Coleta em Matriz

Relatório sobre uma coleta de células de uma matriz

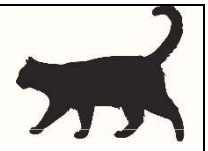
João Pedro Lima Affonso de Carvalho

NUSP: 11260846

joaopolicarvalho99@usp.br

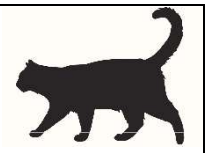
joaopedro@cleancloud.com.br

Moderador 'JoaoPCarv' do fórum TutorBrasil



SUMÁRIO

Nota ao leitor: *Esta documentação faz parte de um modelo padronizado pelo autor e foi revisada pela última vez em 15 de março de 2023. Para quaisquer dúvidas, recomendações ou críticas, por favor contate o autor através dos endereços eletrônicos disponibilizados. Obrigado pela leitura e pelos devidos feedbacks.*



1. Introdução

Este relatório tem como objetivo documentar o projeto que executa a solução de uma problemática lançada através de um exercício de pré-vestibular, onde o objeto em pauta é uma matriz, que será submetida a um algoritmo que coleta suas entradas através de um contexto, conforme será explicado na seção 1.1..

A solução é, em última análise, uma série de algoritmos, e, portanto, a escolha da linguagem Java para este projeto foi apenas por interesse pessoal do autor.

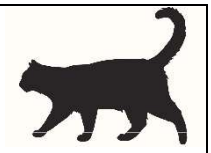
1.1. Problemática

O tópico 'Matriz' foi postado em 16 de agosto de 2022 no fórum de estudos TutorBrasil, onde o autor é moderador e usuário ativo. Nele, é descrito um exercício de vestibular, que traz a seguinte questão:

> *Uma matriz é uma tabela numérica muito utilizada na computação para o armazenamento de dados e informações e é conhecida, nesse meio, como banco de dados.*
>
> *As matrizes na computação são sempre referidas pelas suas linhas e colunas, nessa ordem. Por exemplo, considerando uma matriz A que começa sua contagem no zero, para se referir à célula na terceira linha e primeira coluna, utiliza-se A[2,0].*
>
> *Uma matriz com 12 linhas e 5 colunas foi utilizada durante o cadastro dos dados dos cinco setores de serviços de uma empresa, ao longo de todos os meses de um ano, sendo uma linha para cada mês e uma coluna para o cadastro dos dados de cada serviço prestado.*
>
> *No ato do resgate de alguns dados, foram emitidos ao computador os comandos de coletar todos os dados que obedecessem a cada uma das regras a seguir:*

I	A[l,c]	l < c
II	A[l,c]	l = c
III	A[l,c]	l + c = 6
IV	A[l,c]	l > c
V	A[l,c]	l – c > 5

Tabela 1: Regras



Entre todos os comandos, aquele que coletou mais células dessa matriz foi:

a) I.
b) II.
c) III.
d) IV.
e) V.

, cuja resposta é D) IV.

O autor propôs uma resposta a essa questão (via joaopcarv[MOD]), que contextualiza álgebra e combinatória, de uma forma computacional. O tópico, com a resposta numérica, pode ser acessado em https://www.tutorbrasil.com.br/forum/viewtopic.php?f=3&t=103241. Nessa resposta, o autor também deixou uma implementação de um código em Java, que faz exatamente o que o algoritmo do exercício descreveu, e esse código é uma versão inicial e mais simples do programa final deste relatório.

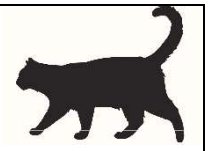
Lendo o enunciado, é interessante o fato que a questão se trata justamente de uma atividade computacional, e é muito conveniente escrever uma simulação dessa atividade, o que serve de motivação para este projeto.

Uma matriz 12x5, mapeada de acordo com a convenção computacional, ou seja, com range de endereços [(0,0); (11,4)], é populada com os dados de cinco serviços de uma empresa ao longo de um ano (doze meses). A partir de então, a matriz é tratada por comandos de computador que vão coletar as células de acordo com regras (vide *Tabela 1: Regras*). Essas regras são relações matemáticas entre os parâmetros linha (l) e coluna (c) de cada entrada.

O intuito da questão é treinar o aluno na contagem de pares ordenados (l,c), ou seja, é uma questão de combinatória (contagem), mas o cenário hipotético construído em cima disso, mencionando comandos computacionais para essa tarefa, motivou o autor a de fato fazer uma instância de implementação dessa contagem.

Observações:

a) A combinatória envolve raciocínio lógico e aborda várias situações que envolvem quantidades expressivas de possibilidades, e a computação torna-se uma ferramenta propícia para a listagem e a contagem de casos nesse quesito;
b) Matrizes são objetos tanto matemáticos quanto computacionais, onde são chamadas de arrays.



1.2. Ferramentas implementadas

A solução programada foi feita inicialmente pelo autor em 17 de agosto de 2022, tendo sido utilizado apenas a IDE Eclipse. Durante a retomada e aprimoramento do projeto (que são explicados em 2.1.), a IDE utilizada foi Intellij.

Para a impressão de informações relevantes ao usuário, foi utilizado o **PrintStream** padrão do sistema, **System**.*out*, com o método de impressão *println*(...).

O programa aproveita as mesmas mensagens que são impressas ao usuário e escreve um relatório simples em documento de texto (**.txt**), e isso é possível graças aos pacotes de manuseio de arquivos do Java. Além disso, existe um documento de texto para controle interno chamado **LeiaMeRegras.txt** (veja a seção 2.2.).

Por fim, após a primeira fase de retomada do projeto, o autor instalou em sua IDE a funcionalidade SonarLint*, da SonarQube, que faz uma série de sugestões, desde correção de potenciais bugs a boas práticas, ainda em tempo de execução. Essas informações a mais fomentaram manutenções no código.

1.3. Casos de uso

Devido ao fato de que o programa é de uso bastante específico e puramente matemático, os casos de uso se resumem em apenas um: a relação 1:1 entre cliente e programa, este nomeado '*ColetaMatriz*'.

Nesta única relação, o cliente faz a requisição com parâmetros e *ColetaMatriz* obrigatoriamente envia uma resposta, através de logs.

Os parâmetros passados pelo cliente são o número de regras, linhas e colunas da matriz. Aqui, ao invés de fixar a resolução em apenas a problemática específica da questão, é dada a liberdade do usuário de escalar quantas regras devem ser levadas em conta, o que também abre margem para o programador adicionar mais regras ao código conforme demandado. Além disso, o usuário também pode escolher quaisquer dimensões válidas para a matriz em análise.

O programa, por sua vez, responde ao usuário de duas formas possíveis, chamadas de "*happy day*" e "*corner cases*" (fazendo o uso de nomenclaturas padrões na engenharia de softwares).

São estes:

- *Happy day*:
  Em que o cliente informa dados de entrada válidos, sendo as dimensões da matriz pelo menos 1x1 e o número de regras compatível com a quantidade de regras pré-programada pelo autor.
  Neste cenário, o programa deve retornar dois logs: um na forma impressa, usando o método padrão já mencionado, e um relatório em **.txt**, também já mencionado, guardando as mesmas impressões feitas em tempo de execução.
- *Corner cases*:
  Em que ou parâmetros inválidos foram passados na solicitação, ou algum problema interno ocorreu, mais provavelmente com o gerenciamento de arquivos feito em tempo de execução. Esses casos são tratados com o lançamento e a captura de exceções, que param o programa e imprimem a mensagem de erro padrão, formatada através do método printStackTrace().

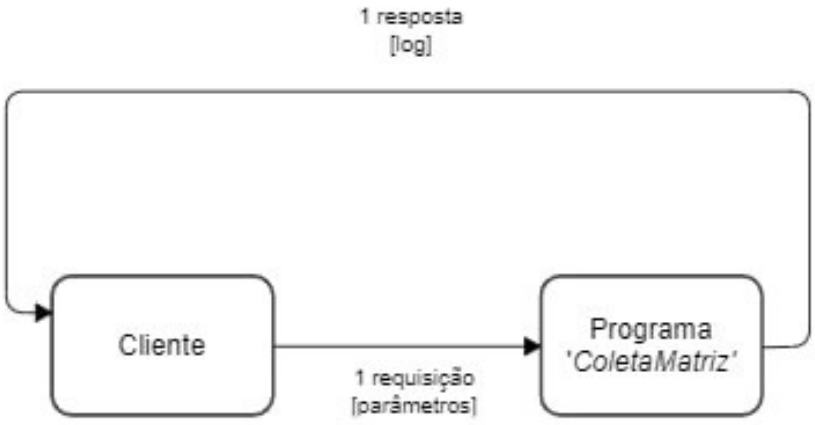
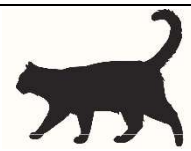


Figura 1: Diagrama de casos de uso



2. Desenvolvimento

Neste capítulo, será apresentado o desenvolvimento em si de *ColetaMatriz*. A linguagem utilizada, Java, é de preferência do autor, e, enquanto a implementação do algoritmo não deve ser exclusiva a apenas uma linguagem, a solução proposta a seguir é fortemente ligada a Java, porque são importados pacotes particulares, com comportamento nativo à linguagem.

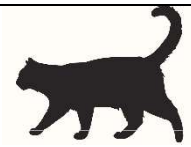
Portanto, o uso de outra tecnologia envolve o desenho da solução específico à essa escolha.

2.1. Desenho da solução

A partir dos Casos de Uso (seção 1.3.), o autor decidiu por fazer uma implementação simples, dirigida a funcionalidades*, adequada a fins lógico-aritméticos. A seguinte subseção apresenta os princípios de desenho (**design**) adotados:

&*Metodologias e padrões utilizados:*

- ✓ ***Programação dirigida a funcionalidades**: mesmo ao se utilizar Java, não é necessário explorar sempre a orientação a objetos. O autor entende que *POO* aplica um modelo de abstração-mapeamento com um propósito muito mais geral do que o necessário para esta aplicação. Por isso, o programa em questão é escrito em apenas uma abstract class, que contém um conjunto de static methods.
  Ou seja, a única classe de *ColetaMatriz* não pode ser instanciada, porque esse comportamento não é necessário nem previsto do desenho, e seus métodos são acessíveis estaticamente, sem particularidade instanciável entre as execuções.
  Por outro lado, isso não significa que paradigmas da orientação a objetos são explorados no desenho, pelo contrário. A escolha por Java foi tal que o autor pudesse usufruir dessa tecnologia e de seus pacotes inerentes, como a criação de objetos **File**. É a classe principal em si que deve ser funcional e não exibir comportamentos de orientação a objetos.



- ✓ **Encapsulamento**: o ato de encapsular campos (atributos) e métodos (funções), deixando apenas disponíveis publicamente os desenhados para esse devido mérito, é um "*key component*" no desenho de aplicações. Com essa metodologia, é garantido um nível previsto de segurança, que dispõe uma série de validações e tratamentos ao longo da **pipeline** de funcionamento do programa.
  Na seção 1.3., é citada a existência de "*corner cases*" dentro único caso de uso desta solução, e esses casos geram exceções, lançadas no Java a partir da tratativa throw e verificadas dentro de blocos try-catch. Esse dispositivo virtual de validação é potencializado ao se aplicar encapsulamento.
  Para encapsular métodos e atributos em Java, é compilado o modificador private (e, em outros casos, pode-se usar também protected). Todos os métodos da classe única são privados, exceto por dois, o típico main, e o método público (public), exposto intencionalmente para o acesso aos algoritmos do programa.
  Por fim, essa metodologia encaixa adequadamente na programação dirigida a funcionalidades. Essa relação não é obrigatória, até porque o encapsulamento faz parte dos paradigmas de *POO*, mas, neste caso, o fato de só existir uma classe, e esse ser projetada para ser funcional, também é um tipo de prática de encapsulamento, no caso o de ocultar a possibilidade de instanciação dessa classe.

- ✓ **Boas práticas**: O autor fez uma primeira implementação de *ColetaMatriz* (ainda com o nome de *ExemploMatriz*), mais enxuta, com menos suporte e que satisfazia plenamente o objetivo da simulação da coleta, e que corroborou na resposta correta do tópico do fórum.
  Posteriormente, o autor retomou a aplicação, para adicioná-la a seu portfólio. Foram escritas então as funções de log em arquivo de texto, e essa foi a segunda fase do agora projeto.
  Por fim, o autor instalou SonarLint* em sua IDE para fins corporativos, e este plug-in trouxe certas potencialidades a serem exploradas no âmbito de melhorias. Essas foram: a substituição de um método que gera o **path** constante do arquivo **.txt** criado como log por uma *constante estática* que guarda a mesma informação; ,a adoção de blocos *try-with-resources** em objetos de manipulação de arquivos e o uso de **StringBuilder*** para compor textos.

Nota: * *try-with-resources* e ***StringBuilder** são explicados na seção 2.2..

2.2. Apresentação de algoritmos

Esta seção sintetiza as duas anteriores, e a seguir será explicado de forma prática o alinhamento entre essas seções. Ao final, são apresentadas informações adicionais relevantes.

Os códigos apresentados seguirão o seguinte padrão de cores:

- package para pacotes;
- import para importações;
- class para classes;
- fields para campos (atributos);
- methods para métodos (funções).

Todo o código será apresentado aqui, incluindo a estruturação dos pacotes e das importações (importantes para a JVM).

✓ Pacotes (diretórios): **estruturação** →

- package br\com\JoaoPCarv\exemplo\tutorbrasil: diretórios do projeto.

```
package br.com.JoaoPCarv.exemplo.tutorbrasil;
```

Figura 2: Estrutura dos pacotes no código.

✓ Importações (dependências): **estruturação** →

- Import java.**[...]**:
  Todas as dependências de *ColetaMatriz* são feitas a partir de pacotes padrões do Java [.io para manipulação de entradas-saídas, .text para manipulação de textos, neste caso em formatação de data e hora, .time, como a API de caso geral para a contagem do tempo, .util para o serviço de coleções e outros utilitários].

```java
import java.io.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
```

Figura 3: Estrutura das importações no código.

✓ Classes: **estruturação →**

- class ColetaMatriz:
  Classe única, abstrata e funcional do projeto. Serve como um **container** para a aplicação.

```java
public abstract class ColetaMatriz{...}
```

Figura 4: Classe no código.

✓ Campos (atributos): **lógica →**

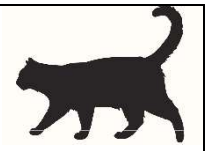
- field [**String**] **path**:
  Propriedade*, guarda a diretiva do caminho do relatório **.txt** dentre os diretórios do projeto. É um campo constante e privado (encapsulado), cuja manutenção é feita internamente pelo projetista.

```java
private static final String path = "report/Relatorio.txt";
```

Figura 5: Especificação do caminho do relatório no código.

Observações:

a) O modificador **final** marca classes que não devem ser herdadas, métodos que não devem ser sobrescritos e campos constantes.
b) *Propriedades são abordadas na seção 3.2.



✓ Métodos (funções): **lógica** →

- method getRule **[**entradas:
  {código da regra,
  elemento linha da entrada da matriz,
  elemento coluna da entrada da matriz**}**;
  saída:
  expressão lógica correspondente à regra.**]**

  Método *interno*, tratado por exceção de regra inválida, que retorna a expressão lógica entre o valor numérico entre os elementos linha e coluna da matriz, conforme o código da regra estabelecido. Por exemplo, para a primeira regra da *Tabela 1*, a função retorna (l < c).

```
private static boolean getRule(int regra, int l, int c) throws Exception {...}
```

Figura 6: Método getRule(...) no código.

  Este método lança uma exceção genérica se o código de regra passado não tiver sentido ou se extrapolar o range programado até então. Esse código é submetido a um bloco switch-case, o que facilita a escalabilidade caso o programador quiser inserir mais regras lógicas. O documento de texto **LeiaMeRegras.txt** faz o controle das regras já inseridas.

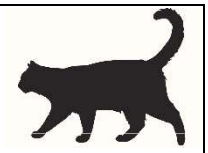
- method getMatriz **[**entradas:
  {número de linhas da matriz,
  número de colunas da matriz**}**;
  saída:
  matriz (array) com as dimensões pedidas.**]**

  Método *interno*, tratado por exceção de dimensões inválidas, que fornece uma matriz populada com as dimensões solicitadas pelo cliente.

```
private static double[][] getMatriz(int nLinhas, int nColunas) throws Exception {...}
```

Figura 7: Método getMatriz(...) no código.

  Este método lança uma exceção se ou o número de linhas ou o número de colunas forem menores ou iguais a zero, e preenche a



matriz com valores aleatórios, obtidos pelo método nativo Math.*random()* - retorno: tipo **double**, antes de entregar a solicitação.

- method getUniqueBufferedWriter**[**entrada: caminho do arquivo **.txt** de relatório; saída: escritor de arquivos padrão do Java.

Método *interno*, tratado por exceção de entrada-saída, que fornece ao **script** um objeto leitor de arquivos único, universal para todo o tempo de execução (objeto do tipo **BufferedWriter**).

```
private static BufferedWriter getUniqueBufferedWriter(String path) throws IOException {...}
```

Figura 8: Método getUniqueBufferedWriter(...) no código.

Essa técnica de compartilhamento se faz necessária porque são feitas várias escritas no log durante o caso "*happy day*", e sem uma unificação nas requisições e domínios de output, há conflito de uso, perdas de informação, segurança e eficiência. O objeto gerado neste método consome um arquivo e escreve no mesmo, e faz uso de **buffer**, que aliado à distribuição unificada deste mesmo objeto por toda execução, garante uma operação eficaz e segura (veja seção 2.3.). O arquivo passado é o gerado pelo método ColetaMatriz.*fileManager*(...), o que cria **binding** entre os algoritmos.

- method getDataFormatada**[**saída: data da execução formatada.**]**

Método *interno*, supostamente seguro, que fornece a data do tempo de execução em texto formatado.

```
private static String getDataFormatada() {...}
```

Figura 9: Método getDataFormatada(...) no código.

Este método não lança exceções, sendo confiado o uso seguro do utilitário de calendário do Java. A data é escrita em **String** segundo o formato '***dd/MM/yyyy***'.

- method getHorario**[**saída:
  horário da data de execução formatado.**]**

  Método *interno*, supostamente seguro, que retorna o horário do tempo de execução em texto formatado.

```
private static String getHorario() {...}
```

Figura 10: Método getHorario(...) no código.

  Este método não lança exceções, sendo confiado o uso seguro da classe de tempo local do pacote .time do Java. O horário é escrito em **String** segundo o formato '***hh:MM:ss***'.

- method fileManager**[**entrada:
  caminho do arquivo **.txt** do relatório;
  saída:
  arquivo do log instanciado.**]**

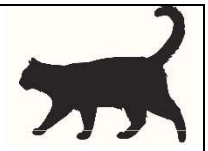
  Método *interno*, tratado por exceção de entrada-saída, que gerencia e fornece o arquivo do log instanciado em *ColetaMatriz*.

```
private static File fileManager(String path) throws IOException {...}
```

Figura 11: Método fileManager(...) no código.

  Esta função faz o seguinte gerenciamento: instancia em **File** o documento de texto de log e verifica se já existe um arquivo criado no diretório. Se não existe, é feita a tentativa de se criar o arquivo, podendo ser lançada uma **IOException** em circunstância crítica.

  Se já existe, o gerenciador procede lendo o texto gravado no documento por completo, até a última linha, chamada de **End of File** (*'eof'*).
  Considerando o caso "*happy day*", onde o roteiro é seguido da forma esperada, caso '*eof*' comece por "Executado em", isso significa que o log diz respeito a uma execução antiga, e todo o texto é apagado



para que este mesmo arquivo esteja limpo para o registro da execução em curso.

Do contrário, '*eof*' não começa por "Executado em", porque ainda estão sendo feitas as chamadas que escrevem os dados no documento, e o '*eof*'é temporário, estando em alguma linha intermediária. Assim, o escritor universal não apaga os registros sendo feitos no tempo da execução.

- Este método faz uso de *try-with-resources**, que são explicados ao final desta seção.

  - method getColetados**[**entradas:
    {código da regra,
    número de linhas da matriz,
    número de colunas da matriz,
    escritor universal do projeto**};**
    saída:
    lista de entradas coletadas a partir da regra.**]**

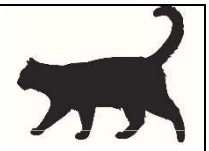
    Método *interno*, tratado por exceções de regra e dimensões inválidas de forma encadeada*, que retorna uma lista, em objeto do tipo escolhido **ArrayList**, contendo todas as células da matriz que foram coletadas pela regra passada.

```
private static List<Double> getColetados(int regra, int nLinhas, int nColunas, BufferedWriter bWriter) throws Exception {...}
```

Figura 12: Método getColetados(...) no código.

*Esta função possui lançamento de exceções encadeado, porque dentro de seu corpo são chamados os métodos ColetaMatriz.*getMatriz*(...) e ColetaMatriz.*getRule*(...) (**binding**), cujas exceções lançadas são discutidas nesta presente seção.

No cenário ideal, o método faz a ligação entre os trechos supracitados de *ColetaMatriz* e performa as ações de criação da matriz, coleta de células e registro no arquivo de log da coletas feitas.



❖ Este método faz uso de **StringBuilder***, que são explicados ao final desta seção.

- method comparadorDeColetas**[**entradas:
  {número de regras desejado*,
  número de linhas da matriz,
  número de colunas da matriz**}**;
  saída:
  sem saída - vácuo | **void.]**

Método *público*, invocador da aplicação, que faz o **binding** fundamental. Essa vinculação consiste em:

- criar o objeto único de **String** que guarda a Propriedade* do caminho do log para toda a execução, atribuindo-a univocamente ao field path (constante);
- invocar os métodos *internos* na ordem correta;
- iniciar o serviço de escrita, instanciando o escritor universal;
- escrever o '*eof*' definitivo no log, iniciado por "Executado em" e que grava as informações de data e hora da execução;
- unificar todos os tratamentos de exceções.

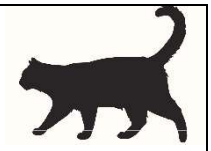
Esta função lança exceções no formato genérico, o que comtempla todas as exceções passíveis de serem lançadas por cada método em particular.

```
public static void comparadorDeColetas(int numeroDeRegras, int nLinhas, int nColunas) throws Exception {...}
```

Figura 13: Método comparadorDeColetas(...) no código.

É o método mestre | **master** de *ColetaMatriz*, evidenciado pelo seu nome, que sumariza a própria funcionalidade do projeto.

Se os argumentos passados forem inválidos, as exceções apropriadas são lançadas, e/ou se houver problema externo de **stream**, a exceção de entrada-saída é lançada.



Já no caso ideal, o programa imprime na saída padrão as mensagens da coleta e cria o relatório em arquivo **.txt**.

- Este método faz uso de *try-with-resources** e **StringBuilder***, que são explicados ao final desta seção.

  - method main**[**entrada:
    argumentos em texto que parametrizam a execução;
    saída:
    sem saída - vácuo | **void]**

    Método *público*, com assinatura típica, que invoca a execução padrão do Java.

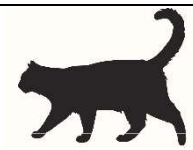
```
public static void main(String[] args) {
    try {
        comparadorDeColetas(...);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 14: Método main(...) no código.

Este método faz o controle maior de todas as exceções passíveis de serem lançadas com a invocação da função **master** através de um bloco try-catch. Este bloco faz a tentativa da execução de todos os algoritmos de *ColetaMatriz*, e se houver anormalidade (monitoradas por lançamento de exceções), a exceção é capturada e tratada, sendo o rastreamento de **error streams** impresso na saída padrão.

Além da apresentação dos códigos, também é relevante expor outras partes operacionais do **script**, sendo essas o uso de comentários (partes não executáveis) e o uso de *try-with-resources** e **StringBuilder***, adições estas feitas a partir de recomendações do SonarLint* (conforme já dito).

*SonarLint é um **plug-in** que, ativado junto à IDE, provê ao programador revisão de código a nível de **clean code**.



- ✓ Quanto aos comentários, as linhas começadas por // são seus identificadores. Fazem parte da documentação e são informações em código.

```
//Classe abstrata, cujos todos métodos são estáticos, por se tratar de uma classe puramente funcional,
// sem a necessidade de instanciação.
```

Figura 15: Exemplo de comentário no código.

- ✓ **try-with-resources* é uma técnica introduzida no Java 7[1] que manipula recursos (neste contexto, manipuladores de dados) de forma segura, ao envolver a criação e o uso desses recursos em um bloco try, de forma que independentemente do êxito dos serviços planejados, o recurso em si é fechado automaticamente ao final do bloco. Ou seja, é o uso de um gerenciador de recursos automatizado pela própria linguagem.

```
try (BufferedWriter uniqueWriter = ColetaMatriz.getUniqueBufferedWriter(path)) {...}
```

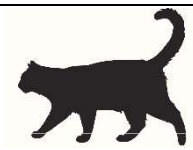
Figura 16: Exemplo de try-with-resources no código.

- ✓ ***StringBuilder** é uma classe que administra uma sequência mutável de caracteres[2]. É adequada para a manipulação dinâmica de textos, provendo ao projetista uma ferramenta uniforme para esse fim. Em *ColetaMatriz*, objetos dessa classe são criados e com eles são invocados o método *append*(...) para a composição contínua de textos, ao invés da concatenação clássica [(...) **+=** (...)]. Por fim, para a obtenção do texto composto, utiliza-se o método *toString*(). É um suprimento elegante.

```
StringBuilder msgStream = new StringBuilder();
{...}
msgStream.append(...);
{...}
msgStream.toString();
```

*Figura 17: Exemplo de uso (ciclo completo) de **StringBuilder** no código*

São estes os principais pontos dos algoritmos deste projeto.



2.3. Testes

Esta seção discute tanto os testes feitos pelo autor durante a fase de desenvolvimento, quanto o teste final do software, injetando exatamente a requisição do enunciado da questão.

Após a retomada do projeto, durante a fase de desenvolvimento, o autor inicialmente fazia requisições de escrita no **script** independentes e não acopladas entre si, o que acontecessem sobrescritas indesejadas no arquivo de log, que configurou tanto perda de informação quanto ineficiência do uso dos serviços de **IO** por parte do autor. Esses comportamentos não previstos foram observados através de testes intermediários dos blocos de funções do código.

Conforme o descrito em 2.2., a solução desse erro foi a adoção de um escritor universal, compartilhado por todas as requisições da aplicação, e, ao se substituir os vários objetos de escrita por esse único, o código foi novamente testado e esse teste foi um sucesso, concluindo assim o ciclo de implementação de uma solução a uma adversidade presente no **runtime**.

Após o evidenciado acima, a contribuição do SonarLint trouxe propostas de **clean code**, explicadas em 2.2.. O autor acatou algumas dessas propostas e fez as manutenções relativas no código, o que levou a mais um ciclo de testes. Com a aprovação destes, mais um ciclo de produção foi concluído.

Sendo relatados esses pontos, a seguir encontra-se o **print** de um teste com a injeção dos parâmetros originais do enunciado – as 5 regras da *Tabela 1* e a matriz em análise dimensionada em 12x5.

- **Teste padrão**: **[** →*Requisição*: ciclo de vida de *ColetaMatriz*;;
  →*Injeção*: argumentos do enunciado;
  →*Expectativa*: impressão da coleta e criação instantânea de log;**]**

```java
public static void main(String[] args) {
    try {
        comparadorDeColetas( numeroDeRegras: 5,  nLinhas: 12,  nColunas: 5);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 18: Caracterização do teste padrão.

Configurado o teste padrão, o seu processamento gerou duas saídas:

- Saída impressa por **System**.*out*.*println*(...):

```
C:\Users\joaop\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.3\lib\idea_rt.jar=56989:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.3\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "G:\Meu Drive\DOCUMENTOS JP\POLI-USP\Projetos em Java\AlgoritmosEmJava\TutorBrasil\JoaoPCarv\ColetaMatriz\out\production\ColetaMatriz" br.com.JoaoPCarv.exemplo.tutorbrasil.ColetaMatriz
Coletado pela regra 1: elemento A[0][1];
Coletado pela regra 1: elemento A[0][2];
Coletado pela regra 1: elemento A[0][3];
Coletado pela regra 1: elemento A[0][4];
Coletado pela regra 1: elemento A[1][2];
Coletado pela regra 1: elemento A[1][3];
Coletado pela regra 1: elemento A[1][4];
Coletado pela regra 1: elemento A[2][3];
Coletado pela regra 1: elemento A[2][4];
Coletado pela regra 1: elemento A[3][4];
A regra 1 coletou 10 elementos da matriz.

Coletado pela regra 2: elemento A[0][0];
Coletado pela regra 2: elemento A[1][1];
Coletado pela regra 2: elemento A[2][2];
Coletado pela regra 2: elemento A[3][3];
Coletado pela regra 2: elemento A[4][4];
A regra 2 coletou 5 elementos da matriz.

Coletado pela regra 3: elemento A[2][4];
Coletado pela regra 3: elemento A[3][3];
Coletado pela regra 3: elemento A[4][2];
Coletado pela regra 3: elemento A[5][1];
Coletado pela regra 3: elemento A[6][0];
A regra 3 coletou 5 elementos da matriz.

Coletado pela regra 4: elemento A[1][0];
Coletado pela regra 4: elemento A[2][0];
Coletado pela regra 4: elemento A[2][1];
Coletado pela regra 4: elemento A[3][0];
Coletado pela regra 4: elemento A[3][1];
Coletado pela regra 4: elemento A[3][2];
Coletado pela regra 4: elemento A[4][0];
Coletado pela regra 4: elemento A[4][1];
Coletado pela regra 4: elemento A[4][2];
Coletado pela regra 4: elemento A[4][3];
Coletado pela regra 4: elemento A[5][0];
Coletado pela regra 4: elemento A[5][1];
Coletado pela regra 4: elemento A[5][2];
Coletado pela regra 4: elemento A[5][3];
Coletado pela regra 4: elemento A[5][4];
Coletado pela regra 4: elemento A[6][0];
Coletado pela regra 4: elemento A[6][1];
Coletado pela regra 4: elemento A[6][2];
Coletado pela regra 4: elemento A[6][3];
Coletado pela regra 4: elemento A[6][4];
Coletado pela regra 4: elemento A[7][0];
Coletado pela regra 4: elemento A[7][1];
Coletado pela regra 4: elemento A[7][2];
Coletado pela regra 4: elemento A[7][3];
Coletado pela regra 4: elemento A[7][4];
Coletado pela regra 4: elemento A[8][0];
Coletado pela regra 4: elemento A[8][1];
Coletado pela regra 4: elemento A[8][2];
Coletado pela regra 4: elemento A[8][3];
Coletado pela regra 4: elemento A[8][4];
Coletado pela regra 4: elemento A[9][0];
Coletado pela regra 4: elemento A[9][1];
Coletado pela regra 4: elemento A[9][2];
Coletado pela regra 4: elemento A[9][3];
Coletado pela regra 4: elemento A[9][4];
Coletado pela regra 4: elemento A[10][0];
Coletado pela regra 4: elemento A[10][1];
Coletado pela regra 4: elemento A[10][2];
Coletado pela regra 4: elemento A[10][3];
Coletado pela regra 4: elemento A[10][4];
Coletado pela regra 4: elemento A[11][0];
Coletado pela regra 4: elemento A[11][1];
Coletado pela regra 4: elemento A[11][2];
Coletado pela regra 4: elemento A[11][3];
Coletado pela regra 4: elemento A[11][4];
A regra 4 coletou 45 elementos da matriz.

Coletado pela regra 5: elemento A[6][0];
Coletado pela regra 5: elemento A[7][0];
Coletado pela regra 5: elemento A[7][1];
Coletado pela regra 5: elemento A[8][0];
Coletado pela regra 5: elemento A[8][1];
Coletado pela regra 5: elemento A[8][2];
Coletado pela regra 5: elemento A[9][0];
Coletado pela regra 5: elemento A[9][1];
Coletado pela regra 5: elemento A[9][2];
Coletado pela regra 5: elemento A[9][3];
Coletado pela regra 5: elemento A[10][0];
Coletado pela regra 5: elemento A[10][1];
Coletado pela regra 5: elemento A[10][2];
Coletado pela regra 5: elemento A[10][3];
Coletado pela regra 5: elemento A[10][4];
Coletado pela regra 5: elemento A[11][0];
Coletado pela regra 5: elemento A[11][1];
Coletado pela regra 5: elemento A[11][2];
Coletado pela regra 5: elemento A[11][3];
Coletado pela regra 5: elemento A[11][4];
A regra 5 coletou 20 elementos da matriz.


A regra 4 foi quem coletou mais dados.


Process finished with exit code 0

```

Figura 19: Saída impressa na saída padrão.

- Saída impressa no arquivo **Relatorio.txt**:

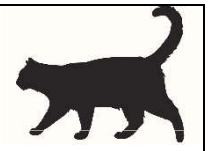
```
Coletado pela regra 1: elemento A[0][1]; Coletado pela regra 4: elemento A[6][2]; Coletado pela regra 5: elemento A[10][1];
Coletado pela regra 1: elemento A[0][2]; Coletado pela regra 4: elemento A[6][3]; Coletado pela regra 5: elemento A[10][2];
Coletado pela regra 1: elemento A[0][3]; Coletado pela regra 4: elemento A[6][4]; Coletado pela regra 5: elemento A[10][3];
Coletado pela regra 1: elemento A[0][4]; Coletado pela regra 4: elemento A[7][0]; Coletado pela regra 5: elemento A[10][4];
Coletado pela regra 1: elemento A[1][2]; Coletado pela regra 4: elemento A[7][1]; Coletado pela regra 5: elemento A[11][0];
Coletado pela regra 1: elemento A[1][3]; Coletado pela regra 4: elemento A[7][2]; Coletado pela regra 5: elemento A[11][1];
Coletado pela regra 1: elemento A[1][4]; Coletado pela regra 4: elemento A[7][3]; Coletado pela regra 5: elemento A[11][2];
Coletado pela regra 1: elemento A[2][3]; Coletado pela regra 4: elemento A[7][4]; Coletado pela regra 5: elemento A[11][3];
Coletado pela regra 1: elemento A[2][4]; Coletado pela regra 4: elemento A[8][0]; Coletado pela regra 5: elemento A[11][4];
Coletado pela regra 1: elemento A[3][4]; Coletado pela regra 4: elemento A[8][1];

                                         Coletado pela regra 4: elemento A[8][2]; A regra 1 coletou 10 elementos da matriz.
Coletado pela regra 2: elemento A[0][0]; Coletado pela regra 4: elemento A[8][3]; A regra 2 coletou 5 elementos da matriz.
Coletado pela regra 2: elemento A[1][1]; Coletado pela regra 4: elemento A[8][4]; A regra 3 coletou 5 elementos da matriz.
Coletado pela regra 2: elemento A[2][2]; Coletado pela regra 4: elemento A[9][0]; A regra 4 coletou 45 elementos da matriz.
Coletado pela regra 2: elemento A[3][3]; Coletado pela regra 4: elemento A[9][1]; A regra 5 coletou 20 elementos da matriz.
Coletado pela regra 2: elemento A[4][4]; Coletado pela regra 4: elemento A[9][2];
                                         Coletado pela regra 4: elemento A[9][3]; A regra 4 foi quem coletou mais dados.
Coletado pela regra 3: elemento A[2][4]; Coletado pela regra 4: elemento A[9][4];
Coletado pela regra 3: elemento A[3][3]; Coletado pela regra 4: elemento A[10][0]; Executado em 15/03/2023 às 19:20:14.
Coletado pela regra 3: elemento A[4][2]; Coletado pela regra 4: elemento A[10][1];
Coletado pela regra 3: elemento A[5][1]; Coletado pela regra 4: elemento A[10][2];
Coletado pela regra 3: elemento A[6][0]; Coletado pela regra 4: elemento A[10][3];
                                         Coletado pela regra 4: elemento A[10][4];
Coletado pela regra 4: elemento A[1][0]; Coletado pela regra 4: elemento A[11][0];
Coletado pela regra 4: elemento A[2][0]; Coletado pela regra 4: elemento A[11][1];
Coletado pela regra 4: elemento A[2][1]; Coletado pela regra 4: elemento A[11][2];
Coletado pela regra 4: elemento A[3][0]; Coletado pela regra 4: elemento A[11][3];
Coletado pela regra 4: elemento A[3][1]; Coletado pela regra 4: elemento A[11][4];
Coletado pela regra 4: elemento A[3][2];
Coletado pela regra 4: elemento A[4][0]; Coletado pela regra 5: elemento A[6][0];
Coletado pela regra 4: elemento A[4][1]; Coletado pela regra 5: elemento A[7][0];
Coletado pela regra 4: elemento A[4][2]; Coletado pela regra 5: elemento A[7][1];
Coletado pela regra 4: elemento A[4][3]; Coletado pela regra 5: elemento A[8][0];
Coletado pela regra 4: elemento A[5][0]; Coletado pela regra 5: elemento A[8][1];
Coletado pela regra 4: elemento A[5][1]; Coletado pela regra 5: elemento A[8][2];
Coletado pela regra 4: elemento A[5][2]; Coletado pela regra 5: elemento A[9][0];
Coletado pela regra 4: elemento A[5][3]; Coletado pela regra 5: elemento A[9][1];
Coletado pela regra 4: elemento A[5][4]; Coletado pela regra 5: elemento A[9][2];
Coletado pela regra 4: elemento A[6][0]; Coletado pela regra 5: elemento A[9][3];
Coletado pela regra 4: elemento A[6][1]; Coletado pela regra 5: elemento A[10][0];
```

Figura 20: Saída impressa no arquivo de log.

A análise do resultado impresso deste teste corresponde à expectativa, pois ambas as impressões estão de acordo com o esperado e a execução foi instantânea, finalizando com **code 0**. Por conseguinte, conclui-se que o teste padrão foi um sucesso, e *ColetaMatriz* funciona corretamente.

- Conclusão do **teste padrão**: Sucesso ✓.



3. Conclusão

Partindo de uma ideia inicial, voltada à didática e complementar à resposta postada no fórum pelo autor, *ColetaMatriz* passou por uma maturação e atualmente é um software acertadamente finalizado.

As seções a seguir documentam as considerações finais desse projeto do ponto de vista do autor.

3.1. Análise da solução | overview

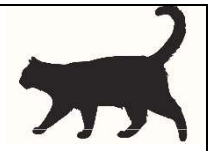
O aplicativo *ColetaMatriz* foi desenvolvido para a premissa inicial de simulação do exercício descrito no tópico do fórum TutorBrasil, mas durante as suas fases de desenvolvimento, implementou técnicas que agregaram valor ao produto final.

Desde o **script** inicial, *ColetaMatriz* (ainda sob o nome de *ExemploMatriz*) já satisfazia a problemática basal de coleta de células de *arrays* conforme regras lógicas entre seus índices de linhas e colunas, e a agregação dessas funcionalidades demandou novos testes para a averiguação da qualidade da solução. Com base no sucesso dos testes (vide seção 2.3.), conclui-se que o projeto foi bem-sucedido.

O aplicativo trabalha com uma execução única e instantânea, o que vai ao encontro com o caso de uso 1:1 simples (seção 1.3.), e em última análise, é a evidência empírica do ciclo de vida bastante básico pensado pelo autor desde a primeira versão, porque não convém criar uma complexidade de diagramas para uma solução numérica e de uso acessório a um tema maior.

Em face destes dois últimos parágrafos, o autor conclui que este software cumpre a qualidade de eficiência aliada à viabilidade segundo o propósito máximo da problemática.

Em todo o caso, implementação de valores e uso de práticas não se limitam ao que foi abordado nesta proposta. A próxima seção expõe novas possibilidades, pelo menos aquelas apreciadas pelo autor.



3.2. Limitações e melhorias

Premissas, **skills** e escopos produzem discussões qualitativas em relação aos projetos de soluções computacionais, e um ponto importante e interessante é decidir o que deve ser feito: quais são as prioridades e o que é relevante (ou não) de ser incluso no **design**. Sobrepondo tais discussões a *ColetaMatriz*, o autor traz, nessa seção, os seguintes pontos:

- Limitação: O projeto não possui interface gráfica alguma, muito menos **user friendly**;
  - Melhoria: Criação de um subsistema de telas com o pacote javax.swing do Java para o fornecimento de interface gráfica; O autor propõe essa abordagem porque as telas **Swing** possuem nível de **Desktop**, sendo a implementação mais fácil;
    - Comentário: O autor classifica esta como o par limitação-melhoria mais pertinente desta série.

- Limitação: O dado **path** é declarado como um field constante, o que vai de encontro a boas práticas relativas a Propriedades*;
  - Melhoria: O uso de *Propriedades em Java pode ser feito de forma padrão pela classe **Properties**, o que abre a possibilidade de aplicação desse padrão no projeto;
    - Comentário: O autor considera essa uma melhoria robusta demais para o escopo desta proposta.

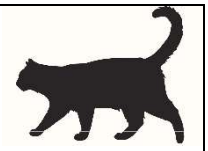
- Limitação: O arquivo de controle interno **LeiaMeRegras.txt** é mantido separadamente do código, e sua manutenção é manual.
  - Melhoria: Automatização do controle desse arquivo incluído nos algoritmos do projeto, usando inclusive **Properties**;
    - Comentário: O autor considera essa uma melhoria robusta demais para o escopo desta proposta.

- Limitação: O SonarLint recomenda que a saída padrão utilizada não é apropriada;
    - Melhoria: O próprio **plug-in** recomenda a implementação de objetos **Logger** para mostrar mensagens ao usuário adequadamente;
        - Comentário: O autor considera essa uma melhoria robusta demais para o escopo desta proposta.

- Limitação: O SonarLint aponta que exceções de estados particulares do programa são lançadas usando um objeto genérico;
    - Melhoria: Criação de exceções dedicadas aos "*corner cases*" do ciclo de vida;
        - Comentário: O autor considera essa uma melhoria robusta demais para o escopo desta proposta.

- Limitação*: O SonarLint sugere que marcadores fortemente tipados caracterizam má prática;
    - Melhoria*: Substituição de declarações fortemente tipadas no código pelo uso da diretiva **var**, introduzida no Java 10[3];
        - **Comentário: O autor não interpreta este levantamento como uma limitação de fato, e comunidades[4] sugerem que tal modificação não resulta em uma melhoria perceptível.

- Limitação*: Em todo o projeto foi adotado **camelCase**, porém nomenclaturas estão mescladas entre português e inglês, o que pode ser discutido como não sendo boa prática;
    - Melhoria*: Adoção de convenção de nomenclaturas em uma só língua, preferencialmente inglês;
        - **Comentário: O autor não considera tal descuido como uma limitação de fato, mas a resolução devida é relativamente simples com o uso de **refactoring**. O autor preservou as nomenclaturas originais.



3.3. Considerações finais

ColetaMatriz foi um projeto simples, mas que levou o autor a novos conhecimentos e cuidados quanto à formalidade de projetos. A documentação em si também foi uma adição aos interesses do autor, sendo este relatório o primeiro de um modelo padronizado.

Como último adendo, a seção [2].3. – Testes não trouxe exemplos dos vários casos de falha para deixar este relatório mais conciso. O mesmo vale para a não exibição de trechos de código inteiros, como os de um método completo.

4. Referências

1. JAVA – Try with Resources. **Baeldung**, 2022. Disponível em: <https://www.baeldung.com/java-try-with-resources>. Acesso em: 14 de mar. de 2023.
2. STRINGBUILDER (Java Platform SE 7) - Oracle Help Center. **Oracle**, 2023. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html>. Acesso em: 14 de mar. de 2023.
3. GOETZ, Brian. JEP 286: Local-Variable Type Inference. **OpenJDK**, 2016. Disponível em: <https://openjdk.org/jeps/286>. Acesso em: 14 de mar. de 2023.
4. WHEN to use var in Java 10. **The First Cry of Atom**, 2020. Disponível em: <https://www.lewuathe.com/when-to-use-var-in-java-10.html>. Acesso em: 14 de mar. de 2023.