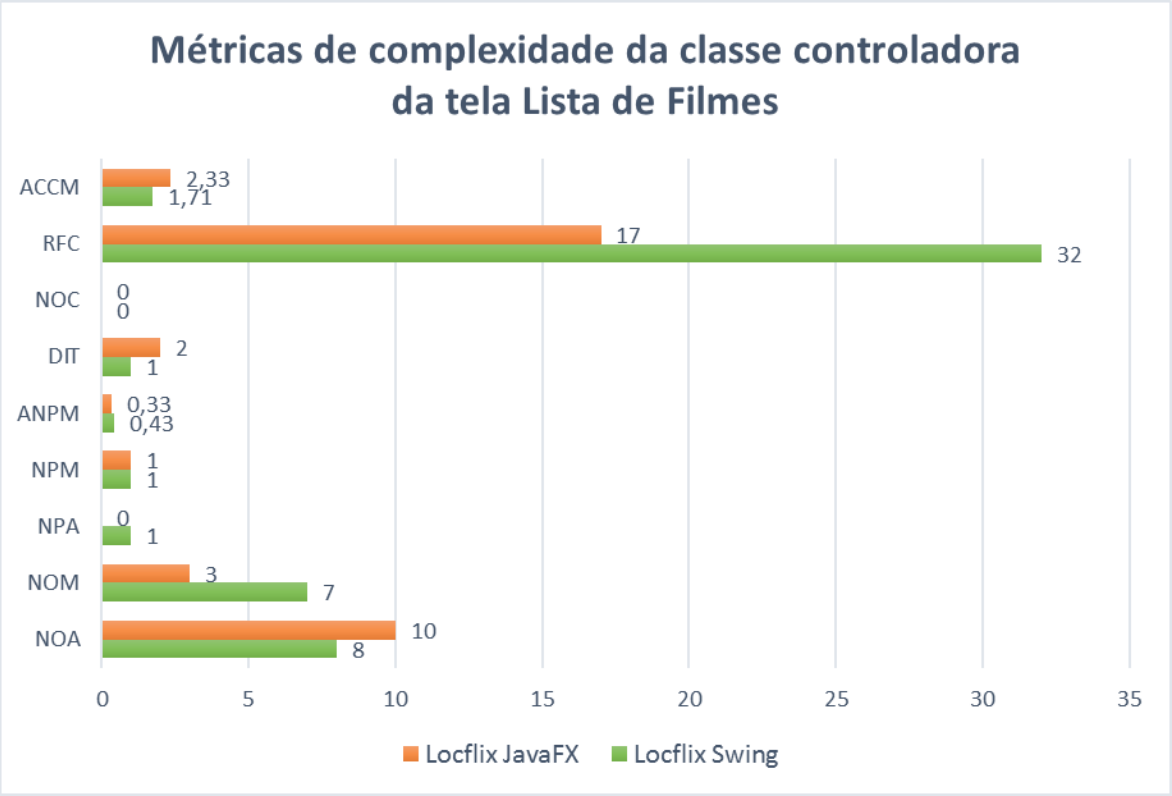


Gráfico 34 - Métricas de complexidade da classe controladora da tela Lista de Filmes.
Fonte: Próprio autor.

Os resultados obtidos da métrica NOA (número de atributos) desta classe ficaram 8 no projeto Locflix Swing e 10 no Locflix JavaFX. Já a métrica NOM (número de métodos) ficaram 7 no Locflix Swing e 3 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) ficaram 1 no projeto Locflix Swing e 0 no Locflix JavaFX. A métrica NPM (número de métodos públicos) desta classe resultou 1 em ambos projetos.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 0,43 no Locflix Swing e 0,33 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 1 no projeto Locflix Swing e 2 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) ficaram 0 em ambos projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 32 no projeto Locflix Swing e 17 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 1,71 no Locflix Swing e 2,33 no Locflix JavaFX.

Os resultados das métricas de complexidade para a classe controladora da tela Inicial/Principal de cada *software* são apresentados no gráfico a seguir.

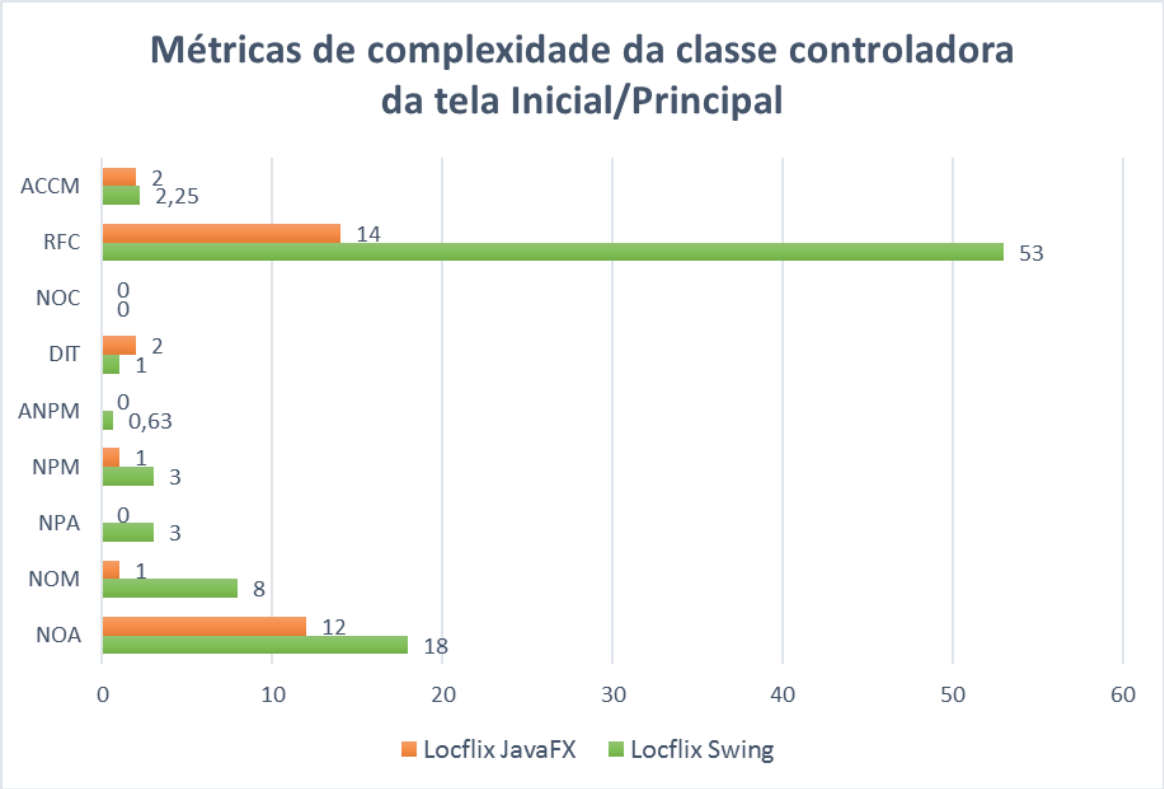


Gráfico 35 - Métricas de complexidade da classe controladora da tela Inicial/Principal.
Fonte: Próprio autor.

Os resultados obtidos da métrica NOA (número de atributos) desta classe ficaram 18 no projeto Locflix Swing e 12 no Locflix JavaFX. Já a métrica NOM (número de métodos) ficaram 8 no Locflix Swing e 1 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) ficaram 3 no projeto Locflix Swing e 0 no Locflix JavaFX. A métrica NPM (número de métodos públicos) resultou em 3 no projeto Locflix Swing e 1 no Locflix JavaFX.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 0,63 no Locflix Swing e 0 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 1 no projeto Locflix Swing e 2 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) ficaram 0 em ambos projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 53 no projeto Locflix Swing e 14 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 2,25 no Locflix Swing e 2 no Locflix JavaFX.

Os resultados das métricas de complexidade para a classe controladora da tela Configurações de usuário de cada *software* são exibidos no gráfico a seguir.

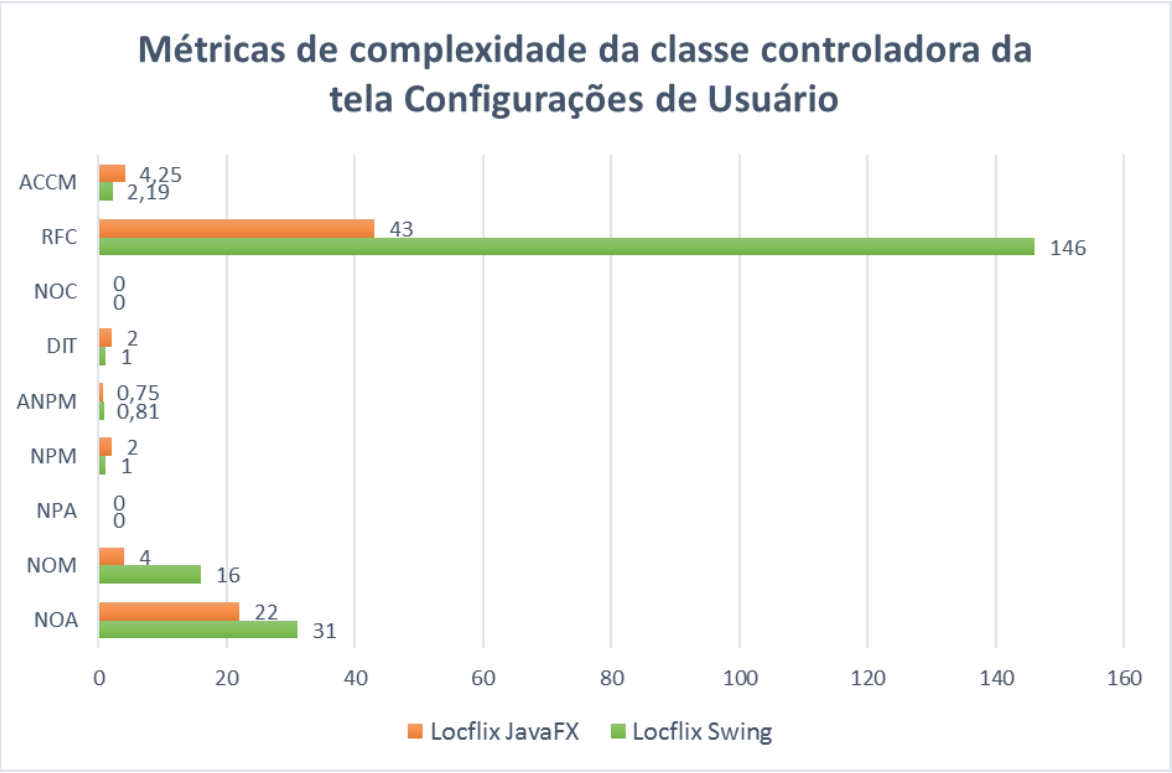


Gráfico 36 - Métricas de complexidade da classe controladora da tela de Configurações de Usuário.
Fonte: Próprio autor.

Os resultados obtidos da métrica NOA (número de atributos) desta classe ficaram 31 no projeto Locflix Swing e 22 no Locflix JavaFX. Já a métrica NOM (número de métodos) ficaram 16 no Locflix Swing e 4 no Locflix JavaFX.

Os resultados de NPA (número de atributos públicos) ficaram 0 em ambos projetos. Já métrica NPM (número de métodos públicos) resultou em 1 no projeto Locflix Swing e 2 no Locflix JavaFX.

Para a métrica ANPM (média do número de parâmetros por método) desta classe, os resultados foram 0,81 no Locflix Swing e 0,75 no Locflix JavaFX.

Os resultados da métrica DIT (profundidade da árvore de herança) ficaram 1 no projeto Locflix Swing e 2 no Locflix JavaFX. Já os resultados da métrica NOC (número de filhos) ficaram 0 em ambos projetos.

Para a métrica RFC (respostas para uma classe) desta classe resultou 146 no projeto Locflix Swing e 43 no Locflix JavaFX. E para métrica ACCM (média de complexidade ciclomática por método) desta classe resultou em 2,19 no Locflix Swing e 4,25 no Locflix JavaFX.

Em geral, nos resultados obtidos pelas métricas NOA (número de atributos) e NOM (número de métodos), pôde-se observar que o projeto Locflix Swing teve um número maior de métodos e atributos na maioria das classes, o que de acordo com Meirelles (2013) pode causar pouca coesão no código-fonte deixando uma classe com muitas responsabilidades.

Através dos resultados da métrica NPA (número de atributos públicos), foi visto que o Locflix Swing possui mais atributos públicos e nos resultados da métrica NPM (número de métodos públicos), o Locflix JavaFX possui mais métodos públicos que o Locflix Swing, mas em todas as classes analisadas, os dois projetos se adequaram nos intervalos sugeridos para essas métricas.

Para métrica ANPM (média do número de parâmetros por método), todas as classes analisadas, tanto no Locflix Swing, quanto no Locflix JavaFX resultaram números baixos para esta (métrica), na maioria delas menor que 1 ou 0, sendo o Locflix JavaFX com números maiores na maioria dos casos.

Nos resultados das métricas DIT (profundidade da árvore de herança) as classes do projeto Locflix JavaFX tiveram um número maior do que as do Locflix Swing, mas também estão de acordo com os intervalos sugeridos. Nos resultados de NOC (número de filhos) somente o projeto Locflix JavaFX teve classes filhas de outras classes mostradas no Gráfico 30. Como os números de DIT e NOC foram mais elevados no projeto usando o JavaFX, de acordo com os estudos sobre estas métricas, é considerado que o projeto Locflix JavaFX é mais complexo que o Locflix Swing. Entretanto valores baixos para DIT indicam pouco reúso de código via herança.

Já nos resultados da métrica RFC (respostas para uma classe), o projeto Locflix Swing demostrou valores maiores para esta (métrica) do que o Locflix JavaFX em todas classes

analisadas, com isso pode indicar baixa coesão e alto acoplamento, o que causa uma difícil manutenção dos métodos.

Nos resultados da métrica ACCM (média de complexidade ciclomática por método) o projeto Locflix JavaFX apresentou valores mais altos do que o Locflix Swing, na maioria das classes analisadas, com isso, é provado que o projeto Locflix JavaFX é mais complexo que o Locflix Swing.

3.2.4 Métricas de acoplamento

Nesta seção serão apresentados os resultados das métricas de acoplamento COF (fator de acoplamento), CBO (acoplamento entre objetos) e ACC (conexões aferentes de uma classe).

Para avaliar cada métrica Meirelles (2013) sugeriu alguns intervalos e recomendações para interpretar os valores:

- Para a métrica COF, altos valores nesta, indica um baixo grau de independência entre os módulos, alta complexidade e difíceis entendimento e manutenção. Os intervalos sugeridos são: até 0,02 (bom); entre 0,02 e 0,14 (regular); de 0,14 em diante (ruim).
- Para a métrica CBO, quanto maior o valor, mais acoplado são os objetos.
- Para métrica ACC, altos valores nesta, uma mudança na classe tem potencialmente mais efeitos colaterais, tornando mais difícil a manutenção.

Os resultados das métricas de acoplamento dos projetos Locflix Swing e Locflix JavaFX inteiros são mostradas no Gráfico 37.

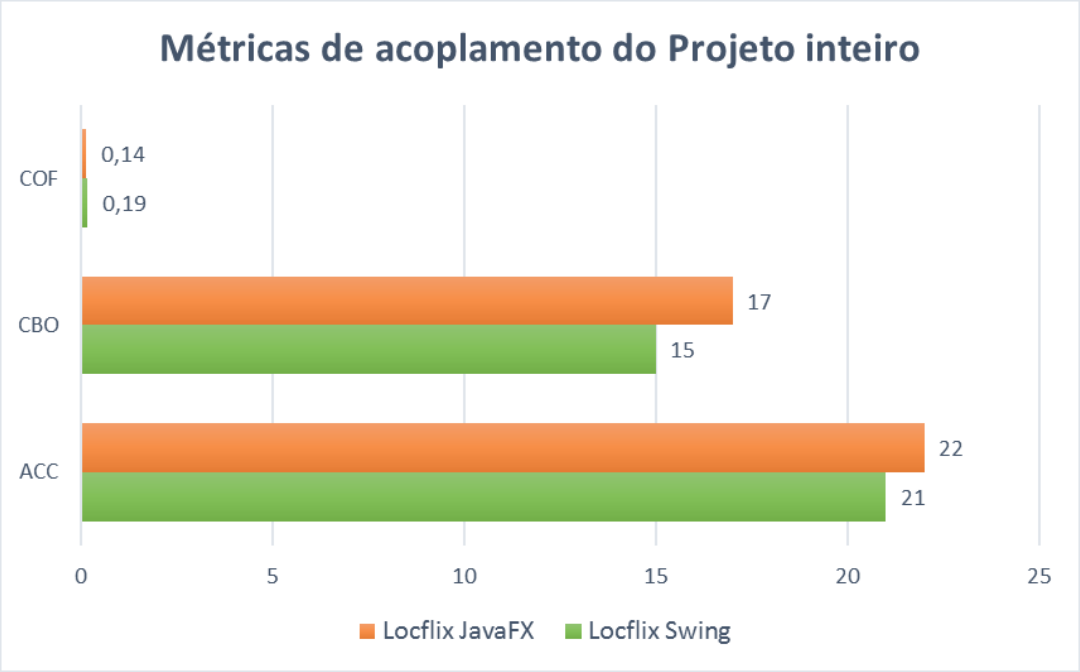


Gráfico 37 - Métricas de acoplamento do Projeto inteiro.
Fonte: Próprio autor.

A métrica COF é calculada somente no projeto inteiro, sendo o projeto Locflix Swing tendo como resultado 0,19 e o Locflix JavaFX com 0,14.

Na métrica CBO o Locflix Swing ficou com 15 e o Locflix JavaFX com 17, e na métrica ACC o projeto Locflx Swing ficou com 21 e o Locflix JavaFX com 22.

3.2.5 Métricas de coesão

Nesta seção serão apresentados os resultados das métricas de coesão LCOM4 (ausência de coesão em métodos) e SC (complexidade estrutural).

Para avaliar cada métrica Meirelles (2013) sugeriu alguns intervalos e recomendações para interpretar os valores:

- Para a métrica LCOM4, quanto maior seu valor, menor é a coesão do *software*.
- Para a métrica SC, altos valores nesta, indica maior complexidade no *software*. Quanto mais complexo for um software, mais difícil será para alterá-lo e evoluí-lo.

Os resultados das métricas de coesão dos projetos Locflix Swing e Locflix JavaFX inteiros são mostradas no Gráfico 38.

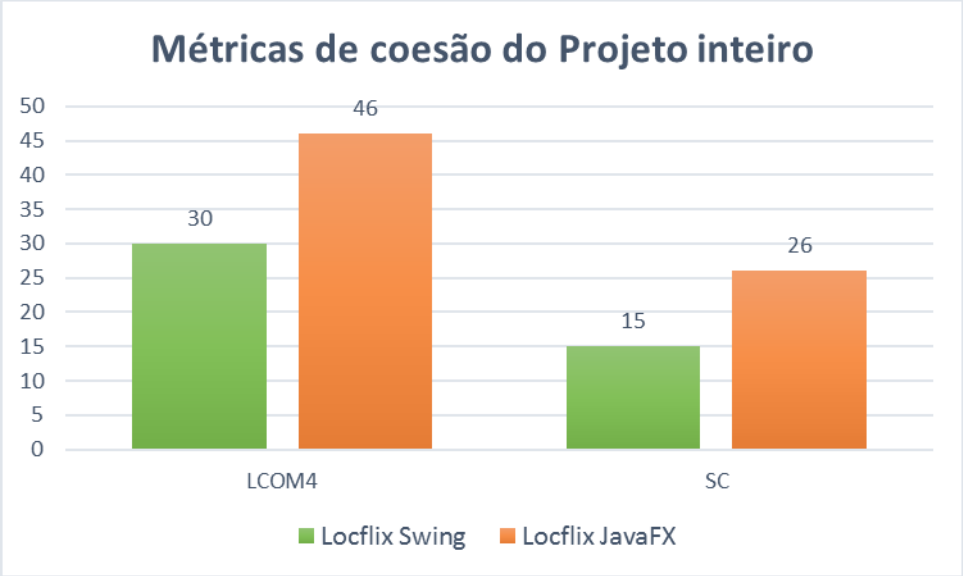


Gráfico 38 - Métricas de coesão do Projeto inteiro.
Fonte: Próprio autor.

Na métrica LCOM4 o Locflix Swing ficou com 30 e o Locflix JavaFX com 46. A partir disso pode-se dizer que o projeto Locflix JavaFX apresenta menos coesão do que o Locflix Swing.

E na métrica SC o projeto Locflx Swing ficou com 15 e o Locflix JavaFX com 26. Pode-se considerar que o projeto Locflix JavaFX tem uma complexidade estrutural maior do que o Locflix Swing.

CONCLUSÃO

Com a elaboração deste trabalho foi possível constatar que os profissionais e estudantes da área de Tecnologia da Informação têm consciência de que a Usabilidade é importante para um *software* e que as duas bibliotecas de interface gráfica do Java, Swing e JavaFX satisfazem às métricas de usabilidade da norma SQuaRE ISO/IEC 25010 e as heurísticas de usabilidade propostas por Jakob Nielsen. Sendo que o *software* feito com a biblioteca JavaFX obteve mais resultados positivos do que o *software* feito com a biblioteca Swing, por ser considerado ter componentes mais atraentes, consistentes e agradáveis para o uso.

Através das análises feitas com as métricas de código-fonte pôde-se ver que para criar as mesmas funcionalidades com a biblioteca JavaFX foram escritas menos linhas de código do que com a biblioteca Swing. Pôde-se ver também que as classes criadas do projeto com a biblioteca JavaFX ficaram menos coesas com valores mais altos de acoplamento e as classes criadas com a biblioteca Swing obtiveram valores altos para coesão e baixos para acoplamento. Classes de valores altos de acoplamento indicam que estão ligadas a outras classes no *software*, tendenciado o mesmo (*software*) a ser menos flexível, mais difícil de se adaptar e modificar. E as classes de valores altos de coesão indicam que todos atributos presentes, se tratam de um mesmo “assunto”.

Portanto, de acordo com os resultados das métricas de código-fonte o *software* feito com a biblioteca Swing apresentou ser mais flexível e mais fácil de se adaptar do que o *software* feito com a biblioteca JavaFX, que apresentou ser mais complexo em diversas métricas.

A partir disso, é possível demonstrar que os *softwares* criados com as bibliotecas Swing e JavaFX, apesar de terem ficado semelhantes visualmente e atendido as métricas e heurísticas de usabilidade, a forma de desenvolvimento para cada um, é bem diferente, e cabe aos engenheiros e desenvolvedores de *software* identificarem qual das bibliotecas será a mais adequada para seus requisitos de projeto de *software.*

TRABALHOS FUTUROS

Como direção para possíveis trabalhos futuros tem-se:

- Análise comparativa entre as bibliotecas de acordo com as métricas de eficiência de desempenho da norma SQuaRE ISO/IEC 25010.
- Análise comparativa entre as bibliotecas de acordo com as métricas de manutenibilidade da norma SQuaRE ISO/IEC 25010.

REFERÊNCIAS

BANSIYA, J.; DAVI, C. **Automated Metrics and Object-Oriented Development: Using QMOOD++ for Object-Oriented Metrics**. 1997.

BECK, Kent. **Smalltalk: best practice patterns**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

CAMPOS, Pedro F.; NUNES, Nuno J. **Introdução ao Java Swing e AWT**. Guia do Laboratório 2006.

CHIDAMBER, S. R.; KEMERER, C. F. **A Metrics Suite for ObjectOriented Design**. IEEE Transactions on Software Engineering, 1994.

DEITEL, Paul; DEITEL, Harvey. **Java – Como programar.** 10ªed. São Paulo: Editora Pearson Education 2016.

Google Trends. **Tendências Java AWT, Java Swing e JavaFX**. Disponível em: <https://trends.google.com.br/trends/explore?q=JavaFX,Java%20Swing,Java%20AWT>. Acesso em 8 de junho de 2017.

GUERRA, Ana Cervigni; COLOMBO, Regina Maria Thienne. **Tecnologia da Informação: Qualidade de Produto de Software**. PBQP Software 2009.

HILDRETH, S. **Buggy Software: Up from a Low Quality Quagmire**. Computer Word, 2005.

IEEE Spectrum. **The 2016 top programming languages**. Disponível em: <http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>. Acesso em 8 de maio de 2017.

ISO 25000. **ISO/IEC 25010**. Disponível em: <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010>. Acesso em 23 de abril de 2017.

JONES, T. C. **Applied Software Measurement: Assuring Productivity and Quality**. McGraw-Hill, New York, 1991.

LORENZ, M.; KIDD, J. **Object-Oriented Software Metrics**. Prentice Hall, 1994.

MEIRELLES, Paulo R. M. **Monitoramento de métricas de código-fonte em projetos de software livre**. São Paulo, 2013.

NIELSEN, Jakob. **Usability engineering**. EUA: Morgan Kaufmann, 1993.

NIELSEN, Jakob. **10 Usability Heuristics for User Interface Design**. Disponível em: <https://www.nngroup.com/articles/ten-usability-heuristics/>. Acesso em 10 de agosto de 2017.

OLIVEIRA, Bruno. **JavaFX Interfaces com qualidade para aplicações desktop**. Casa do Código 2014.

OLIVEIRA FILHO, Carlos M. **Kalibro: interpretação de métricas de código-fonte**. São Paulo, 2013.

Oracle. **JavaFX Developer Home**. Disponível em: <www.oracle.com/technetwork/pt/java/javafx/overview/index.html>. Acesso em 10 de junho de 2017.

Oracle. **The Java Programming Language Platforms**. Disponível em: <https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html>. Acesso em 10 de agosto de 2017.

Oracle. **Java Platform Standard Edition 8 API Specification**. Disponível em: <https://docs.oracle.com/javase/8/docs/api/>. Acesso em 15 de agosto de 2017.

PEREIRA JÚNIOR, Marcos Ronaldo. **Estudo de métricas de código-fonte no sistema Android e seus aplicativos**. Brasília, 2015.

PLATZ, Wolfgang. **Software Fail Watch: 2016 in Review**. Tricents 2017.

PRESSMAN, Roger S. **Engenharia de Software – Uma abordagem profissional.** 7ªed. Porto Alegre: AMGH Editora 2011.

ROSENBERG, L. H.; HYATT L.E. **Software Quality Metrics for ObjectOriented Environments**. Crosstalk - the Journal of Defense Software Engineering. 1997.

SHARBLE, R.; COHEN, S. **The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods**. Software Engineering Notes. 1993.

SOMMERVILLE, Ian. **Engenharia de Software.** 9ªed. São Paulo: Editora Pearson 2011.

STANDARDIZATION, International Organization for. **ISO/IEC 25010 Systems and software Quality Requirements and Evaluation (SQuaRE)**. 1ª ed. Switzerland: International Organization for Standardization and and International Electrotechnical Commission, 2011.

STANDARDIZATION, International Organization for. **ISO/IEC 25023 Systems and software Quality Requirements and Evaluation (SQuaRE)**. 1ª ed. Switzerland: International Organization for Standardization and and International Electrotechnical Commission, 2011.

APÊNDICE 1 – QUESTIONÁRIO

Análise de Usabilidade entre as bibliotecas de interface gráfica Swing e JavaFX do Java

Olá, meu nome é Amanda Neves do Carmo, sou graduanda do curso Ciência da Computação na Rede de Ensino Doctum.

Este questionário tem o intuito de coletar dados para realizar meu Trabalho de Conclusão de Curso, que tem como objetivo analisar duas bibliotecas de interface gráfica do Java, o Swing e o JavaFX e verificar quais destas atendem melhor a parte de usabilidade, tendo como base a norma ISO 25010 e as heurísticas de Nielsen.

Para respondê-lo não é preciso ter conhecimentos técnicos de Java ou bibliotecas de interface gráfica. É só executar e utilizar as duas aplicações que estão no link abaixo:
https://drive.google.com/open?id=0B2RaN2iYBAKQbURZekM4SFVaZVU
Neste link, encontram-se também os manuais para a utilização das aplicações e outras informações à respeito.

Todos os dados coletados serão utilizados somente para fins acadêmicos e serão divulgados para a conclusão do trabalho.

Peço a colaboração de todos que receberem este questionário, sua opinião é muito importante para a conclusão deste trabalho. Desde já, obrigada.

*Obrigatório

Se desejar receber o resultado do trabalho quando o mesmo estiver disponível, deixe seu e-mail.

Não é obrigatório.

Sua resposta

Perfil do Participante

Qual seu nível de formação acadêmica? *

- ◯ Ensino médio ou fundamental
- ◯ Técnico completo ou incompleto
- ◯ Ensino superior completo ou incompleto
- ◯ Especialização completa ou incompleta
- ◯ Mestrado completo ou incompleto
- ◯ Doutorado completo ou incompleto
- ◯ Outro: _____

Há quanto tempo você exerce qualquer atividade relacionada à área da computação? *

- ◯ Nunca exerci
- ◯ Menos de 1 ano
- ◯ De 1 a 5 anos
- ◯ De 5 a 10 anos
- ◯ Mais de 10 anos

Há quanto tempo você faz uso de qualquer dispositivo relacionado à tecnologia? *

Exemplo: Smartphones, computadores, vídeo games, entre outros.

- ○ Menos de 1 ano
- ○ De 1 a 5 anos
- ○ De 5 a 10 anos
- ○ Mais de 10 anos

Você é ou já foi um(a) desenvolvedor(a) Java? *

- ○ Sim
- ○ Não

Na sua opinião, qual a importância da usabilidade em um software? *

	1	2	3	4	5	
Pouco importante	○	○	○	○	○	Muito importante

Análise de Usabilidade entre as bibliotecas de interface gráfica Swing e JavaFX do Java

*Obrigatório

Informações sobre as questões

Após utilizar as aplicações disponibilizadas juntamente com este questionário, responda as perguntas com sinceridade.

Cada pergunta a seguir descreve uma métrica de usabilidade de software, tendo como base as heurísticas de Nielsen e a norma ISO 25010 como foi dito na sessão anterior.
Após a descrição da métrica, em alguns casos, terão imagens de algumas partes das aplicações que atendem e exemplifiquem a esta (métrica) lembrando que, são só alguns exemplos, nas interfaces originais podem ter mais partes que atendem a mesma. E posteriormente você dará sua opinião, se a interface de cada biblioteca atendeu satisfatoriamente a métrica.

Questões

Visibilidade do estado do sistema *

O sistema informa ao usuário o que está acontecendo em cada interação. Isso é feito por meio de feedbacks instantâneos (a cada click) para orientá-lo. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

• Conforme o usuário vai adicionando/cancelando os filmes, a quantidade no botão "Minha lista de filmes" vai atualizando.

• Quando o usuário se cadastra, o botão "Cadastre-se" atualiza para o nome do próprio.

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Correspondência entre o sistema e o mundo real *

A comunicação da aplicação "fala" a linguagem do usuário e não é orientada ao sistema, ou seja, não usa linguagens técnicas ou termos que são de conhecimento específico. Todas as nomenclaturas são contextualizadas e coerentes com o modelo mental do usuário. Isso também é aplicado à ícones e imagens ilustrativas. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

• Termos como “Sinopse” está descrito no botão como “Mais”, pois como já tem uma parte da sinopse visível na interface principal, a palavra "Mais" é mais comum e desperta curiosidade ao usuário.

• Termos como "Lista de filmes" está como "Minha lista de filmes", pois o usuário se sente mais confortável em acessar a aplicação, como se fosse somente dele.

• Os ícones dos botões principais estão coerentes ao contexto do mesmo para o usuário.

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Liberdade de controle fácil pro usuário *

Permite liberdade ao usuário das decisões e ações que podem ser tomadas na aplicação, exceto a regras de negócio ou que interfere outra funcionalidade. Permite desfazer ou refazer alguma ação no sistema e retornar ao ponto anterior quando estiver perdido ou em situações inesperadas. Não força o usuário a fazer algo e também não toma a decisão por ele. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

• Quando o usuário clica em "Adicionar" um filme, imediatamente o botão muda para "Cancelar", caso ele mudar de ideia, ele pode desfazer/desistir clicando em cancelar.
• No caso do JavaFX o usuário pode expandir/esconder uma ou mais sessões de filmes para uma melhor navegação clicando nos títulos (já é um objeto nativo da biblioteca).

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

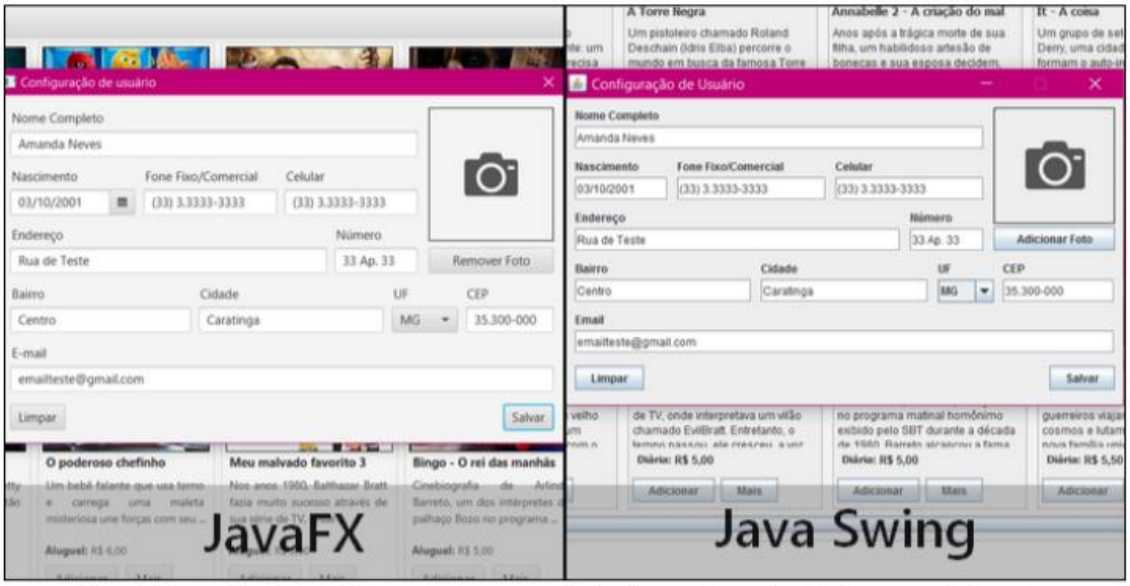
Consistência e padrões *

Mantém uma consistência e padrão visual (texto, cor, desenho do elemento, som e etc). Fala a mesma língua o tempo todo, e nunca identifica uma mesma ação com ícones ou fontes diferentes. Trata coisas similares da mesma maneira, facilitando a identificação do usuário e ensinando-o a usar o sistema. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Prevenção de erros *

Não deixa o usuário errar sem explicar previamente o motivo do erro. Têm interfaces com características que não permite o usuário errar, como máscaras (data, telefone, CEP, entre outras) nos campos de cadastro e mensagens explicativas caso o usuário tenta fazer algo que está contra as regras de negócio da aplicação. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?



- Na configuração de usuário, campos como data de nascimento, telefones e CEP, contém máscaras para evitar possíveis erros do usuário de formatação dos campos.
- No campo da data de nascimento já sugere a data mínima de nascimento do usuário seguindo as regras da aplicação (o usuário deve ter pelo menos 16 anos).
- Ao tentar salvar, caso alguns campos considerados importantes pelas regras da aplicação não forem preenchidos ou forem preenchidos incorretamente, algumas mensagens (alertas) serão mostrados para usuário indicando quais os campos precisam ser preenchidos ou corrigidos.

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Reconhecimento em vez de memorização (aprendizagem) *

Todos os meios de acesso à funcionalidades da aplicação estão de fácil acesso sem precisar acionar a memorização do usuário, permitindo o aprendizado em pouco tempo de uso e proporciona eficácia e eficiência em situações de pressa (afobação ou emergência).Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Flexibilidade e eficiência de uso (operabilidade) *

A aplicação é ágil para usuários avançados e fácil de utilizar pelos usuários leigos. O uso de atalhos de teclados, preenchimento automático a partir de dados anteriores e máscaras de campos são exemplos de itens que aprimoram a eficiência do sistema com flexibilidade. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

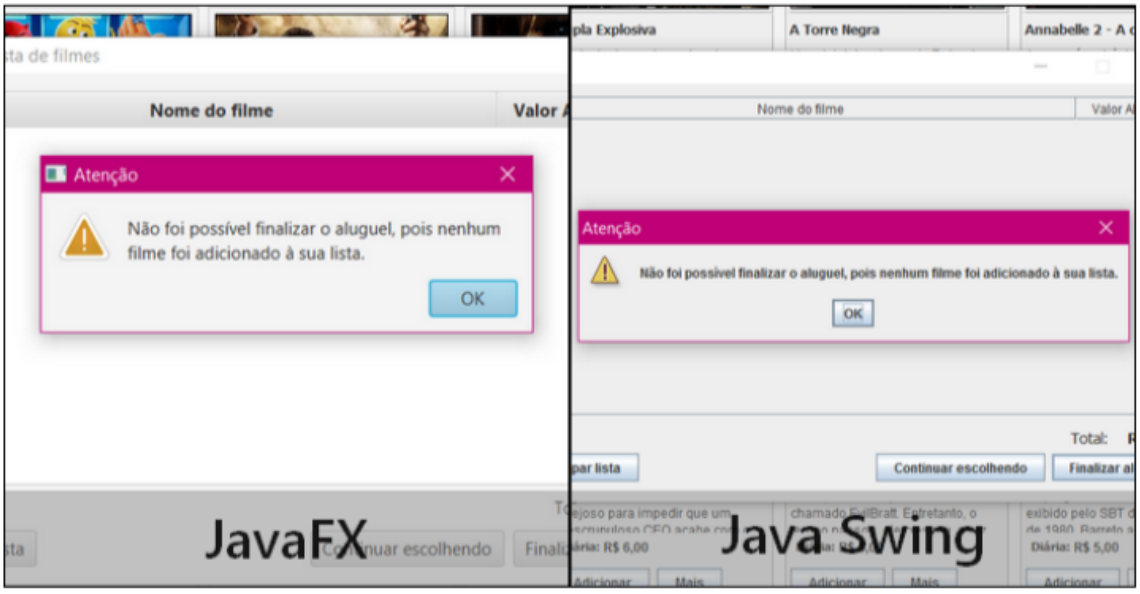
Estética e design minimalista *

A aplicação não usa desnecessariamente excessos de cores e elementos visuais que confundem o usuário. Dialoga de forma simples e direta, com um layout mais limpo, com diálogos naturais, de fácil entendimento e aparecem em momentos necessários. Tem uma interface que permite uma interação agradável e satisfatória para o usuário. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Recuperação de erros *

A aplicação têm mensagens de erro claras, com textos simples e diretos, não intimidando o usuário e sim ajudando a reconhecerem, diagnosticarem e conduzindo à possíveis soluções do erro. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?



	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Reconhecimento de adequação *

A aplicação permite ao usuário reconhecer se a mesma é adequada ou não para suprir todas suas necessidades.Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Acessibilidade *

A aplicação permite o uso de usuários com determinadas características e deficiências (usuários com limitações motoras, auditivas ou visuais). Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Ajuda e documentação *

A aplicação é intuitiva e clara para o usuário e quase não é necessário o uso de documentações complementares. Mesmo assim é preciso ter documentações (manuais e explicações sobre o uso) ao alcance do usuário. Você concorda que as interfaces gráficas criadas pelas bibliotecas JavaFX e Java Swing atendem satisfatoriamente esta métrica?

	Discordo totalmente	Discordo parcialmente	Indiferente	Concordo parcialmente	Concordo totalmente
JavaFX	○	○	○	○	○
Swing	○	○	○	○	○

Após conhecer um pouco das interfaces gráficas que as bibliotecas proporcionam e das métricas de usabilidade mencionadas neste questionário, de forma geral, qual delas (bibliotecas) você achou que proporcionou uma interface mais agradável para o uso? *

- ○ JavaFX
- ○ Swing

Você tem considerações à fazer em relação à biblioteca JavaFX?

Sua resposta

Você tem considerações à fazer em relação à biblioteca Swing?

Sua resposta

APÊNDICE 2 - MANUAL LOCFLIX BIBLIOTECA SWING

Manual de utilização do Locflix

Locflix Swing

Manual básico explicativo da aplicação Locflix Swing para o usuário.

Sumário

Introdução

Este Manual foi planejado para servir como um guia de utilização da aplicação Locflix que tem como objetivo simular um aluguel de filmes. A leitura do usuário é recomendada para conhecer melhor o funcionamento geral da mesma (aplicação).

Informação Importante

O Locflix foi projetado e desenvolvido para fins acadêmicos, portanto todos dados que forem cadastrados nesta aplicação não serão salvos, ou seja, não existe um banco de dados para armazenar dados. Os dados informados na aplicação só estarão visíveis e disponíveis enquanto a mesma estiver em execução.

Compatibilidade

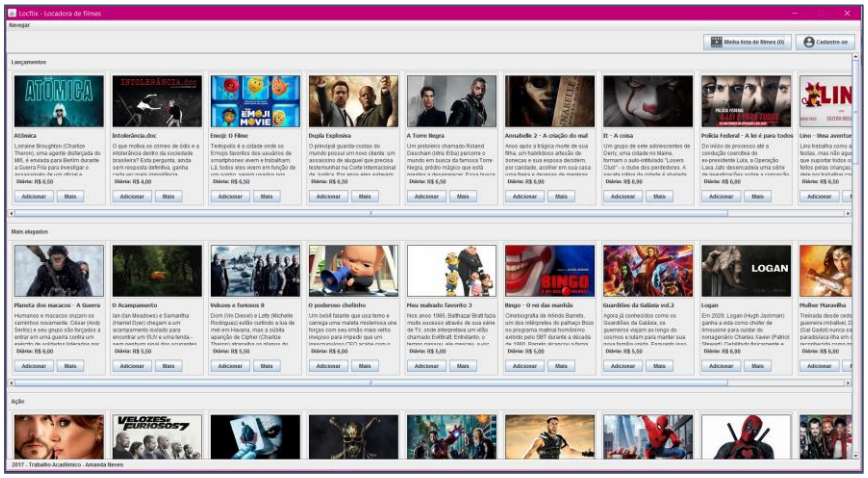
Uma característica importante do Locflix, é a compatibilidade com diversos Sistemas Operacionais, no qual ele pode ser acessado pelo Linux, Windows, entre outros. E não há necessidade de conexão à internet para o acesso, a partir que se tenha o executável (extensão .jar) e as bibliotecas auxiliares que ficam dentro de uma pasta chama "lib".

Estrutura da aplicação

Tela Principal

A interface gráfica da tela principal foi desenhada para proporcionar acesso rápido à todas funcionalidades da aplicação. Nela você encontrará todos os filmes disponíveis separados em categorias, proporcionando uma livre navegação entre eles, possibilitando ver mais informações sobre os mesmos e adicionar/cancelar à lista de filmes à serem alugados.

Nela também são visíveis os botões de acesso para as principais funcionalidades que são a Configuração de usuário (Cadastre-se) e a lista de filmes escolhidos (Minha lista de filmes). Além do menu "Navegar" que também possibilita acesso a estas funcionalidades.



Categorias dos filmes

A aplicação contém várias categorias de filmes nas quais estão em formato de seções. As categorias são:

- Lançamentos (alguns filmes que esteve em cartaz nos cinemas no período entre agosto e setembro de 2017)
- Mais alugados (suposições)
- Ação
- Comédia
- Documentários
- Drama
- Ficção científica
- Romance
- Terror

Filme

Cada categoria contém vários filmes para o usuário alugar. Cada filme contém uma imagem que representa a capa ou alguma cena marcante do mesmo. Logo abaixo o título, uma breve sinopse e dois botões:

- Adicionar: ao clicar neste, o filme é adicionado para a lista de filmes escolhidos, e imediatamente sua legenda é alterada para "Cancelar", caso o usuário queira desfazer sua escolha, o filme é retirado da lista de filmes escolhidos.
- Mais: ao clicar neste, abrirá uma tela com a foto da capa do filme e várias outras informações como: sinopse completa, data de lançamento, valor do aluguel, classificação e duração do filme.

Exemplo de um filme no Locflix

Menu Principal

No menu principal, contém a opção "Navegar", na qual contém dois submenus:

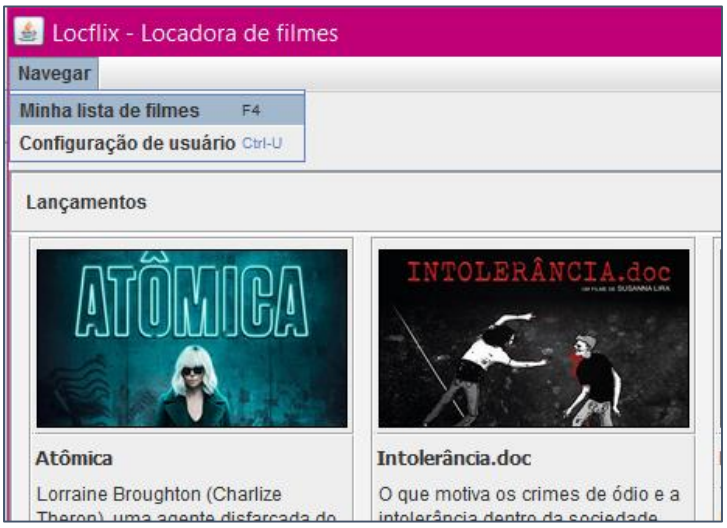
Minha lista de filmes – Uma tela que contém uma tabela com todos os filmes escolhidos e informações de valores. Logo abaixo desta tabela contém o valor total dos filmes e três opções de navegação:

- Limpar lista: cancela todos os filmes escolhidos para aluguel.
- Continuar escolhendo: fecha a tela com a tabela e permite a continuação para escolha dos filmes.
- Finalizar aluguel: Finaliza aluguel.

Configuração de usuário – Uma tela que contém vários campos para o usuário se cadastrar. Contém dois botões:

- Limpar: limpa todos os dados informados na tela pelo usuário.
- Salvar: valida e salva todos os dados informados pelo usuário.

Nos submenus também mostram as teclas de atalho que podem ser pressionadas para acessar estas telas a qualquer momento durante o uso da aplicação.



Menu principal

Botões na barra superior

Estes botões possibilitam mais uma opção (além das teclas de atalho) ao acesso rápido para as telas: lista de filmes escolhidas pelo usuário e da configuração de usuário.

O botão "Minha lista de filmes (0)" é atualizado a cada filme adicionado ou cancelado pelo usuário. O número ao lado da legenda representa a quantidade atual de filmes escolhidos pelo usuário.

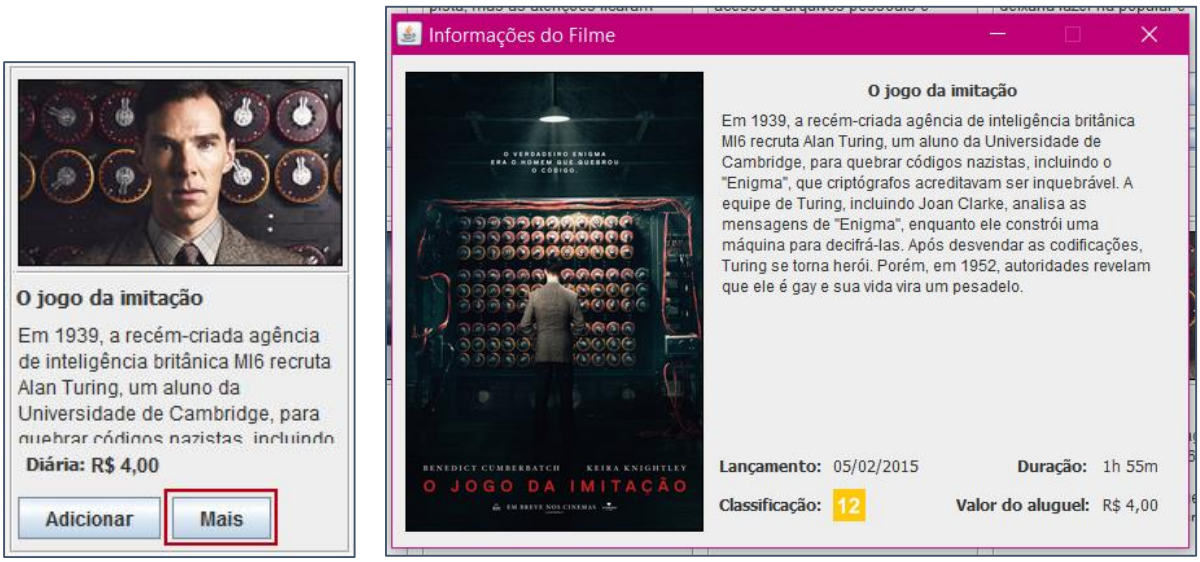
Botões na barra superior

Passo a passo

Nesta seção será mostrado passo a passo de várias funcionalidades da aplicação.

Como ver informações completas do filme

Na interface principal, em cada filme existe um botão com legenda "Mais". Ao clicar neste, abrirá uma tela com várias informações do filme como: imagem da capa do filme, sinopse completa, valor do aluguel, data de lançamento, classificação e duração.



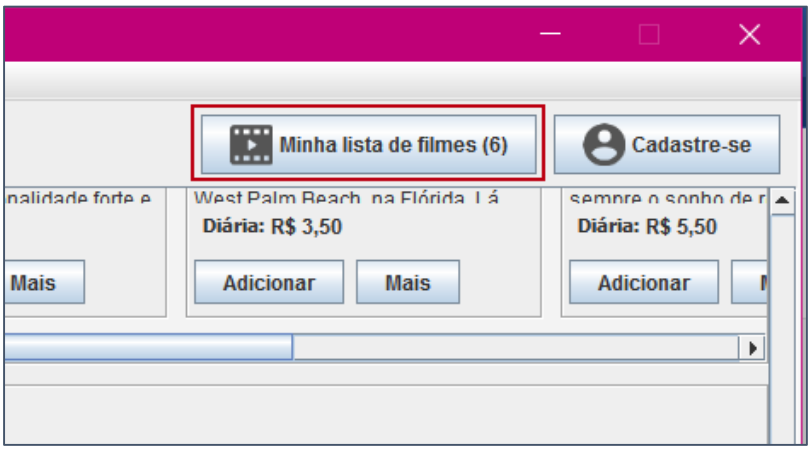
Botão "Mais" e tela Informações do Filme

Como escolher um ou mais filmes

Na interface principal, em cada filme existe um botão "Adicionar", ao clicar neste, imediatamente a legenda do botão "Adicionar" é alterada para "Cancelar" caso o usuário queira desfazer a ação anterior. Pode-se adicionar vários filmes.

O número do botão superior "Minha lista de filmes (0)" sempre é atualizado para a quantidade atual de filmes escolhidos a medida em que o usuário adiciona ou cancela um filme.

Exemplo de alguns filmes adicionados e outro não

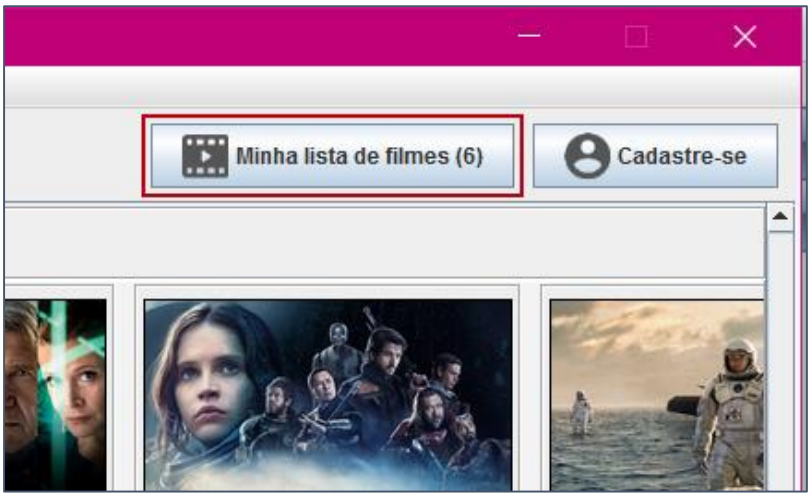


Botão "Minha lista de filmes" com a quantidade de filmes escolhidos

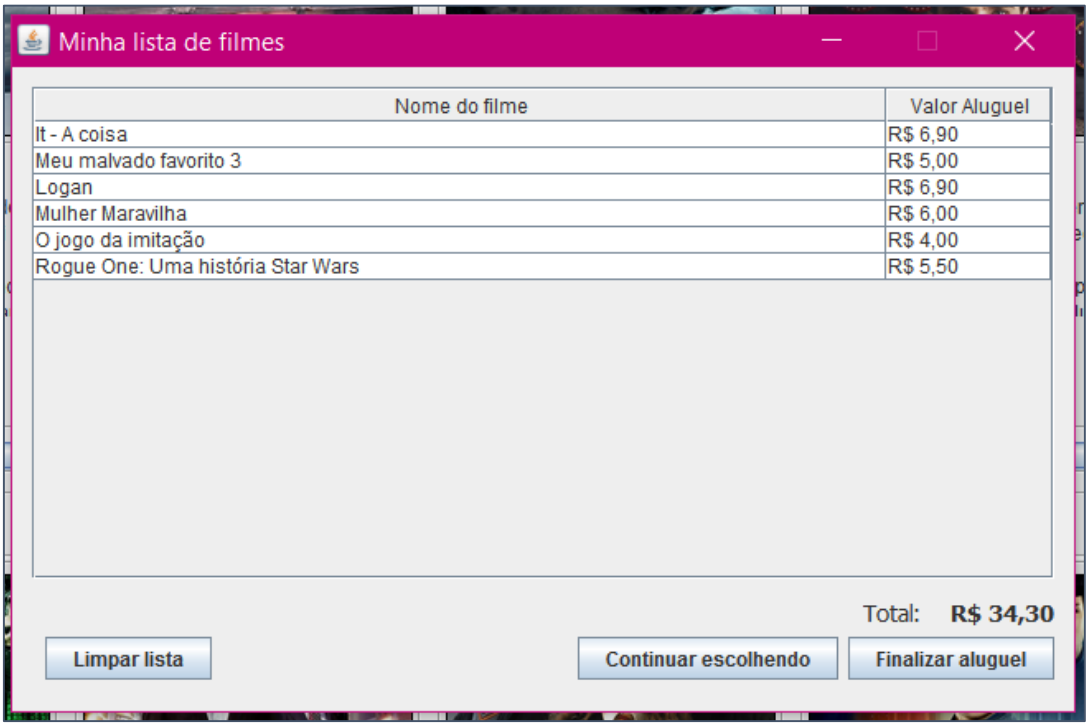
Como saber quais filmes foram escolhidos

No lado superior esquerdo da aplicação existe um botão "Minha lista de filmes (x)", ao clicar neste, abrirá uma tela com uma tabela com todos os nomes dos filmes escolhidos e valores.

Para acessar esta tela também pode-se pressionar a tecla F4 ou acessar o menu Navegar > Minha lista de filmes.



Botão "Minha lista de filmes" com a quantidade de filmes escolhidos



Tela "Minha lista de filmes"

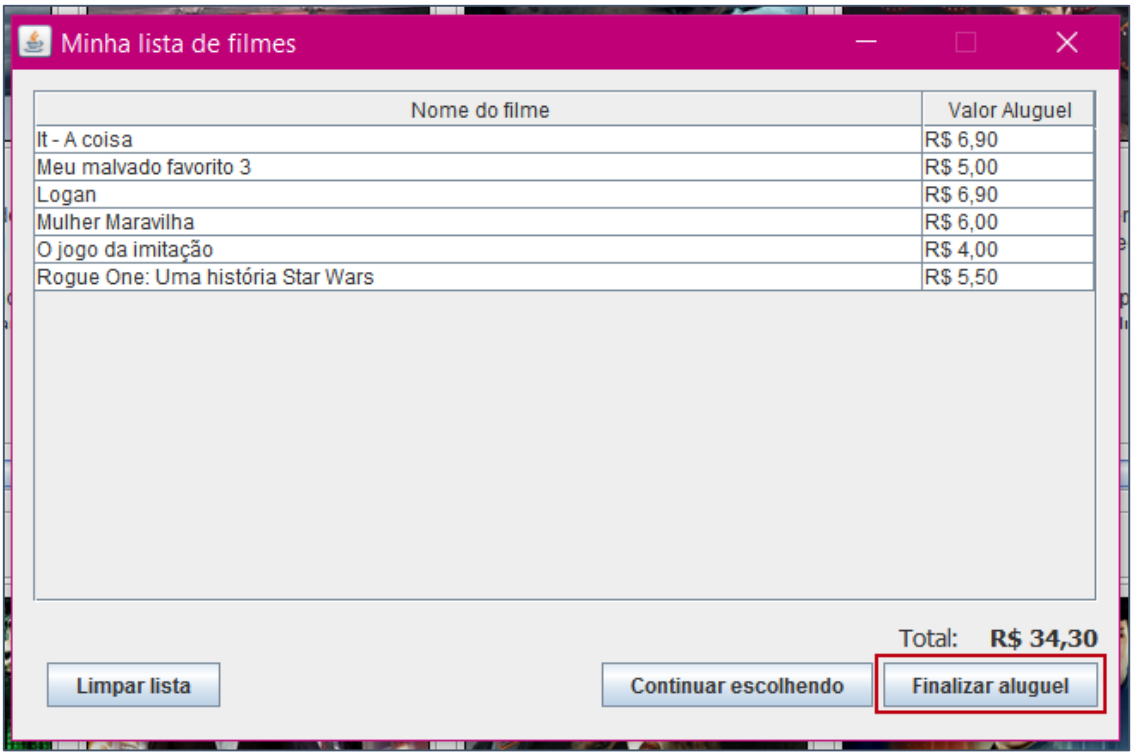
Como finalizar o aluguel dos filmes

Para finalizar o aluguel dos filmes é necessário que se tenha pelo menos um filme escolhido pelo usuário e que o mesmo (usuário) esteja cadastrado na aplicação.

Para saber como cadastra/configura o usuário, a explicação está no tópico posterior a este.

Após a escolha dos filmes pode-se clicar no botão superior esquerdo "Minha lista de filmes (x)", abrirá uma tela com a tabela de todos os filmes escolhidos e valores, logo abaixo terá um botão com a legenda "Finalizar aluguel", ao clicar neste, irá aparecer a mensagem: "Aluguel realizado com sucesso". Caso não aparecer esta mensagem, irá aparecer outra instruindo o usuário para o que se deve ser feito.

Para acessar esta tela também pode-se pressionar a tecla F4 ou acessar o menu Navegar > Minha lista de filmes.



Botão "Finalizar aluguel" na tela "Minha lista de filmes"

Como cadastrar/configurar o usuário

Para acessar a tela de configuração de usuário pode-se clicar no botão superior esquerdo "Cadastre-se", ou pressionar as teclas Ctrl+U ou acessar pelo menu Navegar > Configuração de usuário. Abrirá um formulário com vários campos a serem preenchidos.

Campos obrigatórios:

- Nome completo
- Data de nascimento (o usuário deve ter no mínimo 16 anos de idade)
- Telefone Fixo/Comercial (caso não houver pode-se colocar o celular)
- Endereço
- Número
- Bairro
- Cidade
- UF
- CEP

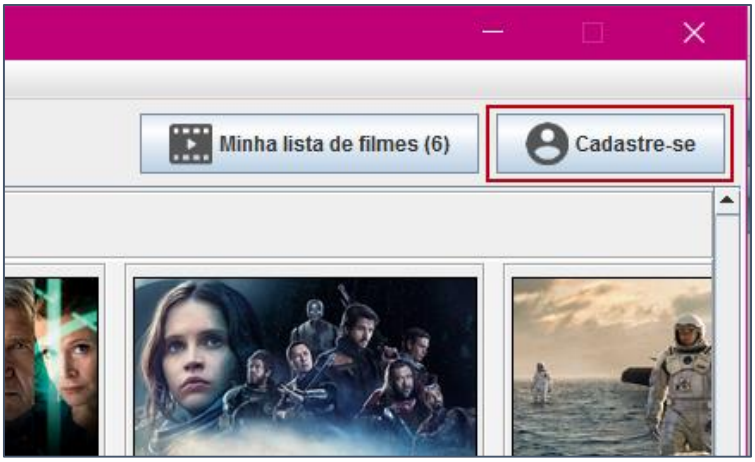
Campos opcionais:

- Foto

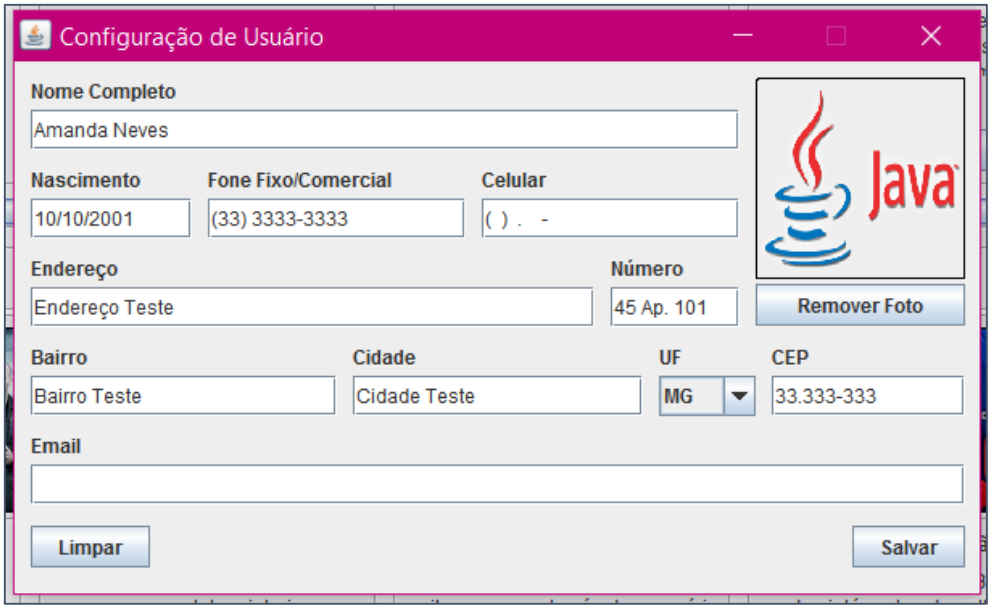
- E-mail

Para adicionar uma foto, pode-se clicar no botão "Adicionar Foto" que fica abaixo de um ícone com uma câmera, escolha a foto de sua preferência e clique em Salvar. Imediatamente a legenda do botão muda para "Remover Foto" caso queira remover.

Após serem preenchidos os campos, pode-se clicar no botão "Salvar", no qual os dados serão validados e salvos.



Botão "Cadastre-se" na tela principal



Tela de configuração de usuário com alguns dados preenchidos

Dúvidas e sugestões

Caso tenha qualquer dúvida ou sugestão referente à aplicação Locflix Swing ou a este manual, entre em contato:

E-mail: amandaneves.carmo@gmail.com

APÊNDICE 3 - MANUAL LOCFLIX BIBLIOTECA JAVAFX

Manual de utilização do Locflix

Locflix JavaFX

Manual básico explicativo da aplicação Locflix JavaFX para o usuário.

Sumário

Introdução

Este Manual foi planejado para servir como um guia de utilização da aplicação Locflix que tem como objetivo simular um aluguel de filmes. A leitura do usuário é recomendada para conhecer melhor o funcionamento geral da mesma (aplicação).

Informação Importante

O Locflix foi projetado e desenvolvido para fins acadêmicos, portanto todos dados que forem cadastrados nesta aplicação não serão salvos, ou seja, não existe um banco de dados para armazenar dados. Os dados informados na aplicação só estarão visíveis e disponíveis enquanto a mesma estiver em execução.

Compatibilidade

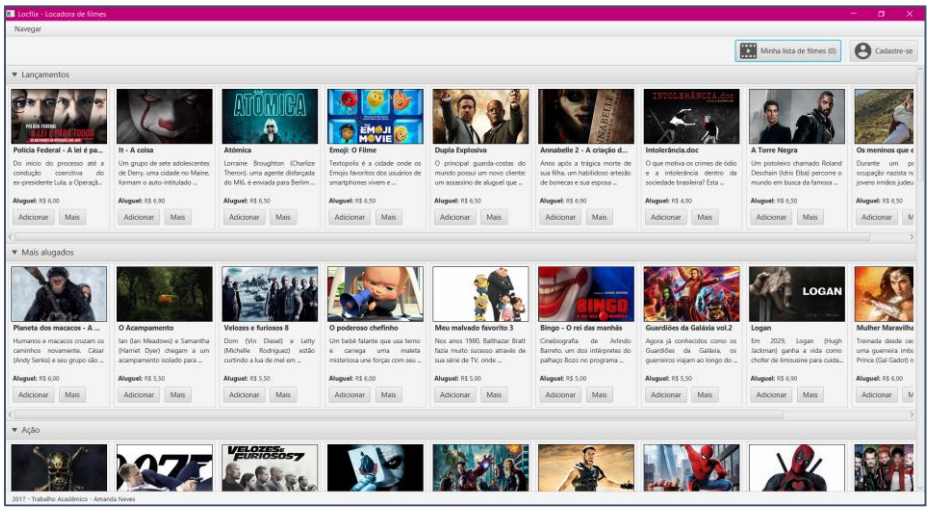
Uma característica importante do Locflix, é a compatibilidade com diversos Sistemas Operacionais, no qual ele pode ser acessado pelo Linux, Windows, entre outros. E não há necessidade de conexão à internet para o acesso, a partir que se tenha o executável (extensão .jar).

Estrutura da aplicação

Tela Principal

A interface gráfica da tela principal foi desenhada para proporcionar acesso rápido à todas funcionalidades da aplicação. Nela você encontrará todos os filmes disponíveis separados em categorias, proporcionando uma livre navegação entre eles, possibilitando ver mais informações sobre os mesmos e adicionar/cancelar à lista de filmes à serem alugados.

Nela também são visíveis os botões de acesso para as principais funcionalidades que são a Configuração de usuário (Cadastre-se) e a lista de filmes escolhidos (Minha lista de filmes). Além do menu "Navegar" que também possibilita acesso a estas funcionalidades.



Categorias dos filmes

A aplicação contém várias categorias de filmes nas quais estão em formato de seções que podem ser expansíveis. As categorias são:

- Lançamentos (alguns filmes que esteve em cartaz nos cinemas no período entre agosto e setembro de 2017)
- Mais alugados (suposições)
- Ação
- Comédia
- Documentários
- Drama
- Ficção científica
- Romance
- Terror

Filme

Cada categoria contém vários filmes para o usuário alugar. Cada filme contém uma imagem que representa a capa ou alguma cena marcante do mesmo. Logo abaixo o título, uma breve sinopse e dois botões:

- Adicionar: ao clicar neste, o filme é adicionado para a lista de filmes escolhidos, e imediatamente sua legenda é alterada para "Cancelar", caso o usuário queira desfazer sua escolha, o filme é retirado da lista de filmes escolhidos.
- Mais: ao clicar neste, abrirá uma tela com a foto da capa do filme e várias outras informações como: sinopse completa, data de lançamento, valor do aluguel, classificação e duração do filme.

Exemplo de um filme no Locflix

Menu Principal

No menu principal, contém a opção "Navegar", na qual contém dois submenus:

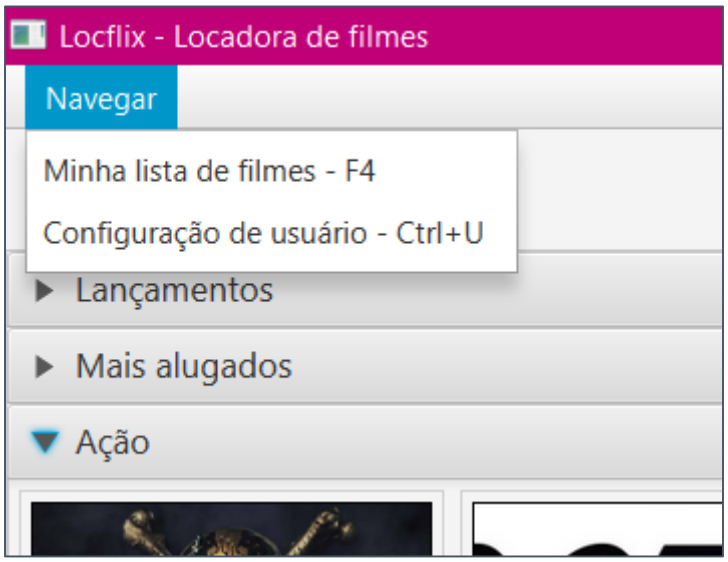
Minha lista de filmes – Uma tela que contém uma tabela com todos os filmes escolhidos e informações de valores. Logo abaixo desta tabela contém o valor total dos filmes e três opções de navegação:

- Limpar lista: cancela todos os filmes escolhidos para aluguel.
- Continuar escolhendo: fecha a tela com a tabela e permite a continuação para escolha dos filmes.
- Finalizar aluguel: Finaliza aluguel.

Configuração de usuário – Uma tela que contém vários campos para o usuário se cadastrar. Contém dois botões:

- Limpar: limpa todos os dados informados na tela pelo usuário.
- Salvar: valida e salva todos os dados informados pelo usuário.

Nos submenus também mostram as teclas de atalho que podem ser pressionadas para acessar estas telas a qualquer momento durante o uso da aplicação.

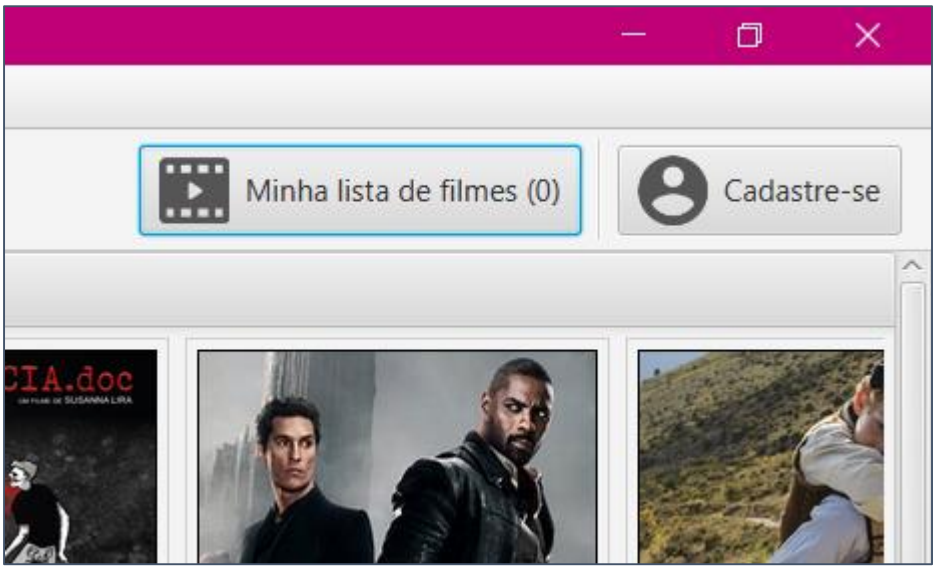


Menu principal

Botões na barra superior

Estes botões possibilitam mais uma opção (além das teclas de atalho) ao acesso rápido para as telas: lista de filmes escolhidas pelo usuário e da configuração de usuário.

O botão "Minha lista de filmes (0)" é atualizado a cada filme adicionado ou cancelado pelo usuário. O número ao lado da legenda representa a quantidade atual de filmes escolhidos pelo usuário.



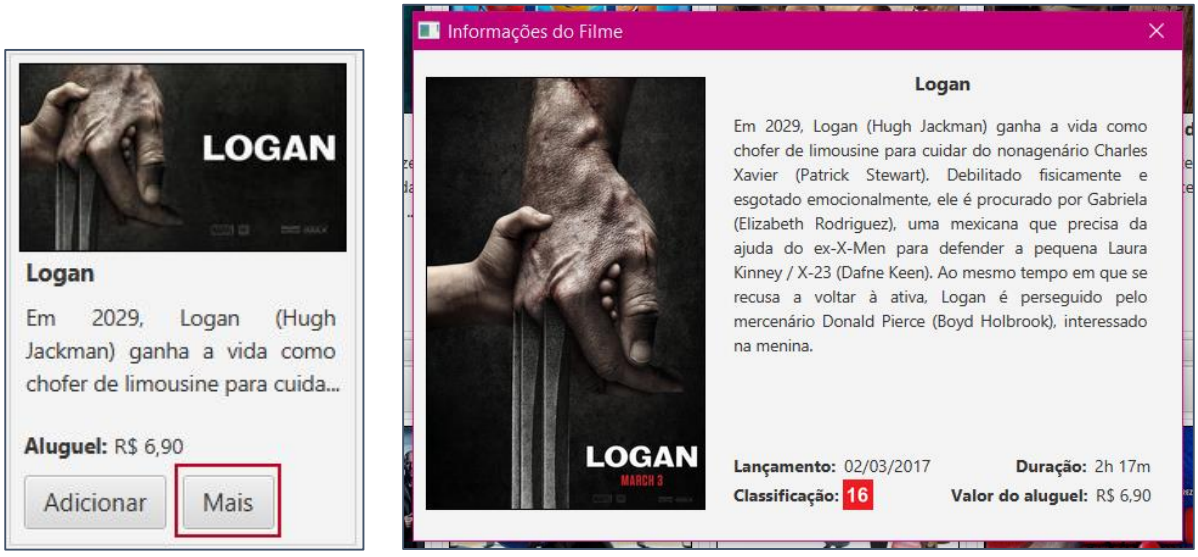
Botões na barra superior

Passo a passo

Nesta seção será mostrado passo a passo de várias funcionalidades da aplicação.

Como ver informações completas do filme

Na interface principal, em cada filme existe um botão com legenda "Mais". Ao clicar neste, abrirá uma tela com várias informações do filme como: imagem da capa do filme, sinopse completa, valor do aluguel, data de lançamento, classificação e duração.



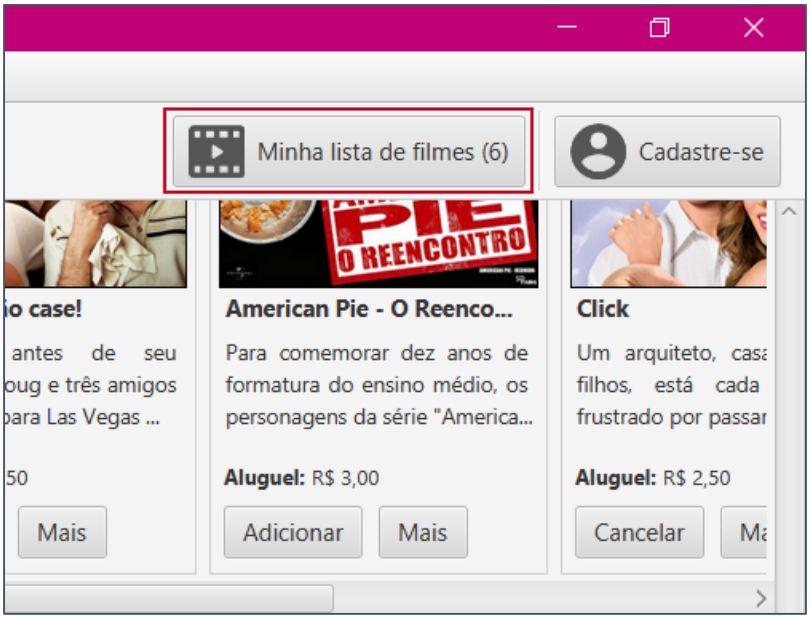
Botão "Mais" e tela Informações do Filme

Como escolher um ou mais filmes

Na interface principal, em cada filme existe um botão "Adicionar", ao clicar neste, imediatamente a legenda do botão "Adicionar" é alterada para "Cancelar" caso o usuário queira desfazer a ação anterior. Pode-se adicionar vários filmes.

O número do botão superior "Minha lista de filmes (0)" sempre é atualizado para a quantidade atual de filmes escolhidos a medida em que o usuário adiciona ou cancela um filme.

Exemplo de alguns filmes adicionados e outro não

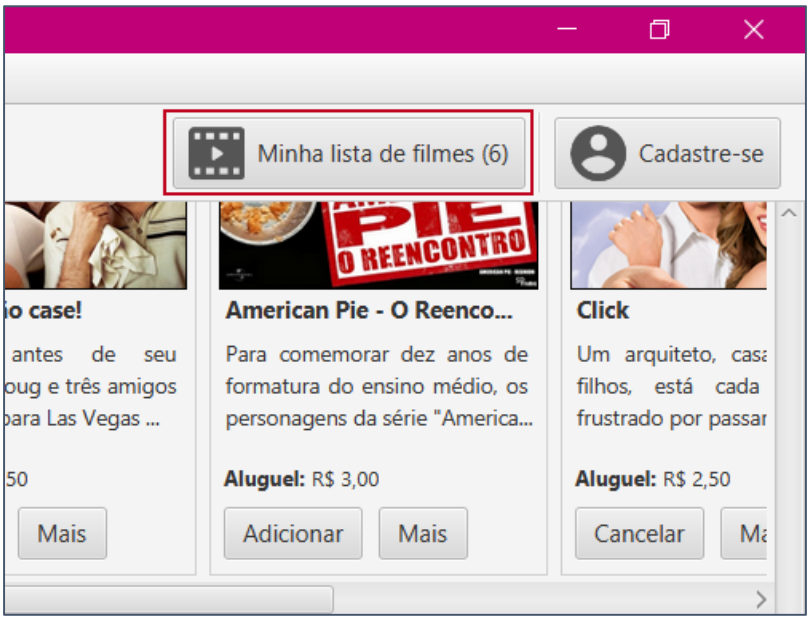


Botão "Minha lista de filmes" com a quantidade de filmes escolhidos

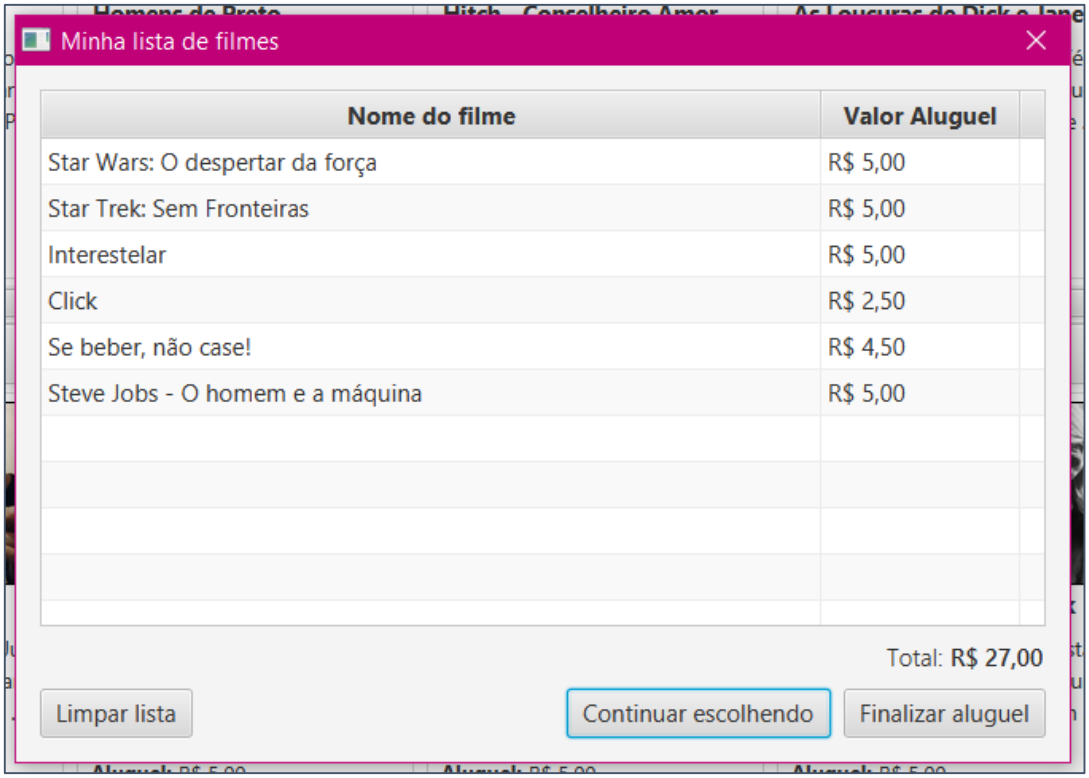
Como saber quais filmes foram escolhidos

No lado superior esquerdo da aplicação existe um botão "Minha lista de filmes (x)", ao clicar neste, abrirá uma tela com uma tabela com todos os nomes dos filmes escolhidos e valores.

Para acessar esta tela também pode-se pressionar a tecla F4 ou acessar o menu Navegar > Minha lista de filmes.



Botão "Minha lista de filmes" com a quantidade de filmes escolhidos



Tela "Minha lista de filmes"

Como finalizar o aluguel dos filmes

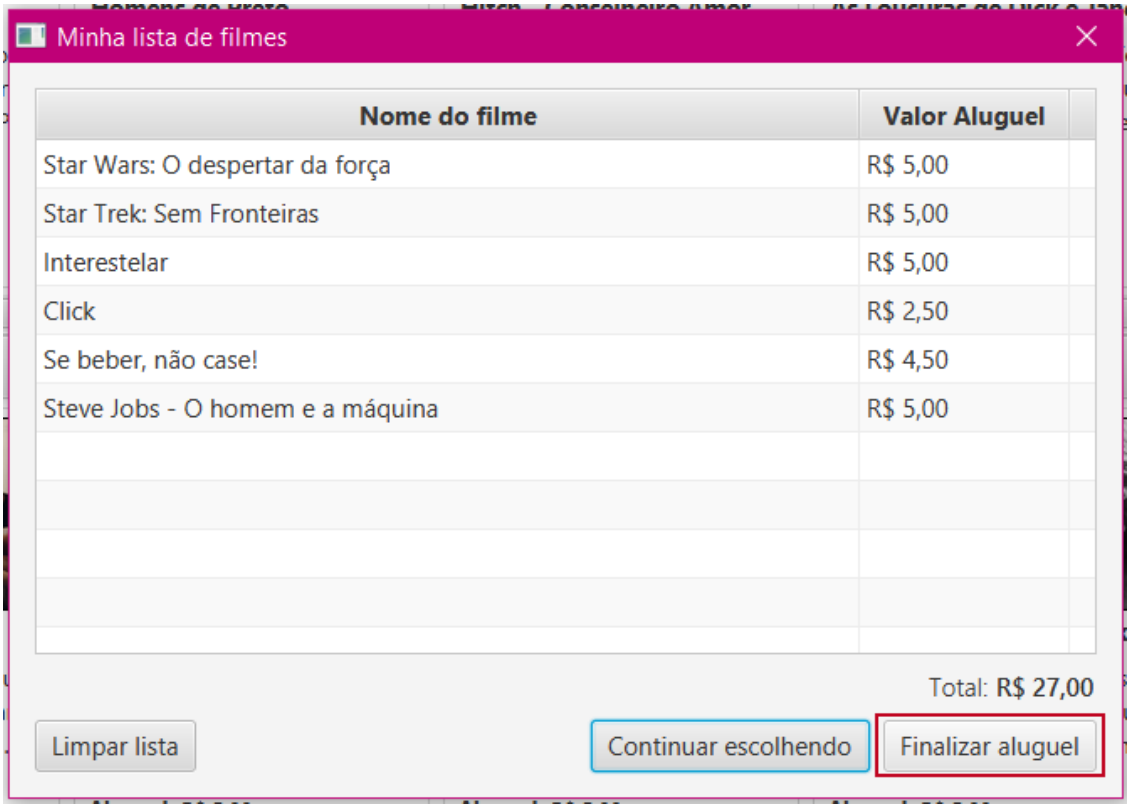
Para finalizar o aluguel dos filmes é necessário que se tenha pelo menos um filme escolhido pelo usuário e que o mesmo (usuário) esteja cadastrado na aplicação.

Para saber como cadastra/configura o usuário, a explicação está no tópico posterior a este.

Após a escolha dos filmes pode-se clicar no botão superior esquerdo "Minha lista de filmes (x)", abrirá uma tela com a tabela de todos os filmes escolhidos e valores, logo abaixo terá um botão

com a legenda "Finalizar aluguel", ao clicar neste, irá aparecer a mensagem: "Aluguel realizado com sucesso". Caso não aparecer esta mensagem, irá aparecer outra instruindo o usuário para o que se deve ser feito.

Para acessar esta tela também pode-se pressionar a tecla F4 ou acessar o menu Navegar > Minha lista de filmes.



Botão "Finalizar aluguel" na tela "Minha lista de filmes"

Como cadastrar/configurar o usuário

Para acessar a tela de configuração de usuário pode-se clicar no botão superior esquerdo "Cadastre-se", ou pressionar as teclas Ctrl+U ou acessar pelo menu Navegar > Configuração de usuário. Abrirá um formulário com vários campos a serem preenchidos.

Campos obrigatórios:

- Nome completo
- Data de nascimento (o usuário deve ter no mínimo 16 anos de idade)
- Telefone Fixo/Comercial (caso não houver pode-se colocar o celular)
- Endereço
- Número
- Bairro
- Cidade
- UF

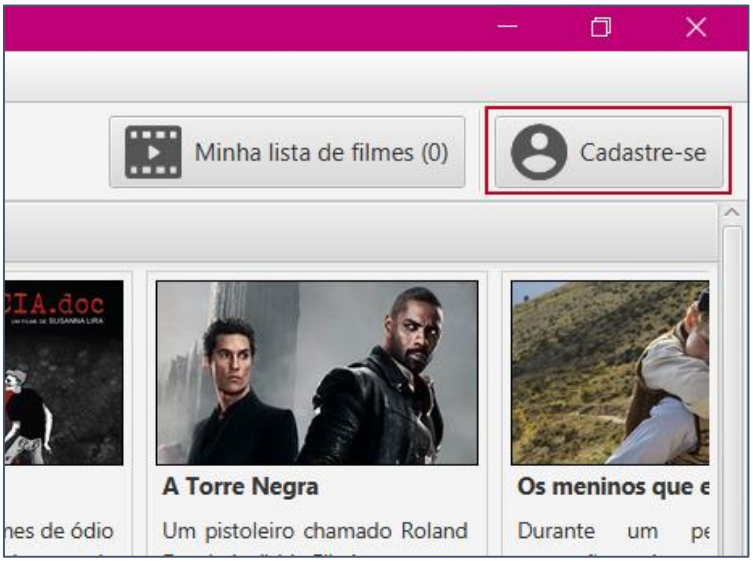
- CEP

Campos opcionais:

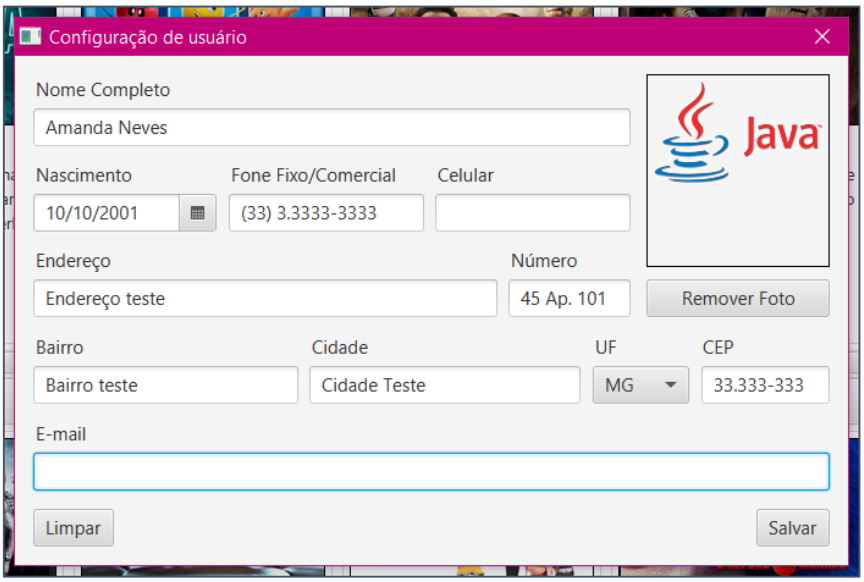
- Foto
- E-mail

Para adicionar uma foto, pode-se clicar no botão "Adicionar Foto" que fica abaixo de um ícone com uma câmera, escolha a foto de sua preferência e clique em Salvar. Imediatamente a legenda do botão muda para "Remover Foto" caso queira remover.

Após serem preenchidos os campos, pode-se clicar no botão "Salvar", no qual os dados serão validados e salvos.



Botão "Cadastre-se" na tela principal



Tela de configuração de usuário com alguns dados preenchidos

Dúvidas e sugestões

Caso tenha qualquer dúvida ou sugestão referente à aplicação Locflix JavaFX ou a este manual, entre em contato:

E-mail: amandaneves.carmo@gmail.com