UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMA

CESAR AUGUSTO BOTH

ESTUDO DAS APIS JAVA PARA A GERAÇÃO DE GRÁFICOS COM SWING E JAVAFX

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

CESAR AUGUSTO BOTH

ESTUDO DAS APIS JAVA PARA A GERAÇÃO DE GRÁFICOS COM SWING E JAVAFX

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Marcio Angelo Matté.

Co-orientador: Prof. Me. Ricardo Sobjak.

MEDIANEIRA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Estudo das APIs Java para geração de gráficos com Swing e JavaFX

Por

Cesar Augusto Both

Este Trabalho de Diplomação (TD) foi apresentado às 14:40 h do dia 09 de Junho de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Manutenção Industrial, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O acadêmico foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Marcio Angelo Matté
UTFPR – Campus Medianeira
(Orientador)

Prof. Me. Juliano Rodrigo Lamb
UTFPR – Campus Medianeira
(Convidado)

Prof. Me. Ricardo Sobjak
UTFPR – Campus Medianeira
(Co-Convidado)

Prof. Me. Juliano Rodrigo Lamb
UTFPR – Campus Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

BOTH, Cesar A. ESTUDO DAS APIS JAVA PARA GERAÇÃO DE GRÁFICOS COM SWING E JAVAFX. 2015. 45 f. Trabalho de Diplomação (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas). Universidade Tecnológica Federal do Paraná – UTFPR. Medianeira. 2015.

O desenvolvimento de aplicações consiste em inúmeras etapas como escolher a plataforma e a tecnologia, que serão utilizadas para o desenvolvimento e são de suma importância e requerem atenção. Se tratando de plataforma *desktop* e linguagem de programação Java existe a disponibilidade das APIs *Swing* e *JavaFX*. Como *Swing* está em um processo de transição para *JavaFX*, cabe aos desenvolvedores de aplicações analisarem se vale a pena utilizar ainda *Swing* ou se devem migrar para *JavaFX*. Com base nisto, este estudo tem como objetivo analisar estas duas APIs e verificar pontos relevantes para a geração de gráficos, avaliando diferenças entre elas e realizar métricas de desempenho capazes de definir uma opção preferível para o estudo elaborado.

Palavras-chave: Interfaces gráficas. Desempenho. Programação Desktop.

ABSTRACT

BOTH, Cesar A. STUDY OF APIS JAVA FOR GENERATION OF GRAPHS WITH SWING AND JAVAFX. 2015. 45 f. Working graduation (Degree in Technology Analysis and Systems Development). Federal Technological University of Paraná – UTFPR. Medianeira. 2015.

The development of applications consists of innumerable stages as to choose the platform and the technology, that will be used for the development and are of utmost importance and require attention. Being about platform desktop computer and programming language Java the availability of the APIs Swing and JavaFX exists. As Swing it is in a process of transition for JavaFX, it fits to the desenvolvedores of applications to analyse uses the penalty still to use Swing or they must be migrated for JavaFX. On the basis of this, this study it has as objective to analyse these two APIs and to verify important points for the generation of graphs, evaluating differences between them and to accomplish metric of performance capable to define a preferable option for the elaborated study.

Keywords: Graphical interfaces. Performance. Programming Desktop computer.

LISTA DE SIGLAS

API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
CPU	*Central Processing Unit*
CSV	*Comma-separated values*
DDR	*Double-Data-Rate*
EPS	*Encapsulated Post Script*
F3	*Form Follows Function*
GB	Gigabytes
GB/s	Gigabytes por segundo
GHz	Gigahertz
GUI	*Grafic User Interface*
JDK	*Java Development Kit*
JPG	*Joint Photographic Experts Group*
JRE	*Java Runtime Enviroment*
JVM	*Java Virtual Machinne*
LAF	*Look-and-Feel*
LGPL	*GNU Lesser General Public License*
MB	*Megabyte*
MHz	Megahertz
PDF	*Portable Document Format*
PNG	*Portable Network Graphics*
RAM	*Random Acess Memory*
RIA	*Rich Internet Application*
RPM	Rotações por minuto
SATA	*Serial Advanced Technology Attachment*
SO	Sistema Operacional
SVG	*Scalable Vector Graphics*

TI	Tecnologia da Informação
XML	*Extensible Markup Language*

LISTA DE FIGURAS

Figura 1 - Estados de uma thread.. 21

Figura 2 - Acesso de threads a um método não sincronizado. 23

Figura 3 - Acesso de threads a um método sincronizado.. 24

Figura 4 - Diagrama de Classes genérico para o cenário proposto... 31

Figura 5 - Diagrama de Atividades para execução dos testes de desempenho. 32

Figura 6 - Gráfico de barra gerado pelo *software Swing*. 36

Figura 7 - Gráfico de barra gerado pelo *software JavaFX*.. 36

Figura 8 - Gráfico de linha gerado pelo *software Swing*... 37

Figura 9 - Gráfico de linha gerado pelo *software JavaFX*. 37

LISTA DE QUADROS

Quadro 1- Utilização da palavra reservada "synchronized". 22

Quadro 2 - Classe Java criada utilizando herança a partir da classe java.lang.Thread. 25

Quadro 3- Classe Java criada implementando a interface Runnable. 25

Quadro 4- Classe criada para executar comandos Swing de dentro de outra thread. 27

Quadro 5- Invocação da classe criada para execução de comandos Swing dentro de outra thread. 27

Quadro 6 - Métricas de desempenho do *software* baseado em Swing para uso de memória *Heap*. 38

Quadro 7 - Métricas de desempenho do software baseado em Swing para uso de CPU. 38

Quadro 8 - Métricas de desempenho do software baseado em JavaFX para uso de memória Heap. 39

Quadro 9 - Métricas de desempenho do software baseado em JavaFX para uso de CPU. 39

SUMÁRIO

1 INTRODUÇÃO ... 11

1.1 OBJETIVO GERAL ... 12

1.2 OBJETIVOS ESPECÍFICOS ... 12

1.3 JUSTIFICATIVA ... 12

2 REVISÃO BIBLIOGRÁFICA ... 14

2.1 INTERFACE GRÁFICA COM O USUÁRIO ... 14

2.2 AWT ... 14

2.3 SWING ... 14

2.4 JAVAFX ... 15

2.4.1 JavaFX e sua evolução ... 16

2.4.1.1 JavaFX 2.0 ... 17

2.5 JFREECHART ... 19

2.6 PROGRAMAÇÃO CONCORRENTE ... 19

2.7 THREADS ... 20

2.7.1 Os estados de uma thread ... 21

2.7.2 Prioridades de uma thread ... 21

2.7.3 Sincronismo de threads ... 22

2.7.4 Utilizando threads em Java ... 24

2.7.4.1 A utilização de threads com *Swing* ... 25

2.8 O PADRÃO MVC ... 28

2.8.1 Fluxo de interações do MVC ... 28

3 MATERIAL E MÉTODOS ... 30

3.1 METODOLOGIA ... 30

3.1.1 Procedimento de desenvolvimento dos softwares ... 30

3.1.2 Parâmetros de avaliação ... 31

3.1.3 Metodologia de testes ... 31

3.2 TECNOLOGIAS UTILIZADAS ... 32

3.2.1 Interface de Desenvolvimento Integrado Netbeans ... 33

3.2.2 *JavaFX* Scene Builder 34
3.2.3 JConsole 34
4 RESULTADOS E DISCUSSÃO 35
4.1 REALIZAÇÃO DOS TESTES 35
4.2 MÉTRICAS DE DESEMPENHO 37
4.2.1 Métricas de desempenho para o software em Swing 38
4.2.2 Métricas de desempenho para o software em JavaFX 38
4.3 COMPARAÇÕES ENTRE TESTES 39
5 CONSIDERAÇÕES FINAIS 41
5.1 CONCLUSÃO 41
5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO 41
6 REFERÊNCIAS BIBLIOGRÁFICAS 42

1 INTRODUÇÃO

É notória a utilização da tecnologia nas mais diversas e distintas atividades desenvolvidas no mundo. Cada vez mais *softwares* vêm sendo desenvolvidos para atender demandas dos mais variados tipos. Um *software* pode gerenciar, desde uma lanchonete até um controle automático de um torno mecânico, dentre outros. As empresas do ramo de Tecnologia da Informação (TI) lançam novos produtos e tornam o mercado cada vez mais competitivo. Para conquistar seus clientes, as empresas devem focar em diferenciais que tornem seus produtos e serviços melhores do que a concorrência, em fatores ligados a desempenho, organização e também no aspecto visual.

Mesmo o mercado de *software,* que atualmente, está voltado à programação para a Internet, existem muitos *softwares desktop* sendo utilizados por diversos setores, sendo que estes atendem às necessidades de uma grande parte das empresas, pois estes *softwares* demandam de um *hardware* com condições de receber a instalação do sistema e executá-lo.

Em busca deste ideal os *softwares* destinados para ambiente *desktop* utilizando a linguagem de programação Java estão em um processo de transição. Esta por sua vez, se dá por intermédio de *Swing* e *JavaFX,* duas APIs[1] (*Application Programming Interface*) focadas em interfaces gráficas para aplicações *desktop*. Todavia *Swing* começa a se tornar obsoleto e esta cada vez mais perdendo espaço para *JavaFX,* que traz maior facilidade na sua utilização, melhoramento dos aspectos gráficos, de desempenho e afins.

Esta transição se torna mais nítida pelo fato de que a partir do JDK[2] (*Java Development Kit*) 8, a API *JavaFX* já está inclusa no pacote, além disso, a Oracle (2014) afirma que *Swing* ainda fará parte da especificação *Java SE,* mas os desenvolvedores devem migrar suas novas aplicações para *JavaFX* e utilizar da possibilidade de estender um aplicativo *Swing* com *JavaFX* para que esta transição seja mais suave possível.

---

[1] "[...]API especifica um conjunto de funções que estão disponíveis a um programador de aplicações, incluindo os parâmetros que são passados a cada função e os valores de retorno que o programador pode esperar." (SILBERSCHATZ; GALVIN; GAGNE, 2008).

[2] JDK, ou Kit de Desenvolvimento Java, é um conjunto de ferramentas elaborado inicialmente pela Sun Microsystems, para o desenvolvimento de aplicações na linguagem Java. Tendo como principal objetivo, facilitar e difundir o uso da tecnologia Java (SERSON, 2007).

1.1 OBJETIVO GERAL

Estudo das tecnologias de construção de interface gráfica para aplicações *desktop* na linguagem de programação Java, utilizando *Swing* em conjunto com a biblioteca JFreeChart e *JavaFX* 2.0, em aplicação concorrente (multithread).

1.2 OBJETIVOS ESPECÍFICOS

- Realizar um estudo sobre as tecnologias para a construção de interfaces gráficas para aplicações *desktop* com Java – *Swing* e *JavaFX;*
- Descrever as principais mudanças do *Swing* para o *JavaFX,* no uso para a geração de interfaces gráficas;
- Estruturar a aplicação em um ambiente concorrente;
- Observar o comportamento em tempo de execução dos *softwares* desenvolvidos por meio do uso da ferramenta JConsole.

1.3 JUSTIFICATIVA

Desenvolver *softwares* requer uma vasta gama de variáveis em relação à coleta de informações, estudo de tecnologias a serem empregadas, encontrar os requisitos da aplicação, desenvolvimento, testes e afins. Este é um processo que exige de todos os envolvidos na elaboração do *software,* não somente pelo fato de programar em si, mas existe uma série de variáveis que devem ser observadas antes do início de um projeto nesta área. É necessário primeiramente coletar informações juntamente com o cliente para saber de suas necessidades perante o produto final, escolher plataforma de desenvolvimento, linguagem de programação, dentre outros tantos passos. Este trabalho visa à busca por elucidação de paradigmas com relação a duas APIs Java para o desenvolvimento de aplicações baseadas em plataforma *desktop,* o *Swing* e o *JavaFX.*

Foca-se em *desktop*, pois muitos *software*s não necessitam das funcionalidades existentes em aplicações *Web,* tão difundidas atualmente, mas que requerem um grau de preocupação muito maior em vários aspectos, como: viabilidade, segurança, praticidade, funcionalidade, acessibilidade, entre outros.

Java está passando por um processo transitório em suas tecnologias voltadas para aplicações *desktop*. É nesse propósito que este estudo se baseia para poder dispor um material abordando as principais diferenças entre *Swing* e *JavaFX* no desenvolvimento destas aplicações.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são abordados os principais temas de interesse para este trabalho, tais como: Interface Gráfica com o Usuário, *Swing*, *JavaFX* e threads.

2.1 INTERFACE GRÁFICA COM O USUÁRIO

Para Serson (2007), Interface Gráfica com o Usuário (GUI), do inglês *Graphical* U*ser Interface*, consiste em um meio de interação amigável entre o cliente e a aplicação. Sendo que o cliente pode interagir com a aplicação por intermédio de dispositivos de entrada/saída, como mouse, teclado, voz dentre outros. Uma aplicação que utiliza alguma espécie de GUI em sua estrutura faz com que seus clientes tenham um grau de familiarização muito maior, no sentido de compreender a aplicação como um todo e levando assim a uma maior produtividade.

2.2 AWT

Segundo Costa (2008), *Abstract Window* Toolkit (AWT) é uma API original do Java para a criação de GUI's, que foi introduzida no JDK 1.0 e ainda está presente. As GUI's criadas através dessa API têm suas características de interface totalmente dependentes do sistema operacional em que elas estão sendo executadas. Desta maneira, um mesmo *software* criado por meio de AWT possui diferenças se rodar em um sistema Windows ou Linux, por exemplo.

2.3 SWING

Segundo Costa (2008), na versão 1.0 do JDK os desenvolvedores Java tinham disponível para a criação de GUIs a API AWT, do inglês *Abstract Window* Toolkit. Está possuía interfaces dependentes do sistema operacional. Posteriormente na versão 1.1 do JDK, surgiu a API *Swing*, por meio desta nova API os *softwares* já eram independentes de sistema operacional, mantendo um padrão visual mesmo em sistemas operacionais diferentes.

Cavaléro (2014) descreve a evolução de *Swing* perante AWT, ao abordar que no *Swing* foi adicionada uma vasta biblioteca de componentes gráficos para o Java, sendo que o uso

destes componentes tornam as aplicações mais ricas e interativas com o cliente. Esta API possibilita a criação de sistemas multiplataforma, internacionalizáveis, com aparência customizada ou nativa, permitindo que o desenvolvedor possa criar novos componentes e adicioná-los ao seu *software.* Uma API totalmente orientada a objetos e que foi elaborada seguindo padrões de projetos em seu código-fonte.

Para evoluir o *Swing* utilizou de aspectos presentes no AWT, pois segundo Cavaléro (2014), *Swing* manteve a compatibilidade com as aplicações feitas com AWT, mas melhorou a mesma retirando o problema de portabilidade e adicionou o conceito de *Look-and-Feel,* ou visual e comportamento, onde todos os componentes visuais foram desenhados utilizando a biblioteca Java 2D e buscam imitar os visuais nativos dos mais diversos sistemas operacionais, permitindo assim que a mesma aplicação rode semelhantemente em qualquer Sistema Operacional (SO).

Caelum (2014) afirma que "Com *Swing*, não importa qual sistema operacional, qual resolução de tela, ou qual profundidade de cores: sua aplicação se comportará da mesma forma em todos os ambientes.", trazendo ao cliente que utiliza um *software* desenvolvido com esta API, uma estabilidade na questão operacional e visual do mesmo, em qualquer estação de trabalho que o cliente utilizar o *software*.

Conforme Neto (2009), *Look-and-Feel* (LAF) é o conceito utilizado para facilitar o entendimento das informações visuais através de cores, os estilos das janelas, dos componentes em geral, dentre outros. Desde o JDK 1.2 são suportados os seguintes LAFs: Windows, MacOS, MOTIF, X-WINDOWS, CDE, KDE, GNOME e o conceito padrão Metal.

2.4 JAVAFX

Para Giongo; Araújo; Rodrigues (2014), *JavaFX* é uma API Java que atende os requisitos para o desenvolvimento de *softwares* com interface RIA[3] (*Rich Internet Application*) .

---

[3] "O termo RIA é usado para descrever Aplicações Ricas para Internet, ou seja, aplicações que são executadas em ambiente web, mas que possuem características similares a *software*s desenvolvidos para execução em ambiente desktop". (GIONGO; ARAÚJO; RODRIGUES, 2014)

Para Marquezini; Aguinaldo; Almeida (2013), quando uma aplicação é desenvolvida utilizando o *JavaFX,* ela automaticamente já é uma aplicação RIA tendo um visual muito melhor e com a capacidade de se poder usar a mesma aplicação em diferentes plataformas. Outro benefício apontado segundo os autores, é a integração que o *JavaFX* também traz com os mais diversos *frameworks*[4] para programação em Java, além de ter fácil manutenção, adaptação e melhoria nas interfaces com o usuário.

Segundo Leal (2009), mesmo *JavaFX* atendendo os requisitos de *softwares* com interface RIA, subentendendo que são somente aplicações para a Internet, *JavaFX* da a possibilidade de construir interfaces gráficas para ambiente *desktop*. Tendo como alguns exemplos jogos e painéis de informação.

2.4.1 JavaFX e sua evolução

Conforme Giongo; Araújo; Rodrigues (2014), *JavaFX* foi lançada em 2008 pela então Sun Microsystems, que na sua versão 1.1, foi incorporada uma linguagem própria para trabalhar com os recursos trazidos por esta nova API, chamada de *JavaFX Script.*

Uma linguagem estaticamente tipada, orientada a objetos e rodava na JVM[5]. *JavaFX Script* foi adquirido pela Sun Microsystems, por intermédio da SeeBeyond que tinha esse projeto originalmente com o nome de F3, sigla para *Form Follows Function.*

As aplicações desenvolvidas seguiam o conceito de "Desenvolver uma vez, para executar em qualquer lugar", isso ocorria pelo fato dessas aplicações executarem diretamente na JRE[6], assim tanto *desktops* como navegadores de Internet podiam executar tais aplicações, tendo somente a necessidade de ter disponível uma versão da JRE compatível com a da aplicação pretendida, sendo esse conceito muito mais confiável após ser lançada a versão 1.1

---

[4] Frameworks são desenvolvidos com o principio de reutilização parcial ou total de uma aplicação, para que possam ser utilizados para desenvolver outras aplicações. Sendo que as partes comuns para a obtenção de uma solução, são implementadas dentro do framework para serem utilizadas em outras aplicações. "O framework geralmente também é construído por especialistas de um domínio e reusado por leigos naquele domínio." (PIMENTEL; FUKS, 2012).

[5] "[...] é um conjunto de programas de *software* que permite a execução de instruções – geralmente escritas em *bytecode* Java. Os JVMs estão disponíveis para todas as plataformas de *software* e hardware mais comuns" (ORACLE, 2014).

[6] "O JRE é composto pela Java *Virtual Machine* (JVM), pelas classes de núcleo da plataforma Java e bibliotecas da plataforma Java para suporte. O JRE é a parte de runtime do *software* do *software* Java, que é o necessário para executá-lo no seu Web browser" (ORACLE, 2014).

com suporte para a criação de aplicações *JavaFX* para dispositivos móveis (GIONGO; ARAÚJO; RODRIGUES, 2014).

Já na versão 1.2 foi dado suporte para novos sistemas operacionais, sendo eles Linux e Solaris, houve também a inclusão de novas classes no pacote de *layouts*, além do aumento de efeitos visuais para os componentes disponíveis. Houve mudanças cruciais quanto à sintaxe da linguagem *JavaFX Script*, fazendo com que muita programação que era útil para o desenvolvimento de novas aplicações, se tornasse incompatível e obsoleto (GIONGO; ARAÚJO; RODRIGUES, 2014).

Segundo Giongo; Araújo; Rodrigues (2014) a versão 1.3, lançada em 2010, melhorou nos aspectos de desempenho em tempo de execução, suporte para aplicativos que rodam em televisores, 3D e a adição de novos controles para as interfaces. A partir dessa versão, os desenvolvedores podiam contar com seu primeiro editor visual para interfaces diretamente no Netbeans, até então na versão 6.9, chamado de *JavaFX Composer*.

Um grande marco aconteceu em 2010 quando a Oracle comprou a Sun. Isto pelo fato do *JavaFX* ter sido totalmente reestruturado, fazendo com que versões anteriores dessa tecnologia, se tornassem incompatíveis. Nesse processo a linguagem *JavaFX Script* foi deixada de lado e a programação antes realizada nessa linguagem, passou para Java e FXML. O grande ganho para isso é que os desenvolvedores não necessitam aprender uma nova linguagem se quiserem programar em *JavaFX*. Todavia mesmo se a vontade for continuar usando uma linguagem declarativa, os mesmos podem usufruir de *Visage*, *Scala* ou *Groovy*, por exemplo (GIONGO; ARAÚJO; RODRIGUES, 2014).

2.4.1.1 JavaFX 2.0

Giongo; Araújo; Rodrigues (2014) afirmam que *JavaFX* 2.0 foi desenvolvido para simplificar o desenvolvimento de *software*s que necessitam a utilização de conteúdo multimídia. Isto é possível pela utilização de avançados recursos de motores de mídia, além de gráficos com aceleração por *hardware*. Além destes recursos, esta versão do *JavaFX* possui uma nova coleção de controles para Interfaces gráficas, agregando requisitos fundamentais para a sua escolha quando se trata de desenvolvimento com aplicações RIA com linguagem Java. Além disso, podem ser utilizadas todas as funcionalidades pertinentes à linguagem já existentes.

Porém segundo Schmitz (2012), a API *JavaFX* 2.0 só começou a ter um lugar de destaque quando a Oracle, adicionou esta API na versão 7 do JDK. Deste modo, a tecnologia se tornou nativa das aplicações Java, não tendo mais a necessidade de realizar instalações adicionais para utilizar esta tecnologia.

Tendo *JavaFX Script* sido descontinuada, a nova versão do *JavaFX* introduziu uma nova linguagem de marcação que se assemelha com XML[7], o *FXML.* Utilizando-se desta nova linguagem é possível modelar a interface gráfica fora da lógica da aplicação, desta maneira se ganha em produtividade, pelo fato de não ser necessário recompilar toda vez o código quando uma mudança na interface gráfica ocorrer. As semelhanças com XML, que é comum a maioria dos desenvolvedores, tornam esta linguagem fácil e intuitiva para ser aprendida (GIONGO; ARAÚJO; RODRIGUES, 2014);

Além disso, para realizar a customização dos componentes do *JavaFX* 2.0, o desenvolvedor pode utilizar-se de CSS[8] (*Cascading Style Sheets*), sendo muito semelhante ao seu uso em aplicações para a Internet, com grande diferenciação principal somente na nomenclatura dos atributos, onde todos possuem o prefixo `-fx-`. (ZANARDO, 2013)

Conforme Giongo; Araújo; Rodrigues (2014), *JavaFX* 2.0 possui integração com *Swing*. Isto se da pelo fato de *JavaFX* conter uma classe denominada de JFXPanel, dentro do pacote `JavaFX.embed.Swing`, que permite dentre suas funcionalidades, reproduzir mídias e toda a sua API gráfica dentro do *Swing*. Isso foi possibilitado pela Oracle, mas a mesma recomenda a migração para utilização de somente *JavaFX.*

Zanardo (2013) explica que o *JavaFX* 2.0 oferece uma solução para facilitar a distribuição dos *softwares* desenvolvidos, dando a possibilidade de empacotar a aplicação, a Máquina Virtual do Java, ou JVM. Sendo assim, o *software* fica independente da versão instalada da maquina do cliente, facilitando sua distribuição e evitando alguns transtornos possíveis quando esta situação ocorre.

---

7 "XML, ou *eXtended Markup Language*, é um padrão para a formatação de dados, ou seja, uma maneira de organizar informações [...] Todas as informações contidas no XML estão dentro de *tags*" (MEDINA, 2014).

[8] CSS, ou Folhas de Estilos em Cascata, é uma linguagem de estilo/formatação para construção de sites. Sendo utilizado para definir a apresentação dos elementos visuais contidos nas páginas dos sites e aplicações em geral (JOBSTRAIBIZER, 2009).

2.5 JFREECHART

Conforme Caelum (2015), é comum verificar a necessidade de implementação de relatórios em formato de gráficos e afins. É nesse propósito que existe a biblioteca "JFreeChart" que simplifica esse cenário com a plataforma de desenvolvimento Java, podendo ser utilizada para gerar inúmeros gráficos, como: gráfico de pontos, de barra, de linha, de pizza, gráficos em 2 ou 3 dimensões dentre outros. Além de disponibilizar o resultado em JPG, PNG, SVG, EPS ou exibir diretamente na aplicação.

Segundo JFreeChart (2015) o projeto desta biblioteca foi iniciado em fevereiro de 2000 por David Gilbert, sendo que já foram realizados mais de 2,2 milhões de downloads da biblioteca.

2.6 PROGRAMAÇÃO CONCORRENTE

A grande maioria das estações de trabalho é baseada em apenas um processador que é responsável pela execução de diversas tarefas simultâneas, em que vários processos utilizam a capacidade do processador em diversas faixas de tempo para a execução de suas atividades. Tal procedimento é denominado multiprocessamento (JEVEAUX, 2015).

Segundo Caelum (2015) existem várias situações em que *softwares* necessitam realizar mais de uma tarefa ao mesmo tempo. Um exemplo está na geração de relatórios em PDF, caso o relatório seja muito extenso, este processo se tornará relativamente lento para apresentar o resultado ao cliente. Tal lentidão pode levar o cliente a ter uma falsa impressão do *software*. Sendo assim, há a necessidade de exibir o andamento da confecção do relatório por intermédio de uma barra de progresso, ou seja, duas tarefas simultâneas. Desta maneira existe a necessidade de se utilizar multitarefas.

Para Horstmann; Cornell (2001) essas multitarefas podem ser manipuladas de duas maneiras. A multitarefa preemptiva é quando o SO realiza a interrupção do programa sem realizar consultas, já na multitarefa cooperativa, os programas são interrompidos somente quando de fato é a ordem dada pelo programa. Exemplos de sistemas cooperativos são: Windows 3.1, 95, 98 e NT. Para sistemas com multitarefa preemptiva, há o exemplo do QNX que é uma versão do UNIX para sistemas em tempo real. Nos preemptivos, embora sejam muito mais complexos de serem implementados, possuem mais eficiência se comparados aos cooperativos. Isto porque se o SO estiver executando um programa com defeito, o mesmo

poderá travar todo o sistema, um caso bem familiar para os donos de computadores com versões cooperativas mais antigas.

Conforme Mmosgame (2015) na multitarefa preemptiva, o SO possui uma lista com o escalonamento dos processos, ou programa, que é elaborada levando em consideração a prioridade das tarefas e seu gerenciamento é feito por ele próprio, caracterizando um ambiente multitarefa muito estável. Neste ambiente cada programa é encapsulado em uma área da memória e recebe uma prioridade atribuída pelo próprio SO, além disso, é ele que decide quando irá executar o processo. Deste modo, se o processo travar ou apresentar alguma anomalia em seu funcionamento, como o processo está encapsulado separadamente dos demais, ele pode ser eliminado sem que trave a execução dele como um todo. Este modo de gerenciamento de multitarefas também está presente no Windows XP e sistemas posteriores da Microsoft.

Já nos SOs implementados com tratamento de multitarefas cooperativas, existe também uma lista com os processos em execução, porém cada processo gere a si mesmo. Assim ele próprio decide quando irá abandonar o processador. Não tendo proteção de memória como no modo preemptivo, cada processo decide o que e faz com o processador e pode ocupar o espaço onde está outro processo, além de poder ele mesmo consumir todos os recursos da estação de trabalho sozinho, gerando instabilidade (MMOSGAME, 2015).

2.7 THREADS

Thread, ou linha de execução, nada mais é que um subprograma que trabalha independente do programa principal, sendo responsável por uma tarefa específica. A divisão das tarefas relacionadas a um programa em várias *threads* pode trazer grandes melhorias em questão de desempenho (MORIMOTO, 2005).

Segundo Jeveaux (2015) as *threads* surgiram da necessidade de elaborar uma maneira de agilizar o modo de execução dos processos, já que os mecanismos de proteção que garantem que um processo não interfira na execução de outro, tornariam o SO mais lento, justamente pelo fato de que a criação de novos processos serem muito mais custosos para a estação de trabalho, além da comunicação entre processos ser mais lenta e complexa do que com *threads*.

2.7.1 Os estados de uma thread

Segundo Horstmann; Cornell (2001), durante o ciclo de vida de uma *thread,* a mesma pode assumir 4 estados. São eles: novo (*new*), passível de execução (*run*), bloqueada e morta. Inicialmente ao ser criada a *thread* assume o estado de novo, sendo que para entrar em execução a mesma deve ser ativada posteriormente a sua criação. Após assumir o estado de passível de execução, a mesma poderá ser bloqueada e posteriormente retomar sua execução, ou findar suas atribuições e assumir o estado de morta. A Figura 1 aborda todas as transições e a ação reversa para as modificações de estado.

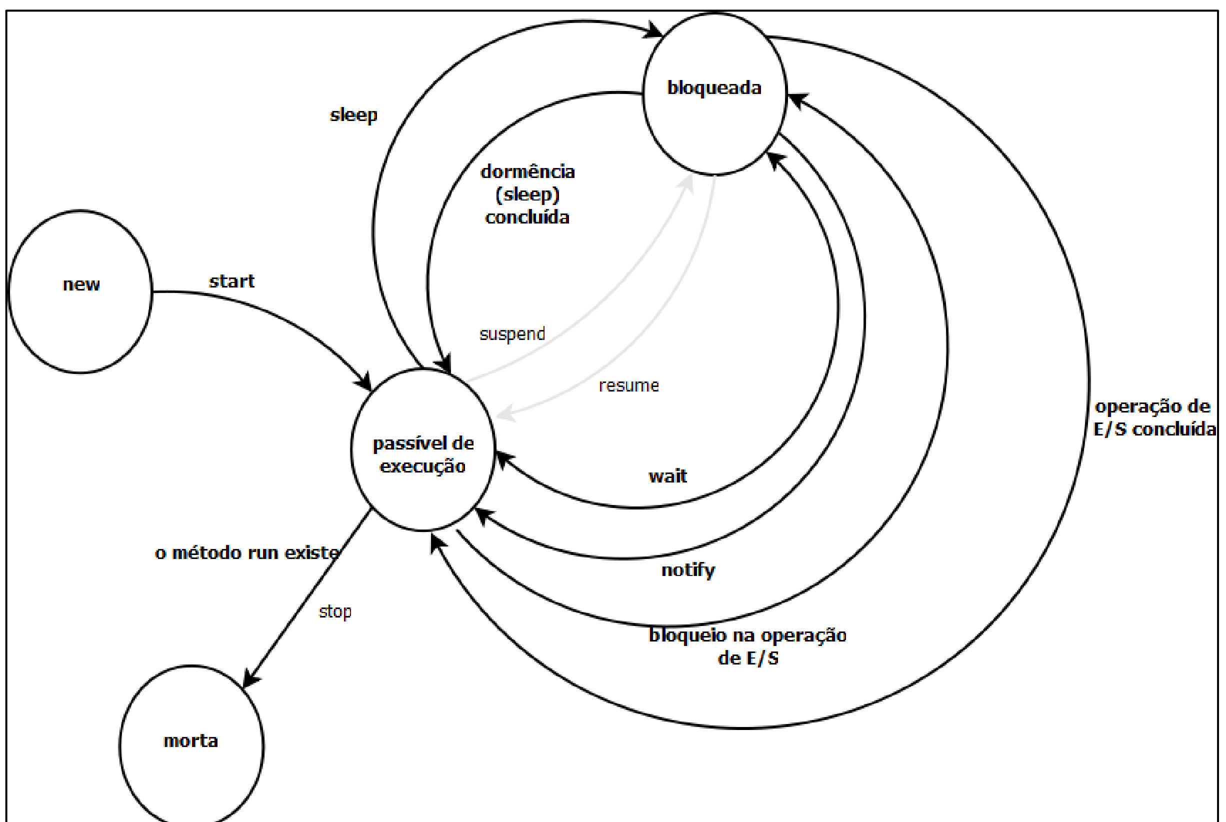

Figura 1 - Estados de uma thread.
Fonte: Adaptado de Hortsmann; Cornell (2001).

2.7.2 Prioridades de uma thread

Horstmann; Cornell (2001) explicam que em Java, toda *thread* tem uma prioridade que vem pré-definida ou pode-se modificar. Por padrão uma *thread* nova herda a prioridade da *thread* que a originou. Contudo existe a possibilidade de modificar o nível de prioridade da mesma através do método `Thread.setPriority()`, sendo o valor mínimo atribuído a

variável MIN_PRIORITY igual a 1, e o valor máximo para a priorização da *thread* através da variável MAX_PRIORITY igual a 10. Além dos extremos existe uma variável, denominada NORM_PRIORITY, que media estes valores, ou seja, com valor 5.

Tendo a oportunidade de executar outra *thread*, o agendador de *threads* possivelmente escolherá uma que tenha um nível de prioridade mais alto e que esteja com estado passível de execução. Todavia isso não é regra, pelo fato que a priorização das *threads* depende do SO vigente. Sendo que se a JVM contar com a implementação de threads conforme o SO em que ela está sendo executada, a mesma fica dependente dessa implementação. Desta maneira a JVM irá mapear as prioridades das *threads* conforme os níveis estabelecidos pelo SO, que em casos pode ter mais ou menos níveis.

2.7.3 Sincronismo de threads

O sincronismo de *threads* é utilizado basicamente para garantir que uma *thread* possa acessar um objeto com segurança, de modo que enquanto esta *thread* não finalizar sua execução, o objeto não será liberado para outras *threads*, evitando que os objetos sejam danificados pelo fato de duas ou mais *threads* terem utilizado e modificado o estado de um mesmo objeto (HORSTMANN; CORNELL, 2001).

Existem situações que podem ocasionar discrepâncias de valores ou situações onde várias *threads* possam utilizar o mesmo objeto, ou realizar a mesma operação simultaneamente. Nestes casos a linguagem Java resolve este impasse, fazendo com que somente seja adicionada a palavra reservada "*synchronized*" ao bloco da operação que não pode ser interrompida sem finalizar. O Quadro 1mostra como é adicionado esta característica aos blocos em questão.

```
public synchronized void depositar(double valor){
    double novoSaldo = this.saldo + valor;
    this.saldo = novoSaldo;
}

public synchronized void atualizar(double taxa){
    double saldoAtualizado = this.saldo * (1 + taxa);
    this.saldo = saldoAtualizado;
}
```

Quadro 1- Utilização da palavra reservada "synchronized".
Fonte: PALHETA (2013)

Horstmann; Cornell (2001) garantem que um método anotado com `synchronized` inicia e termina sua execução sem concorrência entre as *threads* que possam vir a utilizar deste método.

Bloch (2008) frisa a importância de diminuir ao máximo as instruções executadas dentro de um bloco sincronizado. Em suas palavras: “Obtenha o bloqueio, examine os dados compartilhados, transforme-os quando necessário e libere o bloqueio. Se tiver que executar alguma atividade demorada, encontre uma maneira de movê-la para fora da região sincronizada[...]”.

Na Figura 2, a *thread* denominada “thread01” iniciou sua execução e está utilizando um referido objeto. Contudo, como este objeto não foi sincronizado, o agendador de *threads* permitiu que mais outra *thread,* no caso a “thread02”, iniciasse e utilizasse o mesmo objeto.

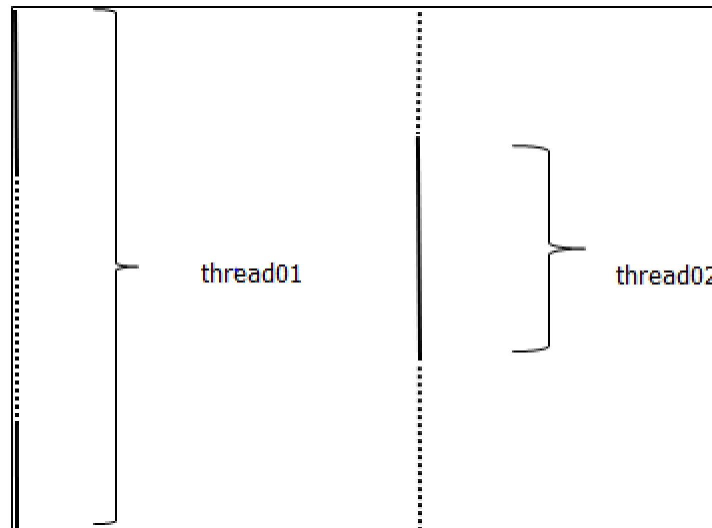

Figura 2 - Acesso de threads a um método não sincronizado.
Fonte: Adaptado de Horstmann; Cornell (2001)

Já na Figura 3, a “thread01” iniciou sua execução a um objeto sincronizado. Note que a “thread02” tentou executar novamente em paralelo com a primeira *thread.* Porém como este objeto é sincronizado a “thread02” foi impedida de executar, sendo desativada e precisou esperar o término do processamento por parte da “thread01” para que em um momento futuro a mesma possa ser escalonada pelo agendador de *threads* para executar e assim poder utilizar o método sincronizado.

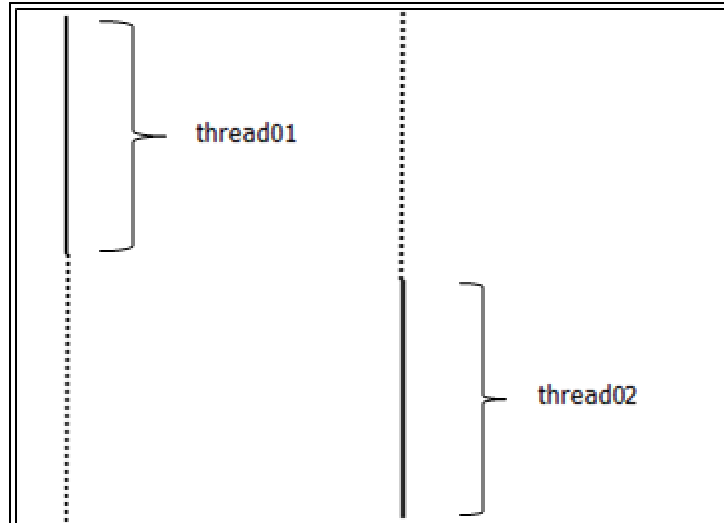

Figura 3 - Acesso de threads a um método sincronizado.
Fonte: Adaptado de Horstmann; Cornell (2001)

Conforme Horstmann; Cornell (2001) esse sincronismo ocorre pelo fato de que quando uma *thread* chama um método `synchronized`, o objeto fica bloqueado. Cada objeto possui uma chave e quando uma *thread* entra, ela ocupa essa chave e enquanto ela não executar o método a mesma não irá liberar esta chave. Quando outra *thread* tentar executar este método, ela irá procurar por uma chave disponível. Contudo a única chave para este método esta sendo ocupada, então esta outra *thread* é bloqueada, assim sendo até que a chave seja liberada pela *thread* inicial.

O agendador de *threads* ativa periodicamente as *threads* que estão no aguardo da chave para a execução do método, garantindo assim que quando a chave estiver liberada, a mesma possa ser utilizada por outra *thread*, realizando o andamento da fila.

2.7.4 Utilizando threads em Java

Para realizar diversas instruções em paralelo em programas desenvolvidos com Java, utiliza-se essencialmente a classe `Thread` e a interface `Runnable` situadas no pacote `java.lang`.

Existem duas maneiras para se utilizar *threads* em uma classe. Uma é criando uma classe que estende a classe `java.lang.Thread`, seu uso é abordado por meio do Quadro 2.

```
public class GeraPDF extends Thread {
  public void run () {
    // ...
  }
}
```

Quadro 2 - Classe Java criada utilizando herança a partir da classe java.lang.Thread.
Fonte: CAELUM (2015)

Já a outra, e também a mais indicada pelo fato de Java não ter suporte a herança múltiplas, é criando a classe que implementa a interface `Runnable`. Pois se sua classe já possuísse herança de outra classe qualquer, a mesma não poderia herdar da classe `java.lang.Thread`, não sendo possível usar ela como *thread.* Desta maneira, deve-se dar preferência em implementar `Runnable` que surte o mesmo efeito, do que herdar da classe `java.lang.Thread`. No Quadro 3 demostra-se o uso de uma classe qualquer com a implementação de `Runnable`.

Ainda nas palavras de Bloch (2008), "Já que a linguagem Java só permite a herança única, essa restrição sobre as classes abstratas limita severamente seu uso como definição de tipo.".

```
public class Programa1 implements Runnable {
  public void run () {
    for (int i = 0; i < 10000; i++) {
      System.out.println("Programa 1 valor: " + i);
    }
  }
}
```

Quadro 3- Classe Java criada implementando a interface Runnable.
Fonte: CAELUM (2015)

2.7.4.1 A utilização de threads com *Swing*

Horstmann; Cornell (2001) indicam que precisa haver cautela ao utilizar *threads* em conjunto com *Swing*, pois o mesmo não tem segurança para utilizar *threads*. Tal insegurança ocorre pelo fato de que a maioria dos métodos das classes *Swing* não são sincronizados. Possivelmente ao manipular elementos de interface por intermédio de várias *threads*, a interface com o usuário ficará danificada.

Conforme Horstmann; Cornell (2001) um aplicativo desenvolvido em *Swing* basicamente se basea em manipuladores de eventos que são controlados por uma *thread,* visando responder as solicitações perante a GUI e seus pedidos de repintura. Para utilizar

threads da maneira mais correta quando se está trabalhando com *Swing,* existem algumas regras para se analisar:

1. Se a ação desenvolvida demorar demais, deve-se criar uma nova *thread* para atender esta ação, pois se ela está demorando irá causar a impressão de que o *software* está travado, e o mesmo não poderá responder a outros eventos antes do término desta ação.
2. Caso uma ação possa bloquear a entrada ou a saída, crie uma *thread* para realizar esta ação, evitando o congelamento da GUI enquanto está ação estiver processando.
3. Havendo a necessidade de esperar por um determinado período de tempo, não deixe o *software* parado na *thread* que trata o envio de eventos do *Swing*, use um temporizador.
4. O processamento dentro das suas *threads* não pode modificar a GUI. Processe todas as informações referentes a GUI antecipadamente, depois ative suas *threads* que irão processar estas informações e após finalizar este processo, atualize a GUI através da *thread* criada pelo *Swing* para tratar o envio de eventos.

Esta última regra apresentada, é comumente chamada de regra de *thread* única, estritamente usada em *Swing.* Todavia existem exceções para o uso desta regra:

1. Existem alguns métodos pertencentes a *Swing* que são seguros para serem utilizados com *threads.* Os mesmos são identificados de uma maneira particular aos demais, na documentação da API *Swing.* Alguns deles são: `JTextComponent.setText()`, `JTextArea.insert()`, `JTextArea.append()`.
2. Os métodos `JComponent.repaint()` e `JComponent.revalidate()` podem ser invocados a partir de qualquer *thread.*
3. Pode-se construir e configurar os componentes *Swing*, além de incluí-los em contêineres, desde que o componente não possa ainda receber eventos de pintura ou validação. Isto ocorre pela invocação dos métodos `show()`, `setVisible(true)` ou `pack()` para o componente ou o contêiner a qual o mesmo foi adicionado. Ou seja, quando tiver ocorrido qualquer uma dessas situações, não haverá mais a possibilidade de manipular o componente a partir de outra *thread* criada pelo programador.

Segundo Horstmann; Cornell (2001) se houver a necessidade de modificar algum componente por meio da *thread* desenvolvida pelo programador, o mesmo deverá utilizar dois

métodos, que tem por funcionalidade incluir ações arbitrárias na fila de eventos do *Swing*. Assim sendo não há como, por exemplo, invocar o método `JLabel.setText()` de dentro da implementação de sua *thread*, mas poderá utilizar os métodos `invokeLater()` ou `invokeAndWait()` pertencentes a classe `EventQueue` para poder realizar a chamada ao método `JLabel.setText()`, e o mesmo ser executado na *thread* correspondente ao envio de eventos da API *Swing*. Desta forma, é necessário assim como visto no Quadro 4, adicionar o código *Swing* que se deseja executar ao método `run()` de uma classe qualquer que implemente `Runnable`.

```
public class LabelUpdater implements Runnable{
    private JLabel label;
    private int percentage;

    public LabelUpdater(JLabel aLabel, int aPercentage){
        label = aLabel;
        percentage = aPercentage;
    }

    public void run(){
        label.setText(percentage + "% concluidos");
    }
}
```

Quadro 4- Classe criada para executar comandos Swing de dentro de outra thread.
Fonte: Adaptado de Horstmann; Cornell (2001)

Após isto, cria-se um objeto pertinente a esta classe dentro da *thread* desenvolvida pelo programador e o parametriza, ou ao método `invokeLater()` ou ao `invokeAndWait()`. Conforme se observa no trecho de código-fonte do Quadro 5.

```
Runnable updater = new LabelUpdater(label, percentage);
EventQueue.invokeLater(updater);
```

Quadro 5- Invocação da classe criada para execução de comandos Swing dentro de outra thread.
Fonte: Adaptado de Horstmann; Cornell (2001)

Quando utilizado o método `invokeLater()`, o mesmo retorna imediatamente logo após o evento ser enviado para a *thread* que é responsável pela manipulação dos eventos *Swing*, executando o método `run()` de maneira assíncrona, mas utilizando o `invokeAndWait()`, só há um retorno quando o método `run()` tenha sido executado. Não é necessário se preocupar com questões de sincronismo, pois a classe `EventQueue` trata das mesmas.

2.8 O PADRÃO MVC

Segundo Fonseca; Alves (2015) um *software* que realiza acesso a dados, tem lógica de negócio implementada, e uma GUI para interação com o cliente, possui grandes dificuldades em ser mantido. Tal dificuldade se dá pelo fato de que a interdependência entre os componentes pode ocasionar um efeito cascata, cada vez que uma alteração é realizada no *software* como um todo.

Com isso adicionar novas telas, realizar manutenção no código-fonte, excluir algo, enfim reformular o *software*, às vezes requer modificações em diversos pontos, e no final das contas o código-fonte se torna confuso, problemático e desastroso.

Para melhorar este impasse, foi desenvolvido o padrão MVC, ou Modelo, Visão e Controle. Tal padrão tem o propósito de separar o código em 3 partes. No modelo fica acondicionado todo o código-fonte responsável pelo acesso aos dados que são consumidos pelo *software*. Na parte de Visão fica toda a parte de interação com o usuário, as já fundamentadas GUI. E por fim na parte de Controle a implementação de regras de negócio.

Esta simples ação de separar o código-fonte por funções ou responsabilidades, deixa o *software* com manutenibilidade mais rápida e este processo se torna muito menos propício a erros e de entendimento mais claro. Sendo um fator que para o programador é primordial, ainda mais em casos que o *software* não foi implementado por ele mesmo. (FONSECA; ALVES, 2015).

O MVC foi criado por Trygve Reenskaug antes mesmo da Internet no ano de 1979 para inicialmente ser usado em aplicações *desktop* na linguagem de programação SmallTalk (AKITA, 2006).

2.8.1 Fluxo de interações do MVC

Não basta saber o que cada parte do padrão MVC deve conter. É preciso entender como devem ser desenvolvidas e como são realizadas as interações entre cada parte deste padrão para que se tenha entendimento como um todo da utilização do mesmo (MEDEIROS, 2015).

Um cliente interage com a GUI, nenhum *software* dá a possibilidade de que o cliente possa interagir diretamente com outra parte do sistema senão as suas telas. Tendo em mente que o processo então começa pela parte de visão (*view*), há a seguinte situação: o cliente

interage com a GUI de uma forma qualquer, ao realizar uma ação perante a visão, essa ação é enviada ao controle (*controller*) que tem a função de verificar o que a camada de visão solicitou por meio do cliente, e deliberar ações que satisfação à requisição. O controle não tem conhecimento direto da existência da visão, somente intermedia as ações quando a visão necessita de algo. Caso a camada de controle verificar que precisa utilizar a camada de modelo (*model*) para buscar alguma informação ou algo do tipo para sanar a necessidade da camada de visão. Neste momento geralmente existe uma modificação nas informações do modelo. A camada de visão usa o modelo para apresentar ao usuário as informações solicitadas por ele. Contudo o modelo não tem conhecimento direto da existência da camada de visão. Ele apenas está para servir, respondendo às requisições que lhe são incumbidas e atualiza os dados do *software*. Apresentado o resultado da solicitação do usuário na camada de visão, o controle fica no aguardo de novas solicitações, sendo esse um ciclo vicioso (MEDEIROS, 2015).

Conforme Medeiros (2015), as vantagens mais chamativas para a incorporação do MVC na elaboração de *softwares* são: possibilidade de reescrita da GUI (camada de visão) ou do *Controller* (camada de controle) sem precisar alterar a camada de modelo, poder reutilizar a camada de visão em outros *softwares* sem a necessidade de modificações excessivas no código-fonte, manutenibilidade mais ágil, código-fonte mais limpo e, portanto, mais organizado.

Nas palavras de Fonseca; Alves (2015), “Certamente a produtividade pode andar junto com a qualidade desde que boas técnicas sejam adotadas no processo. Entre tantas técnicas vale pesquisar e aplicar o MVC talvez a mais simples, obvia e eficaz de todas”.

3 MATERIAL E MÉTODOS

Apresenta-se nesta seção, a metodologia que foi adotada na elaboração deste trabalho, assim como as tecnologias e materiais utilizados para desenvolver os *softwares* que serviram como base para os testes de desempenho e afins.

3.1 METODOLOGIA

Os *softwares* que serviram de base para as métricas de desempenho foram elaborados distintamente. Isso porque um *software* abordava somente programação utilizando *Swing* e a biblioteca JFreeChart e o outro *JavaFX*. Todavia estes *softwares* tinham funcionalidades totalmente similares, sendo que a grande diferença visível estava na parte visual dos mesmos.

O desenvolvimento destes aplicativos seguiu essencialmente o padrão MVC, reutilizando em sua essência a camada de Modelo e Controle. As modificações mais expressivas se deram na camada de Visão como o esperado, por se tratar de programação com duas APIs distintas, estas discrepâncias são inevitáveis levando em consideração as peculiaridades e limitações que existem em cada API na hora de programar com uma ou outra.

3.1.1 Procedimento de desenvolvimento dos softwares

Estes *softwares* na sua essência foram implementados com a capacidade de obter dados de arquivos CSV (*Comma-separated values*), arquivos estes separados por um caractere especial, geralmente a vírgula. A partir da manipulação deste arquivo, que foi realizada através de uma biblioteca Java denominada, JCSV[9], os *softwares* ficam incumbidos de gerar gráficos. Para desenvolver os gráficos com a API Swing optou-se pela utilização da biblioteca JFreeChart(ver capítulo 2.5) pela sua gratuidade e uso bastante difundido no desenvolvimento de gráficos na linguagem Java , todavia existem outras opções para a criação destes gráficos, como por exemplo: *RSwing* que é uma biblioteca gratuita, e *NetCharts Pro*

---

[9] JCSV é uma biblioteca Java para a manipulação de arquivos no formato ".csv", com muito mais agilidade para o desenvolvedor. Sendo que a biblioteca tem a capacidade de vincular arquivos ".csv" diretamente a objetos da linguagem Java (JSCV, 2012).

que por sua vez é uma biblioteca não-gratuita (RIBEIRO; *et al.*, 2015). Já para a API JavaFX foi utilizado os recursos da própria API.

Como a massa de dados utilizada é uma massa com quantidade de dados consideráveis, o uso de *threads* foi utilizado para que o processo tanto de leitura dos dados e a geração dos gráficos, obtivesse um ganho de desempenho mais satisfatório.

Com base neste cenário descrito a Figura 4 ilustra o diagrama de classes genérico criado para a elaboração dos *softwares*.

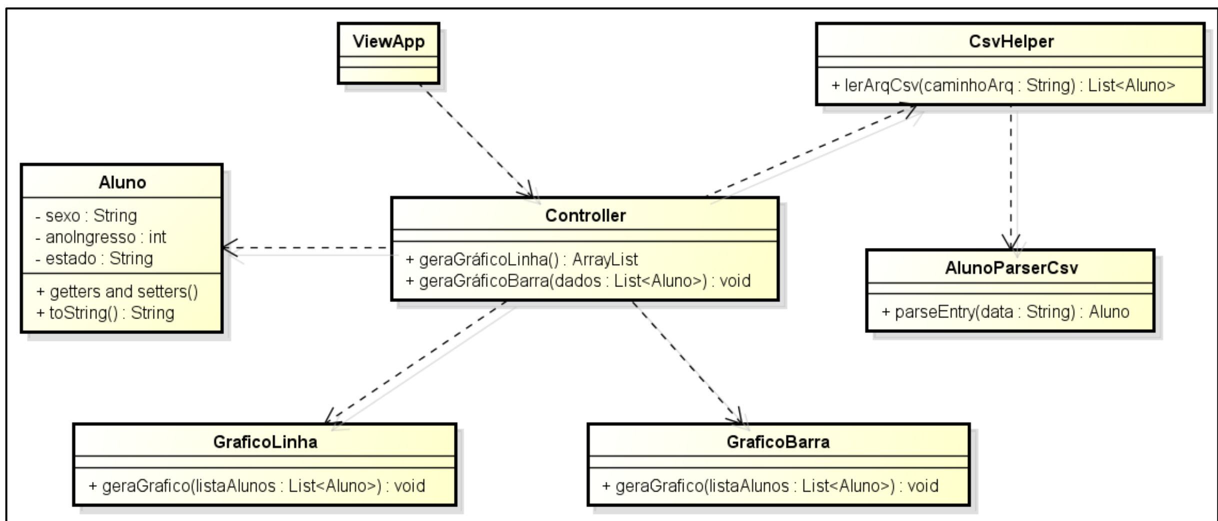

Figura 4 - Diagrama de Classes genérico para o cenário proposto.
Fonte: Autoria própria.

3.1.2 Parâmetros de avaliação

Para poder avaliar os *softwares* em questão de desempenho foram traçados alguns parâmetros de avaliação:

a. Uso de memória *Heap*: avaliação da utilização de memória *Heap* em tempo de execução dos testes nos dois *softwares.*

b. Uso de CPU (*Central Processing Unit*): avaliação do uso percentual de recursos do CPU em tempo de execução dos testes nos dois *softwares.*

3.1.3 Metodologia de testes

Foi elaborada uma rotina de testes comum entre os *softwares* sendo utilizada a mesma massa de dados, a mesma estação de trabalho (ver capítulo 3.2), cuja qual foi seguida em

todos os testes. A Figura 5 demonstra o diagrama de atividades, com todos os passos executados em cada um dos testes realizados.

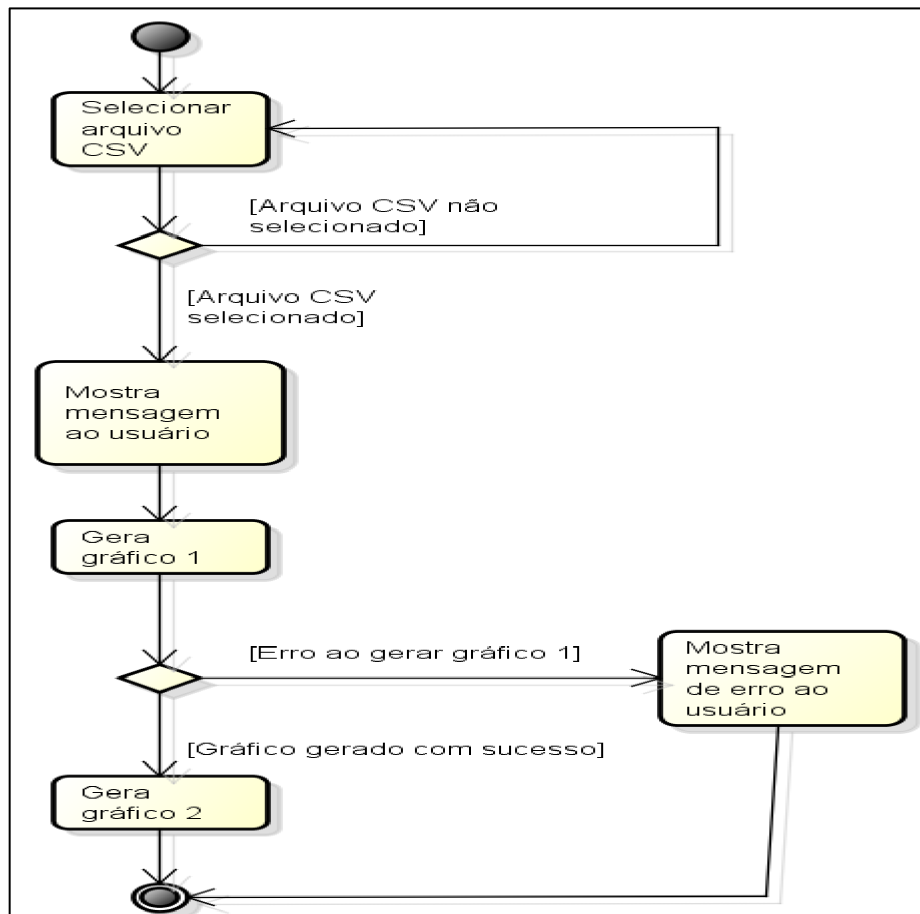

Figura 5 - Diagrama de Atividades para execução dos testes de desempenho.
Fonte: Autoria própria.

A massa de dados utilizada para a realização dos testes contém dados fictícios e foi gerada por intermédio do sítio http://www.generatedata.com/, sendo criados registros que descrevessem o sexo, ano de ingresso e o estado brasileiro de onde os alunos de uma faculdade qualquer vieram.

Os testes foram monitorados por intermédio das ferramentas JConsole e Java VisualVM as quais tinham a finalidade de acompanhar desde a instanciação dos *softwares* junto a JVM da máquina hospedeira, até a término da execução dos mesmos, para assim obter as medidas referentes aos parâmetros de avaliação fixados para tais testes.

3.2 TECNOLOGIAS UTILIZADAS

Para a realização dos testes entre os *softwares* foi utilizada a estação de trabalho com as seguintes configurações de ambiente:

- Sistema Operacional: Windows 7 Ultimate 64 bits;
- Versão Java: 1.8.40;
- Versão *JavaFX*: 2.1.1.

O *hardware* desta estação de trabalho possui as seguintes especificações:

- Memória RAM: 6 GB, modelo: DDR3, *dual channel*, 665 MHz;
- Disco Rígido: 600 GB, velocidade: 5400 RPM, modo de transferência: SATA II 3.0 GB/s;
- Processador: Intel Core I5-2450M CPU @ 2.50 GHz, quantidade de núcleos: 2, threads: 4.

Durante a elaboração dos *softwares* foram utilizados alguns programas para o desenvolvimento dos mesmos. Dentre eles está a IDE[10] Netbeans 8.0.2, *JavaFX* Scene Builder 2.0 e o JConsole 1.7.0.5.

3.2.1 Interface de Desenvolvimento Integrado Netbeans

Segundo Araújo (2011), o programador necessita dispor de maior esforço para realizar atividades ligadas ao raciocínio lógico, para isso é fundamental, conhecer e utilizar de forma eficiente algum Ambiente de Desenvolvimento Integrado, ou IDE. Pois o domínio somente da linguagem de programação pode não ser suficiente para a eficiência do trabalho do programador, já que por sua vez a IDE realiza grande parte do trabalho manual e repetitivo que sem o auxílio da mesma deixaria o programador sobrecarregado, reduzindo o tempo em que ele poderia estar pensando no desenvolvimento de funcionalidades e afins.

O Netbeans é um dos exemplos de IDE, foi originado pelo projeto Xelfi, do ano de 1996, por dois estudantes universitários da Tchecoslováquia. No ano de 1999, ao se tornar um IDE proprietário de nome Netbeans DeveloperX2, foi comprado pela Sun Microsystems, onde poucos meses após sua compra, foi tornado um projeto de licença gratuita, ou *open source*. Mesmo após a compra da Sun Microsystems pela Oracle, em 2009, o Netbeans continua até hoje como um projeto gratuito para programadores do mundo inteiro (ARAÚJO, 2011).

---

[10] "Um ambiente de desenvolvimento integrado (IDE) é uma configuração de programação única na qual você tem todas as ferramentas necessárias à sua disposição. Geralmente, um editor de código-fonte que é cercado por um compilador, um depurador e outras ferramentas de desenvolvimento." (CARVALHO, 2015).

A IDE Netbeans se destaca dentre vários pontos, por possuir um editor de código-fonte e o GUI Builder, para a construção de interfaces gráficas com o recurso conhecido como "*drag-and-drop*", ou arrastar e soltar (ARAÚJO, 2011).

3.2.2 *JavaFX* Scene Builder

Como visto anteriormente o *JavaFX* utiliza um arquivo com recursos bastante semelhante ao XML para elaborar as GUIs, o FXML. Contudo se não for utilizado uma IDE para criar este arquivo, o trabalho se torna bastante improdutivo para o programador.

Para facilitar a criação destes arquivos que contém a descrição das GUIs dos *softwares* desenvolvidos em *JavaFX,* foi criada a IDE, *JavaFX* Scene Builder. Também possui a funcionalidade de arrastar e soltar, sendo que o programador molda a tela e a IDE automaticamente atualiza o arquivo FXML e disponibiliza o mesmo para ser utilizado na elaboração das GUIs. Além disso, com essa ferramenta da Oracle, o programador pode trabalhar com CSS diretamente nos componentes adicionados, mudando estilos, efeitos, dentre outros (OLIVEIRA, 2015).

3.2.3 JConsole

JConsole é uma ferramenta de monitoramento gráfico para aplicações desenvolvidas em Java. Tal ferramenta vem inclusa na plataforma Java SE, sendo que a mesma implementa a API JMX[11], tendo como finalidade monitorar o desempenho dos aplicativos que estão sendo executados em JVMs tanto locais como remotas, para assim descobrir falhas de desempenho. O JConsole foi introduzido na versão 5 da plataforma Java SE, mas o suporte oficial foi dado na versão 6 (ORACLE, 2015).

Conforme Neward (2010) esclarece, muitos programadores nem sabem da existência dessa ferramenta que vem embutida dentro do diretório de instalação do JDK, sendo parte do conteúdo da pasta "bin", por meio do arquivo denominado "jconsole".

---

[11] "A API Java *Management Extensions* (JMX) é uma API padrão para gerenciamento e monitoramento dos recursos como aplicações, dispositivos, serviços e o Java *virtual machine*. A tecnologia JMX foi desenvolvida através do Java Community Process (JCP) como Java *Specification Request* (JSR) 3, Java *Management Extensions* e JSR 160, JMX remoto API." (ORACLE, 2015).

4 RESULTADOS E DISCUSSÃO

Com interface simples, os *software*s têm a finalidade de dispor ao cliente a escolha do arquivo CSV, que contém os dados dos alunos e a invocação dos dois gráficos que foram elaborados em cada *software*. Um dos gráficos foi desenvolvido no formato de linhas, sendo que o mesmo têm como finalidade mostrar a evolução do gênero (masculino x feminino) ao longo de cada ano, compreendendo, como informado anteriormente os anos entre 2007 a 2011. O segundo gráfico por sua vez foi desenvolvido no estilo de barras, e o mesmo têm como propósito, ilustrar a somatória de alunos por estado que ingressaram durante os anos de 2007 a 2011.

Após a seleção do arquivo CSV, o sistema irá realizar todo o processamento de maneira automática apresentando o gráfico de barra, sendo que será necessária somente a intervenção do usuário para realizar a chamada do outro gráfico, no caso o de linha. Foi implementado para a seleção do arquivo uma regra de negócio onde o sistema permite somente a entrada de arquivo no formato CSV, evitando com que seja realizada a manipulação de dados que ocasionariam erro à execução do *software*.

4.1 REALIZAÇÃO DOS TESTES

Tendo finalizado o desenvolvimento dos *softwares* foi realizado os testes de desempenho comparando um *software* com o outro. Tais testes foram elaborados monitorando os aplicativos com as ferramentas JConsole e Java VisualVM do próprio JDK instalado na máquina hospedeira.

Os testes foram executados inúmeras vezes para garantir a integridade dos aplicativos e para servirem como base para gerar as métricas de desempenho e posteriormente as devidas comparações entre o aplicativo desenvolvido com Swing e JavaFX.

Seguindo a metodologia de testes especificada, obtêm-se para cada *software* dois gráficos. As Figura 6 e Figura 7 ilustram os gráficos de barra que são exibidos automaticamente pelos *softwares*.

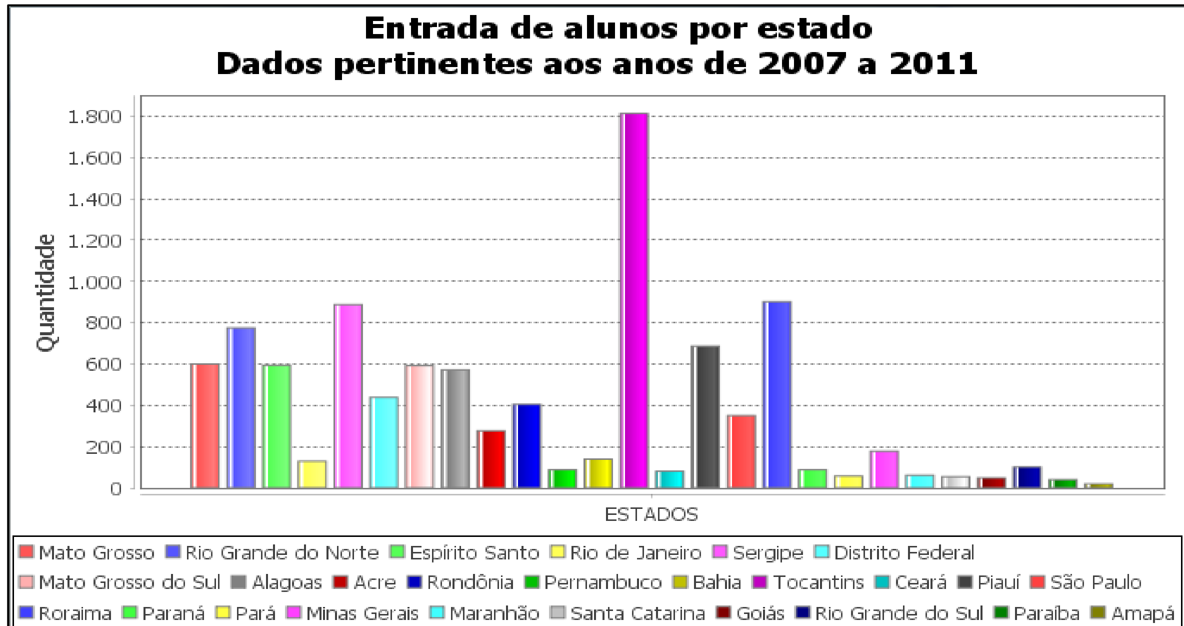

Figura 6 - Gráfico de barra gerado pelo *software Swing*.
Fonte: Autoria própria.

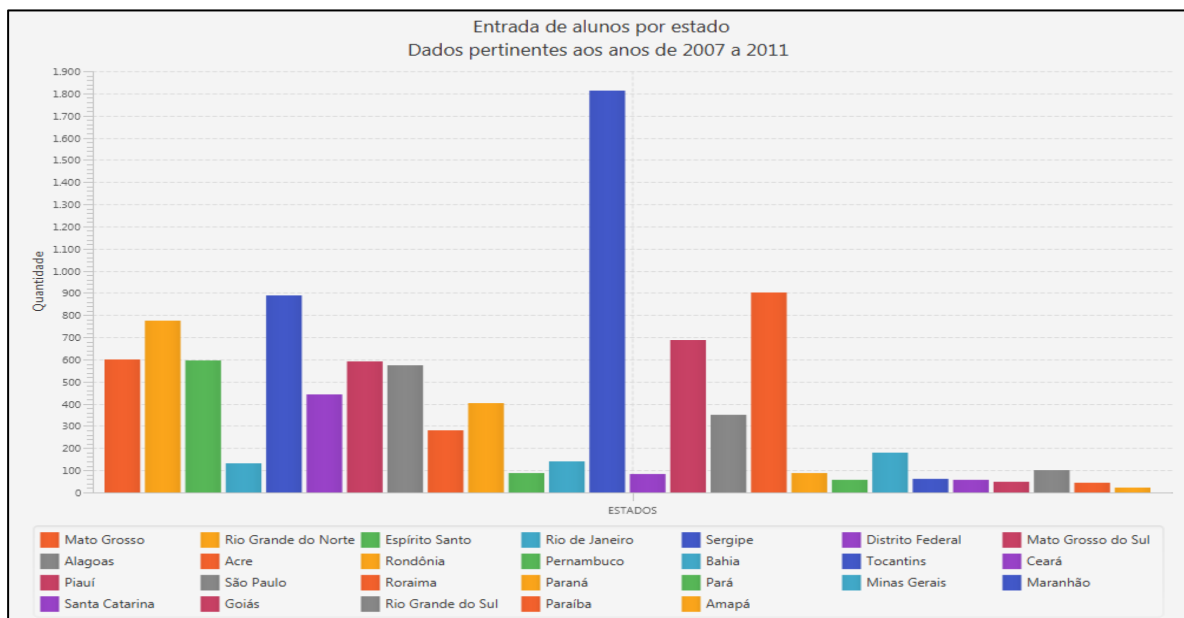

Figura 7 - Gráfico de barra gerado pelo *software JavaFX*.
Fonte: Autoria própria.

A intervenção que cabe ao usuário para a geração e exibição do gráfico de linha acontece na tela inicial dos *softwares* onde se encontra um botão autoexplicativo para tal evento. Veja nas Figura 8 e Figura 9, o resultado obtido na geração destes gráficos com a mesma massa de dados que geraram os primeiros gráficos.

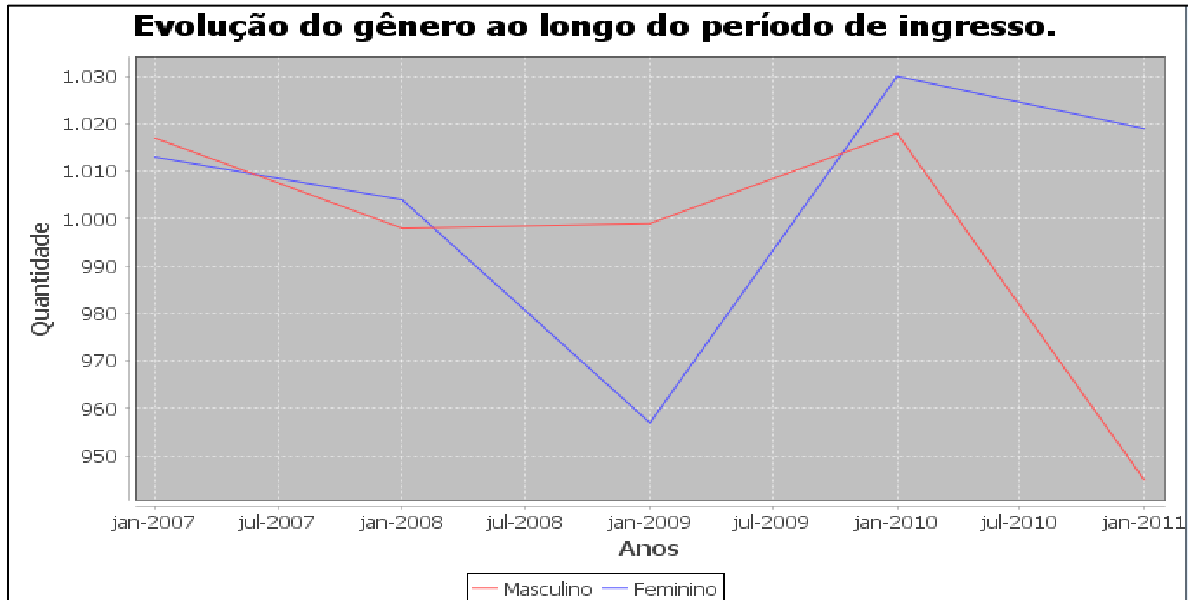

Figura 8 - Gráfico de linha gerado pelo *software Swing*.
Fonte: Autoria própria.

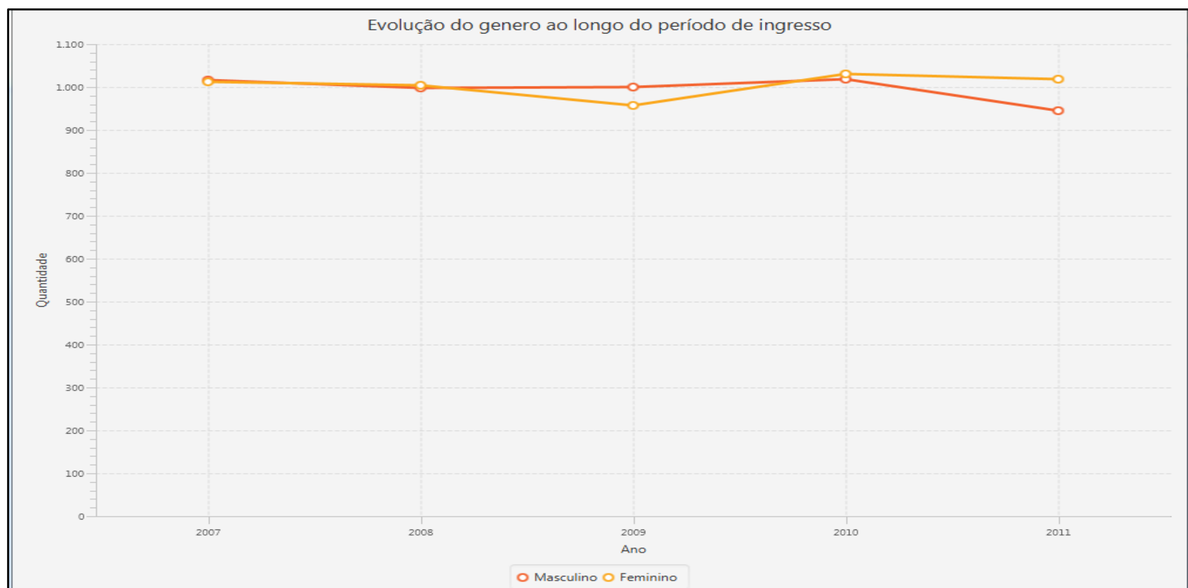

Figura 9 - Gráfico de linha gerado pelo *software JavaFX*.
Fonte: Autoria própria.

4.2 MÉTRICAS DE DESEMPENHO

As métricas de desempenho obtidas pela manipulação do arquivo CSV descrito anteriormente e que possuía 10.000 registros, estão discriminadas a seguir e serão comparadas num segundo momento.

4.2.1 Métricas de desempenho para o software em Swing

As métricas de uso de memória *Heap* para o aplicativo desenvolvido em *Swing*, usando tanto o JConsole como o Java VisualVM para monitoramento foram as seguintes (Quadro 6) .

Ferramenta	Uso de memória Heap (MB) Teste n.						Média	Desvio Padrão	Variância
	1	2	3	4	5	6			
JConsole	19,8	16,6	17,4	24	21,5	16,8	19,3	2,71	7,38
Java VisualVM	25,4	23,1	22,6	26,3	22,1	23,1	23,7	1,53	2,35

Quadro 6 - Métricas de desempenho do *software* baseado em Swing para uso de memória *Heap*.
Fonte: Autoria própria.

Já para o uso da CPU os dados obtidos por meio dos testes estão discriminados a seguir (Quadro 7).

Ferramenta	Uso de CPU (%) Teste n.						Média	Desvio Padrão	Variância
	1	2	3	4	5	6			
JConsole	12,6	13,8	15,2	14,9	12,7	13	13,7	1,02	1,06
Java VisualVM	24,2	19,2	30	26,5	23,1	39,2	27	6,34	40,32

Quadro 7 - Métricas de desempenho do software baseado em Swing para uso de CPU.
Fonte: Autoria própria.

4.2.2 Métricas de desempenho para o software em JavaFX

As métricas de uso de memória *Heap* para o aplicativo desenvolvido em *JavaFX*, usando tanto o JConsole como o Java VisualVM para monitoramento foram as seguintes (Quadro 8) .

Ferramenta	Uso de memória Heap (MB) Teste n.						Média	Desvio Padrão	Variância
	1	2	3	4	5	6			
JConsole	34	33,7	34,2	29,2	32	20	30,5	5,00	25,06
Java VisualVM	32,6	18,3	33,3	34,1	33,3	30,3	30,3	5,50	30,28

Quadro 8 - Métricas de desempenho do software baseado em JavaFX para uso de memória Heap.
Fonte: Autoria própria.

Já para o uso da CPU com o aplicativo desenvolvido em *JavaFX*, os dados obtidos por meio dos testes estão discriminados a seguir (Quadro 9Quadro 7).

Ferramenta	Uso de CPU (%) Teste n.						Média	Desvio Padrão	Variância
	1	2	3	4	5	6			
JConsole	17	15	17,4	17,8	23,6	19,9	18,4	2,71	7,35
Java VisualVM	42	44,8	41,6	47,6	48,9	44,2	44,8	2,67	7,17

Quadro 9 - Métricas de desempenho do software baseado em JavaFX para uso de CPU.
Fonte: Autoria própria.

4.3 COMPARAÇÕES ENTRE TESTES

Realizando comparações entre as métricas apresentadas (ver capítulo 4.2), percebe-se que os testes realizados tiveram amostras homogêneas, pois o coeficiente de variação representado pela fórmula:

$$CV = \frac{\sigma(x)}{\bar{x}} \text{ x } 100$$

apresentou valores inferiores a 20% (por cento). Somente os testes realizados com o Java VisualVM para a métrica de uso de CPU com o *software* em *Swing* (Quadro 7) obteve um coeficiente de variação maior que 20%, tornando esta amostra heterogênea.

Contudo os testes demonstraram que o *software* desenvolvido com *Swing* requer menos demanda de recursos, tanto para a memória quanto para o uso de CPU, se comparado com o *software* em *JavaFX*.

Enquanto a média, considerando os valores dos dois *softwares* de testes, para o uso de memória *Heap* do aplicativo em *Swing* alcançou 21,5MB (Quadro 6), para *JavaFX* este valor alcançou 30,4MB (Quadro 8). Já para o uso de CPU está média levando em consideração o

mesmo ambiente de testes, para *Swing* obteve valor de 20,35% (Quadro 7) e para *JavaFX* a média alcançada chegou a 31,6% (Quadro 9).

O *software* baseado em *Swing* ao realizar acompanhamento através do JConsole e Java VisualVM, mostrou-se mais estável e com um consumo mais gradativo de memória *Heap*, conforme a analogia entre as variâncias apresentadas entre o Quadro 6 e Quadro 8. Contudo ao comparar o uso do CPU, está variância fica desfavorável ao *Swing* que possui um nível de variância bem mais elevado do que se comparado com *JavaFX* (Quadro 7 e Quadro 9).

Talvez o sistema *JavaFX* requer maior uso dos recursos da máquina hospedeira, pelo fato de como ser uma API voltada para RIA tendo seus componentes visuais um custo maior para serem gerados e manipulados pelo sistema.

Ressaltasse que os testes foram refeitos inúmeras vezes (quantidade de testes maior que 6) e os resultados mantiveram no geral a mesma linha de pensamento.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Após o levantamento de todas as informações sobre o estudo realizado e analisando os resultados obtidos como um todo pode-se verificar que:

a) Mesmo a plataforma de desenvolvimento Java estar em um processo de transição, admitido pela própria detentora de seus direitos, a Oracle, no que diz respeito a ambiente *desktop*, *Swing* ainda é uma opção mais confiável e prática pelo fato de já ser uma API consolidada. *JavaFX* tem potencial considerável, mas ainda está amadurecendo em alguns pontos, como questões de desempenho e afins.
b) Em uma avaliação generalizada, Swing se mostrou ser uma API com recursos menos custosos à máquina hospedeira em comparativo com *JavaFX*.
c) Se o seu *software* for desenvolvido em *Swing* e o ambiente em que está propagado o mesmo tiver limitações de *hardware*, cabe avaliar se uma possível migração para *JavaFX* será suportada, pois além de todo o custo para realizar a migração do sistema *Swing* para *JavaFX*, ainda poderá ser necessário uma atualização de *hardware*.
d) Se a equipe de desenvolvimento tiver a necessidade de elaborar um novo *software* para *desktop* e esta equipe dispor de profissionais com capacitação além de dispor de um tempo mais flexível, e o cliente presar muito pela parte visual do *software*, a equipe pode optar por utilizar *JavaFX*, pois seus componentes são mais apresentáveis e sofisticados na questão visual, até mesmo por se tratar de uma API para aplicações RIA.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Sugere-se como trabalho futuro a realização de testes de desempenho por meio das ferramentas JConsole, Java VisualVM e Java Mission Control, provenientes do próprio JDK, em aplicações com maior quantidade de funcionalidades e carga de dados.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AKITA, F. **Repensando a WEB com Rails**. 1ª. ed. Rio de Janeiro: Brasport, v. I, 2006.

ARAÚJO, C. O IDE Netbeans - Parte 1. **Javafree.org**, 2011. Disponivel em: <http://javafree.uol.com.br/artigo/882453/O-IDE-NetBeans.html>. Acesso em: 08 Abril 2015.

BLOCH, J. **Java Efetivo**. 2ª. ed. Rio de Janeiro: ALTA BOOKS, v. I, 2008.

CADENHEAD, R.; LEMAY, L. **Aprenda em 21 dias Java 2**. 1ª. ed. Rio de Janeiro: Elsevier Editora Ltda, v. I, 2005.

CAELUM **Apostila Laboratório Java com Testes, XML e Design Patterns**. **Caelum**, 2014. Disponivel em: <http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/interfaces-graficas-com-swing/>. Acesso em: 11 Dezembro 2014.

CAELUM **Apostila Java e Orientação a Objetos**. **Caelum**, 2015. Disponivel em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/programacao-concorrente-e-threads/>. Acesso em: 25 Março 2015.

CAELUM. Gráficos com JFreeChart. **Caelum**, 2015. Disponivel em: <http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/graficos-com-jfreechart/#7-1-jfreechart>. Acesso em: 1 Abril 2015.

CARVALHO, M. L. B. D. **Ambiente de Desenvolvimento Integrados (IDE)**., 2015. Disponivel em: <http://homepages.dcc.ufmg.br/~mlbc/cursos/internet/java/>. Acesso em: 24 Março 2015.

CAVALÉRO, P. Swing: O Desktop Java. **DevMedia**, 2014. Disponivel em: <http://www.devmedia.com.br/swing-o-desktop-java-revista-easy-java-magazine-12/22860>. Acesso em: 10 Dezembro 2014.

COSTA, R. G. P. D. **Universo Java**. São Paulo: Digerati Books, v. I, 2008.

FONSECA, M. V.; ALVES, A. L. **MVC & FRAMEWORK**., 2015. Disponivel em: <http://www.cpgls.ucg.br/ArquivosUpload/1/File/V%20MOSTRA%20DE%20PRODUO%20CIENTIFICA/EXATAS/2-.PDF>. Acesso em: 07 Abril 2015.

GIONGO, S.; ARAÚJO, E. C. D.; RODRIGUES, D. Introdução ao JavaFX 2.0 - Revista easy Java Magazine 23. **DevMedia**, 2014. Disponivel em: <http://www.devmedia.com.br/introducao-ao-javafx-2-0-revista-easy-java-magazine-23/26035>. Acesso em: 11 Dezembro 2014.

HORSTMANN, C. S.; CORNELL, G. **Core Java 2 - Recursos Avançados**. 1ª. ed. São Paulo: MAKRON Books, v. II, 2001.

JEVEAUX, C. M. Utilizando Threads - parte 1. **DEVMEDIA**, 2015. Disponivel em: <http://www.devmedia.com.br/utilizando-threads-parte-1/4459>. Acesso em: 31 Março 2015.

JFREECHART. Welcome To JFreeChart! **JFreeChart**, 2015. Disponivel em: <http://www.jfree.org/jfreechart/>. Acesso em: 1 Abril 2015.

JOBSTRAIBIZER, F. **Criação de Sites com CSS**. 1ª. ed. São Paulo: Digerati Books, v. I, 2009.

JSCV. JCSV Simple CSV library for Java. **Google Code**, 2012. Disponivel em: <https://code.google.com/p/jcsv/wiki/Welcome?tm=6>. Acesso em: 20 Maio 2015.

LEAL, M. JavaFX. **Globalcode**, 2009. Disponivel em: <http://www.globalcode.com.br/noticias/EntrevistaMauricioLeal>. Acesso em: 12 Dezembro 2014.

LOPES, S. Falando em Java: Introdução ao JavaFX. **Caelum**, 2007. Disponivel em: <http://blog.caelum.com.br/falando-em-java-introducao-ao-javafx/>. Acesso em: 12 Dezembro 2014.

MARQUEZINI, A. D. S.; AGUINALDO, E. C.; ALMEIDA, O. P. D. Desenvolvimento de Aplicação de Ponto de Venda usando JavaFX. **Tekhne e Logos**, Botucatu, v. IV, n. 3, Março 2013. ISSN ISSN. Disponivel em: <www.fatecbt.edu.br/seer/index.php/tl/article/download/242/191>. Acesso em: 11 Dezembro 2014.

MEDEIROS, H. Introdução ao Padrão MVC. **DevMedia**, 2015. Disponivel em: <http://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. Acesso em: 07 Abril 2015.

MEDINA, R. D. **Informática Universidade Federal de Santa Maria**., 2014. Disponivel em: <http://www-usr.inf.ufsm.br/~rose/curso3/cafe/XML-Cap1-Linguagem.pdf>. Acesso em: 13 Dezembro 2014.

MMOSGAME. Monotarefa e Multitarefa Preemptiva e Cooperativa. **MMOSGAME**, 2015. Disponivel em: <http://mmosgame.com/monotarefa-multitarefa-preemptiva-cooperativa/>. Acesso em: 31 Março 2015.

MORIMOTO, E. Thread. **Guia do Hardware.net**, 2005. Disponivel em: <http://www.hardware.com.br/termos/thread>. Acesso em: 25 Março 2015.

NETO, O. M. **Entendendo e Dominando o Java**. 3ª. ed. São Paulo: Digerati Books, v. I, 2009.

NEWARD, T. **5 things you didn't know about. Java performance monitoring, Part 1**., 2010. Disponivel em: <http://www.ibm.com/developerworks/java/library/j-5things7/j-5things7-pdf.pdf>. Acesso em: 13 Abril 2015.

OLIVEIRA, B. H. D. Agilizando a Criação de Telas em JavaFX com JavaFX Scene Builder. **UniVale**, 2015. Disponivel em: <http://www.univale.com.br/unisite/mundo-j/artigos/61_Javafx.pdf>. Acesso em: 08 Abril 2015.

ORACLE. Glossário de Conceitos e Definições Úteis. **Java**, 2014. Disponivel em: <https://www.java.com/pt_BR/download/faq/helpful_concepts.xml>. Acesso em: 11 Dezembro 2014.

ORACLE. O que é a Tecnologia Java e porque preciso dela?. **Java**, 2014. Disponivel em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 10 Dezembro 2014.

ORACLE. Obtenha Informações sobre a Tecnologia Java. **Oracle**, 2014. Disponivel em: <http://www.java.com/pt_BR/about/>. Acesso em: 10 Dezembro 2014.

ORACLE. Java Management Extensions (JMX). **Oracle**, 2015. Disponivel em: <http://docs.oracle.com/javase/6/docs/technotes/guides/jmx/>. Acesso em: 13 Abril 2015.

ORACLE. Overview of Java SE Monitoring and Management. **Oracle**, 2015. Disponivel em: <http://docs.oracle.com/javase/6/docs/technotes/guides/management/overview.html>. Acesso em: 13 Abril 2015.

PALHETA, M. **SlideShare**., 2013. Disponivel em: <http://pt.slideshare.net/marciopalheta/trabalhando-com-threads-em-java>. Acesso em: 04 Abril 2015.

PIMENTEL, M.; FUKS, H. **Sistemas Colaborativos**. 1ª. ed. Rio de Janeiro: Elsevier Editora Ltda, v. I, 2012.

RIBEIRO, D. M. et al. **RSwing: uma biblioteca de componentes de código livre para geração de gráficos estatísticos**., 2015. Disponivel em: <http://www.cin.ufpe.br/~mmr3/publications/SUCESU05_medeirosribeiro.pdf>. Acesso em: 22 Junho 2015.

SCHMITZ, D. Conheça JavaFX 2 e o seu potencial. **iMasters**, 2012. Disponivel em: <http://imasters.com.br/linguagens/java/conheca-javafx-2-e-o-seu-potencial/>. Acesso em: 13 Dezembro 2014.

SERSON, R. R. **Programação Orientada a Objetos com Java 6**. 1ª. ed. Rio de Janeiro: BRASPORT Livros e Multimídia Ltda, v. I, 2007.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Sistemas Operacionais com Java**. 7ª. ed. Rio de Janeiro: Elsevier Editora Ltda, v. I, 2008.

ZANARDO, E. JavaFX – O Java para desktop. **Dextra**, 2013. Disponivel em: <http://www.dextra.com.br/javafx-o-java-para-desktop/>. Acesso em: 09 Março 2015.