Okapi

Documentação do Projeto

Felipe Siqueira
9847706

Edylson Torikai
5248962

Pedro Avellar
9779304

Matheus Bernardes
9277979

20 de Junho de 2017

Universidade de São Paulo (USP)
Instituto de Ciências Matemáticas e de Computação (ICMC)
Bacharelado em Ciências da Computação (BCC)

Sumário

1 Introdução **3**

I Instruções pré-execução **3**

2 Arquivo JAR Okapi **3**

3 Procedimento para a criação do jar Okapi **3**

4 Informação sobre o ambiente de desenvolvimento **3**

II Arquitetura do projeto **4**

5 Interpretador de comando **4**

6 Comandos disponíveis **4**

7 Detalhes da funcionalidade *Plotting* **5**
7.1 Pontos (*dot*) . . . 6
7.2 Linhas (*line*) . . . 6
7.3 Barras (*bar*) . . . 6
7.4 Pizza (*pie*) . . . 6
7.5 *BoxPlot* (*box*) . . . 6

8 Interpretador Aritmético **7**

9 Gerenciador de tabelas de dados embarcado **7**

10 Exemplos comentados de uso **8**

III Organização do projeto **10**

11 Linguagem de Programação Java **10**

12 Alguns dos principais recursos utilizados **10**
12.1 Java.util.regex . . . 10
12.2 Java.lang.reflect . . . 10
12.3 Javax.swing . . . 10

13 Descrição das Classes do projeto **11**
13.1 *MainInterface* . . . 11
13.2 *Interpreter* . . . 11
13.3 *GeneralOkapi* . . . 11
13.4 *Arithmetic* . . . 11
13.5 *IllegalArithmeticExpression* . . . 11
13.6 *TableManager* . . . 11
13.7 *OkapiTable* . . . 11
13.8 *Matrix* . . . 11
13.9 *GeneralPlot* . . . 12
13.10 *PlotLine* . . . 12
13.11 *PlotBar* . . . 12
13.12 *PlotDot* . . . 12
13.13 *PlotBox* . . . 12
13.14 *PlotPie* . . . 12
13.15 *HelpSystem* . . . 12

14 Documentação interna **12**

1 Introdução

Okapi é um programa simples para visualização de dados em duas dimensões. Através dele, é possível construir gráficos dos tipos Pizza, Barra, Linha, Pontos e *BoxPlot*. Para auxiliar nesta tarefa, o programa conta com um sistema interno simples de gerenciamento de tabelas.

O usuário deve passar linhas de comandos ao programa, a fim de realizar alguma operação. Este último, por sua vez, avalia o comando dado, e procura atendê-lo como especificado, sempre que possível. Como recursos extras, o programa também conta com um interpretador de expressões aritméticas interno, e também opera com matrizes. Todos estes itens serão descritos com mais detalhes nas seções que seguem, sob duas diferentes perspectivas.

A primeira parte desta documentação visa instruir o usuário sobre o processo necessário para a correta execução do programa.

A segunda parte, intitulada "Arquitetura do Projeto", busca explorar o projeto sob uma perspectiva de interesse do usuário final. A terceira parte, com o título "Organização do Projeto", explora itens em comum da primeira parte, porém sob a perspectiva de interesse do programador, isto é, com maior profundidade nas decisões de projeto e detalhes de implementação.

Parte I
Instruções pré-execução

2 Arquivo JAR Okapi

Juntamente com o código fonte do projeto Okapi, foi enviado o arquivo **Okapi.jar**, a fim de evitar eventuais problemas de incompatibilidade em função do versionamento do compilador Java. A fim de executar o programa, basta digitar em seu console/terminal:

java -jar Okapi.jar

3 Procedimento para a criação do jar Okapi

Nesta seção será descrito o processo tomado para a criação de Okapi.jar, a fim de tornar o procedimento tomado reproduzível ao usuário. O procedimento que segue foi efetuado no sistema operacional de distribuição Linux. O procedimento deve variar em outros sistemas operacionais.

Primeiro, deve-se compilar os principais módulos *.java, presentes na pasta "src", com o seguinte comando:

javac Interpreter.java MainInterface.java

Depois, para gerar o arquivo Okapi.Jar, na pasta okapi, foi usado o seguinte comando em Bash, dentro do subdiretório "/okapi/src":

find . -name "*.class" | xargs jar cvfm ../Okapi.jar Manifest images/ OkapiHelp.xml

Este comando deve ser digitado no console/terminal no subdiretório "src". O jar será gerado automaticamente no diretório raiz "okapi".

4 Informação sobre o ambiente de desenvolvimento

Tais informações foram retiradas diretamente do comando "java -version".

openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-0ubuntu1.16.04.2-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Parte II
Arquitetura do projeto

5 Interpretador de comando

O sistema interno de interpretação de comando procura construir uma ponte entre o objetivo do usuário e os recursos disponíveis do programa. Seu funcionamento interno baseia-se principalmente em expressões regulares e o recurso de reflexão, próprio do Java.

Para utilizá-lo, o usuário deve digitar, em seu terminal/console, um comando completo por vez, no formato que se segue:

$$(\text{Funcionalidade})\ (\text{Arg}_1) = (\text{Valor}_1)\ (\text{Arg}_2) = (\text{Valor}_2)\ \ldots\ (\text{Arg}_n) = (\text{Valor}_n)$$

Para atribuir o valor a um argumento, o usuário pode utilizar de "equal", "equals", "is" ou, como visto no modelo anterior, "=". A combinação inter-paramétrica é permitida (e.g "magplot type = bar color is red"), mas não intra-paramétrica (e.g "magplot type equal= bar").

Nota-se que o comando fornecido pelo usuário, internamente, passa por um processo de canonização, removendo alguns caracteres especiais e a diferenciação entre letras maiúsculas em minúsculas. Assim sendo, os comandos "HELP opEratiON === magplot" e "help operation = MAGPLOT" são interpretados de forma idêntica.

Caso o usuário digite uma expressão aritmética pura (isto é, apenas números reais, operações aritméticas com os símbolos {*, /, +, -, ∧} e parênteses), o interpretador terá a capacidade de detectá-la automaticamente e, com isto, apresentar o resultado ao usuário. Tal funcionalidade será analisada com maior profundidade na seção 8.

Por fim, o interpretador de comando também é responsável por detectar linhas de comando inválidas e ausência de parâmetros obrigatórios (descritos nas seções que seguem) para determinadas funcionalidades, ou redundantes (dupla especificação de um mesmo parâmetro, no mesmo comando).

6 Comandos disponíveis

O usuário pode obter uma lista de todos as funcionalidades disponíveis no programa, durante a sua execução, digitando "commands". A fim de suplementar a orientação do usuário, existe ainda a funcionalidade "help operation = FuncionalidadeDesejada", que oferece uma breve descrição da função buscada, além de exibir os seus parâmetros internos.

As funcionalidades disponíveis em Okapi, juntamente com suas especificações particulares, podem ser vistas no quadro 1.

Quadro 1: Tabela de comandos disponíveis em Okapi.

Funcionalidade	**Descrição**	**Parâmetros**
commands	Exibe uma lista de funcionalidades disponíveis.	Nenhum.
exit	Encerra o programa, fechando todas as janelas.	Nenhum.
help	Busca no documento XML do Okapi(presente no diretório ./src) informações sobre a operação desejada	operation
list	Lista todas as tabelas ativas na seção atual de Okapi.	Nenhum.
magplot	Permite a visualização dos dados de uma dada tabela, segundo a perspectiva de barras (*bar*), pontos (*dot*), linhas (*line*), "BoxPlot" (*box*) ou Pizza (*pie*). Esta configuração é controlada pelo parâmetro "type". Cada tipo de *plotting* será descrito com mais detalhe na seção 7.	type table color xlim ylim xlab ylab xint yint
matrix	Permite uma operação matricial entre duas dadas tabelas "a" e "b" e, se especificado, gera uma nova tabela resultante "r".	operation a b r
table	Utiliza o gerenciador de tabelas interno para operar com as estruturas de dados.	operation name rownum colnum file

7 Detalhes da funcionalidade *Plotting*

A funcionalidade de *plotting*, apresentada em Okapi como "**magplot**", é a principal do programa. Para utilizá-la, é necessário fornecer dois parâmetros obrigatórios: "type" e "table".

O parâmetro "type" é responsável por identificar o tipo de *plotting*, podendo este apresentar os valores "*pie*", "*dot*", "*bar*", "*line*" ou "*box*".

O parâmetro "table" é responsável por identificar qual tabela de dados será realizada a interpretação, para a construção do gráfico. O formato ideal da tabela de entrada varia dentre os tipos de *plotting* e, por isso, será feita uma análise para cada um deles, separadamente, nas subseções que seguem.

O usuário também tem a possibilidade de definir manualmente, de forma puramente opcional, os parâmetros "*xlab*" e "*ylab*" que controlam, respectivamente, a identificação dos eixos das abscissas e ordenadas, respectivamente.

Também pode-se informar, caso desejado, os parâmetros "*xlim*" e "*ylim*", que definem, respectivamente, os limites dos eixos das abscissas e ordenadas. O formato desejado dos valores destes parâmetros é um pouco particular, sendo formado por um par de valores reais x e y, separados por vírgula, e dentro de um par de parênteses, colchetes ou chaves, sem espaçamento interno. Tal formato pode ser observado no quadro 2. Quando estes parâmetros não são especificados explicitamente num comando de *magplot*, o programa realiza a adaptação dos eixos coordenados para os dados atuais, automaticamente.

Quadro 2: Formato de entrada dos parâmetros "*xlim*" e "*ylim*", da funcionalidade de *plotting* "*magplot*".

Modelo	(x,y)	[x,y]	{x, y}
Exemplos	(25.5,54.5)	[-100,100]	{-50.50,0}

O par de parâmetros "*xint*" e "*yint*" que informam o número de intervalos desejado nos eixos das abscissas e ordenadas, respectivamente. Tais parâmetros, assim como os dois pares anteriores, são facultativos.

Por último, pode-se informar a cor predominante no *plotting*, através do parâmetro "*color*". As cores padrões dos *plottings* são branco (interior das figuras geométricas) e preto (bordas).

Como último detalhe final em comum aos *plottings*, a janela, que será aberta com o gráfico construído ao uso da funcionalidade "*magplot*", oferece a opção de gerar um arquivo de saída com a imagem resultante, através do botão "Save". O arquivo é gerado, por padrão, no diretório local, com o mesmo nome da tabela que foi usada como recurso de dados, e com a extensão *.jpg*.

Em caso de conflito de nomes (isto é, caso já exista um aquivo com o mesmo nome da imagem gerada), então o programa irá gerar o arquivo de saída com o nome complementado com o próximo número natural que possa diferenciar dos nomes dos arquivos já existentes (e.g, "tabela1.png", "tabela2.png", "tabela23.png", ...).

7.1 Pontos (*dot*)

Este tipo de *plotting* aceita apenas tabelas 2xN ou Mx2, isto é, tabelas que possam expressar uma relação de valores (x,y).

Caso o usuário especifique uma tabela horizontal (2xN), a primeira linha será interpretada como os valores de x, e os valores da segunda linha, os valores de y. Do contrário, numa tabela vertical (Mx2), os valores da primeira coluna são interpretados como os valores de x, e os da segunda coluna, valores de y. Neste último caso, um aviso é lançado ao usuário, apenas para se certificar que tal interpretação da tabela está ocorrendo.

7.2 Linhas (*line*)

As especificações deste tipo de *plotting* são as mesmas do de tipo "*dot*".

7.3 Barras (*bar*)

Este tipo de *plotting* aceita tabelas com quaisquer dimensões. O número de barras é baseado no valor dado ao parâmetro "*xint*".

7.4 Pizza (*pie*)

Este tipo de *plotting* aceita tabelas com quaisquer dimensões, e ignora quaisquer valores relacionados aos eixos coordenados.

7.5 *BoxPlot* (*box*)

Este tipo de *plotting* aceita tabelas com quaisquer dimensões, e ignora quaisquer parâmetros relacionados ao eixo das ordenadas.

Tal estilo de gráfico deixa claro os três quartis (Quartil inferior, Mediana e Quartil superior), os valores máximo e mínimo global, bem como a detecção de dados não-representativos (*outliers*) do conjunto de dados. Como regra geral, qualquer valor 50% maior ou menor que o intervalo da diferença interquartil (IQR = Q3 - Q1) é considerado um dado *outlier* e, nesta situação, é eleito um valor máximo ou mínimo local (*thresholds*).

8 Interpretador Aritmético

O interpretador de comandos de Okapi é capaz de detectar e resolver expressões aritméticas. Para acessar tal funcionalidade, o usuário precisa apenas digitar sua expressão aritmética, em uma única linha de comando, e o interpretador deverá devolver a solução numérica, se possível.

Os operandos podem haver diferentes casas de precisão. As operações aritméticas suportadas são adição (+), subtração (-), multiplicação (*), divisão (/), e potência ($\wedge$). As expressões podem ser parentesadas (e.g (- 2 + 3) * (2.5 / (2.5 $\wedge$ 2))).

9 Gerenciador de tabelas de dados embarcado

O gerenciador de tabelas interno em Okapi é responsável por criar estruturas de dados lógicas, as quais as funções de operações matriciais e de *plotting* possam funcionar corretamente. Para o uso desta funcionalidade, os parâmetros "*operation*" e "*name*" devem ser sempre explicitados.

O parâmetro "*name*" é responsável por identificar a tabela na qual será realizada a operação especificada.

O parâmetro "*operation*" pode tomar os valores "create", "remrow", "addrow", "remcol", "addcol", "addemptyrow", "addemptycol", "colname", "rowname", "print", "remlastcol" e "remlastrow". Em suma, os parâmetros da funcionalidade "table" podem ser observados no quadro 3

Quadro 3: Lista de parâmetros da funcionalidade "table" de Okapi.

Operação	**Descrição**	**Subdependências**
remrow	Remove a coluna correspondente ao índice dado.	index
remcol	Remove a linha correspondente ao índice dado.	index
addrow	Adiciona uma nova linha, com os valores no arquivo especificado.	file
addcol	Adiciona uma nova coluna, com os valores no arquivo especificado.	file
rowname	Atualiza o nome das linhas de acordo com o conteúdo do arquivo especificado.	file
colname	Atualiza o nome das colunas de acordo com o conteúdo do arquivo especificado.	file
remlastrow	Remove a última linha da tabela.	Nenhuma.
remlastcol	Remove a última coluna da tabela.	Nenhuma.
addemptyrow	Adiciona uma linha vazia na tabela.	Nenhuma.
addemptycol	Adiciona uma coluna vazia na tabela.	Nenhuma.
print	Imprime a dada tabela no terminal/console.	Nenhuma.

Na operação "create", é obrigatório especificar os parâmetros "rownum", "colnum", "file" e "name". Estes campos definem, respectivamente, o número de linhas, colunas, o arquivo onde os novos valores serão buscados, e o nome de identificação da nova tabela.

Para as operações "addrow", "addcol", "colname" e "rowname" também é necessário especificar o parâmetro "file", local onde será buscado os novos valores. Nota que o valor deste parâmetro pode ser um caminho completo de arquivo, e pode ser necessária a inclusão da extensão do arquivo, a depender do Sistema Operacional.

As operações "remrow" e "remcol" exigem o parâmetro "index" preenchido.

10 Exemplos comentados de uso

A seguir são expostos alguns comandos válidos.

Conjunto de dados de teste (conteúdo de “test.temp”):

24,4 16,1 18,0 17,5 17,6 16,4 40,0 -1,0 -13,9 -2,0
17,4 18,0 13,5 14,5 14,9 14,3 37,0 -1,0 -3,4 9,3

1) table operation = create name = hellotable rownum = 2 colnum = 10 file = test.temp

Este comando cria uma nova tabela 2x10, identificada como “hellotable”, com os dados numéricos contidos em um arquivo “test.temp”. Vale lembrar que valores reais, com casas de precisão (2,87) devem seguir o padrão de números decimais máquina do usuário (isto é, diferencia-se pontos de vírgulas de separação decimal de acordo com a máquina do operador).

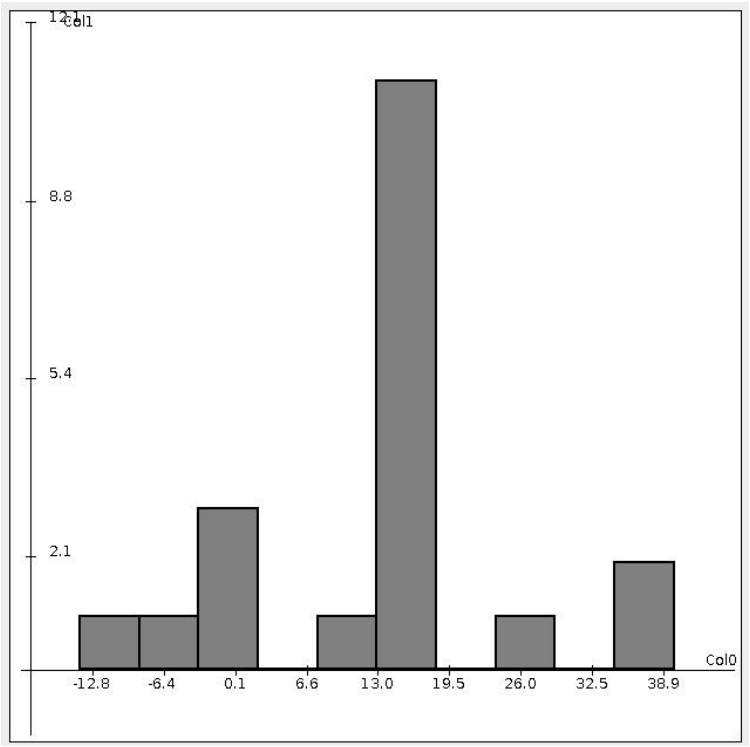
2) table operation is print name = hellotable

Tal comando imprime no terminal/console a tabela anteriormente criada.

3) magplot type is bar table is hellotable color = gray xint = 10

Aqui é criado um gráfico de barras utilizando a tabela existente, com cinco intervalo de barras, e cor predominantemente vermelha. O resultado pode ser observado na imagem 2.

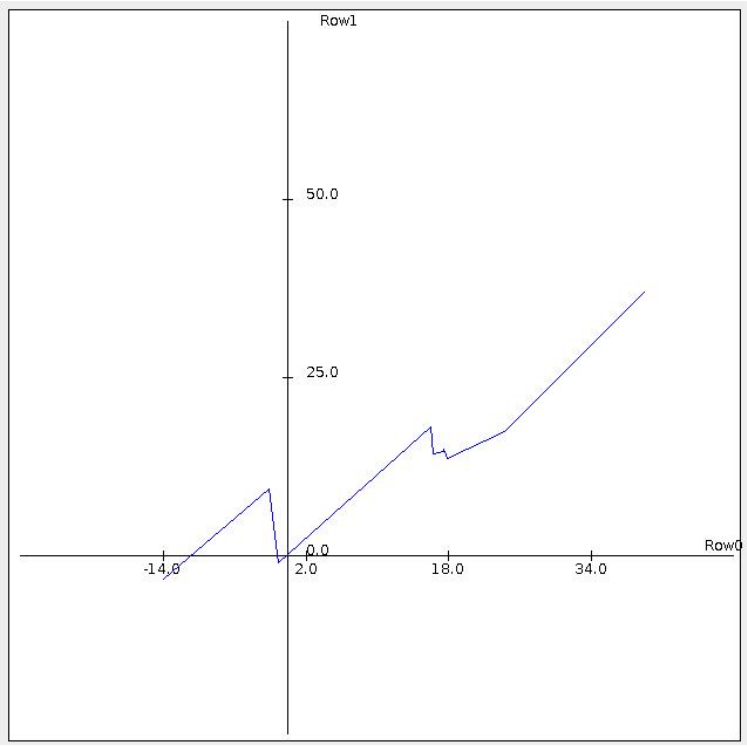
Imagem 1: Exemplo de gráfico em barras.



4) magplot type = line table is hellotable color equals blue xlim is (-30,50) ylim is [-25,75].

Desta vez, é criado um gráfico de linhas, da cor azul, com os limites dos eixos coordenados explicitamente indicados. Interessante recordar que, caso os limites dos eixos coordenados não fossem mencionados, o programa ajustaria os eixos automaticamente. O resultado pode ser verificado na imagem 2.

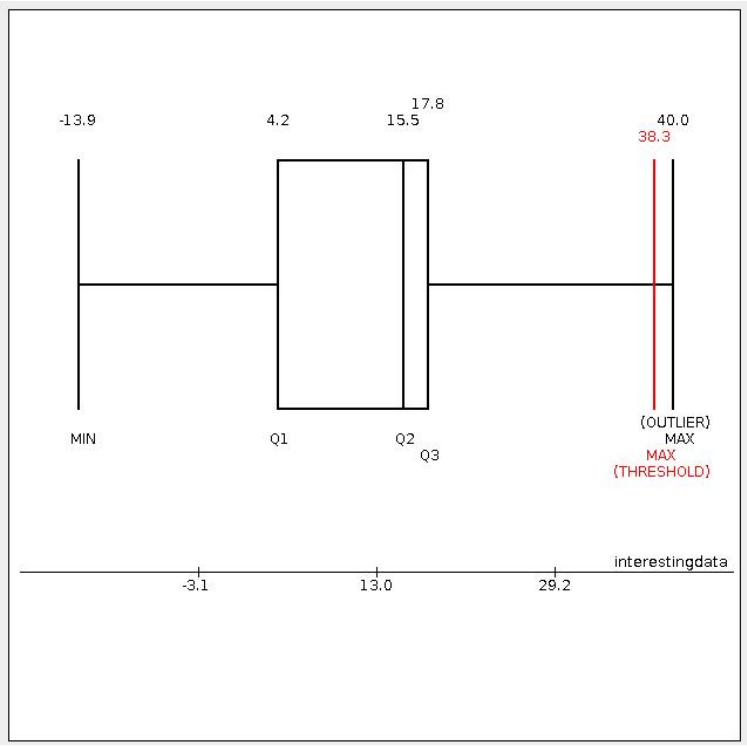
Imagem 2: Exemplo de gráfico de linha.



5) magplot table = hellotable type = box xlab = InterestingData

Este é um exemplo típico de *plotting* do tipo *BoxPlot*. Este *plotting* é unidimensional, isto é, toda a tabela é considerada (independentemente do número de linhas e colunas) para a construção do gráfico em relação ao eixo das abscissas. O resultado gerado pode ser observado no gráfico 3.

Imagem 3: Exemplo de gráfico do tipo *BoxPlot*.



6) help operation = magplot

Exemplo de uso do sistema interno de ajuda ao usuário. As informações apresentadas são buscadas no arquivo "./src/OkapiHelp.xml". Este comando forma um bom par com o comando "commands", que lista todos os comandos disponíveis ao usuário em Okapi.

Parte III
Organização do projeto

11 Linguagem de Programação Java

O projeto todo foi feito em Linguagem de Programação Java, como determinado pela ementa do curso. Todos os recursos utilizados neste projetos são nativos da própria linguagem e, portanto, devem vir em conjunto com qualquer distribuição oficinal do pacote da máquina virtual da Oracle, sem necessidade de configuração adicional de *plugins* ou qualquer outro recurso externo.

12 Alguns dos principais recursos utilizados

Esta seção é dedicada a dar ênfase sobre como foi aplicado alguns dos pacotes mais presentes no código do projeto.

12.1 Java.util.regex

O uso de expressões regulares foi fundamental para a construção dos módulos [Interpreter.java], [Arithmetic.java] e [HelpSystem.java]. Os principais *Patterns* utilizados são declarados como *Strings* constantes na seção de constantes de ambas as classes, a fim de facilitar sua manutenção.

Como pode ser observado no código fonte da classe [Interpreter.java], a mesma trata-se de uma classe abstrata, isto é, não pode ser instanciada. Em contrapartida, suas expressões regulares devem ser compiladas antes do uso. Para solucionar este problema, foi criado um método estático, dentro da própria classe [Interpreter.java], denominado compile(). O simples uso deste método prepara os *Patterns* das expressões regulares, permitindo o uso posterior.

12.2 Java.lang.reflect

A técnica de reflexão fora explorada de diversas formas no decorrer do projeto, especialmente na classe [Interpreter.java].

Através dele, foi possível o mapeamento direto entre nomes de métodos e campos e os próprios parâmetros que são especificados em cada linha de comando do usuário final, visando beneficiar a escalabilidade do projeto.

Em suma, todos os métodos que podem ser diretamente chamados pelo usuário (com exceção do método *getInput()*) estão contidos diretamente na classe [Interpreter.java]. Todos os campos que podem ser diretamente definidos, pelo usuário final, como parâmetros de funcionalidades, através de linhas de comandos, estão na numa classe interna privada do [Interpreter.java], identificada por [Interpreter.InterpreterParameters].

12.3 Javax.swing

Embora existam pacotes gráficos Java mais sofisticados, foi optado pelo uso exclusivo do pacote gráfico SWING e, portanto, fora realizada toda a implementação interna da construção dos gráficos, sem facilidades externas.

O pacote javax.swing fora explorado, em sua maioria, na classe [GeneralPlot.java], embora também presente em [MainInterface.java] e [HelpSystem.java], e também nas classes restantes relacionadas a *Plotting*.

13 Descrição das Classes do projeto

Esta seção é dedicada à breves comentários sobre as particularidades de implementação de cada uma das classes que compõe este projeto. Informações aprofundadas podem ser encontradas na documentação interna do módulo correspondente.

13.1 *MainInterface*

Esta é a classe que contém o método *main*. Ela é responsável por chamar a janela de inicialização Okapi, realizar a chamada do método Interpreter.compile(), manter o loop principal do programa (até que o comando "exit" seja chamado pelo usuário), e fechar todas as janelas quando este último é quebrado (isto é, ao fim do programa).

13.2 *Interpreter*

Classe abstrata responsável por interpretar as linhas de comandos dadas pelo usuário, e com isto ou estabelecer os parâmetros corretos embarcados na mesma, ou chamar a classe [Arithmetic.java] para a resolução de expressões aritméticas.

13.3 *GeneralOkapi*

Esta classe procura estabelecer um elo em comum entre as classes [Arithmetic.java], [TableManager.java] e [Matrix.java], em função da possível necessidade de criação de métodos comuns entre as mesmas.

13.4 *Arithmetic*

Classe responsável pela resolução de expressões aritméticas (através de seu método principal, "solve(String arithmeticExpression)"). Caso a expressão aritmética seja inválida, a mesma lança uma exceção customizada *IllegalArithmeticExpression*, que, por sua vez, é recebida pelo [Interpreter.java], e ali tratada adequadamente.

13.5 *IllegalArithmeticExpression*

Exceção customizada, lançada pela classe [Arithmetic.java], na ocasião do recebimento de uma expressão aritmética não suportada.

13.6 *TableManager*

Classe abstrata construída para intermediar com quaisquer operações entre o usuário final e a classe [OkapiTable.java], auxiliando o primeiro em relação à operações sobre o último. Esta classe trata todas as operações disponíveis com relação a tabela de dados, citadas na seção 9.

13.7 *OkapiTable*

Estrutura de dados padrão do projeto Okapi. Internamente, esta classe possui uma List¡List¡T¿¿ (Sendo T padronizado, neste projeto, em *Double*), usada para representar arquivos de dados carregados pelo usuário, e duas List¡String¿, sendo uma para a listagem de nomes de colunas, e a segunda, para nomes de linhas.

13.8 *Matrix*

Classe responsável por realizar operações matriciais (soma, subtração e multiplicação) entre duas tabelas de dados.

13.9 *GeneralPlot*

Classe abstrata responsável por realizar todas as etapas em comum para todos as subclasses de *plotting*, isto é, construção de uma janela de visualização separada, com pano de fundo, eixos coordenados (sem houver), título de gráfico e botões de "Save" e "Close".

13.10 *PlotLine*

Esta classe constrói uma representação visual de uma dada tabela de dimensões 2xN ou Nx2 (N ¿= 1), em um gráfico de linhas. Este processo ocorre durante a chamada do método PlotBar.setupPlotLine(OkapiTable, Color). O gráfico ocorre durante a sua instanciação, onde internamente é chamado o construtor do da classe [GeneralPlot.java].

13.11 *PlotBar*

Esta classe constrói uma representação visual de uma dada tabela de quaisquer dimensões (desde que não nula), em um gráfico de barras. Este processo ocorre durante a chamada do método PlotBar.setupPlotBar(OkapiTable, Color). O gráfico ocorre durante a sua instanciação, onde internamente é chamado o construtor do da classe [GeneralPlot.java].

13.12 *PlotDot*

Esta classe constrói uma representação visual de uma dada tabela de dimensões 2xN ou Nx2 (N ¿= 1), em um gráfico de pontos. Este processo ocorre durante a chamada do método PlotBar.setupPlotDot(OkapiTable, Color). O gráfico ocorre durante a sua instanciação, onde internamente é chamado o construtor do da classe [GeneralPlot.java].

13.13 *PlotBox*

Esta classe constrói uma representação visual de uma dada tabela de quaisquer dimensões (desde que não nula), em um gráfico do tipo *BoxPlot*. Este processo ocorre durante a chamada do método PlotBar.setupPlotBox(OkapiTable, Color). O gráfico ocorre durante a sua instanciação, onde internamente é chamado o construtor do da classe [GeneralPlot.java].

13.14 *PlotPie*

Esta classe constrói uma representação visual de uma dada tabela de quaisquer dimensões (desde que não nula), em um gráfico de pizza. Este processo ocorre durante a chamada do método PlotPie.setupPlotPie(OkapiTable, Color). O gráfico ocorre durante a sua instanciação, onde internamente é chamado o construtor do da classe [GeneralPlot.java].

13.15 *HelpSystem*

Este módulo é instanciado através da funcionalidade "*help*". Em suma, esta classe carrega as informações do arquivo "./src/OkapiHelp.xml" em memória, e então utiliza expressões regulares para obter as partes relevantes para a busca atual do usuário. Se bem-sucedido, esta mesma classe cria uma janela, com as informações recuperadas.

14 Documentação interna

A documentação interna do projeto foi construída em dois níveis: *Javadoc*, no escopo de classes e métodos, e comentários internos aos métodos.

O primeiro visa justificar, de maneira sucinta, a criação de cada método (ou classe), dando ênfase ao seu respectivo valor de retorno (se houver) e exceções internas não tratadas (se houverem).

O segundo procura possibilitar a manutenção do código, de maneira específica e rigorosa, guiando o leitor através das etapas, ou tornando explícito detalhes importantes para o funcionamento dos algoritmos que, muitas vezes, não são óbvios a primeira vista.