



**Faculdade de Tecnologia de Americana**
**Curso Superior de Análise de Sistemas e Tecnologia da Informação**

# DESENVOLVIMENTO DE UMA FERRAMENTA PARA SIMULAÇÃO DE SISTEMAS DE FILAS

**FAISTER CABRERA CARVALHO**

**Americana, SP**
**2011**



**Faculdade de Tecnologia de Americana**
**Curso Superior de Análise de Sistemas e Tecnologia da Informação**

# DESENVOLVIMENTO DE UMA FERRAMENTA PARA SIMULAÇÃO DE SISTEMAS DE FILAS

**FAISTER CABRERA CARVALHO**

**faister.carvalho@gmail.com**

**Trabalho de conclusão de curso desenvolvido em cumprimento à exigência curricular do Curso de Análise de Sistemas e Tecnologia da Informação, sob orientação da Profª. Dra. Mariana Godoy Vazquez Miano.**

**Área: Simulação de Sistemas**

**Americana, SP**
**2011**

**BANCA EXAMINADORA**

**Prof. Dra. Mariana Godoy Vazquez Miano (Orientadora)**

**Prof. Dra. Thais Godoy Vazquez (Co-orientadora)**

**Prof. Clerivaldo José Roccia**

**AGRADECIMENTOS**

Sinto-me grato a todas as pessoas que auxiliaram na realização deste trabalho, direta ou indiretamente.

À minha orientadora, por todo seu apoio durante o curso e pelas oportunidades a mim oferecidas. Sem a sua ajuda este trabalho não teria sido possível.

Aos meus colegas de sala, que tornaram a convivência diária em ambiente escolar muito agradável e ajudaram a manter o ânimo e motivação.

Em especial à minha colega Mayara, que me acompanhou durante todo o curso, e sem o apoio da qual não teria concluído o presente trabalho da forma que é apresentado.

Aos meus pais, pelo apoio ao estudo desde minha criação, o que possibilitou que eu conclua este trabalho.

## RESUMO

Sistemas de filas são bastante comuns no âmbito comercial, e têm impacto financeiro direto nas empresas. O presente trabalho descreve o desenvolvimento de uma ferramenta completa e versátil para simulação de sistemas de filas em linguagem de programação Java, detalhando cada passo no desenvolvimento e sua base teórica.

**Palavras Chave:** Teoria de filas, Simulação, Cálculo numérico

## ABSTRACT

Queuing systems are quite common in commerce, having direct financial impact on companies. This paper describes the development of a complete and versatile tool for queuing systems simulation in the Java programming language, detailing each step of the development and its theoretical basis.

**Keywords:** Queuing theory, Simulation, Numerical calculation

**SUMÁRIO**

## LISTA DE FIGURAS E DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

**GIF:** *Graphics Interchange Format* (Formato de Intercâmbio de Gráficos)

**IDE:** *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)

**JPG:** *Joint Photographic Experts Group*

**PNG:** *Portable Network Graphics* (Gráficos de Rede Portáteis)

**RAM:** *Random Access Memory* (Memória de Acesso Aleatório)

**TIFF:** *Tagged Image File Format* (Formato de Arquivo de Imagem Marcado)

**UI:** *User Interface* (Interface Gráfica)

# 1 INTRODUÇÃO

## 1.1 TRABALHOS ANTERIORES

A modelagem e simulação de sistemas é um recurso bastante versátil e útil no meio empresarial para auxiliar na tomada de decisão, e graças à alta capacidade de processamento dos computadores atuais se tornou facilmente aplicável.

Existem vários modelos de sistemas, aplicáveis a diferentes casos reais com diferentes finalidades. Um desses modelos é o de filas, usado para sistemas com servidores destinados a atender filas de clientes.

Durante o curso de Análise de Sistemas e Tecnologia da Informação foi ministrada a matéria Matemática para Simulação de Sistemas, na qual o conceito de teoria de filas foi abordado. A partir dos estudos realizados nesta matéria, o autor do presente trabalho, em conjunto com a colega Mayara Bruno da Silva, aprofundou os estudos visando o desenvolvimento de uma ferramenta complexa.

Como resultado dos estudos, uma primeira ferramenta foi desenvolvida como protótipo, e apresentada com um artigo, sob orientação da professora Dra. Mariana Godoy Vazquez Miano, no evento ERAD-SP (Escola Regional de Alto Desempenho de São Paulo), realizado entre os dias 27 e 29 de julho de 2011, e no XLIII SBPO (Simpósio Brasileiro de Pesquisa Operacional), realizado entre os dias 15 e 18 de agosto de 2011.

Após a apresentação nos eventos, os dois alunos responsáveis pelo trabalho decidiram refazer a ferramenta de forma mais organizada e funcional, visando o desenvolvimento dos trabalhos de conclusão de curso baseados na experiência obtida.

Ficou acertado que ambos os trabalhos seriam complementares, cada um abordando uma parte do conteúdo. O trabalho de conclusão de curso da aluna Mayara Bruno da Silva, portanto, é complementar a este, e aborda os conceitos matemáticos da teoria de filas e suas aplicações em simulação de sistemas.

O trabalho atual visa detalhar o desenvolvimento da ferramenta, aplicando os conceitos abordados no trabalho complementar de forma prática.

## 1.2 TEORIA DE FILAS

Segundo a teoria de filas, um sistema de filas é um sistema onde entradas discretas aleatórias são atendidas por um ou mais servidores. Em geral as entradas no sistema e o tempo de processamento dos servidores é aleatório, havendo assim a possibilidade de formação de uma fila de espera. Note-se que o processo aleatório que caracteriza a capacidade de atendimento é um processo de Poisson com taxa μ, e, consequentemente, o tempo médio de atendimento do servidor é igual a 1 / μ.

David George Kendall[1] criou um código para definir modelos de filas através de caracteres separados por barras, conhecido como notação de Kendall (KENDALL, 1953). Os caracteres desta notação são escolhidos de acordo com as características de elementos básicos do sistema no formato X/Y/J/K, onde X representa tipo de entrada, Y representa o tipo de serviço, J define o número de servidores e K é o número máximo de usuários no sistema. Os parâmetros X e Y podem ter os valores M, que significa que usam distribuição Poisson/Exponencial, D, quando são determinísticos, ou G, que representa outros.

Se o número máximo de usuários no sistema é infinito omite-se o último caractere. Neste caso o sistema é dito sem bloqueio ou sem perdas, ou seja, todos os usuários que chegam entram no sistema. Por outro lado, um número máximo de usuários finito significa que quando o sistema atingir este número, todo usuário que chega é rejeitado até que o número de clientes no sistema diminua.

Define-se intensidade de tráfego como a razão $\rho = \lambda / \mu$ (FOGLIATTI e MATTOS, 2006), onde λ é a taxa de entrada (número médio de chegadas por unidade de tempo) e 1 / μ o tempo médio de atendimento. A unidade para esta medida é denominada Erlang[2].

A vazão em um sistema de filas é a taxa média de saída do sistema. Note-se que a vazão é diferente da taxa de atendimento do servidor. A taxa de atendimento mede a capacidade de atendimento do servidor e é igual à taxa de saída, enquanto

[1] David George Kendall foi um estatístico britânico, nascido dia 15 de janeiro de 1918, especialista em probabilidade e análise de dados. Faleceu dia 23 de outubro de 2007.

houver usuários para serem atendidos. Se em um determinado período não existirem usuários no sistema, nenhum usuário sairá do mesmo, embora a taxa de atendimento (capacidade) continue a mesma.

Como exemplo, pode-se pensar em um conjunto de seis frentes de caixa de um supermercado. Assumindo-se que as taxas de entrada e saída de clientes seguem distribuições exponenciais e que não há limite para as filas de clientes, pode-se dizer que o sistema segue o padrão M/M/6.

## 1.3 SIMULAÇÃO

A simulação consiste em aplicar formalizações em computadores, como expressões matemáticas ou especificações formalizadas, a fim de imitar um processo ou operação do mundo real. Assim, para se realizar uma simulação, é necessário formular um modelo computacional que corresponda à situação real a ser simulada.

Entre as diversas definições de simulação, pode-se citar a que diz que “a simulação é um processo de projetar um modelo computacional de um sistema real e conduzir experimentos com este modelo com o propósito de entender seu comportamento e/ou avaliar estratégias para sua operação” (PEGDEN, SHANNON e SADOWSKI, 1990). Portanto, pode-se entender a simulação como um processo amplo que engloba, além da construção do modelo, todo o método experimental a seguir, objetivando descrever o comportamento do sistema, construir teorias e hipóteses considerando as observações efetuadas e usar o modelo para prever o comportamento futuro, ou seja, os efeitos de alterações no sistema ou nos métodos empregados.

## 1.4 A LINGUAGEM JAVA

A linguagem de programação Java foi criada em 1991 por James Gosling. Inicialmente sendo parte do projeto Green da Sun Microsystems, a linguagem seria chamada Oak (Carvalho) em referência à árvore que era visível pela janela de

[2] Agner Krarup Erlang foi um matemático dinamarquês, nascido dia 1 de Janeiro de 1878, pioneiro em diversos estudos relacionados à telefonia. Faleceu dia 3 de Fevereiro de 1929.

James Gosling. Como já existia uma linguagem de programação com este nome, a linguagem foi rebatizada para Java.

O termo Java é utilizado, geralmente, como referência à linguagem de programação orientada a objetos, ao ambiente de desenvolvimento composto pelo compilador, interpretador e gerador de documentação ou ao ambiente de execução que pode ser qualquer máquina que possua Java Runtime Environment (JRE) instalado.

A linguagem de programação Java é uma linguagem de alto-nível com as seguintes características:

- Simples: O aprendizado da linguagem de programação Java pode ser feito em um curto período de tempo;

- Orientada a objetos: Desde o início do seu desenvolvimento esta linguagem foi projetada para ser orientada a objetos;

- Familiar: A linguagem Java é muito familiar para os programadores habituados com C/C++;

- Robusta: Ela foi pensada para o desenvolvimento de softwares confiáveis, provendo verificações tanto em tempo de execução quanto compilação, o coletor de lixo responsabiliza-se pela limpeza da memória quando houver necessidade;

- Segura: Aplicações Java são executadas em ambiente próprio (JRE) o que inviabiliza a intrusão de código malicioso;

- Portável: Programas desenvolvidos nesta linguagem podem ser executados em praticamente qualquer máquina desde que esta possua o JRE instalado;

## 1.5 CÁLCULO NUMÉRICO

O cálculo numérico é um ramo da matemática que estuda métodos construtivos (na forma de algoritmos) que convergem para entidades matemáticas cuja existência foi demonstrada. Um método numérico aplica uma sucessão que converge para um valor exato. Cada termo dessa sucessão deve ser visto como uma aproximação possível de se calcular com um número finito de operações elementares. É objetivo do cálculo numérico encontrar sucessões que aproximem valores exatos com o mínimo de operações elementares. Em outras palavras, os métodos numéricos são um conjunto de ferramentas usadas para se obter a solução de problemas matemáticos de forma aproximada, sendo aplicados geralmente a problemas que não apresentam uma solução exata (BUFFONI).

Um dos escritos matemáticos mais antigos relacionados ao cálculo numérico é a tábua de argila babilônica YBC 7289 (MELVILLE), que fornece uma aproximação sexagesimal da raiz quadrada de 2, o comprimento da diagonal de um quadrado unitário.

Durante o trabalho é utilizado um método de integração numérica denominado a regra dos trapézios (DA ROCHA LOPES e A. GOMES RUGGIERO, 1996). Além disto, a própria funcionalidade principal da ferramenta acontece de forma numérica, devido à sua aleatoriedade e natureza aproximada.

## 1.6 ESCOPO DO TRABALHO

O presente trabalho detalha o desenvolvimento de uma ferramenta para simulação de sistemas de filas de fácil aplicação em linguagem de programação Java.

O ambiente de desenvolvimento integrado (também chamado de IDE, do inglês “Integrated Development Environment”) escolhido para o desenvolvimento foi o NetBeans, ambiente gratuito e de código aberto que auxilia na codificação em múltiplas linguagens de programação, dentre as quais o Java.

A simulação aplicará conceitos da teoria de filas com uma abordagem numérica, efetuando aproximações e criando situações aleatórias durante a simulação, o que possibilita a previsão de situações prejudiciais ao sistema e facilita o planejamento de sua estratégia.

A ferramenta permitirá que o usuário insira todos os parâmetros da simulação (especificados na teoria de filas), que alguns parâmetros sejam alterados em tempo real durante a execução da simulação, que o usuário interrompa e reinicie a simulação, apresentará visualmente as filas e servidores e apresentará os resultados da simulação em variáveis numéricas relevantes e gráficos.

O desenvolvimento foi dividido em etapas para facilitar o entendimento, e tais etapas são tratadas nos capítulos posteriores.

# 2 DESENVOLVENDO A INTERFACE COM O USUÁRIO

A interface com o usuário (comumente chamada de UI, do inglês “User Interface”) engloba todos os aspectos visíveis da ferramenta para o usuário final, como telas, abas, campos e botões. Uma boa UI consegue tornar as funcionalidades do sistema satisfatoriamente acessíveis ao usuário final, facilitando o manuseio da ferramenta.

A UI criada para a ferramenta será feita em apenas uma janela, e os diferentes parâmetros da simulação serão separados em três diferentes abas para facilitar a compreensão.

Para iniciar a construção da interface com o usuário cria-se um novo projeto de área de trabalho no NetBeans. De forma padrão, é apresentada a tela principal do programa, inicialmente com apenas uma barra de menu acima e uma barra de status abaixo. Para deixar a interface mais limpa serão removidos estes dois componentes.

Com a tela limpa, inicia-se o desenho da ferramenta. Primeiramente, é criado um painel tabulado, que conterá as abas da ferramenta. Este painel deve estar cobrindo praticamente toda a tela, deixando apenas um pequeno espaço em cada lado por questões estéticas, como mostrado na Figura 1.

Figura 1: Tela do sistema com painel tabulado

## 2.1 ABA DE ENTRADA DE CLIENTES NO SISTEMA

Uma simulação de sistema de filas precisa de parâmetros para calcular a entrada de clientes no sistema. Tais parâmetros são usados para determinar quantos clientes chegam a cada intervalo.

Será permitido que o usuário insira dados obtidos através de medições em um sistema real na ferramenta, e tais dados, por sua vez, poderão ser inseridos de três formas diferentes: medição cronometrada; tabela de frequência; e valor fixo.

O objeto que guardará as abas é, então, criado, assim como um painel para a primeira aba do sistema, chamada “Entrada”. É possível nomeá-la selecionando-a e pressionando F2. Neste novo painel, criamos uma caixa de combinação na qual o usuário escolherá a forma na qual os dados da entrada do sistema serão inseridos, contendo três opções: “Medição cronometrada”, “Tabela de frequência” e “Valor fixo”. Neste ponto a tela inicial deve estar como na Figura 2.

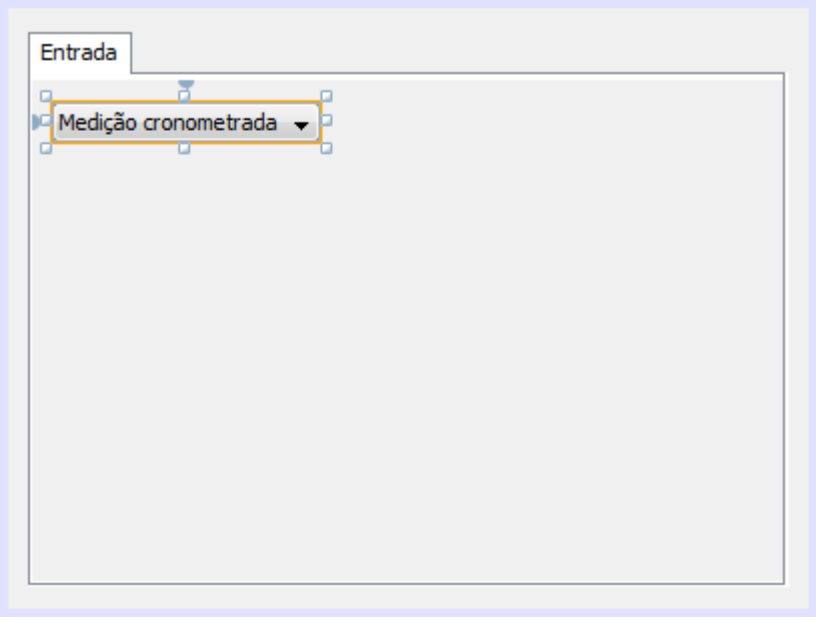


Figura 2: Estágio inicial da aba de entrada de clientes no sistema

### 2.1.1 MEDIÇÃO CRONOMETRADA

Uma das formas de inserção de dados de entrada de clientes na ferramenta será a “medição cronometrada”. Nesta forma, o cliente inserirá os dados obtidos nas

medições de forma detalhada, e a ferramenta usará as medições como base para criar uma curva aproximada que as abranja.

Para a interface com o usuário será criado um painel, e dentro deste uma tabela e outro painel. Na tabela o usuário irá inserir os dados da medição, e no painel um gráfico será gerado com a curva da entrada de clientes no sistema pelo tempo. As informações necessárias sobre as medições são o horário de início da medição, sua duração e o número de clientes que entraram no sistema.

Nesta forma, assim como na forma de “tabela de frequência”, o usuário deverá escolher como os dados inseridos serão interpretados durante a simulação. Criaremos, então, uma caixa de combinação para controlar a distribuição utilizada, contendo duas opções: Observada e Poisson. Esta caixa de combinação ficará fora do painel criado para a “medição cronometrada”, já que será usada também para a “tabela de frequência”. A tela deverá estar, então, como na Figura 3.

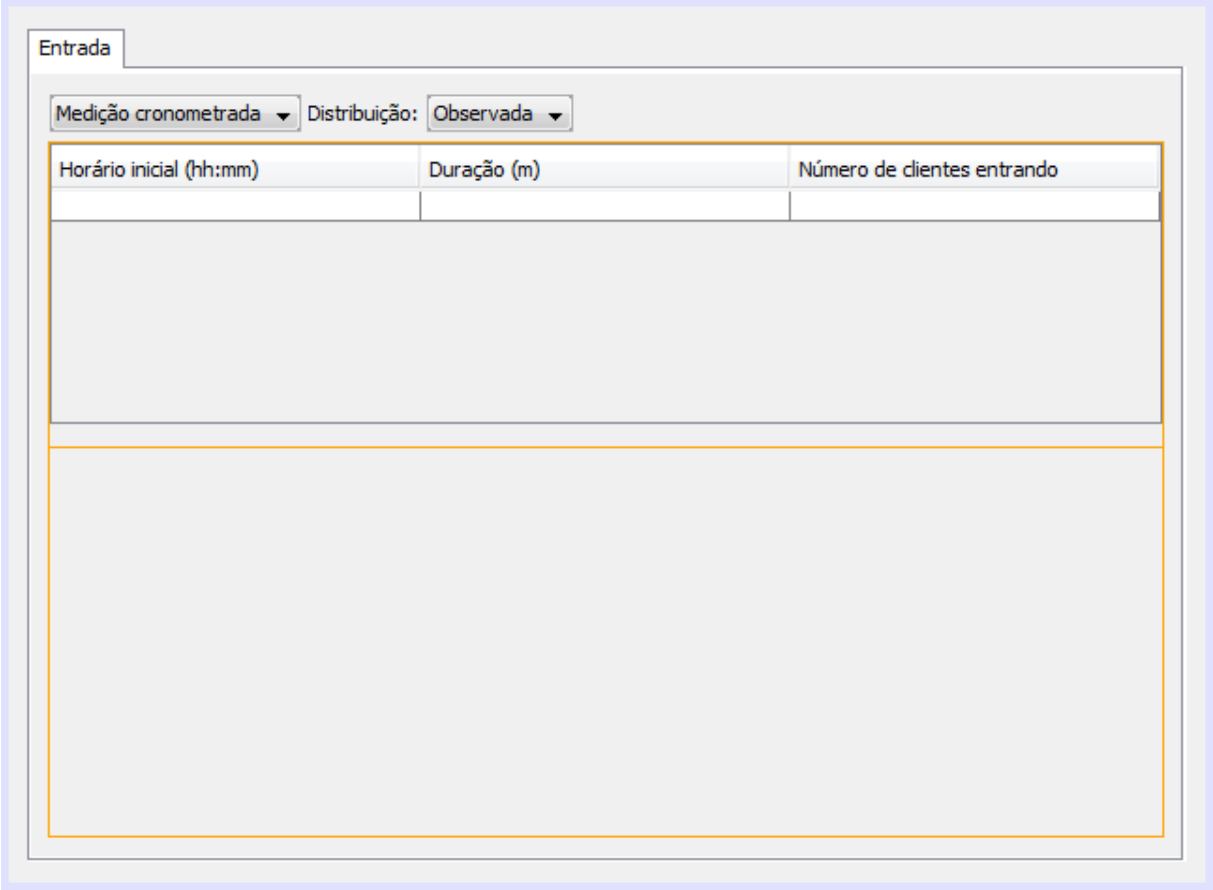


Figura 3: Aparência da forma de entrada “medição cronometrada”

O código gerado pelo NetBeans ao montar a tela deve ser suficiente para que a tela seja desenhada como na Figura 3, porém isso não será o bastante para que a interface com o usuário seja funcional.

A tabela criada será usada pelo usuário para inserir dados na forma de medição cronometrada, e terá o seguinte comportamento dinâmico: assim que a última linha da tabela estiver completa, uma nova linha é criada para permitir que o usuário insira mais registros; se algum valor de uma linha anterior à última for apagado, a linha em questão é removida da tabela; a formatação das células deve ser validada de acordo com seus padrões; as linhas devem ser posicionadas automaticamente para que fiquem ordenadas pelo horário inicial.

Para a validação da tabela será criado um método no modo de visualização de código. Este método retornará “true” caso os dados na tabela sejam válidos ou “false” caso haja algum dado inválido. Segue o código de tal método:

```
private boolean validaTabelaMedicaoCronometrada() {
    String horarioInicial =
(String)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 1, 0);
    Double duracao =
(Double)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 1, 1);
    Long numClientes =
(Long)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 1, 2);
    if ((horarioInicial != null) &&
        (duracao != null) &&
        (numClientes != null)) {
        ((DefaultTableModel)tblMedicaoCronometrada.getModel()).addRow(new Object[]{null, null,
null});
    }
    for (int linha = 0; linha < tblMedicaoCronometrada.getRowCount() - 1; linha++) {
        horarioInicial = (String)tblMedicaoCronometrada.getValueAt(linha, 0);
        duracao = (Double)tblMedicaoCronometrada.getValueAt(linha, 1);
        numClientes = (Long)tblMedicaoCronometrada.getValueAt(linha, 2);
        if ((horarioInicial == null) ||
            (duracao == null) ||
            (numClientes == null)) {
            ((DefaultTableModel)tblMedicaoCronometrada.getModel()).removeRow(linha);
            linha--;
        } else {
            if ((!tblMedicaoCronometrada.isEditing()) ||
                (!tblMedicaoCronometrada.isColumnSelected(0)) ||
                (!tblMedicaoCronometrada.isRowSelected(linha))) {
                try {
                    String horarioInicialAux[] = horarioInicial.split(":");
                    Long.parseLong(horarioInicialAux[0]);
                    Double.parseDouble(horarioInicialAux[1].replace(",", "."));
                } catch (Exception e) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

Este método será utilizado na codificação do comportamento necessário para a ordenação da tabela, implantado usando o evento “PropertyChange” da mesma. Para isso clica-se com o botão direito do mouse na tabela e com o botão esquerdo do mouse em “Eventos”, “PropertyChange” e por último “propertyChange”. O NetBeans deve se encarregar de criar um método e associá-lo ao evento, apresentando o editor para que seja colocado seu conteúdo.

```
private void tblMedicaoCronometradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaMedicaoCronometrada()) {
        for (int linha1 = 0; linha1 < tblMedicaoCronometrada.getRowCount() - 1; linha1++) {
            for (int linha2 = linha1 + 1; linha2 < tblMedicaoCronometrada.getRowCount() - 1;
linha2++) {
                String aux[] = ((String)tblMedicaoCronometrada.getValueAt(linha1,
0)).split(":");
                double valor1 = Long.parseLong(aux[0]) * 60 +
Double.parseDouble(aux[1].replace(",", "."));
                aux = ((String)tblMedicaoCronometrada.getValueAt(linha2, 0)).split(":");
                double valor2 = Long.parseLong(aux[0]) * 60 +
Double.parseDouble(aux[1].replace(",", "."));
                if (valor2 < valor1) {
                    ((DefaultTableModel)tblMedicaoCronometrada.getModel()).moveRow(linha1,
linha1, linha2);
                }
            }
        }
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                 + "O formato correto para o horário inicial é
hh:mm. Ex: 10:00", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}
```

O código acima abrange todos os requisitos especificados, portanto a tabela estará funcional como previsto. Note que a tabela em questão foi nomeada “tblMedicaoCronometrada”.

Além dos ajustes na tabela, deverá ser ajustado o painel para que se comporte como um container para o gráfico gerado. Para isso será alterado o layout do painel para “Layout da borda”, clicando com o botão direito no mesmo, escolhendo em seguida “Definir layout” e por fim “Layout da borda”.

### 2.1.2 TABELA DE FREQUÊNCIA

Além da forma de entrada denominada medição cronometrada, será permitido que o usuário insira os dados na forma de tabela de frequência. Para se criar a interface necessária, será copiado o painel que contém os componentes da medição cronometrada e colado no painel da entrada. Para facilitar a tarefa, essa ação será realizada no Inspetor do NetBeans. Depois de feita a cópia, será necessário ajustar

os limites superior e inferior do novo painel para que o mesmo fique na mesma posição que o painel anterior, e também apagar os dois componentes internos, que também foram copiados (a tabela e o painel da medição cronometrada).

A seguir, basta clicar com o botão direito do mouse no painel recém-criado (usando o Inspetor do NetBeans, novamente) e usar a opção “Desenhar este recipiente”, para facilitar sua edição.

Será adicionado um componente de tabela, que será a tabela de frequência. Esta tabela terá as colunas Número de clientes e Frequência, como observado na Figura 4.

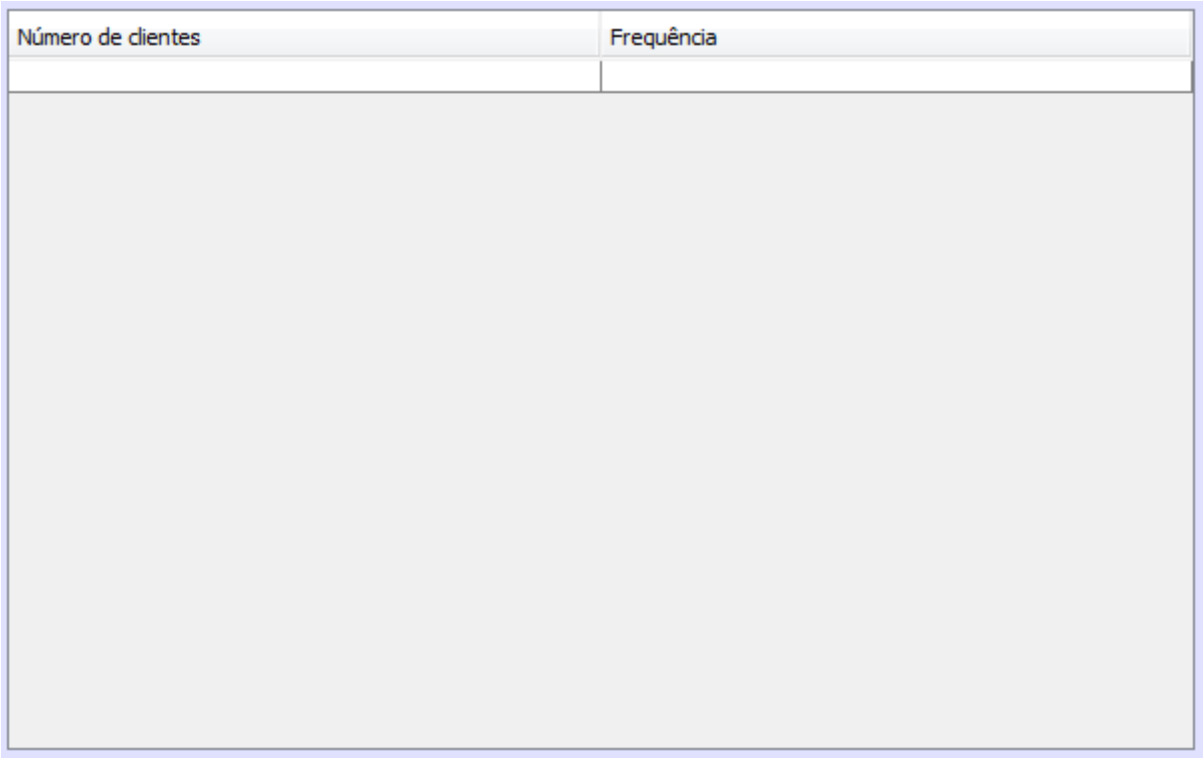


Figura 4: Aparência da forma de entrada “tabela de frequência”

Esta tabela também deverá ter a mesma funcionalidade da tabela de frequência de medição cronometrada, validando os números inseridos para que não sejam negativos.

Similarmente ao que foi feito para a medição cronometrada, será criado um método responsável por validar os dados inseridos na tabela. Este método será chamado “validaTabelaFrequenciaEntrada”, e seu código está descrito a seguir.

```
private boolean validaTabelaFrequenciaEntrada() {
    Long numClientes =
(Long)tblTabelaFrequenciaEntrada.getValueAt(tblTabelaFrequenciaEntrada.getRowCount() - 1, 0);
    Long frequencia =
(Long)tblTabelaFrequenciaEntrada.getValueAt(tblTabelaFrequenciaEntrada.getRowCount() - 1, 1);
    if ((numClientes != null) &&
        (frequencia != null)) {
        ((DefaultTableModel)tblTabelaFrequenciaEntrada.getModel()).addRow(new Object[]{null,
null, null});
    }
    for (int linha = 0; linha < tblTabelaFrequenciaEntrada.getRowCount() - 1; linha++) {
        numClientes = (Long)tblTabelaFrequenciaEntrada.getValueAt(linha, 0);
        frequencia = (Long)tblTabelaFrequenciaEntrada.getValueAt(linha, 1);
        if ((numClientes == null) ||
            (frequencia == null)) {
            ((DefaultTableModel)tblTabelaFrequenciaEntrada.getModel()).removeRow(linha);
            linha--;
        } else {
            if ((!tblTabelaFrequenciaEntrada.isEditing()) ||
                (!tblTabelaFrequenciaEntrada.isColumnSelected(0)) ||
                (!tblTabelaFrequenciaEntrada.isRowSelected(linha))) {
                if ((numClientes <= 0) ||
                    (frequencia <= 0)) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

Para que o código do evento seja inserido, basta clicar com o botão direito do mouse na tabela e com o botão esquerdo do mouse em “Eventos”, “PropertyChange” e por fim “propertyChange”, como já foi feito anteriormente.

```
private void tblTabelaFrequenciaEntradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaFrequenciaEntrada()) {
        for (int linha1 = 0; linha1 < tblTabelaFrequenciaEntrada.getRowCount() - 1; linha1++)
{
            for (int linha2 = linha1 + 1; linha2 < tblTabelaFrequenciaEntrada.getRowCount() -
1; linha2++) {
                long valor1 = (Long) tblTabelaFrequenciaEntrada.getValueAt(linha1, 0);
                long valor2 = (Long) tblTabelaFrequenciaEntrada.getValueAt(linha2, 0);
                if (valor2 < valor1) {
                    ((DefaultTableModel)tblTabelaFrequenciaEntrada.getModel()).moveRow(linha1,
linha1, linha2);
                }
            }
        }
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "Os valores inseridos devem ser maiores que
zero.", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}
```

O objeto da tabela de frequência foi nomeado “tblTabelaFrequenciaEntrada”, e o código acima deve satisfazer as necessidades funcionais de sua interface com o usuário.

Para que a tabela seja interpretada corretamente, também será preciso adicionar um campo numérico no painel de entrada para que o usuário possa informar qual a duração dos intervalos mensurados.

A seguir, clica-se com o botão direito do mouse no painel de entrada, no Inspetor do NetBeans, e então é escolhido “Desenhar este recipiente” para que o mesmo seja editado.

À frente da caixa de combinação de escolha da distribuição é adicionado um campo para inserção da duração dos intervalos. Após feito isto a tela deve estar como na Figura 5.

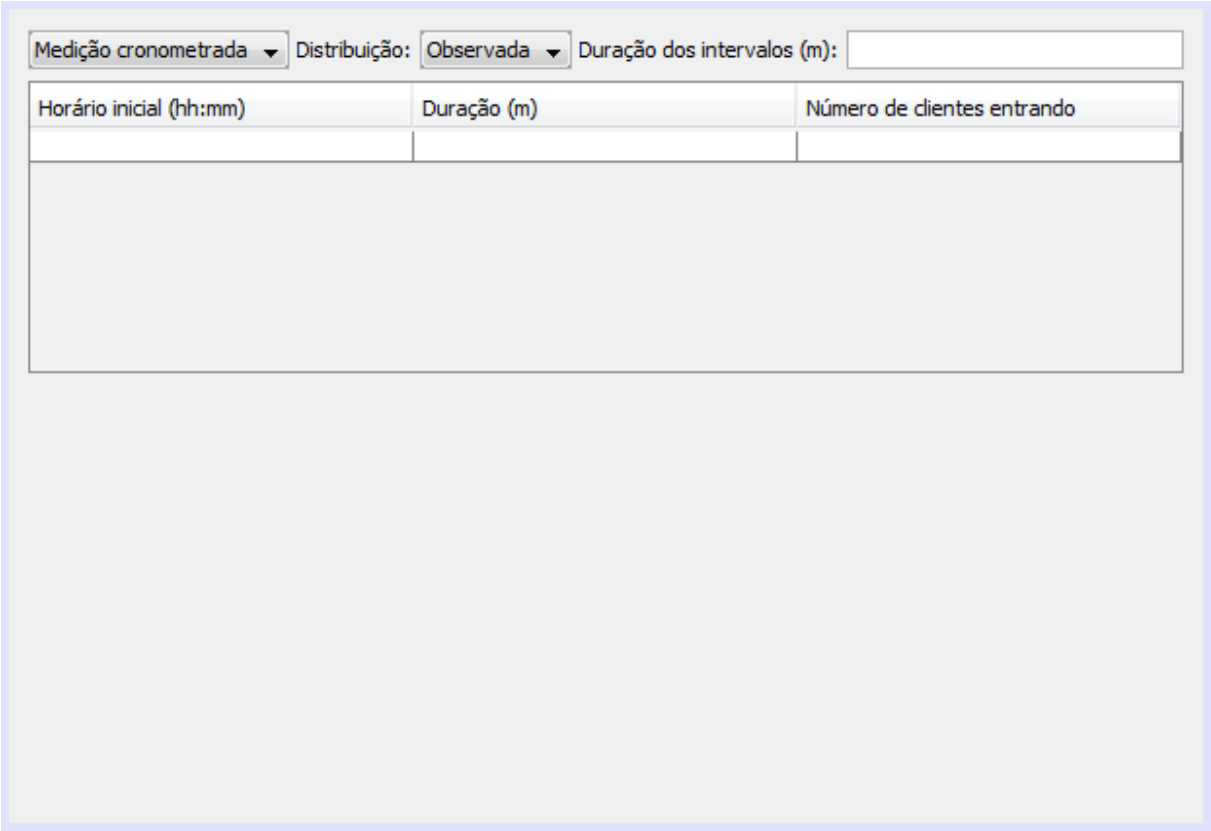


Figura 5: Aba de entrada com campo de duração dos intervalos

Usando um objeto da classe “JFormattedTextField”, o NetBeans fica encarregado da validação numérica do campo, restando apenas a validação lógica de que a duração do intervalo deve ser um número positivo de minutos, já que todo intervalo, por definição, deve ter uma duração maior que zero.

Para fazer tal validação usaremos o evento “PropertyChange”, assim como temos usado nas tabelas até agora. Para codificarmos a validação, clicamos com o botão direito no campo de duração do intervalo e escolhemos “Eventos”, “PropertyChange” e finalmente “propertyChange”.

```
private void nmrDuracaoIntervaloEntradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (nmrDuracaoIntervaloEntrada.getValue() != null) {
        double duracao = Double.parseDouble(nmrDuracaoIntervaloEntrada.getValue().toString());
        if (duracao <= 0.0) {
            JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                    + "A duração do intervalo deve ser um
número positivo", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
            nmrDuracaoIntervaloEntrada.setValue(null);
        }
    }
}
```

O campo foi nomeado “nmrDuracaoIntervaloEntrada”.

### 2.1.3 VALOR FIXO

Nesta forma de entrada, o usuário simplesmente especifica um valor para a taxa de entrada de clientes no sistema (comumente chamada de “Lambda”, ou “λ”, na teoria de filas) a ser usada durante toda a simulação.

O painel da tabela de frequência é copiado, da mesma forma que o painel da medição cronometrada, e colado no painel de entrada, ajustando os limites superior e inferior e excluindo os componentes internos da cópia. Por fim, clica-se com o botão direito do mouse no painel e escolhe-se “Desenhar este recipiente” para editá-lo.

Um campo para o valor de lambda será adicionado utilizando a classe “JFormattedTextField” para que a validação numérica seja feita automaticamente. A aparência do painel estará como na Figura 6.

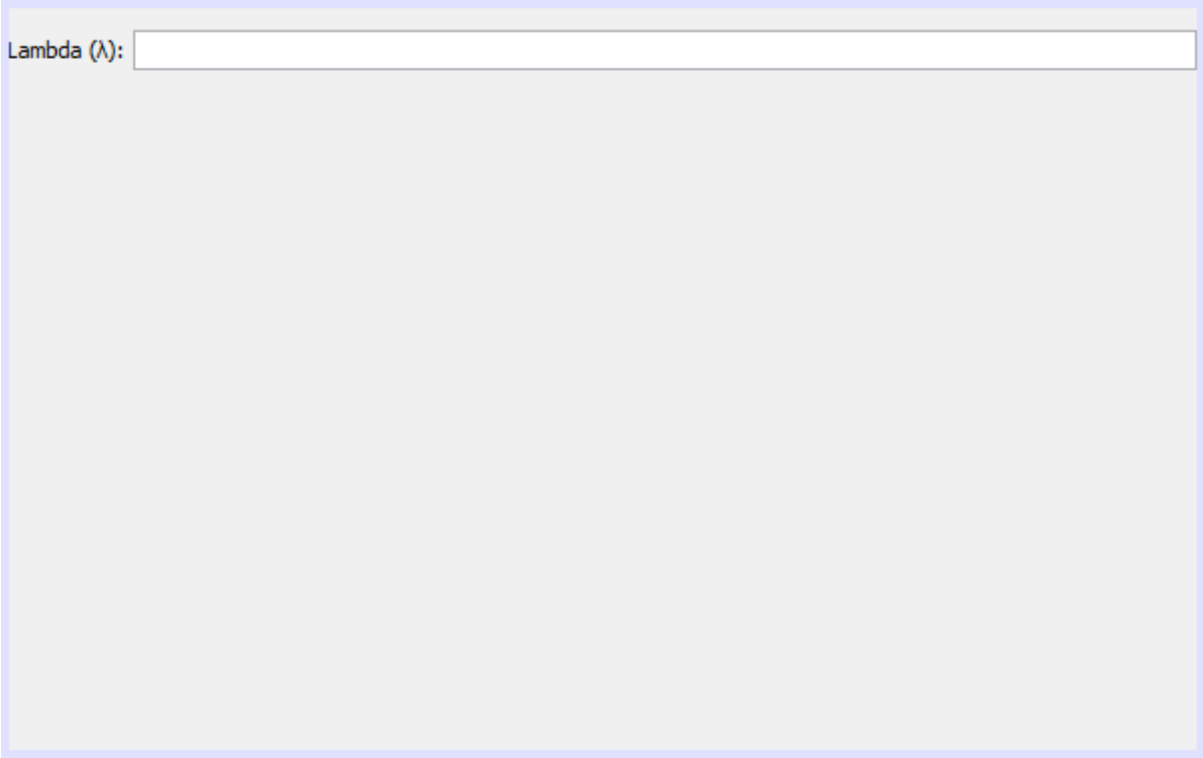


Figura 6: Aparência da forma de entrada “valor fixo”

Valida-se, então, se o número inserido é positivo, já que não é permitido inserir valores negativos para a taxa de entrada de clientes, utilizando o evento “PropertyChange” novamente.

```
private void nmrValorFixoEntradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (nmrValorFixoEntrada.getValue() != null) {
        double lambda = Double.parseDouble(nmrValorFixoEntrada.getValue().toString());
        if (lambda <= 0.0) {
            JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                    + "O valor da taxa de entrada deve ser um
número positivo", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
            nmrValorFixoEntrada.setValue(null);
        }
    }
}
```

O campo foi nomeado “nmrValorFixoEntrada”.

### 2.1.4 AJUSTES FINAIS

Agora que foram criados painéis para as três formas de entrada, é preciso acertar o comportamento da caixa de combinação responsável por alternar entre eles. Esta caixa de combinação irá exibir os componentes que correspondem à forma de entrada escolhida, e esconder os demais.

Para tal, será usado o evento “ActionPerformed” da caixa de combinação. Primeiro será preciso voltar a editar o painel de entrada, clicando com o botão direito no mesmo e escolhendo “Desenhar este recipiente”, e em seguida deve-se clicar com o botão direito na caixa de combinação, escolhendo “Eventos”, “Action” e “actionPerformed”.

```
private void cmbEntradaActionPerformed(java.awt.event.ActionEvent evt) {
    final int MEDICAO_CRONOMETRADA = 0;
    final int TABELA_DE_FREQUENCIA = 1;
    final int VALOR_FIXO = 2;

    switch (cmbEntrada.getSelectedIndex()) {
        case MEDICAO_CRONOMETRADA:
            pnlMedicaoCronometrada.setVisible(true);

            pnlTabelaFrequenciaEntrada.setVisible(false);
            pnlValorFixoEntrada.setVisible(false);
            txtDuracaoIntervaloEntrada.setVisible(false);
            nmrDuracaoIntervaloEntrada.setVisible(false);
            break;
        case TABELA_DE_FREQUENCIA:
            pnlTabelaFrequenciaEntrada.setVisible(true);
            txtDuracaoIntervaloEntrada.setVisible(true);
            nmrDuracaoIntervaloEntrada.setVisible(true);

            pnlMedicaoCronometrada.setVisible(false);
            pnlValorFixoEntrada.setVisible(false);
            break;
        case VALOR_FIXO:
            pnlValorFixoEntrada.setVisible(true);

            pnlMedicaoCronometrada.setVisible(false);
            pnlTabelaFrequenciaEntrada.setVisible(false);
            txtDuracaoIntervaloEntrada.setVisible(false);
            nmrDuracaoIntervaloEntrada.setVisible(false);
            break;
    }
}
```

Note-se que os painéis foram nomeados “pnlMedicaoCronometrada”, “pnlTabelaFrequencia” e “pnlValorFixoEntrada”. O campo numérico de duração do intervalo foi nomeado “nmrDuracaoIntervaloEntrada”, e o label correspondente “txtDuracaoIntervaloEntrada”.

Além de ajustar o evento da caixa de combinação, será preciso assegurar que, inicialmente, os componentes relacionados à medição cronometrada sejam os únicos visíveis, já que esta será a opção padrão da ferramenta.

Para fazer isso é preciso alterar o construtor da tela, e simplesmente chamar o evento “ActionPerformed” que acabamos de criar, que fará tudo que é necessário.

Deve-se, então, alternar à visão de código, e econtrar o construtor. A seguir, será adicionado o comando necessário após a chamada de “initComponents” (um método criado pelo NetBeans para inicializar os componentes).

```
public SimulacaoView(SingleFrameApplication app) {
    super(app);

    initComponents();
    cmbEntradaActionPerformed(null);
}
```

## 2.2 ABA DE SAÍDA DE CLIENTES NO SISTEMA

Com a aba de entrada desenhada e funcional será feita a interface com o usuário para a aba responsável pela taxa de saída de clientes do sistema. Nesta aba o usuário irá inserir dados referentes ao tempo de atendimento do sistema simulado, o que implica na taxa de saída de clientes do sistema quando há demanda.

Para a taxa de saída de clientes do sistema não faria sentido se a taxa variasse com o tempo, afinal pré-supõe-se que o atendimento realizado é o mesmo independente do horário. Portanto, não haverá a “medição cronometrada”.

Deve-se, então, copiar o painel de entrada e colá-lo dentro do painel tabulado. O Inspetor, novamente, facilitará esta tarefa. Isto criará outra aba. Para voltar a editar o painel principal da ferramenta, será necessário clicar no mesmo com o botão direito e escolher “Desenhar este recipiente”. O texto da nova aba deverá ser alterado para “Saída”, clicando na mesma e pressionando F2. O painel de medição cronometrada deste novo painel de saída deverá ser removido, bem como a opção correspondente na caixa de combinação que controlará a forma de inserção de dados, ficando a aparência da tela como na Figura 7.

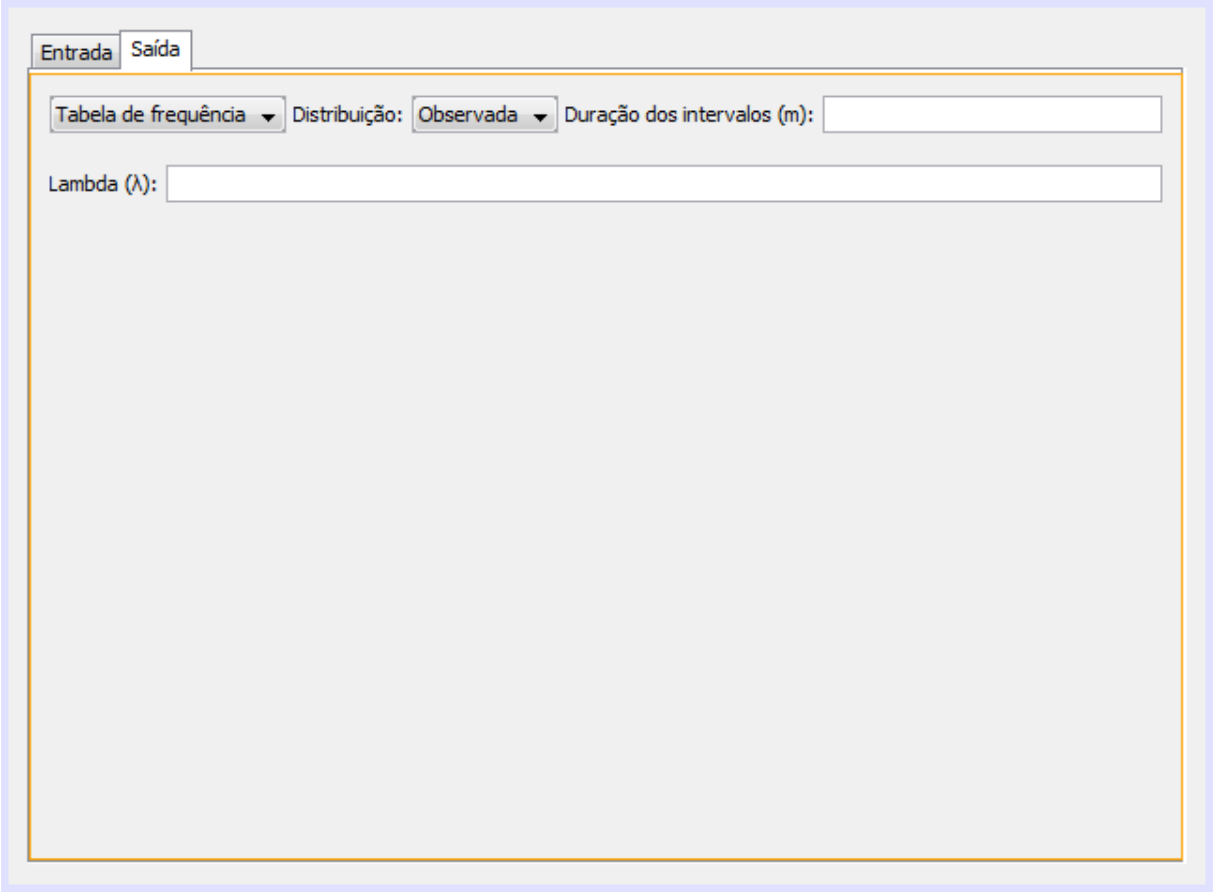


Figura 7: Estágio inicial da aba de saída de clientes do sistema

### 2.2.1 TABELA DE FREQUÊNCIA

A primeira forma de inserção de dados relacionados à taxa de saída de clientes do sistema será a “tabela de frequência”. Será bastante similar à da taxa de entrada, porém agora as amostras de tempo de atendimento observadas serão agrupadas em intervalos. Será preciso oferecer ao usuário uma forma de informar a duração dos intervalos e, para cada intervalo, o início do mesmo e sua frequência, deixando a tela como podemos ver na Figura 8.

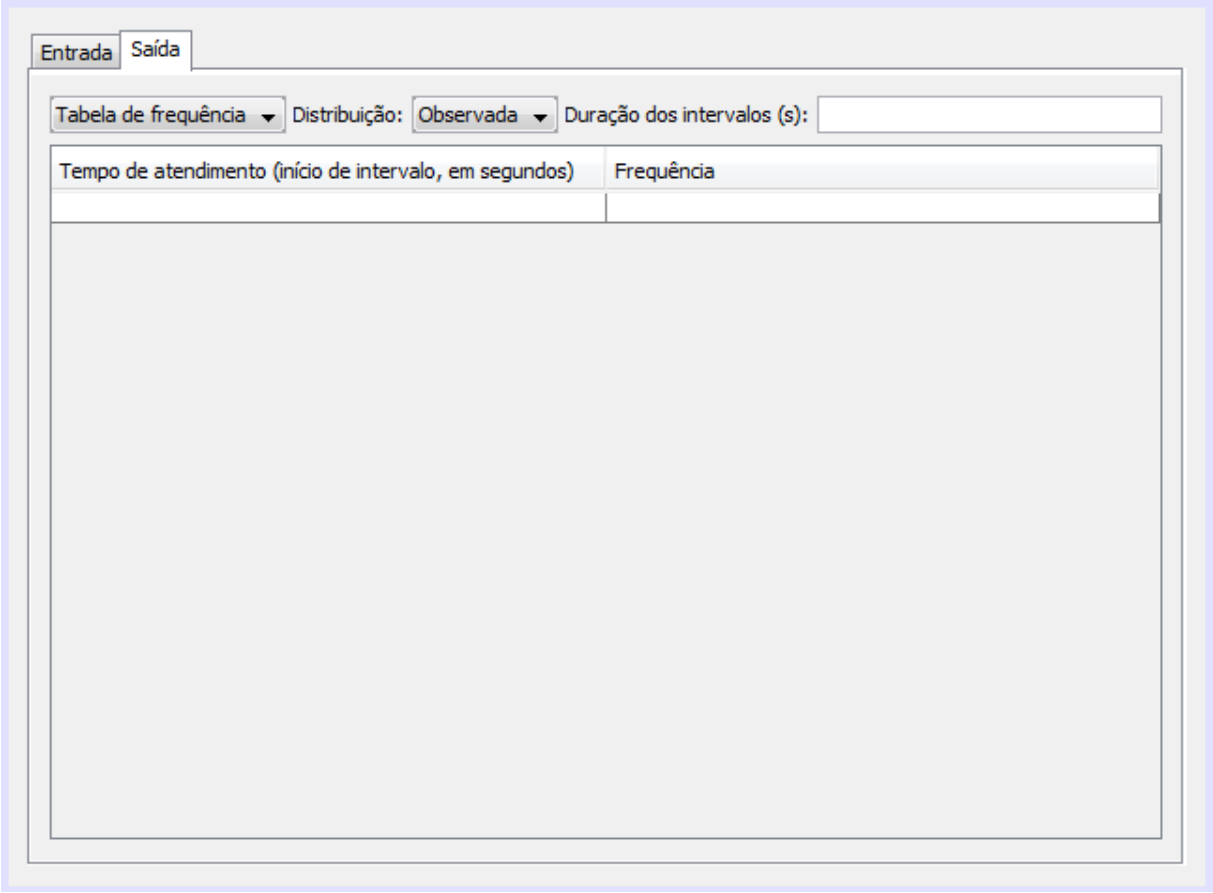


Figura 8: Aparência da forma de saída de clientes “tabela de frequência”

O campo de duração dos intervalos será um objeto da classe “JFormattedTextField”, para que a validação numérica seja feita automaticamente, restando a validação de que seu valor não possa ser menor ou igual a zero. Para esta validação deverá ser utilizado novamente o evento “PropertyChange”.

```
private void nmrDuracaoIntervaloSaidaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (nmrDuracaoIntervaloSaida.getValue() != null) {
        double duracao = Double.parseDouble(nmrDuracaoIntervaloSaida.getValue().toString());
        if (duracao <= 0.0) {
            JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                    + "A duração do intervalo deve ser um
número positivo", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
            nmrDuracaoIntervaloSaida.setValue(null);
        }
    }
}
```

Na tabela será preciso validar todos os números inseridos para que não sejam negativos, da mesma forma que foi feito para a tabela de frequência da taxa de entrada de clientes no sistema. De forma análoga, será criado um método responsável pela validação dos dados, detalhado a seguir.

```
private boolean validaTabelaFrequenciaSaida() {
    Double inicioIntervalo =
(Double)tblTabelaFrequenciaSaida.getValueAt(tblTabelaFrequenciaSaida.getRowCount() - 1, 0);
    Long frequencia =
(Long)tblTabelaFrequenciaSaida.getValueAt(tblTabelaFrequenciaSaida.getRowCount() - 1, 1);
    if ((inicioIntervalo != null) &&
        (frequencia != null)) {
        ((DefaultTableModel)tblTabelaFrequenciaSaida.getModel()).addRow(new Object[]{null,
null, null});
    }
    for (int linha = 0; linha < tblTabelaFrequenciaSaida.getRowCount() - 1; linha++) {
        inicioIntervalo = (Double)tblTabelaFrequenciaSaida.getValueAt(linha, 0);
        frequencia = (Long)tblTabelaFrequenciaSaida.getValueAt(linha, 1);
        if ((inicioIntervalo == null) ||
            (frequencia == null)) {
            ((DefaultTableModel)tblTabelaFrequenciaSaida.getModel()).removeRow(linha);
            linha--;
        } else {
            if ((!tblTabelaFrequenciaSaida.isEditing()) ||
                (!tblTabelaFrequenciaSaida.isColumnSelected(0)) ||
                (!tblTabelaFrequenciaSaida.isRowSelected(linha))) {
                if ((inicioIntervalo <= 0) ||
                    (frequencia <= 0)) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

Então será programado o evento “PropertyChange” da tabela com o código a seguir.

```
private void tblTabelaFrequenciaSaidaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaFrequenciaSaida()) {
        for (int linha1 = 0; linha1 < tblTabelaFrequenciaSaida.getRowCount() - 1; linha1++) {
            for (int linha2 = linha1 + 1; linha2 < tblTabelaFrequenciaSaida.getRowCount() - 1;
linha2++) {
                long valor1 = (Long) tblTabelaFrequenciaSaida.getValueAt(linha1, 0);
                long valor2 = (Long) tblTabelaFrequenciaSaida.getValueAt(linha2, 0);
                if (valor2 < valor1) {
                    ((DefaultTableModel)tblTabelaFrequenciaSaida.getModel()).moveRow(linha1,
linha1, linha2);
                }
            }
        }
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                 + "Os valores inseridos devem ser maiores que
zero.", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}
```

Note que o objeto da tabela de frequência foi nomeado “tblTabelaFrequenciaSaida”, e o método de validação da mesma foi nomeado “validaTabelaFrequenciaSaida”.

### 2.2.2 VALOR FIXO

Além de entrar os dados da taxa de saída de clientes do sistema, o usuário poderá optar por inserir um valor fixo para ser utilizado como taxa de saída de clientes durante toda a simulação.

Para que sua aparência seja criada, é possível reaproveitar o painel de valor fixo copiado da aba de entrada, alterando o texto do campo de “Lambda (λ)”, que representa a taxa de entrada de clientes no sistema, para “Mi (μ)”, que representa a taxa de saída de clientes do sistema na teoria de filas. A aparência ficará, por fim, como na Figura 9.

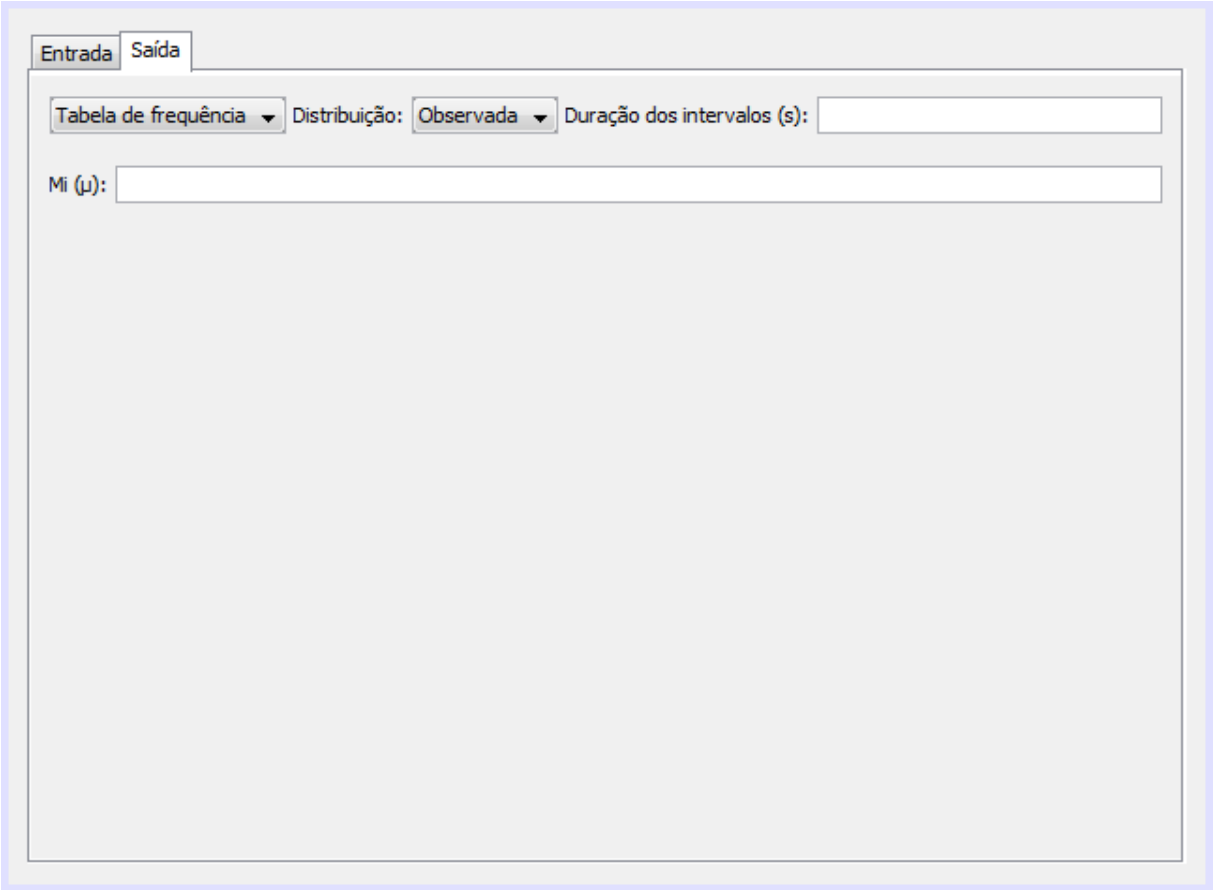


Figura 9: Aparência da forma de saída de clientes “valor fixo”

Após a aparência estar concluída será preciso configurar a validação do campo criado, não permitindo que sejam inseridos valores menores ou iguais a zero. Isto pode ser feito usando o evento “PropertyChange” com o código a seguir.

```
private void nmrValorFixoSaidaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (nmrValorFixoSaida.getValue() != null) {
        double lambda = Double.parseDouble(nmrValorFixoSaida.getValue().toString());
        if (lambda <= 0.0) {
            JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                    + "O valor da taxa de saída deve ser um
número positivo", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
            nmrValorFixoSaida.setValue(null);
        }
    }
}
```

O campo que guardará o valor fixo da taxa de saída de clientes do sistema foi nomeado “nmrValorFixoSaida”.

### 2.2.3 AJUSTES FINAIS

Assim como na aba relacionada à taxa de entrada de clientes no sistema, será preciso configurar o comportamento da caixa de combinação responsável pela escolha da forma de inserção dos dados relacionados à taxa de saída de clientes do sistema, exibindo apenas os componentes relacionados à forma selecionada.

Para isto será usado o evento “ActionPerformed” novamente, e serão feitos os filtros como está detalhado no código a seguir.

```
private void cmbSaidaActionPerformed(java.awt.event.ActionEvent evt) {
    final int TABELA_DE_FREQUENCIA = 0;
    final int VALOR_FIXO = 1;

    switch (cmbSaida.getSelectedIndex()) {
        case TABELA_DE_FREQUENCIA:
            pnlTabelaFrequenciaSaida.setVisible(true);
            txtDuracaoIntervaloSaida.setVisible(true);
            nmrDuracaoIntervaloSaida.setVisible(true);

            pnlValorFixoSaida.setVisible(false);
            break;
        case VALOR_FIXO:
            pnlValorFixoSaida.setVisible(true);

            pnlTabelaFrequenciaSaida.setVisible(false);
            txtDuracaoIntervaloSaida.setVisible(false);
            nmrDuracaoIntervaloSaida.setVisible(false);
            break;
    }
}
```

A caixa de combinação foi nomeada “cmbSaida”.

Assim como na aba de entrada, também será preciso assegurar que apenas estejam visíveis na aba de saída os componentes relacionados à opção padrão do

combo. O construtor da tela deverá ser alterado novamente, para que o mesmo também chame o evento “actionPerformed” do combo de saída.

O construtor ficará, então, como no código a seguir.

```
public SimulacaoView(SingleFrameApplication app) {
    super(app);

    initComponents();
    cmbEntradaActionPerformed(null);
    cmbSaidaActionPerformed(null);
}
```

## 2.3 ABA DE SIMULAÇÃO

A última aba do sistema será a aba que conterá a representação visual da simulação e suas variáveis, bem como campos para configurar parâmetros relativos à simulação.

Primeiramente será criado um novo painel dentro do painel tabulado, para ser a terceira e última aba do sistema, e seu texto será mudado para “Simulação”. Para facilitar esta tarefa será usado o “Inspetor”, clicando com o botão direito do mouse no painel tabulado, escolhendo “Adicionar da paleta”, “Contêineres Swing” e “Painel”, nesta ordem. Para alterar o texto basta clicar na aba e pressionar F2.

Dentro do novo painel campos deverão ser adicionados para que o usuário possa configurar os parâmetros referentes à simulação. O usuário poderá determinar o número de servidores, o número de filas, os horários inicial e final e a velocidade da simulação nesta aba. Tais campos (com exceção do horário inicial da simulação) poderão ser alterados durante a simulação em tempo real.

Além dos parâmetros da simulação, nesta aba também ficarão a representação visual da simulação, todas as informações relevantes e dois gráficos que auxiliarão no entendimento.

### 2.3.1 PARÂMETROS DA SIMULAÇÂO

Para o parâmetro referente ao número de servidores na simulação será adicionado um objeto da classe “JSpinner”, com seu modelo devidamente

configurado para permitir valor numérico inteiro, com valor inicial e mínimo 1. Para fazer isto basta selecionar o campo e clicar em “SpinnerModel” à frente de “model” no painel de propriedades do NetBeans, que pode ser aberto pelo menu “Janela”, no item “Propriedades”.

Para a escolha do número de filas serão adicionados dois botões de opção: o primeiro representará apenas uma fila para todos os servidores e o segundo representará uma fila para cada servidor. Como inicialmente o sistema estará configurado para apenas um servidor, não faria sentido manter a segunda opção ativa, portanto a mesma será deixada inativa. Para fazer isto basta selecionar a opção e desmarcar a caixa ao lado de “enabled” no painel de propriedades do NetBeans.

Para a inserção dos horários inicial e final, serão adicionados dois objetos da classe “JTextField”.

Finalmente, para a velocidade da simulação, será usado um objeto da classe “JSlider”, configurado para permitir valores entre 1 e 1000, e com valor inicial 871. Tais configurações podem ser feitas no painel de propriedades do NetBeans, alterando os valores das propriedades “minimum”, “maximum” e “value”, respectivamente. Será exibido, também, o tempo restante até o fim da simulação logo abaixo, em um objeto da classe “JLabel”. A velocidade inicial representará 1 minuto restante de simulação. A tela ficará, então, como na Figura 10.

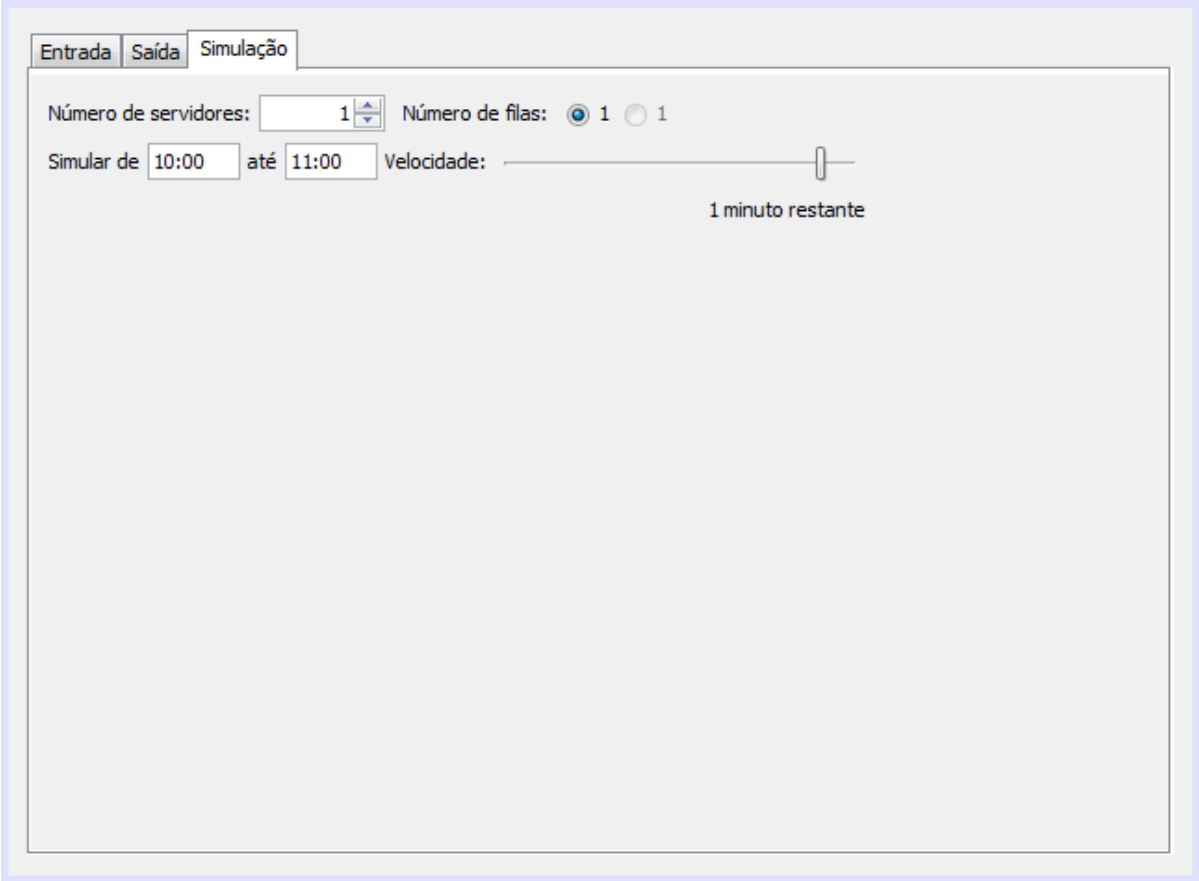


Figura 10: Aparência inicial da aba de simulação

Feito isto a aparência dos parâmetros da simulação deve estar concluída, mas ainda será preciso fazer algumas validações para concluir os requisitos da interface com o usuário.

Assim que o usuário alterar o número de servidores da simulação, a segunda opção do número de filas deverá ser ativada ou desativada, assim como seu texto deverá ser corrigido. Isto será feito no evento “stateChanged” do campo de número de servidores. Para se inserir código no evento basta clicar com o botão direito no campo e escolher “Eventos”, “Change” e “stateChanged”. O código a seguir deve ser suficiente.

```
private void numeroServidoresStateChanged(javax.swing.event.ChangeEvent evt) {
    multiplasFilas.setText(numeroServidores.getValue().toString());
    if (Integer.parseInt(numeroServidores.getValue().toString()) == 1) {
        unicaFila.setSelected(true);
        multiplasFilas.setEnabled(false);
    } else {
        multiplasFilas.setEnabled(true);
    }
}
```

Os campos de horário devem, também, ser validados, para que o usuário não possa inserir horários em formato inválido. Para tal será criado um método responsável por converter um texto em número, assumindo que este texto esteja no formato “hh:mm”. O código a seguir será suficiente para o necessário, tendo o novo método sido nomeado “converteHorario”.

```
private Double converteHorario(String horario) {
    if ((horario != null) && (!horario.isEmpty())) {
        int pos = horario.indexOf(":");
        if ((pos > 0) && (pos < horario.length() - 1)) {
            try {
                return Double.parseDouble(horario.substring(0, pos)) +
Double.parseDouble(horario.substring(pos + 1));
            } catch (Exception e) {
                return null;
            }
        } else {
            return null;
        }
    } else {
        return null;
    }
}
```

Para efetuar a validação em si será feito diferente desta vez: o evento de perda de foco será usado para verificar se o formato está correto usando o método criado, exibindo uma mensagem e retornando o foco para o campo caso haja algum problema. Para se codificar o referido evento basta clicar com o botão direito no campo em questão e escolher “Eventos”, “Focus” e por fim “focusLost”. O código a seguir trata o evento do campo de horário inicial, que foi nomeado “hmHorarioInicial”.

```
private void hmHorarioInicialFocusLost(java.awt.event.FocusEvent evt) {
    if (converteHorario(hmHorarioInicial.getText()) == null) {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                 + "O valor do horário inicial deve estar no
formato hh:mm", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                hmHorarioInicial.requestFocus();
            }
        });
    }
}
```

Nota-se que o método responsável por devolver o foco ao campo tem de ser chamado de forma diferente. Isso é necessário pois enquanto o evento de perda de foco não for concluído o foco continuará no campo, resultando que, ao clicar na mensagem de erro, o evento de perda de foco é invocado novamente. Se o retorno do foco for registrado para depois do fim da execução do evento este impasse é evitado.

De forma análoga, o código seguinte deverá ser suficiente para o tratamento do campo de horário final da simulação, que foi nomeado “hmHorarioFinal”.

```
private void hmHorarioFinalFocusLost(java.awt.event.FocusEvent evt) {
    if (converteHorario(hmHorarioFinal.getText()) == null) {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "O valor do horário final deve estar no
formato hh:mm", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                hmHorarioFinal.requestFocus();
            }
        });
    }
}
```

O tratamento do campo de velocidade é um pouco mais complexo que o tratamento dos demais campos. Nele será usada uma função matemática para permitir que se tenha maior precisão para velocidades maiores, permitindo o ajuste fino dos segundos restantes para o término da simulação, e se tenha menos precisão em velocidades menores, de forma que, na velocidade mínima permitida pela ferramenta, a simulação demore uma hora para ser concluída.

A função desenvolvida para se encontrar o tempo restante, em segundos, a partir da velocidade escolhida no controle deslizante foi $TR = (3(1001 - V) / 50)^2$, na qual TR representa o tempo restante em segundos e V representa a velocidade. Com a fórmula escolhida, na velocidade inicial (V = 871) a simulação durará aproximadamente um minuto, na velocidade máxima (V = 1000) a simulação acabará praticamente instantaneamente, e na velocidade mínima (V = 1) a simulação durará aproximadamente uma hora, permitindo várias opções de velocidade diferentes para diferentes finalidades.

Além de encontrar o tempo restante da simulação, também será necessário escrever este tempo por extenso para facilitar a compreensão. Será criado, então, um método responsável por isto, cujo código está reproduzido a seguir.

```
private String horarioExtenso(int horario, char base, boolean primaria) {
    String superior;
    int valor;
    String baseAtual;
    switch (base) {
        case 'S':
            valor = horario % 60;
            superior = horarioExtenso(horario / 60, 'M', false);
            baseAtual = " segundo";
            break;
        case 'M':
```

```
            valor = horario % 60;
            superior = horarioExtenso(horario / 60, 'H', false);
            baseAtual = " minuto";
            break;
        case 'H':
            valor = horario % 24;
            superior = horarioExtenso(horario / 60, 'D', false);
            baseAtual = " hora";
            break;
        case 'D':
            valor = horario;
            superior = "";
            baseAtual = " dia";
            break;
        default:
            return "";
    }
    String atual;
    if ((valor <= 0) && (!primaria)) {
        atual = "";
    } else {
        atual = valor + baseAtual + (valor != 1 ? "s" : "");
    }
    if (superior.isEmpty()) {
        return atual;
    } else {
        if (atual.isEmpty()) {
            return superior;
        } else {
            return superior + (primaria ? " e " : ", ") + atual;
        }
    }
}
```

O método será chamado “horarioExtenso”, e receberá três parâmetros: o horário a ser convertido como número inteiro, a base do número a ser convertido, podendo ser ‘S’ para segundos, ‘M’ para minutos, ‘H’ para horas ou ‘D’ para dias, e uma variável booleana responsável pelo controle da ligação entre tipos, permitindo que apenas o último tipo seja anexado aos demais com a conjunção aditiva “e” (como em “5 horas**,** 24 minutos **e** 10 segundos”). Será possível configurar a base, pois este método será usado mais uma vez neste trabalho.

Com o método criado, será feito o tratamento do evento do controle deslizante de velocidade. O evento que atenderá às expectativas é chamado “stateChanged”, e pode ser configurado clicando com o botão direito no componente e escolhendo “Eventos”, “Change e “stateChanged”. O código a seguir aplicará o necessário para o comportamento desejado.

```
private void dslVelocidadeStateChanged(javax.swing.event.ChangeEvent evt) {
    int tempoRestante = (int) Math.round(Math.pow(3.0 * (1001 - dslVelocidade.getValue()) /
50, 2));
    txtTempoRestante.setText(horarioExtenso(tempoRestante, 'S', true) + " restante" +
(tempoRestante != 1 ? "s" : ""));
}
```

O componente de controle deslizante referente à velocidade foi nomeado “dslVelocidade”.

### 2.3.2 REPRESENTAÇÃO VISUAL DA SIMULAÇÃO

A simulação será apresentada visualmente em tempo real, portanto será necessário criar os componentes responsáveis por esta apresentação. Será usada para isso a biblioteca Swing do Java, responsável pelas janelas feitas automaticamente no NetBeans, porém com uma classe personalizada para a finalidade do presente trabalho.

Para que seja possível desenhar na tela será criada uma classe herdeira da classe JPanel, da biblioteca Swing, para ser a classe do painel de simulação. Nesta classe o método “paintComponent”, responsável por desenhar o componente na tela, será sobreposto, e usado para desenhar os itens necessários na tela.

Para desenhar corretamente os servidores e filas será necessário criar alguns atributos na nova classe. Serão guardados em atributos as imagens escolhidas para cliente e servidor, o número de clientes nas filas, o tamanho da maior fila (para evitar consultar a todo momento) e o número de servidores.

Redimensionamento de imagem é uma tarefa computacionalmente custosa, portanto é melhor evitá-la quando possível. Por este motivo, serão criados também atributos para guardar as imagens redimensionadas, e um número que indicará qual era uma das dimensões quando as mesmas foram redimensionadas. Ao calcular as dimensões atuais no método “paintComponent” será possível fazer uma comparação com a dimensão armazenada anteriormente, e então as imagens só serão redimensionadas novamente caso essa dimensão tenha mudado. Se as dimensões continuarem as mesmas as imagens redimensionadas anteriormente serão mantidas e será necessário menos processamento.

O código a seguir representa o código da classe, a qual será colocada em um arquivo separado e nomeada “PainelGrafico”.

```
import java.awt.Graphics;
import java.awt.Graphics2D;
```

```
import java.awt.Image;
import javax.swing.JPanel;

public class PainelGrafico extends JPanel {

    public Image  imagemCliente;
    public Image  imagemClienteRedimensionada;
    public Image  imagemServidor;
    public Image  imagemServidorRedimensionada;
    public int    numeroClientesFila[];
    public int    tamanhoMaiorFila;
    public int    numeroServidores;
    public double alturaAnterior;

    public PainelGrafico() {
        imagemCliente = null;
        imagemClienteRedimensionada = null;
        imagemServidor = null;
        imagemServidorRedimensionada = null;
        numeroClientesFila = null;
        tamanhoMaiorFila = 0;
        numeroServidores = 0;
        alturaAnterior = 0;
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        if ((imagemCliente != null) && (imagemServidor != null)) {
            double aspectoCliente = (double)(imagemCliente.getWidth(this)) /
imagemCliente.getHeight(this);
            double aspectoServidor = (double)(imagemServidor.getWidth(this)) /
imagemServidor.getHeight(this);
            double alturaMaxima = ((this.getHeight() - 80) / numeroServidores) * 0.95;
            double larguraServidorPorClientes = 2 * aspectoServidor / aspectoCliente;
            double numeroClientesHorizontal = 1.05 * tamanhoMaiorFila +
larguraServidorPorClientes + 0.05;
            double larguraMaximaCliente = (this.getWidth() - 40) / numeroClientesHorizontal;
            double larguraMaximaServidor = larguraMaximaCliente * larguraServidorPorClientes;
            double alturaServidor = larguraMaximaServidor / aspectoServidor;
            double larguraServidor;
            if (alturaServidor <= alturaMaxima) {
                larguraServidor = larguraMaximaServidor;
            } else {
                alturaServidor = alturaMaxima;
                larguraServidor = alturaServidor * aspectoServidor;
            }
            double alturaCliente = alturaServidor / 2;
            double larguraCliente = alturaCliente * aspectoCliente;
            if (alturaServidor < 1) {
                alturaServidor = 1;
            }
            if (larguraServidor < 1) {
                larguraServidor = 1;
            }
            if (alturaCliente < 1) {
                alturaCliente = 1;
            }
            if (larguraCliente < 1) {
                larguraCliente = 1;
            }
            if (alturaServidor != alturaAnterior) {
                alturaAnterior = alturaServidor;
                imagemClienteRedimensionada = imagemCliente.getScaledInstance((int)
larguraCliente, (int) alturaCliente, Image.SCALE_SMOOTH);
                imagemServidorRedimensionada = imagemServidor.getScaledInstance((int)
larguraServidor, (int) alturaServidor, Image.SCALE_SMOOTH);
            }
            Graphics2D g2D = (Graphics2D) g;
            int margemDireita = this.getWidth() - 20 - (int) larguraServidor;
            for (int i = 0; i < numeroServidores; i++) {
                g2D.drawImage(imagemServidorRedimensionada, margemDireita, (int)(20 + i *
(alturaServidor * 1.05)), this);
            }
            for (int i = 0; i < numeroClientesFila.length; i++) {
                for (int j = 0; j < numeroClientesFila[i]; j++) {
```

```
                    g2D.drawImage(imagemClienteRedimensionada, margemDireita -
(int)(larguraCliente * 0.10) - (int)((j + 1) * larguraCliente * 1.05), 20 + (int)(i *
alturaServidor * 1.05) + (int)(alturaCliente / 2), this);
                }
            }
        }
    }

}
```

Será adicionado, então, um novo painel dentro da aba de simulação, ocupando o espaço que a simulação deverá ocupar. Dentro deste painel serão adicionados objetos da classe “JLabel”, que serão utilizados durante a simulação para exibir o número de clientes no sistema e o número de clientes já atendidos pelo sistema. A tela ficará, então, como na Figura 11.

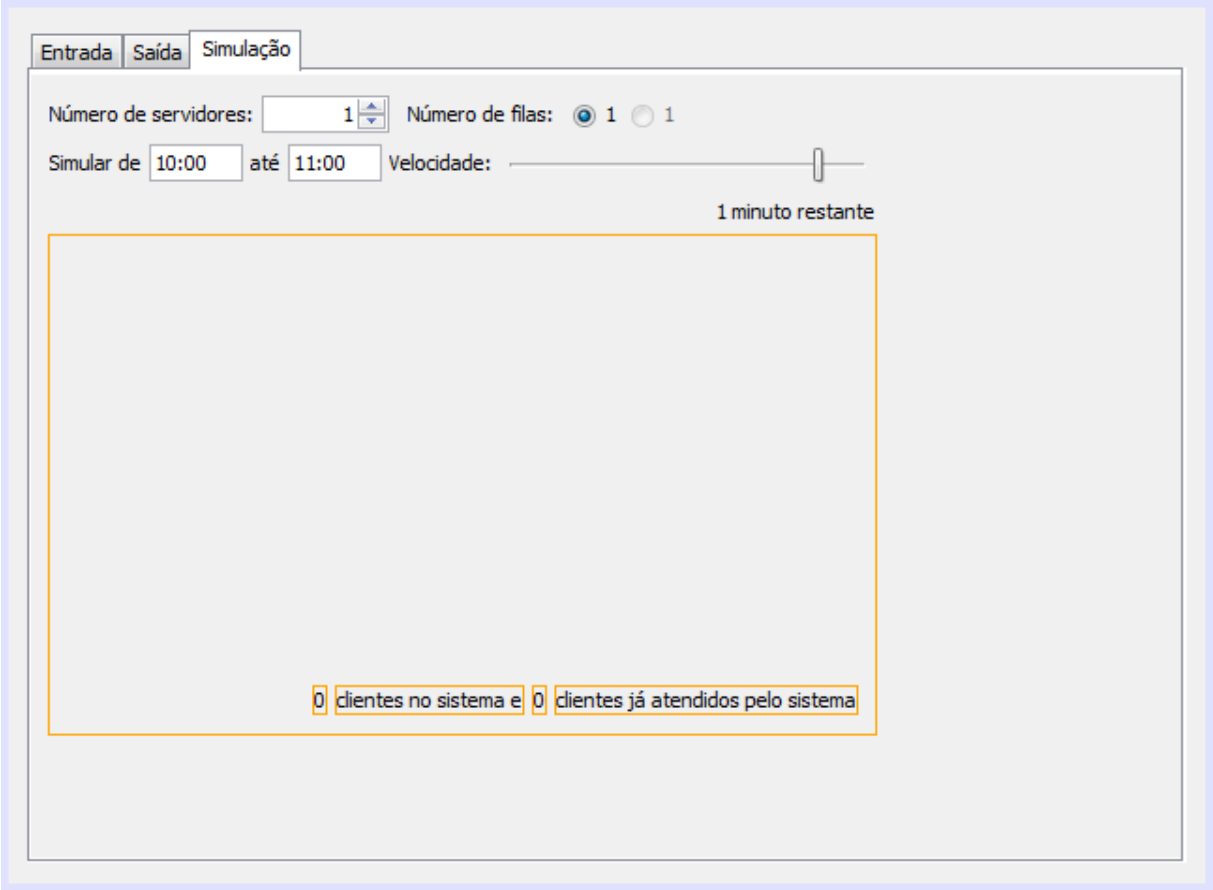


Figura 11: Aparência da aba de simulação com painel gráfico

Para que o painel original seja da classe “PainelGrafico” será necessário alterar a linha de código que o constrói. Para fazer isto será preciso clicar com o botão direito no painel original (usando o Inspetor, para facilitar) e escolher “Personalizar código”. O NetBeans apresentará uma janela com o código pronto

relacionado ao componente, e a opção referente à construção do mesmo deverá ser mudada para um código personalizado. Por fim, esta tela deverá ficar como na Figura 12.

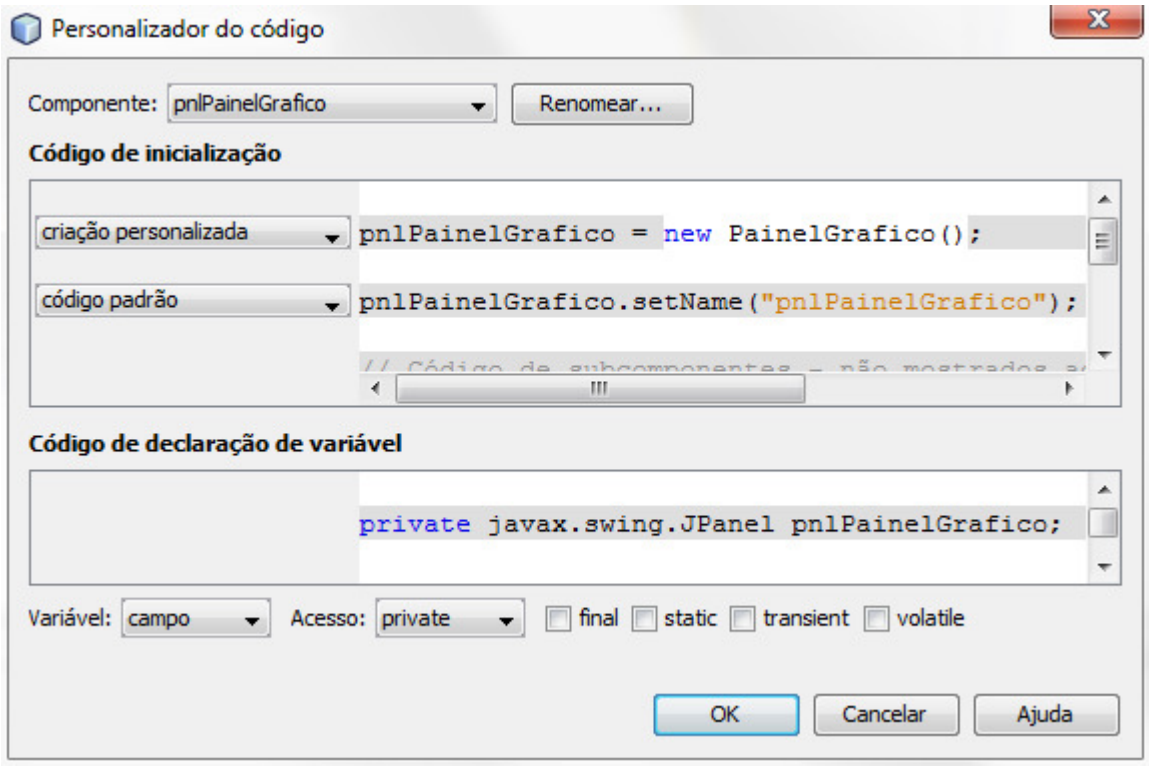


Figura 12: Código personalizado para construir o painel gráfico

Repare que o painel original foi nomeado “pnlPainelGrafico”.

Antes do início da simulação, porém, este mesmo espaço será usado para que o usuário possa escolher as imagens que representarão os clientes e servidores. Um novo painel será, então, criado dentro deste, e será usado como container para as opções de aparência da simulação.

Será adicionado um campo de texto, da classe “JTextField”, um botão para buscar imagens no computador e um objeto da classe “JLabel” no novo painel. O campo de texto guardará o caminho da imagem escolhida, e não permitirá edição direta. Para se definir isto basta selecionar o mesmo e desmarcar a caixa ao lado de “editable” no painel de propriedades do NetBeans. O “JLabel” exibirá uma prévia da

imagem escolhida. Estes campos deverão ser adicionados para cliente e servidor, portanto serão duas cópias de cada.

Feito isto a aparência da tela ficará como mostrada na Figura 13.

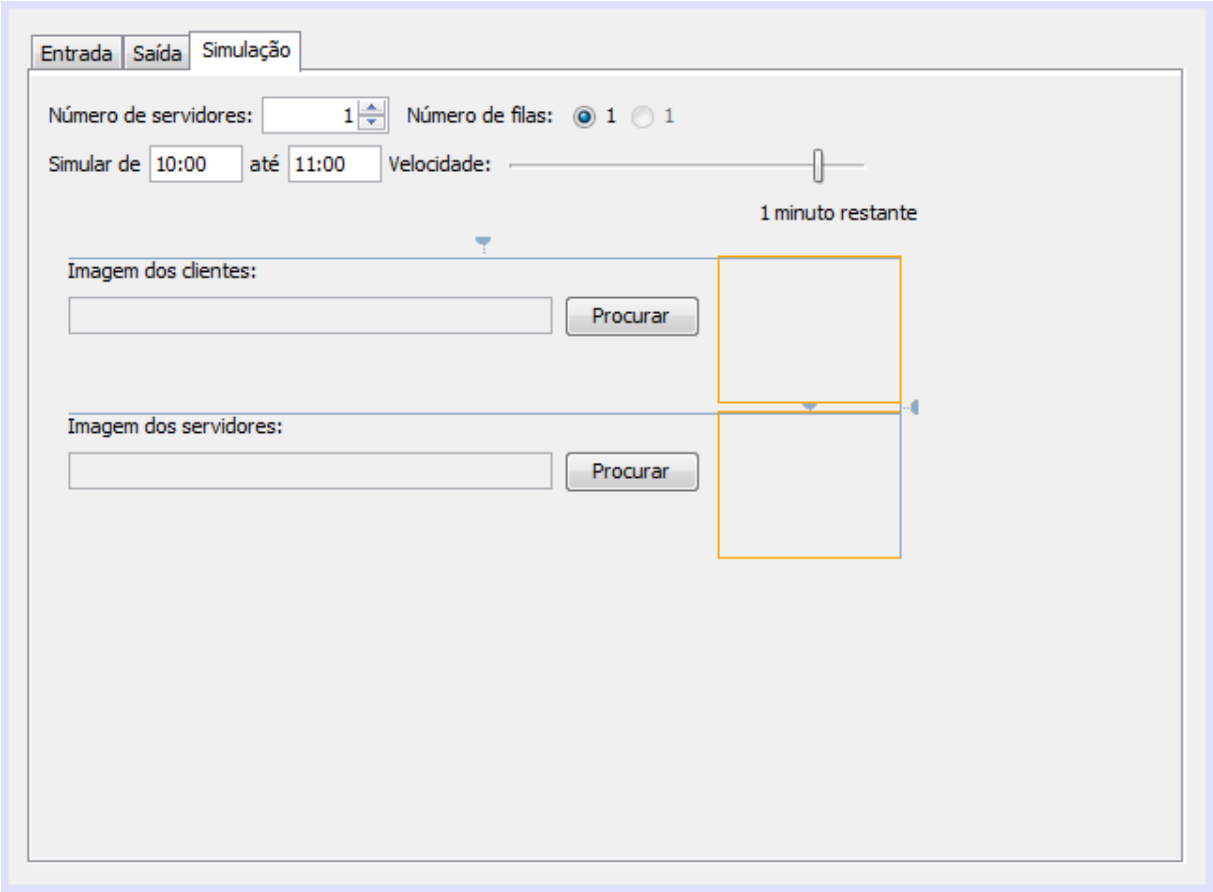


Figura 13: Aparência da aba de simulação com painel de aparência

Agora que os componentes estão lá será necessário fazê-los funcionar. Os botões deverão abrir caixas de escolha de arquivos, e estas caixas deverão permitir apenas a escolha de arquivos de imagem.

Para fazer isto será criado um novo arquivo de classe, e dentro deste arquivo será codificado tudo que se fizer necessário.

Além dos códigos também será necessário incluir quatro arquivos de imagem no projeto. Estes arquivos conterão os ícones dos quatro tipos de imagens

suportadas: GIF, JPG, PNG e TIF. Estes arquivos serão incluídos em um pacote novo, chamado “imagens”.

O código a seguir, levemente modificado de um exemplo chamado FileChooserDemo2 (ORACLE), disponibilizado pela Oracle[3] em seu site oficial, será suficiente para cumprir os requisitos do novo arquivo, que será chamado “Imagem.java”.

```
/*
* Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* - Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in the
* documentation and/or other materials provided with the distribution.
*
* - Neither the name of Oracle or the names of its
* contributors may be used to endorse or promote products derived
* from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
* IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.io.File;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileView;

public class Imagem {
    public static String abrir(Component pai) {
        JFileChooser fc = new JFileChooser();
        fc.addChoosableFileFilter(new ImagemFilter());
        fc.setAcceptAllFileFilterUsed(false);
        fc.setFileView(new ImagemFileView());
        fc.setAccessory(new ImagemPreview(fc));
        if (fc.showDialog(pai, "Selecione a imagem") == JFileChooser.APPROVE_OPTION) {
            return fc.getSelectedFile().getAbsolutePath();
        } else {
```

[3] Oracle Corporation é uma empresa multinacional de tecnologia e informática dos Estados Unidos, especializada no desenvolvimento e comercialização de hardware e software.

```
            return null;
        }
    }
}

class ImagemFilter extends FileFilter {

    @Override
    public boolean accept(File f) {
        if (f.isDirectory()) {
            return true;
        }

        String extension = Utils.getExtension(f);
        if (extension != null) {
            if (extension.equals(Utils.tiff) ||
                extension.equals(Utils.tif) ||
                extension.equals(Utils.gif) ||
                extension.equals(Utils.jpeg) ||
                extension.equals(Utils.jpg) ||
                extension.equals(Utils.png)) {
                    return true;
            } else {
                return false;
            }
        }

        return false;
    }

    @Override
    public String getDescription() {
        return "Apenas imagens";
    }
}

class ImagemFileView extends FileView {
    ImageIcon jpgIcon = Utils.createImageIcon("JPG.png");
    ImageIcon gifIcon = Utils.createImageIcon("GIF.png");
    ImageIcon tiffIcon = Utils.createImageIcon("TIF.png");
    ImageIcon pngIcon = Utils.createImageIcon("PNG.png");

    @Override
    public String getName(File f) {
        return null;
    }

    @Override
    public String getDescription(File f) {
        return null;
    }

    @Override
    public Boolean isTraversable(File f) {
        return null;
    }

    @Override
    public String getTypeDescription(File f) {
        String extension = Utils.getExtension(f);
        String type = null;

        if (extension != null) {
            if (extension.equals(Utils.jpeg) ||
                extension.equals(Utils.jpg)) {
                type = "Imagem JPEG";
            } else if (extension.equals(Utils.gif)){
                type = "Imagem GIF";
            } else if (extension.equals(Utils.tiff) ||
                       extension.equals(Utils.tif)) {
                type = "Imagem TIFF";
            } else if (extension.equals(Utils.png)){
                type = "Imagem PNG";
            }
        }
        return type;
    }
```

```
    @Override
    public Icon getIcon(File f) {
        String extension = Utils.getExtension(f);
        Icon icon = null;

        if (extension != null) {
            if (extension.equals(Utils.jpeg) ||
                extension.equals(Utils.jpg)) {
                icon = jpgIcon;
            } else if (extension.equals(Utils.gif)) {
                icon = gifIcon;
            } else if (extension.equals(Utils.tiff) ||
                       extension.equals(Utils.tif)) {
                icon = tiffIcon;
            } else if (extension.equals(Utils.png)) {
                icon = pngIcon;
            }
        }
        return icon;
    }
}

class ImagemPreview extends JComponent
                        implements PropertyChangeListener {
    ImageIcon thumbnail = null;
    File file = null;

    public ImagemPreview(JFileChooser fc) {
        setPreferredSize(new Dimension(100, 50));
        fc.addPropertyChangeListener(this);
    }

    public void loadImage() {
        if (file == null) {
            thumbnail = null;
            return;
        }

        ImageIcon tmpIcon = new ImageIcon(file.getPath());
        if (tmpIcon != null) {
            if (tmpIcon.getIconWidth() > 90) {
                thumbnail = new ImageIcon(tmpIcon.getImage().
                                          getScaledInstance(90, -1,
                                                  Image.SCALE_DEFAULT));
            } else {
                thumbnail = tmpIcon;
            }
        }
    }

    @Override
    public void propertyChange(PropertyChangeEvent e) {
        boolean update = false;
        String prop = e.getPropertyName();

        if (JFileChooser.DIRECTORY_CHANGED_PROPERTY.equals(prop)) {
            file = null;
            update = true;

        } else if (JFileChooser.SELECTED_FILE_CHANGED_PROPERTY.equals(prop)) {
            file = (File) e.getNewValue();
            update = true;
        }

        if (update) {
            thumbnail = null;
            if (isShowing()) {
                loadImage();
                repaint();
            }
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        if (thumbnail == null) {
```

```
            loadImage();
        }
        if (thumbnail != null) {
            int x = getWidth()/2 - thumbnail.getIconWidth()/2;
            int y = getHeight()/2 - thumbnail.getIconHeight()/2;

            if (y < 0) {
                y = 0;
            }

            if (x < 5) {
                x = 5;
            }
            thumbnail.paintIcon(this, g, x, y);
        }
    }
}

class Utils {
    public final static String jpeg = "jpeg";
    public final static String jpg = "jpg";
    public final static String gif = "gif";
    public final static String tiff = "tiff";
    public final static String tif = "tif";
    public final static String png = "png";

    public static String getExtension(File f) {
        String ext = null;
        String s = f.getName();
        int i = s.lastIndexOf('.');

        if (i > 0 &&  i < s.length() - 1) {
            ext = s.substring(i+1).toLowerCase();
        }
        return ext;
    }

    protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = Utils.class.getResource("imagens/" + path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Arquivo não encontrado: " + path);
            return null;
        }
    }
}
```

Com a classe responsável pela escolha de imagens completa, serão codificados os eventos dos botões. O evento utilizado para configurar o comportamento dos botões será o “actionPerformed”, e o código a seguir será suficiente para o botão responsável por carregar a imagem do cliente.

```
private void botImagemClientesActionPerformed(java.awt.event.ActionEvent evt) {
    String caminho = Imagem.abrir(mainPanel);
    File arquivo = new File(caminho);
    if ((caminho != null) && (!caminho.isEmpty()) && (arquivo.exists())) {
        try {
            ((PainelGrafico) pnlPainelGrafico).imagemCliente = ImageIO.read(arquivo);
            ((PainelGrafico) pnlPainelGrafico).alturaAnterior = 0;
            edtImagemClientes.setText(caminho);
            ImageIcon icone = new ImageIcon(caminho);
            imgImagemClientes.setIcon(icone);
            int width, height;
            if (icone.getImage().getWidth(icone.getImageObserver()) /
icone.getImage().getHeight(icone.getImageObserver()) > 100.0 / 80.0) {
                width = 100;
```

```
                height = (int)((100.0 / ((ImageIcon) icone).getImage().getWidth(((ImageIcon)
icone).getImageObserver())) * ((ImageIcon) icone).getImage().getHeight(((ImageIcon)
icone).getImageObserver()));
                if (height < 1) {
                    height = 1;
                }
            } else {
                height = 80;
                width = (int)((80.0 / ((ImageIcon) icone).getImage().getHeight(((ImageIcon)
icone).getImageObserver())) * ((ImageIcon) icone).getImage().getWidth(((ImageIcon)
icone).getImageObserver()));
                if (width < 1) {
                    width = 1;
                }
            }
            icone.setImage(icone.getImage().getScaledInstance(width, height,
Image.SCALE_SMOOTH));
            imgImagemClientes.setSize(width, height);
        } catch (Exception ex) {
            Logger.getLogger(SimulacaoView.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Note-se que o campo de texto, o botão e o objeto da classe “JLabel” foram chamados “edtImagemClientes”, “botImagemClientes” e “imgImagemClientes”, respectivamente, e o último possui 100 pixels de largura por 80 pixels de altura.

Semelhantemente, o código a seguir atenderá aos requisitos do evento “actionPerformed” do botão responsável pela escolha de uma imagem para representar os servidores.

```
private void botImagemServidoresActionPerformed(java.awt.event.ActionEvent evt) {
    String caminho = Imagem.abrir(mainPanel);
    File arquivo = new File(caminho);
    if ((caminho != null) && (!caminho.isEmpty()) && (arquivo.exists())) {
        try {
            ((PainelGrafico) pnlPainelGrafico).imagemServidor = ImageIO.read(arquivo);
            ((PainelGrafico) pnlPainelGrafico).alturaAnterior = 0;
            edtImagemServidores.setText(caminho);
            ImageIcon icone = new ImageIcon(caminho);
            imgImagemServidores.setIcon(icone);
            int width, height;
            if (icone.getImage().getWidth(icone.getImageObserver()) /
icone.getImage().getHeight(icone.getImageObserver()) > 100.0 / 80.0) {
                width = 100;
                height = (int)((100.0 / ((ImageIcon) icone).getImage().getWidth(((ImageIcon)
icone).getImageObserver())) * ((ImageIcon) icone).getImage().getHeight(((ImageIcon)
icone).getImageObserver()));
                if (height < 1) {
                    height = 1;
                }
            } else {
                height = 80;
                width = (int)((80.0 / ((ImageIcon) icone).getImage().getHeight(((ImageIcon)
icone).getImageObserver())) * ((ImageIcon) icone).getImage().getWidth(((ImageIcon)
icone).getImageObserver()));
                if (width < 1) {
                    width = 1;
                }
            }
            icone.setImage(icone.getImage().getScaledInstance(width, height,
Image.SCALE_SMOOTH));
            imgImagemServidores.setSize(width, height);
        } catch (Exception ex) {
            Logger.getLogger(SimulacaoView.class.getName()).log(Level.SEVERE, null, ex);
```

```
            }
        }
}
```

### 2.3.3 APRESENTAÇÃO DE INFORMAÇÕES

A ferramenta apresentará para o usuário, em tempo real, as informações relevantes para a avaliação de desempenho do sistema de filas. Tais informações serão apresentadas em dois gráficos e em variáveis numéricas.

Serão feitos, abaixo do painel gráfico no qual ocorrerá a simulação, dois novos painéis, destinados a conter os dois gráficos. Os layouts de ambos deverão ser alterados para “Layout da borda”, assim como feito com o painel referente ao gráfico da curva da taxa de entrada para a medição cronometrada.

À direita do painel gráfico será criado mais um painel, e neste serão adicionados campos para as variáveis relevantes da teoria de filas, que são: intensidade de tráfego, probabilidade de zero clientes no sistema, probabilidade de enfileiramento, e as seguintes variáveis com suas variâncias: numero médio de clientes no sistema, de clientes na fila, tempo médio no sistema e tempo médio de espera.

Além dos campos, serão adicionados também dois objetos da classe “JLabel”, destinados a apresentar as estabilidades momentânea e média do sistema.

Neste momento a aparência da tela estará como apresentada na Figura 14.

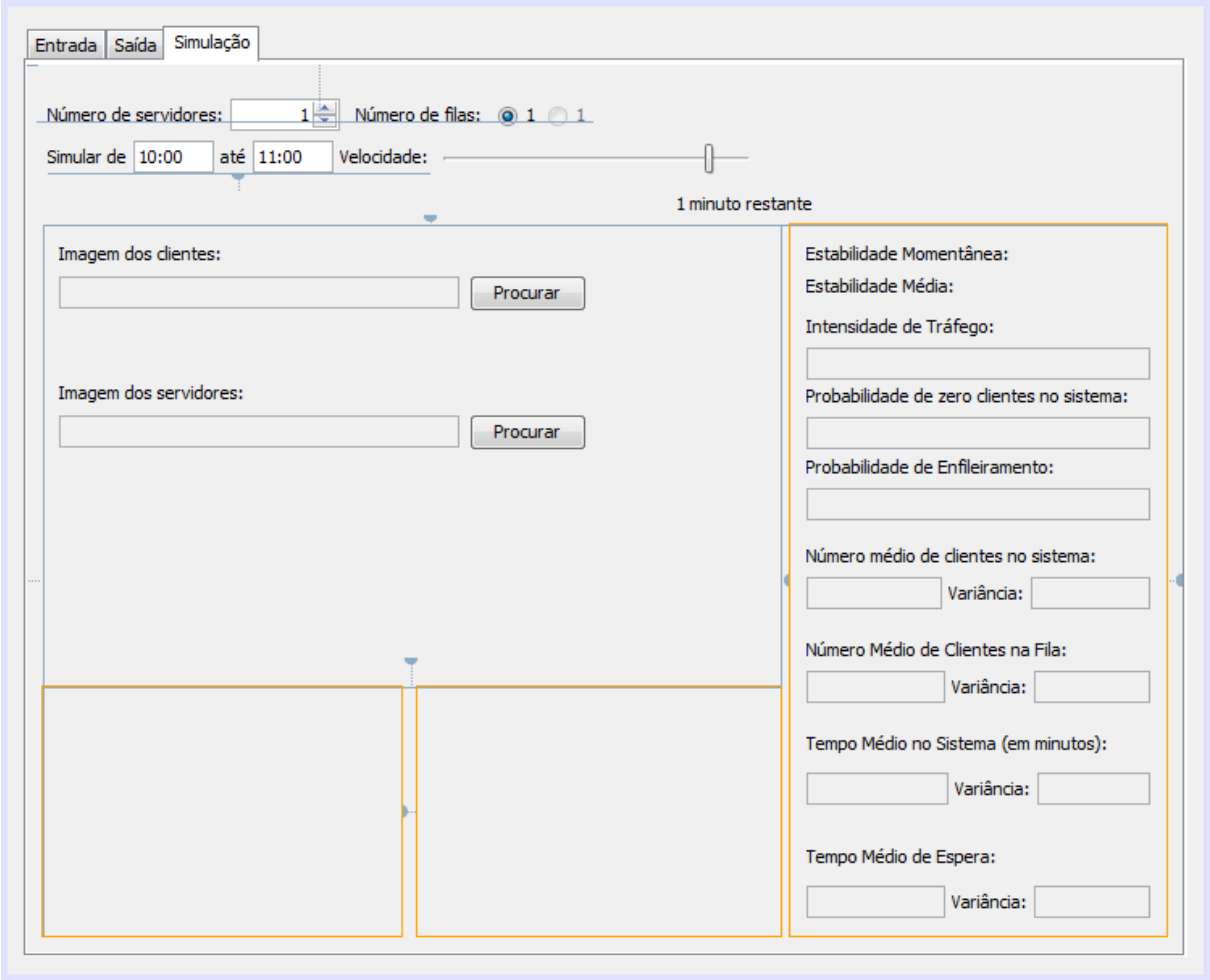


Figura 14: Aparência da aba de simulação com componentes informativos

### 2.3.4 AJUSTES FINAIS

Serão necessários dois botões adicionais na aba de simulação. O primeiro será usado para iniciar e pausar a simulação, e o segundo servirá para reiniciá-la. Ambos serão adicionados no canto superior direito, e a aparência da tela ficará como na Figura 15.

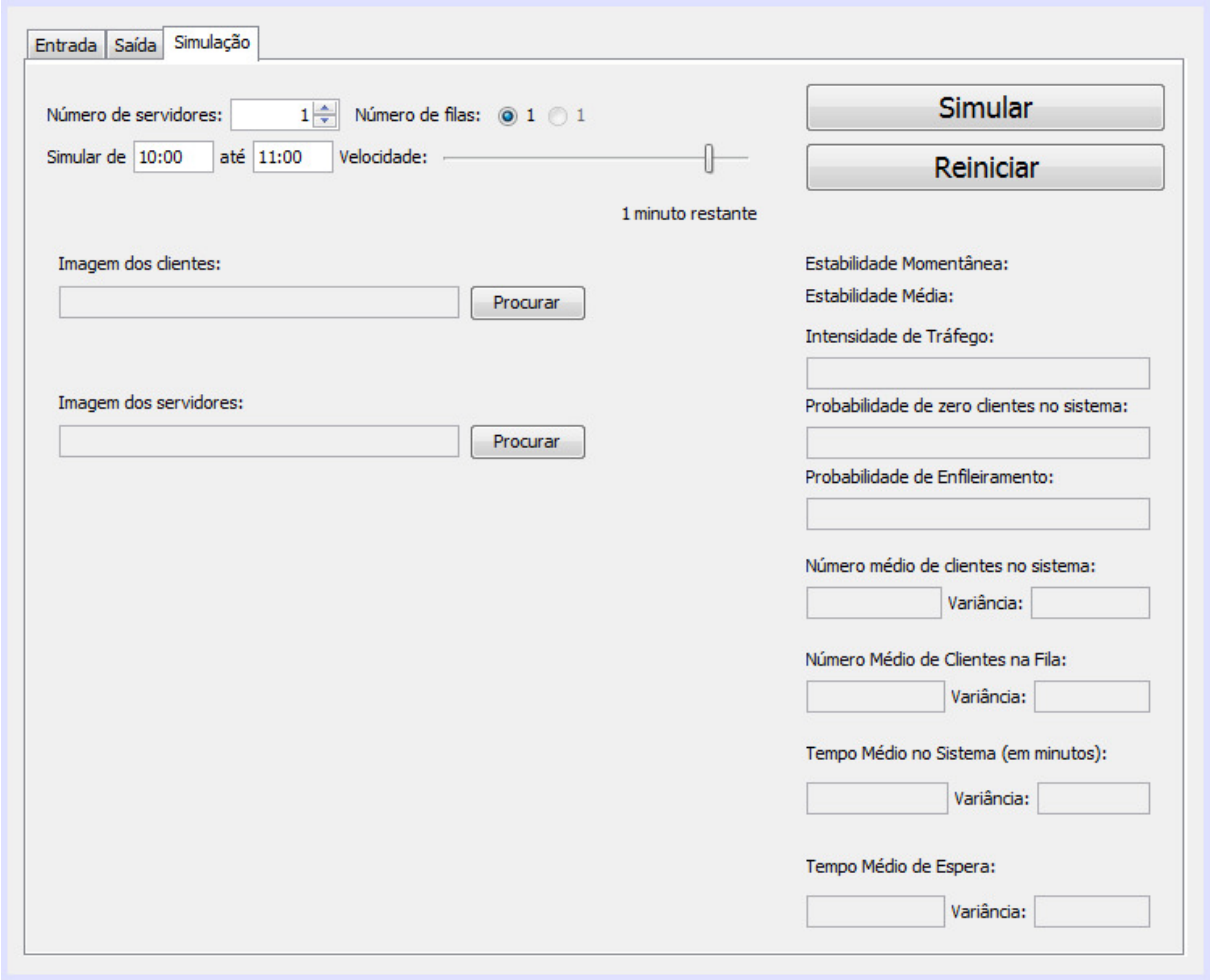


Figura 15: Aparência final da aba de simulação

Também será preciso especificar que os objetos da classe “JLabel” dentro do painel gráfico, que exibirão informações enquanto a simulação estiver sendo executada, sejam invisíveis ao iniciar a ferramenta. Para isto o construtor da tela será alterado novamente, ficando com o código seguinte.

```
public SimulacaoView(SingleFrameApplication app) {
    super(app);

    initComponents();
    cmbEntradaActionPerformed(null);
    cmbSaidaActionPerformed(null);
    txtNumeroClientesSistema.setVisible(false);
    txtTextoClientesSistema.setVisible(false);
    txtNumeroClientesAtendidos.setVisible(false);
    txtTextoClientesAtendidos.setVisible(false);
}
```

Os referidos objetos foram nomeados “txtNumeroClientesSistema”, “txtTextoClientesSistema”, “txtNumeroClientesAtendidos” e “txtTextoClientesAtendidos”.

# 3 DESENVOLVENDO AS FUNCIONALIDADES

Agora que a interface com o usuário está pronta resta apenas implementar as funcionalidades esperadas da mesma.

## 3.1 CURVA DE MEDIÇÃO CRONOMETRADA

Na inserção dos dados relacionados à taxa de entrada de clientes no sistema, uma das formas é a medição cronometrada. Nesta forma o usuário insere todas as medições realizadas, e a ferramenta deve gerar uma curva para a taxa de entrada que melhor represente estes dados.

Para melhor compreender a geração desta curva alguns pontos devem ser esclarecidos. Os dados inseridos informarão o número de clientes que chegaram em intervalos determinados. A curva que será gerada, entretanto, é da taxa de chegada, que é a uma função contínua derivada do número de clientes no sistema.

A tarefa atual é, então, gerar uma curva cuja integral definida em cada intervalo inserido seja tão próxima quanto possível do valor inserido pelo usuário.

Para isso será usada uma biblioteca específica para cálculos científicos chamada JScience (JScience), em sua versão experimental (EDU-DEVELOPER). Uma de suas classes, chamada DoubleCubicSplineInterpolator, implementa interpolação de pontos em um plano, o que facilitará nossa tarefa de criar a curva.

O objeto da curva deverá ser guardado, para que seja possível utilizá-lo futuramente na própria simulação. Criar-se-á, então, um atributo na classe da tela para guardá-la, e se alterará o construtor para que o mesmo seja inicializado. O código a seguir demonstra como o início do código da classe ficará após as modificações.

```
public class SimulacaoView extends FrameView {

    private DoubleCubicSplineInterpolator curva;

    public SimulacaoView(SingleFrameApplication app) {
        super(app);

        initComponents();
        cmbEntradaActionPerformed(null);
        cmbSaidaActionPerformed(null);
        txtNumeroClientesSistema.setVisible(false);
        txtTextoClientesSistema.setVisible(false);
        txtNumeroClientesAtendidos.setVisible(false);
        txtTextoClientesAtendidos.setVisible(false);
        curva = null;
    }
    // [...]
```

A integração será realizada de forma aproximada usando uma técnica de integração numérica comumente chamada de regra dos trapézios (DA ROCHA LOPES e A. GOMES RUGGIERO, 1996), uma fórmula fechada de Newton[4]-Cotes[5] de primeiro grau. Por ser um método numérico, o resultado obtido não será exato, mas uma aproximação do valor desejado.

A integral definida de uma função corresponde à área abaixo da curva em um intervalo determinado. Ao aplicar a regra dos trapézios subdividimos, então, esta área em dois ou mais pontos espaçados uniformemente e montamos trapézios a partir dos pontos obtidos nos extremos de cada subdivisão. O resultado da soma das áreas de todos os trapézios será um número próximo ao valor da integral definida, tendo mais precisão com intervalos menores (e consequentemente mais trapézios).

Será aplicado um mecanismo de controle de erro na integral, comparando uma aproximação com a próxima aproximação, feita com mais trapézios. Se a diferença entre as duas estiver muito alta o resultado ainda não estará bom o bastante. Assim que a diferença entre uma aproximação e a próxima estiver abaixo de um centésimo de seu valor a aproximação será aceita.

Para simplificar o código será criado um método específico para realizar a integração numérica. Este método receberá o início do intervalo e sua duração, e

[4] Sir Isaac Newton, nascido em 4 de janeiro de 1643, foi um cientista inglês. Ganhou renome por suas contribuições à física e matemática, porém também estudou astronomia, alquimia, filosofia natural e teologia. Faleceu em 31 de março de 1727.
[5] Roger Cotes, nascido em 10 de julho de 1682, foi um matemático inglês conhecido por realizar trabalhos em parceria com Isaac Newton. Faleceu em 5 de junho de 1716.

retornará o valor aproximado da integral definida no mesmo. O código a seguir será suficiente para o referido.

```
private double integrar(double inicio, double tamanho) {
    double valorMedio = (curva.map(inicio) + curva.map(inicio + tamanho)) / 2.0;
    if(valorMedio<0){
        valorMedio = 0;
    }
    double aproximacao = tamanho * valorMedio;
    int cont = 1;
    double anterior;
    do{
        cont++;
        double divisao = tamanho / cont;
        anterior = aproximacao;
        aproximacao = 0;
        double pontoAtual = curva.map(inicio);
        for (int i = 1; i <= cont; i++) {
            double proximoPonto = curva.map(inicio + i * divisao);
            valorMedio = (pontoAtual + proximoPonto) / 2;
            if (valorMedio > 0) {
                aproximacao += divisao * valorMedio;
            }
            pontoAtual = proximoPonto;
        }
    } while (Math.abs(aproximacao - anterior) > aproximacao / 100);
    return aproximacao;
}
```

Por fim, para apresentar a curva para o usuário, será gerado um gráfico de linha a partir do objeto da curva. Uma biblioteca Java chamada JFreeChart (JFreeChart), específica para esta tarefa, será usada, e em seguida o gráfico será adicionado ao painel reservado para o mesmo.

O código criado anteriormente para o evento “propertyChange” da tabela de medição de frequência será, então, alterado para adicionar as funcionalidades desejadas. O código a seguir demonstra como o novo código do evento ficará contemplando todas estas funcionalidades.

```
private void tblMedicaoCronometradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaMedicaoCronometrada()) {
        if (tblMedicaoCronometrada.getRowCount() > 1) {
            for (int linha1 = 0; linha1 < tblMedicaoCronometrada.getRowCount() - 1; linha1++)
{
                for (int linha2 = linha1 + 1; linha2 < tblMedicaoCronometrada.getRowCount() -
1; linha2++) {
                    String aux[] = ((String)tblMedicaoCronometrada.getValueAt(linha1,
0)).split(":");
                    double valor1 = Long.parseLong(aux[0]) * 60 +
Double.parseDouble(aux[1].replace(",", "."));
                    aux = ((String)tblMedicaoCronometrada.getValueAt(linha2, 0)).split(":");
                    double valor2 = Long.parseLong(aux[0]) * 60 +
Double.parseDouble(aux[1].replace(",", "."));
                    if (valor2 < valor1) {
                        ((DefaultTableModel)tblMedicaoCronometrada.getModel()).moveRow(linha1,
linha1, linha2);
                    }
                }
            }
```

```
            double x[] = new double[tblMedicaoCronometrada.getRowCount() + 3];
            double y[] = new double[tblMedicaoCronometrada.getRowCount() + 3];
            x[0] = x[1] = x[2] = converteHorario(tblMedicaoCronometrada.getValueAt(0,
0).toString());
            for (int i = 1; i < tblMedicaoCronometrada.getRowCount() - 2; i++) {
                x[i+2] = converteHorario(tblMedicaoCronometrada.getValueAt(i, 0).toString()) +
((Double)tblMedicaoCronometrada.getValueAt(i, 1)) / 2;
            }
            x[tblMedicaoCronometrada.getRowCount()] = x[tblMedicaoCronometrada.getRowCount() +
1] = x[tblMedicaoCronometrada.getRowCount() + 2] =
converteHorario(tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 2,
0).toString()) +
((Double)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 2, 1));
            double aux = (x[tblMedicaoCronometrada.getRowCount()] - x[0]) / 10000.0;
            x[0] -= 2 * aux;
            x[1] -= aux;
            x[tblMedicaoCronometrada.getRowCount() + 1] += aux;
            x[tblMedicaoCronometrada.getRowCount() + 2] += 2 * aux;
            y[0] = y[1] = ((Long)tblMedicaoCronometrada.getValueAt(0, 2)) /
((Double)tblMedicaoCronometrada.getValueAt(0, 1));
            for (int i = 0; i < tblMedicaoCronometrada.getRowCount() - 1; i++) {
                y[i+2] = ((Long)tblMedicaoCronometrada.getValueAt(i, 2)) /
((Double)tblMedicaoCronometrada.getValueAt(i, 1));
            }
            y[tblMedicaoCronometrada.getRowCount() + 1] =
y[tblMedicaoCronometrada.getRowCount() + 2] =
((Long)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 2, 2)) /
((Double)tblMedicaoCronometrada.getValueAt(tblMedicaoCronometrada.getRowCount() - 2, 1));
            boolean flag;
            double diff;
            do {
                curva = new DoubleCubicSplineInterpolator(x, y);
                flag = false;
                for (int i = 2; i < tblMedicaoCronometrada.getRowCount() + 1; i++) {
                    diff = integrar(converteHorario(tblMedicaoCronometrada.getValueAt(i - 2,
0).toString()), ((Double)tblMedicaoCronometrada.getValueAt(i - 2, 1))) -
((Long)tblMedicaoCronometrada.getValueAt(i - 2, 2));
                    if (Math.abs(diff) > ((Long)tblMedicaoCronometrada.getValueAt(i-2, 2)) /
100.0) {
                        y[i] -= (diff / ((Double)tblMedicaoCronometrada.getValueAt(i-2,
1)))/2;
                        if (i == 2) {
                            y[0] = y[1] = y[2];
                        } else if (i == tblMedicaoCronometrada.getRowCount()) {
                            y[tblMedicaoCronometrada.getRowCount() + 1] =
y[tblMedicaoCronometrada.getRowCount() + 2] = y[tblMedicaoCronometrada.getRowCount()];
                        }
                        curva = new DoubleCubicSplineInterpolator(x, y);
                        flag = true;
                    }
                }
            } while (flag);
            final XYSeries series1 = new XYSeries("Lambda (λ)");
            double intervalo = (x[tblMedicaoCronometrada.getRowCount()] - x[1]) / 700;
            for (double i = x[0]; i <= x[tblMedicaoCronometrada.getRowCount()]; i +=
intervalo) {
                aux = curva.map(i);
                if (aux > 0) {
                    series1.add(i - x[0], aux);
                } else {
                    series1.add(i - x[0], 0);
                }

            }
            final XYSeriesCollection dataset = new XYSeriesCollection();
            dataset.addSeries(series1);
            final JFreeChart chart = ChartFactory.createXYLineChart("Variação da taxa de
chegada", "Tempo (em minutos)", "Clientes / minuto", dataset, PlotOrientation.VERTICAL, true,
true, false);
            chart.setBackgroundPaint(pnlGrafico.getBackground());
            ChartPanel myChartPanel = new ChartPanel(chart, true);
            pnlGrafico.removeAll();
            pnlGrafico.add(myChartPanel, BorderLayout.CENTER);
            pnlGrafico.validate();
        }
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
```

```
                                                + "O formato correto para o horário inicial é
hh:mm. Ex: 10:00", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}
```

Não se deve esquecer de adicionar ao projeto as bibliotecas utilizadas. Caso contrário, as respectivas classes não serão encontradas e o código apresentará erros. Para adicionar uma biblioteca a um projeto do NetBeans basta clicar com o botão direito do mouse no mesmo e então clicar em “Propriedades”. Uma janela será aberta com as propriedades do projeto, com uma categoria chamada “Bibliotecas”. Nesta categoria basta clicar em “Adicionar JAR/pasta” para adicionar os arquivos das bibliotecas desejadas.

## 3.2 TAXAS DE ENTRADA E SAÍDA

Durante a simulação será necessário, a todo o momento, decidir quantos clientes entrarão e sairão do sistema. Para isso serão utilizados os dados das taxas de entrada e saída. Para facilitar esta tarefa algumas funcionalidades devem ser adicionadas ao código.

### 3.2.1 DISTRIBUIÇÕES

Ao se utilizar tabelas de frequência, entretanto, vários valores são inseridos, e um único valor deve ser utilizado.

Na etapa de desenvolvimento da interface com o usuário foram adicionadas caixas de combinação para permitir ao usuário a escolha da distribuição matemática utilizada ao interpretar os dados inseridos nas tabelas de frequência.

O valor utilizado, quando baseado em uma tabela de frequência, será gerado de forma aleatória, mas seguindo uma distribuição que regule as probabilidades para que os valores sorteados representem com maior fidelidade os dados inseridos. Será permitido ao usuário a escolha entre as opções “Observada” e “Poisson[6]”.

Na distribuição observada, o valor a ser utilizado como taxa será um dos valores presentes na tabela, obtido através de um sorteio, porém com maior

probabilidade de escolha para os valores com maior frequência. Desta forma os valores sorteados serão condizentes com os observados, porém nunca serão sorteados valores diferentes destes: sempre será sorteado um valor da lista.

Já com a segunda opção, os valores serão sorteados utilizando uma distribuição matemática conhecida como distribuição de Poisson (FOGLIATTI e MATTOS, 2006). Esta distribuição recebe um número esperado de ocorrências para um evento (chegada de clientes no sistema, por exemplo), e gera aleatoriamente um número de ocorrências para o evento, com a probabilidade maior para números próximos ao número esperado de ocorrências.

Serão criados, então, dois métodos. O primeiro método retornará a frequência esperada de determinado número de ocorrências, recebido por parâmetro, na distribuição com o número esperado de ocorrências também informado em um parâmetro. O segundo método será utilizado para fazer o sorteio de um número de ocorrências na distribuição de Poisson com número esperado de ocorrências igual ao parâmetro recebido.

A fórmula matemática para encontrar a frequência esperada do número de ocorrências de um evento em uma distribuição de Poisson é:

$$F_e = (e^{-\lambda}\lambda^k) / k!,$$

sendo $F_e$ a frequência esperada, $\lambda$ o número esperado de ocorrências e k o número de ocorrências a partir do qual será calculada a frequência.

O código para tal método ficará como descrito a seguir.

```
private double frequenciaEsperadaPoisson(int numero, double numeroEsperado) {
    int fatorial = 1;
    for (int i = 2; i <= numero; i++) {
        fatorial *= i;
    }
    return (Math.pow(Math.E, -numeroEsperado) * Math.pow(numeroEsperado, numero)) / fatorial;
}
```

[6] Siméon Denis Poisson, nascido em 21 de junho de 1781, foi um matemático e físico francês. Criador da distribuição de Poisson, Faleceu em 25 de abril de 1840.

Para sortear um número de ocorrências para um evento usando uma distribuição de Poisson alguns procedimentos deverão ser seguidos. Será criado um contador, que iniciará em zero, e um número real aleatório entre 0 e 1. Se o número aleatório for maior que $e^{-\lambda}$, onde λ representa o número esperado de ocorrências, o contador será incrementado e o número aleatório será multiplicado por outro número aleatório entre 0 e 1.

Assim que o resultado das multiplicações entre números aleatórios se tornar menor ou igual a $e^{-\lambda}$ o valor presente no contador será utilizado como o número de ocorrências sorteado.

O método a seguir implementa a funcionalidade descrita.

```
private int numeroSorteadoPoisson(double numeroEsperado) {
    double comparacao = Math.exp(- numeroEsperado);
    double multiplicador = Math.random();
    int sorteado;
    for (sorteado = 0; multiplicador > comparacao; sorteado++) {
        multiplicador *= Math.random();
    }
    return sorteado;
}
```

Os números esperados de ocorrências para a chegada e para a saída de clientes deverão ser armazenados em atributos, no caso das tabelas de frequência, para evitar a repetição do cálculo empregado para achá-los. O início da classe ficará como o código a seguir.

```
public class SimulacaoView extends FrameView {

    private DoubleCubicSplineInterpolator curva;
    private double                        numeroEsperadoEntrada;
    private double                        numeroEsperadoSaida;

    public SimulacaoView(SingleFrameApplication app) {
        super(app);

        initComponents();
        cmbEntradaActionPerformed(null);
        cmbSaidaActionPerformed(null);
        txtNumeroClientesSistema.setVisible(false);
        txtTextoClientesSistema.setVisible(false);
        txtNumeroClientesAtendidos.setVisible(false);
        txtTextoClientesAtendidos.setVisible(false);
        curva = null;
        numeroEsperadoEntrada = 0;
        numeroEsperadoSaida = 0;
    }
    // [...]
```

Por motivos de compatibilidade futura, serão criados dois métodos responsáveis pelos cálculos dos dois números, que retornarão o total de ocorrências observadas. O código para os métodos ficará da forma seguinte.

```
private double frequenciaTotalObservadaEntrada() {
    double somaValores=0;
    double somaFrequencias=0;
    for (int j = 0; j < tblTabelaFrequenciaEntrada.getRowCount() - 1; j++) {
        somaValores += Double.parseDouble(tblTabelaFrequenciaEntrada.getValueAt(j,
0).toString()) * Double.parseDouble(tblTabelaFrequenciaEntrada.getValueAt(j, 1).toString());
        somaFrequencias += Double.parseDouble(tblTabelaFrequenciaEntrada.getValueAt(j,
1).toString());
    }
    numeroEsperadoEntrada = somaValores / somaFrequencias;
    return somaFrequencias;
}

private double frequenciaTotalObservadaSaida() {
    double somaValores=0;
    double somaFrequencias=0;
    for (int j = 0; j < tblTabelaFrequenciaSaida.getRowCount() - 1; j++) {
        somaValores += Double.parseDouble(tblTabelaFrequenciaSaida.getValueAt(j,
0).toString()) * Double.parseDouble(tblTabelaFrequenciaSaida.getValueAt(j, 1).toString());
        somaFrequencias += Double.parseDouble(tblTabelaFrequenciaSaida.getValueAt(j,
1).toString());
    }
    numeroEsperadoSaida = somaValores / somaFrequencias;
    return somaFrequencias;
}
```

Agora que as funções referentes ao sorteio estão prontas, o nível de significância da distribuição com relação aos dados observados quando a distribuição escolhida for Poisson deverá ser calculado.

Será criado um método para isto, para utilizarmos sempre que este cálculo for necessário. Será calculado o nível de significância utilizando uma distribuição conhecida como Qui-Quadrado (FOGLIATTI e MATTOS, 2006), utilizando uma classe chamada ChiSqrDistribution, específica para este tipo de cálculo, da biblioteca JSci (JSci - A science API for Java), como observado no código a seguir.

```
private int significanciaPoisson(JTable tblTabelaFrequencia, double frequenciaTotalObservada,
double numeroEsperado) {
    if (tblTabelaFrequencia.getRowCount() > 1) {
        boolean acimaDeCinco = false;
        boolean passou = false;
        int maximo =
Integer.parseInt(tblTabelaFrequencia.getValueAt(tblTabelaFrequencia.getRowCount() - 2,
0).toString());
        double somaFrequenciaObservada = 0;
        double somaFrequenciaEsperada = 0;
        double quiQuadrado = 0;
        int j = 0;
        for (int i = 0; i < maximo && (!passou || acimaDeCinco); i++) {
            if (Integer.parseInt(tblTabelaFrequencia.getValueAt(j, 0).toString()) == i) {
                somaFrequenciaObservada +=
Double.parseDouble(tblTabelaFrequencia.getValueAt(j, 1).toString());
                j++;
            }
```

```
            double frequenciaEsperada = frequenciaEsperadaPoisson(i, numeroEsperado) *
frequenciaTotalObservada;
            somaFrequenciaEsperada += frequenciaEsperada;
            acimaDeCinco = frequenciaEsperada >= 5;
            if (acimaDeCinco) {
                passou = true;
                quiQuadrado += Math.pow(somaFrequenciaEsperada - somaFrequenciaObservada, 2) /
somaFrequenciaEsperada;
                somaFrequenciaEsperada = 0;
                somaFrequenciaObservada = 0;
            }
        }
        if (!passou) {
            quiQuadrado += Math.pow(somaFrequenciaEsperada - somaFrequenciaObservada, 2) /
somaFrequenciaEsperada;
        }
        return (int) (100 * (1 - new ChiSqrDistribution(5).cumulative(quiQuadrado)));
    } else {
        return 0;
    }
}
```

Para evitar a repetição do código será criado um método responsável por exibir a significância na aba de entrada, e outro na aba de saída, com os códigos apresentados a seguir.

```
private void exibirSignificanciaEntrada() {
    final int DISTRIBUICAO_OBSERVADA = 0;
    final int DISTRIBUICAO_POISSON = 1;

    final int TABELA_FREQUENCIA = 1;

    switch (cmbDistribuicaoEntrada.getSelectedIndex()) {
        case DISTRIBUICAO_POISSON:
            cmbDistribuicaoEntrada.removeAllItems();
            if (cmbEntrada.getSelectedIndex() == TABELA_FREQUENCIA) {
                double frequenciaTotalObservada = frequenciaTotalObservadaEntrada();
                cmbDistribuicaoEntrada.addItem("Observada");
                cmbDistribuicaoEntrada.addItem("Poisson (significância: " +
significanciaPoisson(tblTabelaFrequenciaEntrada, frequenciaTotalObservada,
numeroEsperadoSaida) + "%)");
            } else {
                cmbDistribuicaoEntrada.addItem("Observada");
                cmbDistribuicaoEntrada.addItem("Poisson");
            }
            cmbDistribuicaoEntrada.setSelectedIndex(DISTRIBUICAO_POISSON);
            break;
        case DISTRIBUICAO_OBSERVADA:
            cmbDistribuicaoEntrada.removeAllItems();
            cmbDistribuicaoEntrada.addItem("Observada");
            cmbDistribuicaoEntrada.addItem("Poisson");
            cmbDistribuicaoEntrada.setSelectedIndex(DISTRIBUICAO_OBSERVADA);
            break;
    }
}

private void exibirSignificanciaSaida() {
    final int DISTRIBUICAO_OBSERVADA = 0;
    final int DISTRIBUICAO_POISSON = 1;

    final int TABELA_FREQUENCIA = 1;

    switch (cmbDistribuicaoSaida.getSelectedIndex()) {
        case DISTRIBUICAO_POISSON:
            cmbDistribuicaoSaida.removeAllItems();
            if (cmbSaida.getSelectedIndex() == TABELA_FREQUENCIA) {
                double frequenciaTotalObservada = frequenciaTotalObservadaSaida();
                cmbDistribuicaoSaida.addItem("Observada");
```

```
                cmbDistribuicaoSaida.addItem("Poisson (significância: " +
significanciaPoisson(tblTabelaFrequenciaSaida, frequenciaTotalObservada, numeroEsperadoSaida)
+ "%)");
            } else {
                cmbDistribuicaoSaida.addItem("Observada");
                cmbDistribuicaoSaida.addItem("Poisson");
            }
            cmbDistribuicaoSaida.setSelectedIndex(DISTRIBUICAO_POISSON);
            break;
        case DISTRIBUICAO_OBSERVADA:
            cmbDistribuicaoSaida.removeAllItems();
            cmbDistribuicaoSaida.addItem("Observada");
            cmbDistribuicaoSaida.addItem("Poisson");
            cmbDistribuicaoSaida.setSelectedIndex(DISTRIBUICAO_OBSERVADA);
            break;
    }
}
```

Estes métodos serão chamados sempre que uma mudança na significância possa ter ocorrido. A significância pode mudar ao adicionar linhas na tabela, portanto os eventos “propertyChange” de ambas as tabelas de frequência deverão alterados para que seus códigos fiquem como os seguintes.

```
private void tblTabelaFrequenciaEntradaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaFrequenciaEntrada()) {
        for (int linha1 = 0; linha1 < tblTabelaFrequenciaEntrada.getRowCount() - 1; linha1++)
{
            for (int linha2 = linha1 + 1; linha2 < tblTabelaFrequenciaEntrada.getRowCount() -
1; linha2++) {
                long valor1 = (Long) tblTabelaFrequenciaEntrada.getValueAt(linha1, 0);
                long valor2 = (Long) tblTabelaFrequenciaEntrada.getValueAt(linha2, 0);
                if (valor2 < valor1) {
                    ((DefaultTableModel)tblTabelaFrequenciaEntrada.getModel()).moveRow(linha1,
linha1, linha2);
                }
            }
        }
        exibirSignificanciaEntrada();
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "Os valores inseridos devem ser maiores que
zero.", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}

// [...]

private void tblTabelaFrequenciaSaidaPropertyChange(java.beans.PropertyChangeEvent evt) {
    if (validaTabelaFrequenciaSaida()) {
        for (int linha1 = 0; linha1 < tblTabelaFrequenciaSaida.getRowCount() - 1; linha1++) {
            for (int linha2 = linha1 + 1; linha2 < tblTabelaFrequenciaSaida.getRowCount() - 1;
linha2++) {
                long valor1 = (Long) tblTabelaFrequenciaSaida.getValueAt(linha1, 0);
                long valor2 = (Long) tblTabelaFrequenciaSaida.getValueAt(linha2, 0);
                if (valor2 < valor1) {
                    ((DefaultTableModel)tblTabelaFrequenciaSaida.getModel()).moveRow(linha1,
linha1, linha2);
                }
            }
        }
        exibirSignificanciaSaida();
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "Os valores inseridos devem ser maiores que
zero.", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
    }
}
```

O texto da caixa de combinação também mudará se o usuário alternar a forma de inserção de dados, portanto também serão alterados os eventos “actionPerformed” das caixas de combinação relacionadas, que ficarão como mostrados a seguir.

```
private void cmbEntradaActionPerformed(java.awt.event.ActionEvent evt) {
    final int MEDICAO_CRONOMETRADA = 0;
    final int TABELA_DE_FREQUENCIA = 1;
    final int VALOR_FIXO = 2;

    exibirSignificanciaEntrada();
    switch (cmbEntrada.getSelectedIndex()) {
        case MEDICAO_CRONOMETRADA:

            pnlMedicaoCronometrada.setVisible(true);

            pnlTabelaFrequenciaEntrada.setVisible(false);
            pnlValorFixoEntrada.setVisible(false);
            txtDuracaoIntervaloEntrada.setVisible(false);
            nmrDuracaoIntervaloEntrada.setVisible(false);
            break;
        case TABELA_DE_FREQUENCIA:
            exibirSignificanciaEntrada();

            pnlTabelaFrequenciaEntrada.setVisible(true);
            txtDuracaoIntervaloEntrada.setVisible(true);
            nmrDuracaoIntervaloEntrada.setVisible(true);

            pnlMedicaoCronometrada.setVisible(false);
            pnlValorFixoEntrada.setVisible(false);
            break;
        case VALOR_FIXO:
            pnlValorFixoEntrada.setVisible(true);

            pnlMedicaoCronometrada.setVisible(false);
            pnlTabelaFrequenciaEntrada.setVisible(false);
            txtDuracaoIntervaloEntrada.setVisible(false);
            nmrDuracaoIntervaloEntrada.setVisible(false);
            break;
    }
}

// [...]

private void cmbSaidaActionPerformed(java.awt.event.ActionEvent evt) {
    final int TABELA_DE_FREQUENCIA = 0;
    final int VALOR_FIXO = 1;

    exibirSignificanciaSaida();
    switch (cmbSaida.getSelectedIndex()) {
        case TABELA_DE_FREQUENCIA:
            pnlTabelaFrequenciaSaida.setVisible(true);
            txtDuracaoIntervaloSaida.setVisible(true);
            nmrDuracaoIntervaloSaida.setVisible(true);

            pnlValorFixoSaida.setVisible(false);
            break;
        case VALOR_FIXO:
            pnlValorFixoSaida.setVisible(true);

            pnlTabelaFrequenciaSaida.setVisible(false);
            txtDuracaoIntervaloSaida.setVisible(false);
            nmrDuracaoIntervaloSaida.setVisible(false);
            break;
    }
}
```

Além destas alterações, serão adicionados eventos “actionPerformed” nas caixas de combinação de distribuição, tanto na aba de entrada quanto na de saída. Os respectivos códigos ficarão como segue.

```
private void cmbDistribuicaoEntradaActionPerformed(java.awt.event.ActionEvent evt) {
    exibirSignificanciaEntrada();
}

private void cmbDistribuicaoSaidaActionPerformed(java.awt.event.ActionEvent evt) {
    exibirSignificanciaSaida();
}
```

As caixas de combinação foram nomeadas “cmbDistribuicaoEntrada” e “cmbDistribuicaoSaida”.

### 3.2.2 MÉTODOS AUXILIARES

Para facilitar a obtenção das taxas de entrada e de saída durante a simulação, serão criados dois métodos auxiliares responsáveis por retornar os mesmos.

Os métodos verificarão qual forma de inserção de dados está selecionada e retornarão o valor correspondente. Caso a forma selecionada seja a tabela de frequência, e a distribuição selecionada seja “Poisson”, serão usados os métodos criados anteriormente para obter o valor correto.

O código para tais métodos ficará como apresentado a seguir.

```
public double pegarTaxaEntrada(double tempoAtualSimulacao) {
    final int MEDICAO_CRONOMETRADA = 0;
    final int TABELA_FREQUENCIA = 1;

    final int DISTRIBUICAO_OBSERVADA = 0;
    final int DISTRIBUICAO_POISSON = 1;

    switch (cmbEntrada.getSelectedIndex()) {
        case MEDICAO_CRONOMETRADA:
            double lambda = curva.map(tempoAtualSimulacao);
            if (lambda < 0) {
                lambda = 0;
            }
            if (cmbDistribuicaoEntrada.getSelectedIndex() == DISTRIBUICAO_POISSON) {
                lambda = numeroSorteadoPoisson(lambda);
            }
            return lambda;
        case TABELA_FREQUENCIA:
            switch (cmbDistribuicaoEntrada.getSelectedIndex()) {
                case DISTRIBUICAO_OBSERVADA:
                    int cont = 0;
                    for (int j = 0; j < tblTabelaFrequenciaEntrada.getRowCount() - 1; j++) {
                        cont += (Long) tblTabelaFrequenciaEntrada.getValueAt(j, 1);
                    }
                    long sort = Math.round(Math.random() * (cont + 1)) + 1;
                    int j = 0;
```

```
                    cont = 0;
                    while (cont < sort) {
                        cont += (Long) tblTabelaFrequenciaEntrada.getValueAt(j, 1);
                        j++;
                    }
                    return (Long) tblTabelaFrequenciaEntrada.getValueAt(j - 1, 0) /
Double.parseDouble(nmrDuracaoIntervaloEntrada.getText());
                default:
                    return numeroSorteadoPoisson(numeroEsperadoEntrada) /
Double.parseDouble(nmrDuracaoIntervaloEntrada.getText());
            }
        default:
            if (cmbDistribuicaoEntrada.getSelectedIndex() == DISTRIBUICAO_OBSERVADA) {
                return Double.parseDouble(nmrValorFixoEntrada.getText());
            } else {
                return (double)
numeroSorteadoPoisson(Double.parseDouble(nmrValorFixoEntrada.getText()));
            }
    }
}

// [...]

public double pegarTaxaSaida() {
    final int TABELA_FREQUENCIA = 0;

    final int DISTRIBUICAO_OBSERVADA = 0;

    switch (cmbSaida.getSelectedIndex()) {
        case TABELA_FREQUENCIA:
            switch (cmbDistribuicaoSaida.getSelectedIndex()) {
                case DISTRIBUICAO_OBSERVADA:
                    int cont = 0;
                    for (int j = 0; j < tblTabelaFrequenciaSaida.getRowCount() - 1; j++) {
                        cont += (Long) tblTabelaFrequenciaSaida.getValueAt(j, 1);
                    }
                    long sort = Math.round(Math.random() * (cont + 1)) + 1;
                    int j = 0;
                    cont = 0;
                    while (cont < sort) {
                        cont += (Long) tblTabelaFrequenciaSaida.getValueAt(j, 1);
                        j++;
                    }
                    return (Long) tblTabelaFrequenciaSaida.getValueAt(j - 1, 0) /
Double.parseDouble(nmrDuracaoIntervaloSaida.getText());
                default:
                    return numeroSorteadoPoisson(numeroEsperadoSaida) /
Double.parseDouble(nmrDuracaoIntervaloSaida.getText());
            }
        default:
            if (cmbDistribuicaoSaida.getSelectedIndex() == DISTRIBUICAO_OBSERVADA) {
                return Double.parseDouble(nmrValorFixoSaida.getText());
            } else {
                return (double)
numeroSorteadoPoisson(Double.parseDouble(nmrValorFixoSaida.getText()));
            }
    }
}
```

## 3.3 SIMULAÇÃO

Por fim, será feita a codificação do que acontece após o usuário apertar o botão “Simular”.

### 3.3.1 ATUALIZAÇÃO

A simulação acontecerá em tempo real, portanto deverá ser atualizada constantemente para que seja fluída, e o andamento da simulação deverá ser relativo ao tempo passado. A taxa de atualização escolhida foi 60 atualizações por segundo.

As atualizações estarão executando constantemente, portanto irão travar a interface com o usuário se rodarem na mesma linha de execução. As atualizações serão feitas, então, em uma “Thread”, para que sejam executadas em uma linha de execução nova.

Para isso será criada uma nova classe que herdará da classe “Thread” do Java. Esta classe terá um método chamado “run”, que será executado em uma nova linha de execução quando o método “start” da classe pai for chamado.

A nova classe deverá conter atributos para armazenar referências aos objetos da tela e do painel gráfico, pois terá contato direto com ambos. Também são necessários atributos para armazenar o tempo atual e final da simulação, o tempo real restante, um valor booleano indicando se a simulação está pausada, a velocidade da simulação, um vetor armazenando a ordem das filas, um vetor armazenando listas com os tempos de entrada de todos os clientes no sistema, um vetor armazenando o tempo em que os clientes entraram para ser atendidos em cada fila, um valor booleano que indica se existe apenas uma fila, um vetor com valores auxiliares de saída, que armazenará a taxa de saída que sobrou após os clientes do passo atual terem saído do sistema (utilizado no próximo passo), um contador indicando quanto tempo já passou desde o início da simulação e um valor booleano que indica se as informações da simulação estão sendo atualizadas.

Também serão necessários atributos para armazenar as informações relevantes cujos valores serão inseridos nos campos da tela. Tais atributos deverão ser públicos, pois serão acessados por um método da classe da tela.

No método “run” será criado um laço de repetição que continuará até que a simulação termine. Dentro deste laço serão controladas as atualizações, limitando a

um máximo de 60 atualizações por segundo, e pausando a “Thread” caso o atributo associado seja verdadeiro.

Serão usados os dois métodos criados anteriormente para obter valores para as taxas de entrada e saída, e então os clientes serão adicionados e removidos de acordo com os valores obtidos.

Após as alterações nas filas, as médias de todas as variáveis de informações serão calculadas.

Além do método “run”, será criado também um método para alterar o tempo restante, que ajustará o atributo de velocidade de acordo com o tempo restante estabelecido no parâmetro.

Mais um método será criado nesta classe para tratar a alteração entre fila única e múltiplas filas, ajustando todos os atributos dependentes.

Por fim, outro método será criado, e tratará da alteração no número de servidores do sistema, transferindo clientes que estavam em filas removidas para as restantes se necessário.

O código da classe ficará, então, como mostrado a seguir.

```
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Atualizacao extends Thread{
    PainelGrafico painel;
    SimulacaoView janela;
    boolean       pausado;
    double        tempoAtualSimulacao;
    int           tempoFinalSimulacao;
    int           tempoRealRestante;
    double        velocidade;
    int           ordemFilas[];
    List<Double>  tempoEntrada[];
    double        tempoAtendimento[];
    boolean       filaUnica;
    double        auxiliarSaida[];
    double        contadorTempoSimulacao;
    boolean       atualizandoInformacoes;
    double        mediaClientesSistema;
    double        mediaVarianciaClientesSistema;
    double        mediaRo;
    double        mediaProbabilidadeSistemaVazio;
    double        mediaProbabilidadeEnfileiramento;
    double        mediaClientesFilas;
```

```
    double        mediaVarianciaClientesFilas;
    double        mediaTempoSistema;
    double        mediaVarianciaTempoSistema;
    double        mediaTempoFila;
    double        mediaVarianciaTempoFila;
    boolean       estabilidadeMomentanea;
    boolean       estabilidadeMedia;

    public Atualizacao(PainelGrafico pg, SimulacaoView j) {
        painel = pg;
        janela = j;
        filaUnica = true;
        tempoFinalSimulacao = 660;
        tempoRealRestante = 60;
        atualizandoInformacoes = false;
        auxiliarSaida = new double[1];
        ordemFilas = new int[1];
        tempoEntrada = new List[1];
        tempoEntrada[0] = new ArrayList<Double>();
        tempoAtendimento = new double[1];
        painel.numeroClientesFila = new int[1];
        painel.numeroClientesFila[0] = 0;
    }

    public void run() {
        long tempoRealAnterior = System.currentTimeMillis();
        double auxiliarEntrada = 0;
        double tempoRestanteSimulacao = tempoFinalSimulacao - tempoAtualSimulacao;
        if (this.tempoRealRestante > 0) {
            velocidade = tempoRestanteSimulacao / (17 * ((int)(1000 * this.tempoRealRestante)
/ 17));
        } else {
            velocidade = tempoRestanteSimulacao / 17;
        }
        while (tempoAtualSimulacao < tempoFinalSimulacao) {
            if (pausado) {
                 try {
                     tempoRealAnterior -= System.currentTimeMillis();
                     synchronized (this) {
                         this.wait();
                     }
                     tempoRealAnterior += System.currentTimeMillis();
                 } catch (InterruptedException ex) {
                     Logger.getLogger(Atualizacao.class.getName()).log(Level.SEVERE, null,
ex);
                 }
            }
            int numeroClientesEntrando = 0;
            long tempoRealPassado = System.currentTimeMillis() - tempoRealAnterior;
            if (tempoRealPassado < 17) {
                try {
                    sleep(17 - tempoRealPassado);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Atualizacao.class.getName()).log(Level.SEVERE, null, ex);
                }
                tempoRealPassado = 17;
            }
            double tempoPassadoSimulacao = tempoRealPassado * velocidade;
            tempoAtualSimulacao += tempoPassadoSimulacao;
            tempoRealAnterior = System.currentTimeMillis();
            double taxaEntrada = janela.pegarTaxaEntrada(tempoAtualSimulacao);
            auxiliarEntrada += taxaEntrada * tempoRealPassado * velocidade;
            while (auxiliarEntrada >= 1) {
                int i = 0;
                while ((i < ordemFilas.length - 1) && (ordemFilas[i + 1] == ordemFilas[0])) {
                    i++;
                }
                if (i > 0) {
                    int posicao = (int)(Math.random() * (i + 1));
                    if (posicao == i) {
                        posicao--;
                    }
                    int aux = ordemFilas[posicao];
                    ordemFilas[posicao] = ordemFilas[i - 1];
                    ordemFilas[i - 1] = aux;
                    painel.numeroClientesFila[aux]++;
                    tempoEntrada[aux].add(tempoAtualSimulacao);
```

```
                } else {
                    painel.numeroClientesFila[ordemFilas[0]]++;
                    tempoEntrada[ordemFilas[0]].add(tempoAtualSimulacao);
                }
                auxiliarEntrada--;
                numeroClientesEntrando++;
            }
            double miMedio = 0;
            int numeroClientesSaindo = 0;
            double tempoMedioSistema = 0;
            double tempoMedioFila = 0;
            for (int i = 0; i < painel.numeroServidores; i++) {
                double mi = janela.pegarTaxaSaida();
                miMedio += mi;
                auxiliarSaida[i] += mi * tempoRealPassado * velocidade;
                double auxTempoMedio = 0;
                if (filaUnica) {
                    if (painel.numeroClientesFila[0] < (int) auxiliarSaida[i]) {
                        while (!tempoEntrada[0].isEmpty()) {
                            auxTempoMedio += tempoAtualSimulacao - tempoEntrada[0].get(0);
                            tempoEntrada[0].remove(0);
                        }
                        tempoMedioSistema += auxTempoMedio;
                        numeroClientesSaindo += painel.numeroClientesFila[0];
                        painel.numeroClientesFila[0] = 0;
                    } else {
                        for (int j = 0; j < (int) auxiliarSaida[i]; j++) {
                            auxTempoMedio += tempoAtualSimulacao - tempoEntrada[0].get(0);
                            tempoEntrada[0].remove(0);
                        }
                        tempoMedioSistema += auxTempoMedio;
                        numeroClientesSaindo += (int) auxiliarSaida[i];
                        painel.numeroClientesFila[0] -= (int) auxiliarSaida[i];
                    }
                    tempoMedioFila += auxTempoMedio - (tempoAtualSimulacao -
tempoAtendimento[0]);
                    tempoAtendimento[0] = tempoAtualSimulacao;
                } else {
                    if (painel.numeroClientesFila[i] < (int) auxiliarSaida[i]) {
                        while (!tempoEntrada[i].isEmpty()) {
                            auxTempoMedio += tempoAtualSimulacao - tempoEntrada[i].get(0);
                            tempoEntrada[i].remove(0);
                        }
                        tempoMedioSistema += auxTempoMedio;
                        numeroClientesSaindo += painel.numeroClientesFila[i];
                        painel.numeroClientesFila[i] = 0;
                    } else {
                        for (int j = 0; j < (int) auxiliarSaida[i]; j++) {
                            auxTempoMedio += tempoAtualSimulacao - tempoEntrada[i].get(0);
                            tempoEntrada[i].remove(0);
                        }
                        tempoMedioSistema += auxTempoMedio;
                        numeroClientesSaindo += (int) auxiliarSaida[i];
                        painel.numeroClientesFila[i] -= (int) auxiliarSaida[i];
                    }
                    tempoMedioFila += auxTempoMedio - (tempoAtualSimulacao -
tempoAtendimento[i]);
                    tempoAtendimento[i] = tempoAtualSimulacao;
                    int j = 0;
                    while ((j < ordemFilas.length - 1) && (ordemFilas[j] != i)) {
                        j++;
                    }
                    while ((j < ordemFilas.length - 1) &&
(painel.numeroClientesFila[ordemFilas[j + 1]] < painel.numeroClientesFila[i])) {
                        ordemFilas[j] = ordemFilas[j + 1];
                        j++;
                        ordemFilas[j] = i;
                    }
                }
                auxiliarSaida[i] = auxiliarSaida[i] - (int) auxiliarSaida[i];
            }
            painel.tamanhoMaiorFila = painel.numeroClientesFila[ordemFilas[ordemFilas.length -
1]];
            if (numeroClientesSaindo > 0) {
                tempoMedioSistema /= numeroClientesSaindo;
                tempoMedioFila /= numeroClientesSaindo;
            }
```

```
            miMedio = miMedio / painel.numeroServidores;
            double ro;
            if (miMedio > 0) {
                ro = taxaEntrada / (miMedio * painel.numeroServidores);
            } else {
                ro = 1;
            }
            estabilidadeMomentanea = (ro <= 1);
            mediaRo = (mediaRo * contadorTempoSimulacao + ro * tempoPassadoSimulacao) /
(contadorTempoSimulacao + tempoPassadoSimulacao);
            estabilidadeMedia = (mediaRo <= 1);
            int numeroClientes = painel.numeroClientesFila[0];
            int numeroClientesFilas = painel.numeroClientesFila[0];
            if (filaUnica) {
                if (numeroClientesFilas > painel.numeroServidores) {
                    numeroClientesFilas -= painel.numeroServidores;
                } else {
                    numeroClientesFilas = 0;
                }
            } else {
                if (numeroClientesFilas > 0) {
                    numeroClientesFilas--;
                }
            }
            for (int i = 1; i < painel.numeroClientesFila.length; i++) {
                numeroClientes += painel.numeroClientesFila[i];
                numeroClientesFilas += painel.numeroClientesFila[i];
                if (painel.numeroClientesFila[i] > 0) {
                    numeroClientesFilas--;
                }
            }
            mediaClientesSistema= (mediaClientesSistema * contadorTempoSimulacao +
numeroClientes * tempoPassadoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
            double auxiliarVariancia = Math.pow(numeroClientes - mediaClientesSistema, 2);
            mediaVarianciaClientesSistema = (mediaVarianciaClientesSistema *
contadorTempoSimulacao + auxiliarVariancia * tempoPassadoSimulacao) / (contadorTempoSimulacao
+ tempoPassadoSimulacao);
            mediaClientesFilas = (mediaClientesFilas * contadorTempoSimulacao +
numeroClientesFilas * tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            mediaClientesFilas = (mediaClientesFilas * contadorTempoSimulacao +
numeroClientesFilas * tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            auxiliarVariancia = Math.pow(numeroClientesFilas - mediaClientesFilas, 2);
            mediaVarianciaClientesFilas = (mediaVarianciaClientesFilas *
contadorTempoSimulacao + auxiliarVariancia*tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            if (numeroClientes == 0) {
                 mediaProbabilidadeSistemaVazio = (mediaProbabilidadeSistemaVazio *
contadorTempoSimulacao + tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            }else{
                mediaProbabilidadeSistemaVazio = (mediaProbabilidadeSistemaVazio *
contadorTempoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
            }
            boolean enfileirado = false;
            for (int i = 0; (i < painel.numeroClientesFila.length) && (!enfileirado); i++) {
                if (painel.numeroClientesFila[i] > 0) {
                    enfileirado = true;
                }
            }
            if(enfileirado){
                 mediaProbabilidadeEnfileiramento = (mediaProbabilidadeEnfileiramento *
contadorTempoSimulacao + tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            }else{
                mediaProbabilidadeEnfileiramento = (mediaProbabilidadeEnfileiramento *
contadorTempoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
            }
            mediaTempoSistema = (mediaTempoSistema * contadorTempoSimulacao +
tempoMedioSistema * tempoPassadoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
            auxiliarVariancia = Math.pow(tempoMedioSistema - mediaTempoSistema, 2);
            mediaVarianciaTempoSistema = (mediaVarianciaTempoSistema * contadorTempoSimulacao
+ auxiliarVariancia * tempoPassadoSimulacao) / (contadorTempoSimulacao +
tempoPassadoSimulacao);
            mediaTempoFila = (mediaTempoFila * contadorTempoSimulacao + tempoMedioFila *
tempoPassadoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
```

```
            auxiliarVariancia = Math.pow(tempoMedioFila - mediaTempoFila, 2);
            mediaVarianciaTempoFila = (mediaVarianciaTempoFila * contadorTempoSimulacao +
auxiliarVariancia * tempoPassadoSimulacao) / (contadorTempoSimulacao + tempoPassadoSimulacao);
            janela.adicionarPontoGraficoClientesSistema(tempoAtualSimulacao, numeroClientes);
            if (numeroClientesSaindo > 0) {
                janela.adicionarPontoGraficoTempoSistema(tempoAtualSimulacao,
tempoMedioSistema);
            }
            contadorTempoSimulacao += tempoPassadoSimulacao;
            atualizandoInformacoes = true;
            janela.atualizarInformacoes(numeroClientesEntrando, numeroClientesSaindo);
            atualizandoInformacoes = false;
            painel.revalidate();
            painel.repaint();
        }
        janela.finalizarSimulacao();
    }

    public void alterarTempoRestante(int tempoRestante) {
        if (!atualizandoInformacoes) {
            this.tempoRealRestante = tempoRestante;
            double tempoRestanteSimulacao = tempoFinalSimulacao - tempoAtualSimulacao;
            if (tempoRestante > 0) {
                velocidade = tempoRestanteSimulacao / (17 * ((int)(1000 *
this.tempoRealRestante) / 17));
            } else {
                velocidade = tempoRestanteSimulacao / 17;
            }
        }
    }

    public void alterarFilaUnica(boolean filaUnica) {
        if (this.filaUnica != filaUnica) {
            if (painel.numeroServidores > 1) {
                int auxiliarFila[];
                int auxiliarOrdemFilas[];
                double auxiliarTempoAtendimento[];
                List<Double> auxiliarTempoEntrada[];
                if (this.filaUnica) {
                    auxiliarFila = new int[painel.numeroServidores];
                    auxiliarOrdemFilas = new int[painel.numeroServidores];
                    auxiliarTempoAtendimento = new double[painel.numeroServidores];
                    auxiliarTempoEntrada = new List[painel.numeroServidores];
                    auxiliarFila[0] = painel.numeroClientesFila[0];
                    for (int i = 0; i < painel.numeroServidores - 1; i++) {
                        auxiliarOrdemFilas[i] = i + 1;
                    }
                    auxiliarTempoEntrada[0] = tempoEntrada[0];
                    for (int i = 1; i < painel.numeroServidores; i++) {
                        auxiliarTempoEntrada[i] = new ArrayList<Double>();
                    }
                    auxiliarTempoAtendimento[0] = tempoAtendimento[0];
                } else {
                    auxiliarFila = new int[1];
                    auxiliarOrdemFilas = new int[1];
                    auxiliarTempoAtendimento = new double[1];
                    auxiliarTempoEntrada = new List[1];
                    auxiliarFila[0] = painel.numeroClientesFila[0];
                    for (int i = 1; i < painel.numeroClientesFila.length; i++) {
                        auxiliarFila[0] += painel.numeroClientesFila[i];
                    }
                    auxiliarOrdemFilas[0] = 0;
                    auxiliarTempoEntrada[0] = tempoEntrada[0];
                    for (int i = 1; i < painel.numeroServidores; i++) {
                        auxiliarTempoEntrada[0].addAll(tempoEntrada[i]);
                    }
                    auxiliarTempoAtendimento[0] = tempoAtendimento[0];
                }
                painel.numeroClientesFila = auxiliarFila;
                ordemFilas = auxiliarOrdemFilas;
                tempoEntrada = auxiliarTempoEntrada;
                tempoAtendimento = auxiliarTempoAtendimento;
            }
            this.filaUnica = filaUnica;
        }
    }
```

```
    public void alterarNumeroServidores(int numeroServidores) {
        if (numeroServidores != painel.numeroServidores) {
            double auxiliarSaida[] = new double[numeroServidores];
            if (this.auxiliarSaida.length <= numeroServidores) {
                for (int i = 0; i < this.auxiliarSaida.length; i++) {
                    auxiliarSaida[i] = this.auxiliarSaida[i];
                }
            } else {
                for (int i = 0; i < numeroServidores; i++) {
                    auxiliarSaida[i] = this.auxiliarSaida[i];
                }
            }
            this.auxiliarSaida = auxiliarSaida;
            if (!filaUnica) {
                int auxiliarFila[] = new int[numeroServidores];
                int auxiliarOrdemFilas[] = new int[numeroServidores];
                double auxiliarTempoAtendimento[] = new double[numeroServidores];
                List<Double> auxiliarTempoEntrada[] = new List[numeroServidores];
                if (numeroServidores > painel.numeroServidores) {
                    for (int i = 0; i < painel.numeroClientesFila.length; i++) {
                        auxiliarFila[i] = painel.numeroClientesFila[i];
                    }
                    int diferenca = numeroServidores - painel.numeroServidores;
                    for (int i = 1; i <= diferenca; i++) {
                        auxiliarOrdemFilas[i - 1] = numeroServidores - i;
                    }
                    for (int i = 0; i < painel.numeroServidores; i++) {
                        auxiliarOrdemFilas[diferenca + i] = ordemFilas[i];
                    }
                    for (int i = 0; i < painel.numeroServidores; i++) {
                        auxiliarTempoEntrada[i] = tempoEntrada[i];
                    }
                    for (int i = 1; i <= diferenca; i++) {
                        auxiliarTempoEntrada[numeroServidores - i] = new ArrayList<Double>();
                    }
                    for (int i = 0; i < painel.numeroServidores; i++) {
                        auxiliarTempoAtendimento[i] = tempoAtendimento[i];
                    }
                } else {
                    int diferenca = painel.numeroServidores - numeroServidores;
                    int clientesRemovidos = 0;
                    List<Double> temposRemovidos = new ArrayList<Double>();
                    for (int i = 1; i <= diferenca; i++) {
                        clientesRemovidos += painel.numeroClientesFila[painel.numeroServidores
- i];
                        temposRemovidos.addAll(tempoEntrada[painel.numeroServidores - i]);
                    }
                    for (int i = 0; i < numeroServidores; i++) {
                        auxiliarFila[i] = painel.numeroClientesFila[i];
                    }
                    int j = 0;
                    for (int i = 0; i < ordemFilas.length; i++) {
                        if (ordemFilas[i] < numeroServidores) {
                            auxiliarOrdemFilas[j] = ordemFilas[i];
                            j++;
                        }
                    }
                    for (int i = 0; i < numeroServidores; i++) {
                        auxiliarTempoEntrada[i] = tempoEntrada[i];
                    }
                    while (clientesRemovidos >= 1) {
                        int i = 0;
                        while ((i < auxiliarOrdemFilas.length - 1) && (auxiliarOrdemFilas[i +
1] == auxiliarOrdemFilas[0])) {
                            i++;
                        }
                        if (i > 0) {
                            int posicao = (int)(Math.random() * (i + 1));
                            if (posicao == i) {
                                posicao--;
                            }
                            int aux = auxiliarOrdemFilas[posicao];
                            auxiliarOrdemFilas[posicao] = auxiliarOrdemFilas[i - 1];
                            auxiliarOrdemFilas[i - 1] = aux;
                            auxiliarFila[aux]++;
                            auxiliarTempoEntrada[aux].add(temposRemovidos.get(0));
                        } else {
```

```
                            auxiliarFila[auxiliarOrdemFilas[0]]++;
                            auxiliarTempoEntrada[0].add(temposRemovidos.get(0));
                        }
                        temposRemovidos.remove(0);
                        clientesRemovidos--;
                    }
                    for (int i = 0; i < numeroServidores; i++) {
                        auxiliarTempoAtendimento[i] = tempoAtendimento[i];
                    }
                }
                painel.numeroClientesFila = auxiliarFila;
                ordemFilas = auxiliarOrdemFilas;
                tempoEntrada = auxiliarTempoEntrada;
                tempoAtendimento = auxiliarTempoAtendimento;
            }
            painel.numeroServidores = numeroServidores;
        }
    }
}
```

Nota-se que os métodos “adicionarPontoGraficoClientesSistema”, “adicionarPontoGraficoTempoSistema” e “atualizarInformacoes”, utilizados no código acima, ainda não foram codificados. Este será, então, o próximo passo.

### 3.3.2 MÉTODOS DA TELA

Os métodos chamados “adicionarPontoGraficoClientesSistema” e “adicionarPontoGraficoTempoSistema” serão chamados para, como o nome sugere, adicionar um ponto ao gráfico correspondente. Para isto, será necessário que se adicione dois atributos na classe da tela, que guardarão objetos da classe “XYSeries”, objetos estes que conterão as séries de pontos dos referidos gráficos.

Também será necessário alterar o construtor da classe da tela para que um dos novos atributos seja iniciado com “null”, indicando que os gráficos ainda não foram criados.

O código de início da classe estará, por fim, como apresentado a seguir.

```
public class SimulacaoView extends FrameView {

    private DoubleCubicSplineInterpolator curva;
    private double                        numeroEsperadoEntrada;
    private double                        numeroEsperadoSaida;
    XYSeries                              graficoClientesSistema;
    XYSeries                              graficoTempoSistema;

    public SimulacaoView(SingleFrameApplication app) {
        super(app);

        initComponents();
        cmbEntradaActionPerformed(null);
        cmbSaidaActionPerformed(null);
        txtNumeroClientesSistema.setVisible(false);
        txtTextoClientesSistema.setVisible(false);
```

```
        txtNumeroClientesAtendidos.setVisible(false);
        txtTextoClientesAtendidos.setVisible(false);
        curva = null;
        numeroEsperadoEntrada = 0;
        numeroEsperadoSaida = 0;
        graficoClientesSistema = null;
    }

    // [...]
```

Agora que os atributos estão na classe, será possível codificar os métodos supracitados. Os respectivos códigos estão apresentados a seguir.

```
public void adicionarPontoGraficoClientesSistema(double tempoAtualSimulacao, int
numeroClientes) {
    graficoClientesSistema.add(tempoAtualSimulacao -
converteHorario(hmHorarioInicial.getText()), numeroClientes);
}

public void adicionarPontoGraficoTempoSistema(double tempoAtualSimulacao, double
tempoMedioSistema) {
    graficoTempoSistema.add(tempoAtualSimulacao - converteHorario(hmHorarioInicial.getText()),
tempoMedioSistema);
}
```

O método chamado “atualizarInformacoes” será chamado sempre que os valores dos campos de informações da simulação precisarem ser atualizados. Para que seja possível codificá-lo mais um atributo na classe da tela deverá ser criado, para guardar o objeto da classe de atualização criado anteriormente.

Também será necessário alterar o construtor para que o mesmo seja construído. O código do início da tela ficará como segue.

```
public class SimulacaoView extends FrameView {

    private DoubleCubicSplineInterpolator curva;
    private double                        numeroEsperadoEntrada;
    private double                        numeroEsperadoSaida;
    Atualizacao                           atualizacao;
    XYSeries                              graficoClientesSistema;
    XYSeries                              graficoTempoSistema;

    public SimulacaoView(SingleFrameApplication app) {
        super(app);

        initComponents();
        cmbEntradaActionPerformed(null);
        cmbSaidaActionPerformed(null);
        txtNumeroClientesSistema.setVisible(false);
        txtTextoClientesSistema.setVisible(false);
        txtNumeroClientesAtendidos.setVisible(false);
        txtTextoClientesAtendidos.setVisible(false);
        curva = null;
        numeroEsperadoEntrada = 0;
        numeroEsperadoSaida = 0;
        graficoClientesSistema = null;
        atualizacao = new Atualizacao((PainelGrafico) pnlPainelGrafico, this);
    }

    // [...]
```

Assim será possível obter os valores para inserir nos campos informativos do objeto da classe “Atualizacao”. O código do método “atualizarInformacoes” está apresentado a seguir.

```
public void atualizarInformacoes(int numeroClientesEntrando, int numeroClientesSaindo) {
    if (atualizacao.estabilidadeMomentanea) {
        txtEstabilidadeMomentanea.setForeground(Color.BLACK);
        txtEstabilidadeMomentanea.setText("Estável");
    } else {
        txtEstabilidadeMomentanea.setForeground(Color.RED);
        txtEstabilidadeMomentanea.setText("Instável");
    }
    if (atualizacao.estabilidadeMedia) {
        txtEstabilidadeMedia.setForeground(Color.BLACK);
        txtEstabilidadeMedia.setText("Estável");
    } else {
        txtEstabilidadeMedia.setForeground(Color.RED);
        txtEstabilidadeMedia.setText("Instável");
    }
    double tempoRestante = (atualizacao.tempoFinalSimulacao - atualizacao.tempoAtualSimulacao)
/ (1000 * atualizacao.velocidade);
    if (tempoRestante < 0) {
        tempoRestante = 0;
    }
    dslVelocidade.setValue((int) (1001 - 50 * Math.sqrt(tempoRestante) / 3));
    txtNumeroClientesSistema.setText("" +
(Integer.parseInt(txtNumeroClientesSistema.getText()) + numeroClientesEntrando));
    if (numeroClientesSaindo < Integer.parseInt(txtNumeroClientesSistema.getText())) {
        txtNumeroClientesSistema.setText("" +
(Integer.parseInt(txtNumeroClientesSistema.getText()) - numeroClientesSaindo));
        txtNumeroClientesAtendidos.setText("" +
(Integer.parseInt(txtNumeroClientesAtendidos.getText()) + numeroClientesSaindo));
    } else {
        txtNumeroClientesAtendidos.setText("" +
(Integer.parseInt(txtNumeroClientesAtendidos.getText()) +
Integer.parseInt(txtNumeroClientesSistema.getText())));
        txtNumeroClientesSistema.setText("0");
    }
    nmrIntensidadeTrafego.setValue(atualizacao.mediaRo);
    nmrProbabilidadeZero.setValue(atualizacao.mediaProbabilidadeSistemaVazio);
    nmrProbabilidadeEnfileiramento.setValue(atualizacao.mediaProbabilidadeEnfileiramento);
    nmrClientesSistema.setValue(atualizacao.mediaClientesSistema);
    nmrVarianciaClientesSistema.setValue(atualizacao.mediaVarianciaClientesSistema);
    nmrClientesFila.setValue(atualizacao.mediaClientesFilas);
    nmrVarianciaClientesFila.setValue(atualizacao.mediaVarianciaClientesFilas);
    nmrTempoSistema.setValue(atualizacao.mediaTempoSistema);
    nmrVarianciaTempoSistema.setValue(atualizacao.mediaVarianciaTempoSistema);
    nmrTempoEspera.setValue(atualizacao.mediaTempoFila);
    nmrVarianciaTempoEspera.setValue(atualizacao.mediaVarianciaTempoFila);
    pnlAbas.setTitleAt(2, "Simulação (" + horarioExtenso((int)
atualizacao.tempoAtualSimulacao, 'M', true) + ")");
}
```

Agora que todos os métodos estão concluídos, será necessário alterar o código do evento “actionPerformed” do botão de simular para que o mesmo inicie a e pause a simulação. O novo código está descrito a seguir.

```
private void botSimularActionPerformed(java.awt.event.ActionEvent evt) {
    if (botSimular.getText().equals("Simular")) {
        if(validarParametros()){
            if (hmHorarioInicial.isEditable()) {
                atualizacao.tempoAtualSimulacao = converteHorario(hmHorarioInicial.getText());
                hmHorarioInicial.setEditable(false);
            }
            pnlAbas.setEnabled(false);
            pnlAparencia.setVisible(false);
```

```
            botSimular.setText("Pausar");
            botReiniciar.setEnabled(true);
            atualizacao.pausado = false;
            if (graficoClientesSistema == null) {
                graficoClientesSistema = new XYSeries("Numero de Clientes no Sistema");
                XYSeriesCollection colecao = new XYSeriesCollection();
                colecao.addSeries(graficoClientesSistema);
                JFreeChart objeto = ChartFactory.createXYLineChart("Número de Clientes no
Sistema", "Tempo", "Número de Clientes", colecao, PlotOrientation.VERTICAL, false, true,
false);
                objeto.setBackgroundPaint(pnlGraficoClientesSistema.getBackground());
                ChartPanel painel = new ChartPanel(objeto, true);
                pnlGraficoClientesSistema.removeAll();
                pnlGraficoClientesSistema.add(painel);
                graficoTempoSistema = new XYSeries("Tempo dos Clientes no Sistema");
                colecao = new XYSeriesCollection();
                colecao.addSeries(graficoTempoSistema);
                objeto = ChartFactory.createXYLineChart("Tempo dos Clientes no Sistema",
"Tempo", "Tempo dos Clientes", colecao, PlotOrientation.VERTICAL, false, true, false);
                objeto.setBackgroundPaint(pnlGraficoTempoSistema.getBackground());
                painel = new ChartPanel(objeto, true);
                pnlGraficoTempoSistema.removeAll();
                pnlGraficoTempoSistema.add(painel);
            }
            pnlGraficoClientesSistema.validate();
            pnlGraficoTempoSistema.validate();
            try {
                atualizacao.start();
            } catch (Exception e) {
                synchronized (atualizacao) {
                    atualizacao.notify();
                }
            }
            txtNumeroClientesSistema.setVisible(true);
            txtTextoClientesSistema.setVisible(true);
            txtNumeroClientesAtendidos.setVisible(true);
            txtTextoClientesAtendidos.setVisible(true);
        }
    } else if (botSimular.getText().equals("Pausar")) {
        atualizacao.pausado = true;
        botSimular.setText("Simular");
        pnlAbas.setEnabled(true);
        txtNumeroClientesSistema.setVisible(false);
        txtTextoClientesSistema.setVisible(false);
        txtNumeroClientesAtendidos.setVisible(false);
        txtTextoClientesAtendidos.setVisible(false);
        pnlAparencia.setVisible(true);
    }
}
```

Semelhantemente, será codificada a funcionalidade do botão “Reiniciar” em seu evento “actionPerformed”, cujo código é apresentado a seguir.

```
private void botReiniciarActionPerformed(java.awt.event.ActionEvent evt) {
    botReiniciar.setEnabled(false);
    atualizacao.pausado = true;
    while ((atualizacao.isAlive()) && (!atualizacao.getState().equals(State.WAITING))) {
        try {
            Thread.sleep(1);
        } catch (InterruptedException ex) {
            Logger.getLogger(SimulacaoView.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    pnlAbas.setTitleAt(2, "Simulação");
    ((PainelGrafico) pnlPainelGrafico).numeroClientesFila = new int[1];
    ((PainelGrafico) pnlPainelGrafico).tamanhoMaiorFila = 0;
    ((PainelGrafico) pnlPainelGrafico).numeroServidores = 1;
    ((PainelGrafico) pnlPainelGrafico).alturaAnterior = 0;
    atualizacao = new Atualizacao((PainelGrafico) pnlPainelGrafico, this);
    atualizacao.alterarFilaUnica(opcUnicaFila.isSelected());
    atualizacao.tempoFinalSimulacao = converteHorario(hmHorarioFinal.getText()).intValue();
    atualizacao.alterarTempoRestante((int) Math.round(Math.pow(3.0 * (1001 -
```

```
dslVelocidade.getValue()) / 50, 2)));

atualizacao.alterarNumeroServidores(Integer.parseInt(nmrNumeroServidores.getValue().toString()
));
    botSimular.setText("Simular");
    hmHorarioInicial.setEditable(true);
    pnlAbas.setEnabled(true);
    graficoClientesSistema = null;
    graficoTempoSistema = null;
    pnlGraficoClientesSistema.removeAll();
    pnlGraficoTempoSistema.removeAll();
    txtEstabilidadeMomentanea.setText("");
    txtEstabilidadeMedia.setText("");
    txtNumeroClientesSistema.setText("0");
    txtNumeroClientesAtendidos.setText("0");
    nmrIntensidadeTrafego.setText("");
    nmrProbabilidadeZero.setText("");
    nmrProbabilidadeEnfileiramento.setText("");
    nmrClientesSistema.setText("");
    nmrVarianciaClientesSistema.setText("");
    nmrClientesFila.setText("");
    nmrVarianciaClientesFila.setText("");
    nmrTempoSistema.setText("");
    nmrVarianciaTempoSistema.setText("");
    nmrTempoEspera.setText("");
    nmrVarianciaTempoEspera.setText("");
    pnlAparencia.setVisible(true);
    txtNumeroClientesSistema.setVisible(false);
    txtTextoClientesSistema.setVisible(false);
    txtNumeroClientesAtendidos.setVisible(false);
    txtTextoClientesAtendidos.setVisible(false);
    botSimular.setEnabled(true);
}
```

### 3.3.3 EVENTOS DOS PARÂMETROS

Para que o usuário possa alterar os parâmetros da simulação em tempo real será necessário codificar essa alteração nos respectivos eventos.

O campo de número de servidores, os campos de números de filas, os campos de horários e o campo de velocidade deverão ter esta funcionalidade. Seus respectivos eventos estão definidos a seguir.

```
private void nmrNumeroServidoresStateChanged(javax.swing.event.ChangeEvent evt) {
    opcMultiplasFilas.setText(nmrNumeroServidores.getValue().toString());
    int numeroServidores = Integer.parseInt(nmrNumeroServidores.getValue().toString());
    if (numeroServidores == 1) {
        opcUnicaFila.setSelected(true);
        opcMultiplasFilas.setEnabled(false);
    } else {
        opcMultiplasFilas.setEnabled(true);
    }
    atualizacao.alterarNumeroServidores(numeroServidores);
}

private void opcMultiplasFilasActionPerformed(java.awt.event.ActionEvent evt) {
    atualizacao.alterarFilaUnica(false);
}

private void opcUnicaFilaActionPerformed(java.awt.event.ActionEvent evt) {
    atualizacao.alterarFilaUnica(true);
}

private void hmHorarioInicialFocusLost(java.awt.event.FocusEvent evt) {
    if (converteHorario(hmHorarioInicial.getText()) == null) {
```

```
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "O valor do horário inicial deve estar no
formato hh:mm", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                hmHorarioInicial.requestFocus();
            }
        });
    }
}

private void hmHorarioFinalFocusLost(java.awt.event.FocusEvent evt) {
    Double horarioFinal = converteHorario(hmHorarioFinal.getText());
    if (horarioFinal != null) {
        atualizacao.tempoFinalSimulacao = horarioFinal.intValue();
    } else {
        JOptionPane.showMessageDialog(mainPanel, "Valor incorreto!\n"
                                                + "O valor do horário final deve estar no
formato hh:mm", "Valor incorreto", JOptionPane.ERROR_MESSAGE);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                hmHorarioFinal.requestFocus();
            }
        });
    }
}

private void dslVelocidadeStateChanged(javax.swing.event.ChangeEvent evt) {
    int tempoRestante = (int) Math.round(Math.pow(3.0 * (1001 - dslVelocidade.getValue()) /
50, 2));
    txtTempoRestante.setText(horarioExtenso(tempoRestante, 'S', true) + " restante" +
(tempoRestante != 1 ? "s" : ""));
    atualizacao.alterarTempoRestante(tempoRestante);
}
```

E assim termina o desenvolvimento da ferramenta.

# 4 CONCLUSÃO

O presente trabalho demonstra que é possível (e viável) o desenvolvimento completo de uma ferramenta comercialmente aplicável e versátil.

A gama de aplicações da ferramenta vai desde problemas simples encontrados no cotidiano das organizações, como a fila para bater ponto, a problemas cruciais para o funcionamento das mesmas, como filas de linha de produção ou filas de clientes.

A linguagem Java, utilizada no desenvolvimento, apresentou vantagens tais que viabilizaram o desenvolvimento da ferramenta. Com ela é possível, e relativamente fácil, encontrar soluções prontas para operações básicas, em forma de bibliotecas.

Além disso, a linguagem de programação Java é de mais fácil abstração e manuseio que linguagens de mais baixo nível, por contar com classes bastante fáceis de utilizar. Não é necessário se preocupar com as nuanças da programação em baixo nível, como o controle de alocação de memória, as chamadas para o sistema operacional, e assim por diante.

Vale notar, também, que a ferramenta, por ter sido desenvolvida em linguagem Java, pode ser facilmente transferida entre diferentes plataformas, desde que as mesmas tenham a máquina virtual Java instalada e suportem a biblioteca Swing, sem perder sua utilidade e sem a necessidade de uma nova compilação.

Adicionalmente, a ferramenta utilizada (NetBeans) auxiliou imensamente no desenvolvimento da interface com o usuário, visto que possui uma forma gráfica e intuitiva de montar telas e componentes visuais, o que seria muito mais trabalhoso se feito via código puro.

Por outro lado, o desempenho do sistema será levemente inferior ao que seria caso o sistema tivesse sido feito em nível mais baixo de programação. O Java executa códigos através de uma máquina virtual, que já é bastante pesada, e, além disso, carrega todas as bibliotecas disponíveis, independente da utilização das mesmas.

Outro ponto importante é que ao programar na linguagem Java se tem pouco controle do que ocorre na memória, o que normalmente deixa o programa um pouco mais pesado. Esta diferença, porém, só seria considerável para sistemas com pouca disponibilidade de memória RAM.

Por fim, é fácil reconhecer a utilidade prática do trabalho, visto que existem poucas ferramentas simplificadas com a mesma finalidade, e menos ainda gratuitas. Como o código é conhecido, quaisquer funcionalidades adicionais podem ser adicionadas facilmente, tornando a ferramenta bastante versátil.

# 5 REFERÊNCIAS BIBLIOGRÁFICAS

**Site oficial da linguagem Java**. Disponivel em: <http://www.java.com>. Acesso em: 7 nov. 2011.

**Site oficial do NetBeans**. Disponivel em: <http://www.netbeans.org>. Acesso em: 7 nov. 2011.

BUFFONI, S. S. D. O. Apostila de Introdução. **Professores da Universidade Federal Fluminense**. Disponivel em: <http://www.professores.uff.br/salete/imn/calnumI.pdf>. Acesso em: 7 nov. 2011.

DA ROCHA LOPES, V. L.; A. GOMES RUGGIERO, M. **Cálculo Numérico:** Aspectos Teóricos Computacionais. 2ª Edição. ed. São Paulo: Makron Books, 1996.

EDU-DEVELOPER. Jade. Disponivel em: <http://java.net/projects/jade/>. Acesso em: 11 set. 2011.

FOGLIATTI, M. C.; MATTOS, N. M. C. **Teoria de Filas**. 1ª. ed. [S.l.]: Interciência, 2006.

JFREECHART. Disponivel em: <http://www.jfree.org/jfreechart/>. Acesso em: 12 set. 2011.

JSCI - A science API for Java. Disponivel em: <http://jsci.sourceforge.net/>. Acesso em: 23 out. 2011.

JSCIENCE. Disponivel em: <http://jscience.org/>. Acesso em: 11 set. 2011.

KENDALL, D. G. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. **Annals of Mathematical Statistics**, p. 338–354, 1953.

MELVILLE, D. J. YBC 7289. Disponivel em: <http://it.stlawu.edu/~dmelvill/mesomath/tablets/YBC7289.html>. Acesso em: 7 nov. 2011.

ORACLE. Using Swing Components: Examples (The Java® Tutorials > Creating a GUI with JFC/Swing > Using Swing Components). **Site da Oracle**. Disponivel em: <http://docs.oracle.com/javase/tutorial/uiswing/examples/components/>. Acesso em: 05 ago. 2011.

PEGDEN, C. D.; SHANNON, R. E.; SADOWSKI, R. P. **Introduction to Simulation Using SIMAN**. 1ª. ed. [S.l.]: Mcgraw-Hill College Division, 1990.