INSTITUTO FEDERAL
Bahia
Campus Vitória da Conquista

EMPILHAMENTO DE CONTÊINERES EM PÁTIO PORTUÁRIO
RELATÓRIO DE DESENVOLVIMENTO

ALLAN FRANÇA DUTRA

FERNANDA BARBOSA MENDES FERRAZ

FRANCISCO MIGUEL MEIRA PESSOA

GUSTAVO XAVIER FARIAS

MILCA VIEIRA SOUZA

SÉRGIO GABRIEL MENDES LIMA

Julho, 2022

Vitória da Conquista, Bahia

INSTITUTO FEDERAL
Bahia
Campus Vitória da Conquista

EMPILHAMENTO DE CONTÊINERES EM PÁTIO PORTUÁRIO

Relatório de desenvolvimento de aplicativo apresentado como requisito parcial para obtenção de aprovação na disciplina de Linguagem de Programação II, do Curso de Bacharelado em Sistemas de Informação, do Instituto Federal de Educação, Ciência e Tecnologia da Bahia, Campus Vitória da Conquista.

Prof. Alexandro dos Santos Silva

ALLAN FRANÇA DUTRA

FERNANDA BARBOSA MENDES FERRAZ

FRANCISCO MIGUEL MEIRA PESSOA

GUSTAVO XAVIER FARIAS

MILCA VIEIRA SOUZA

SÉRGIO GABRIEL MENDES LIMA

Julho, 2022

Vitória da Conquista, Bahia

SUMÁRIO

1. INTRODUÇÃO

A aplicação desenvolvida propõe simular a atividade de um guindaste em um pátio de determinado terminal portuário que faz a função de empilhamento de contêineres que podem ser movimentados de caminhões de carga para as posições de empilhamento e vice-versa. Para simplificação do projeto, apenas uma quadra do pátio foi considerada, dividida em 12 (doze) posições de empilhamento identificadas por letras, de A até L. Além disso, pode-se ser empilhado no máximo 06 (seis) contêineres por posição de empilhamento.

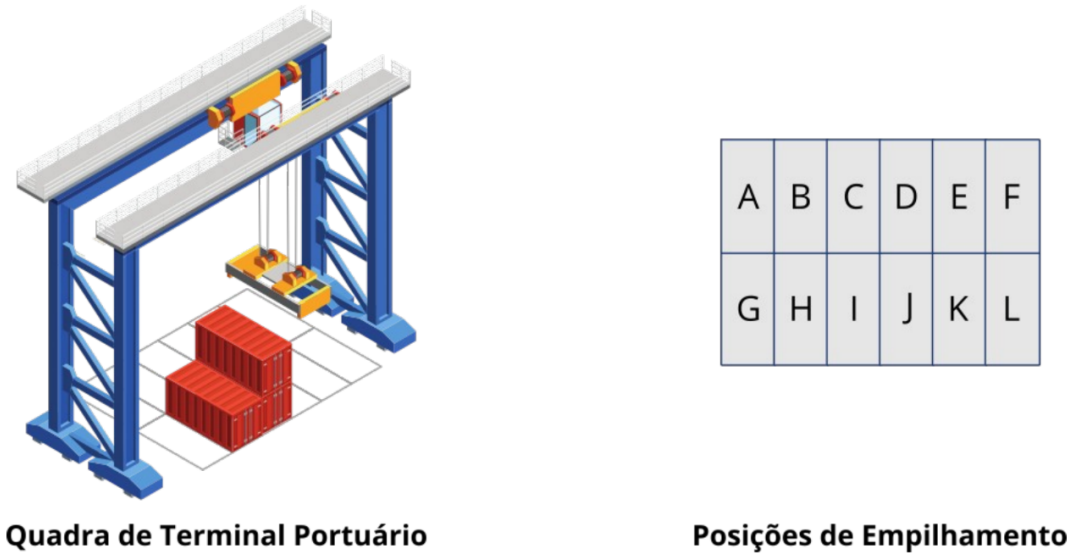

Figura 01 - Quadra de Terminal Portuário e a divisão de Posições de Empilhamento

Outro detalhe importante na aplicação é que para cada contêiner que for empilhado, lhe será atribuído um identificador de valor único, e o mesmo poderá ser utilizado pelo usuário para, por exemplo, desempilhar um contêiner específico. O identificador é na forma **ch.n**, onde **ch** corresponde à posição de empilhamento: letra A a L, e **n** corresponde ao número sequencial de inserção naquela posição. Alguns exemplos de identificadores possíveis são: **A.1**, **B.5**, **D.9**, **L.2**.

Desse modo, para funcionamento do aplicativo, possui no sistema uma pilha (Deque) com 06 (seis) espaços para cada posição de empilhamento que serão preenchidos pelos contêineres. Soma-se a isso, a estrutura de dados Mapa (HashMap) que é a responsável por guardar as 12 (doze) posições de empilhamento enquanto o código estiver sendo compilado.

Os contêineres a serem empilhados durante a aplicação possuem dados que primeiro deverão ser registrados pelo usuário: nome do proprietário, tipo de carga (carga seca, commodities, produtos perigosos ou produtos perecíveis), peso da carga (em quilogramas), tipo de operação (embarque ou desembarque em navio) e a posição de empilhamento que o usuário quer empilhar o contêiner.

2. CONTEXTUALIZAÇÃO

Ao iniciar o aplicativo, o usuário desfrutará de 04 (quatro) opções de funcionalidades disponíveis pelo sistema: 1. Empilhar um contêiner, 2. Desempilhar um contêiner, 3. Consultar Contêineres e 4. Sair. O usuário deve digitar entre os números 1 e 4, qualquer outro caractere será considerado erro e o sistema pedirá por uma entrada válida.

Caso seja escolhido a 1º opção, o usuário então irá cadastrar os dados do contêiner e ele será devidamente empilhado na posição de empilhamento escolhida, caso haja espaço disponível. Não possuindo espaço disponível, será alertado pelo sistema e o usuário poderá escolher outra posição que estiver livre.

Para a 2º opção, o usuário deverá informar o Identificador do contêiner, e se o mesmo estiver empilhado, será portanto desempilhado. Caso não esteja, será informado ao usuário. Além disso, para essa opção existe uma restrição por se está sendo utilizado pilhas para armazenar os contêineres: Em caso do contêiner desejado para desempilhar não esteja no topo da pilha, os outros contêineres acima deste serão desempilhados, o contêiner será retirado e os outros empilhados novamente.

Para a 3° opção, o usuário será apresentado com outro Menu de Operações para escolher entre: a) Dados do contêiner no topo de uma posição de empilhamento, b) Quantidade de contêineres por tipo de carga, c) Peso total de cargas empilhadas por tipo de carga, d) Quantidade de contêineres por tipo de operação, e) Posições de empilhamentos vazias e f) Gráfico que mostra a relação de quantos contêineres há em cada Posição de empilhamento..

No caso da opção a), o usuário também deverá informar qual posição de empilhamento ele deseja ver os dados do contêiner no topo. Portanto, ele deve escrever entre as letras A até L, caso escreva outra letra ou qualquer outro caractere, o sistema solicitará uma entrada válida. Por fim, caso a posição de empilhamento estiver vazia, também será avisado ao usuário. As outras opções apresentarão os dados que o usuário desejou consultar.

A 4º Opção encerrará o aplicativo, perdendo assim qualquer Contêiner que tenha sido previamente cadastrado durante a execução do sistema.

3. IMPLEMENTAÇÃO

Para criação do aplicativo foi utilizado o Diagrama de Classes previamente feito para organização de idéias e melhor entendimento de como seria a relação das classes. No fim, o código foi dividido em 04 (quatro) classes: **App**, **Conteiner**, **Pilha** e **Empilhador**. Além das classes, também foram criadas 02 (duas) *enums*: **TipoCarga** e **TipoOperacao**.

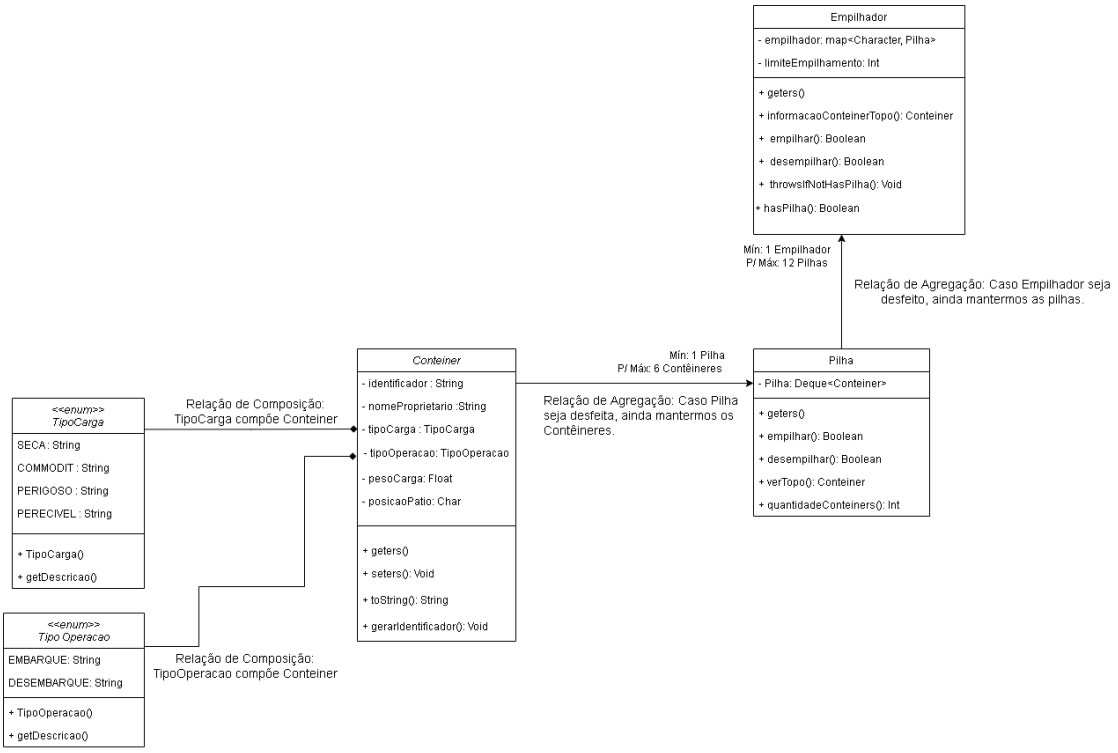

Figura 02 - Diagrama de Classes

3.1 - Classe App

A classe **App** é onde está o código responsável pela interface que o usuário interage para utilizar o aplicativo. Para registrar as respostas do usuário que permitem utilizar as funcionalidades do sistema, é usado o *Scanner* que foi importado no começo do código.

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Scanner in = new Scanner(System.in);
        Empilhador empilhador = new Empilhador();
        boolean sair = false;
        System.out.println("Bem vindo ao gestor de Contêiner Automatizado");
        while (!sair) {
            System.out.println("                              ");
            System.out.println("  1 - Empilhar Contêiner      ");
            System.out.println("  2 - Desempilhar Contêiner   ");
            System.out.println("  3 - Consultar Contêineres   ");
            System.out.println("  4 - Sair                    ");
            System.out.println("                              ");
            int opcao = 0;
            boolean executado = false;
            do {
                try {
                    System.out.println("Digite a opção desejada: ");
                    opcao = in.nextInt();
                    executado = true;
                } catch (Exception e) {
                    in = new Scanner(System.in);
                    System.out.println("Opção inválida, tente novamente!");
                    continue;
                }
            } while (!executado);
```

Figura 03 - Código de Interface na Classe App

Para as diferentes funcionalidades foi utilizado o *switch case* com a variável **opcao**, ou seja, em caso do valor de **opcao** ser igual a 1, 2, 3 ou 4, algo diferente será executado. Se o usuário escolhesse 2, o case 2 seria utilizado para desempilhar algum contêiner.

```java
case 2:
    valido = false;
    do {
        try {
            System.out.println("Digite o identificador do contêiner que deseja desempilhar: ");
            String identificador = in.next();
            if (empilhador.desempilhar(identificador)) {
                System.out.println("Contêiner desempilhado com sucesso!");
            }
            valido = true;
        } catch (Exception e) {
            in = new Scanner(System.in);
            System.out.println("Identificador inválido, tente novamente");
            continue;
        }
    } while (!valido);
    break;
```

Figura 04 - Código do case 2 na Classe App

3.2 - Classe Empilhador

No case 2 é utilizado o método **desempilhar()** que faz parte de um dos métodos presentes na classe **Empilhador**, assim como esse os outros métodos também são chamados pela classe **App**.

```
30  public boolean desempilhar(String indentificador) throws IllegalArgumentException {
31      char posicao = indentificador.toUpperCase().charAt(index: 0);
32
33      this.throwsIfNotHasPilha(posicao);
34
35      return pilhas.get(posicao).desempilhar(indentificador);
36
37  }
```

Figura 05 - Código do método desempilhar() dentro da classe Empilhador

O **Empilhador** é a classe responsável pelo armazenamento das Posições de Empilhamento, para isso a estrutura de dados utilizada foi o HashMap onde um Caractere (A até L) será a chave para a Pilha de respectiva letra.

```
1   import java.util.HashMap;
2   import java.util.Map;
3
    ...
4   public class Empilhador {
5       private Map<Character, Pilha> pilhas = new HashMap<>();
6       private int limiteEmpilhamento = 6;
7
8       Empilhador() {
9           for (int i = 0; i < 12; i++) {
10              char letra = (char) (i + 65);
11              pilhas.put(letra, new Pilha(letra, limiteEmpilhamento));
12          }
13      }
14
```

Figura 06 - Código do Mapa de pilhas na classe Empilhador

Em **Empilhador** também está sendo usado métodos presentes em outra classe, neste caso a classe **Pilha**.

```
71  public Map<String, Float> getPesoTotalPorTipoCarga() {
72      Map<String, Float> pesoTotalPorTipoCarga = new HashMap<>();
73      for (TipoCarga t : TipoCarga.values()) {
74          float peso = 0;
75          for (Pilha pilha : pilhas.values()) {
76              peso += pilha.getPesoTotalPorTipoCarga(t);
77          }
78          pesoTotalPorTipoCarga.put(t.getDescricao(), peso);
79      }
80
81      return pesoTotalPorTipoCarga;
82  }
```

Figura 07 - Código do método getPesoTotalPorTipoCarga() na classe Empilhador

Esses métodos *get* como o **getPesoTotalPorTipoCarga()** são necessários, pois essas estruturas de dados são privadas, então não podem ser acessadas diretamente de uma classe diferente de onde foram criadas.

3.3 - Classe Pilha

Passando para a classe **Pilha**, é onde estão sendo empilhados os Contêineres, utilizando da estrutura ArrayDeque, uma Fila que permite adição ou remoção de elementos em suas duas extremidades. Assim como o **pilhas** é um Mapa privado, **pilha** é uma Fila privada.

```
4   public class Pilha {
5
6       private Deque<Conteiner> pilha = new ArrayDeque<>();
7       private int contador = 1;
8       private int limiteConteiners;
9       private char nome;
10
11      Pilha(char nome, int limiteConteiners) {
12          this.nome = nome;
13          this.limiteConteiners = limiteConteiners;
14      }
```

Figura 08 - Código da Fila de Conteiner na classe Pilha

Os métodos presentes na classe **Pilha** são utilizados para as consultas, 3º Opção do Menu Principal. O método **getPesoTotalPorTipoCarga()**, para conseguir somar os pesos em cada contêiner utiliza uma pilha auxiliar que é criada dentro do método que guarda os contêineres que vão sendo retirados e depois realocados novamente na pilha.

```java
public float getPesoTotalPorTipoCarga(TipoCarga t) {
    float peso = 0;

    Deque<Conteiner> pilhaAux = new ArrayDeque<Conteiner>();
    for (int i = 0; i < pilha.size(); i++) {
        Conteiner conteiner = pilha.pop();
        if (conteiner.getTipoCarga() == t) {
            peso += conteiner.getPesoCarga();
        }
        pilhaAux.push(conteiner);
    }
    for (int i = 0; i < pilhaAux.size(); i++) {
        pilha.push(pilhaAux.pop());
    }

    return peso;
}
```

Figura 09 - Código do método getPesoTotalPorTipoCarga() na classe Pilha

3.4 - Classe Conteiner

Por fim, na classe **Conteiner** é onde os dados de nome do proprietário, tipo e peso de Carga e etc. são guardados. Esses dados são privados e também possuem seus *gets* e *sets*. Além disso, também existe o construtor dentro da classe.

```java
public class Conteiner {
    private TipoCarga tipoCarga;
    private TipoOperacao tipoOperacao;
    private String identificador;
    private String nomeProprietario;
    private float pesoCarga;
    private char posicaoPatio;
    public Conteiner(TipoCarga tipoCarga, TipoOperacao tipoOperacao,
                     String nomeProprietario, float pesoCarga, char posicaoPatio) {
        this.tipoCarga = tipoCarga;
        this.tipoOperacao = tipoOperacao;
        this.nomeProprietario = nomeProprietario;
        this.pesoCarga = pesoCarga;
        this.posicaoPatio = posicaoPatio;
    }
```

Figura 10 - Código do construtor Conteiner e as variáveis na classe Conteiner

Na classe **Conteiner** é onde está sendo gerado o identificador único para cada contêiner. O identificador só é gerado a partir do momento que o contêiner é empilhado com sucesso pelo

usuário ao utilizar o aplicativo. É utilizado da variável **posicaoPatio** concatenada com **posicaoPilha** e separadas por um ponto.

```
16      public void gerarIdentificador(int posicaoPilha){
17          this.identificador = this.posicaoPatio+"."+posicaoPilha;
18      }
```

Figura 11 - Código do método gerarIdentificador() na classe Conteiner

No caso das variáveis **tipoCarga** e **tipoOperacao**, como já dito antes são *enums*, foi feito dessa maneira para que o usuário escolha entre os diferentes tipos de Carga e Operações, invés de escrever. Assim, evitando o problema de um mesmo tipo ser salvo como diferente, por conta de erro de digitação.

```
1   public enum TipoCarga {
2       SECA(descricao: "Seca"),
3       COMMODIT(descricao: "Commodit"),
4       PERIGOSO(descricao: "Produtos Perigosos"),
5       PERECIVEL(descricao: "Produtos Perecivel");
6       private String descricao;
7
8       TipoCarga(String descricao) {
9           this.descricao = descricao;
10      }
11
12      public String getDescricao() {
13          return descricao;
14      }
15  }
```

Figura 12 - Código do enum TipoCarga

3.5 - Tratamento de Exceções

Ao longo do desenvolvimento do aplicativo, houve a necessidade de tratar algumas exceções, com o objetivo de evitar a quebra de código.

A primeira é feita quando o usuário digita alguma opção que não existe no menu. Nesse caso, aparece a mensagem “Opção inválida, tente novamente” seguido de um reset no scanner para que o usuário possa digitar novamente. Esse padrão se repete ao longo da classe **App**, tratando de possíveis erros relacionados ao tipo de carga, tipo de operação, nome do proprietário, entre outros. (Exemplo observado na figura 03, página 06).

Na classe **Empilhar** também foram tratadas algumas outras exceções, como a "IllegalArgumentException" que indica se um método recebeu um argumento ilegal ou inapropriado.

```
22  public boolean empilhar(char posicao, Conteiner conteiner) throws IllegalArgumentException {
23
24      this.throwsIfNotHasPilha(posicao);
25
26      Pilha pilha = pilhas.get(posicao);
27      return pilha.empilhar(conteiner);
28  }
```

Figura 13 - Código da exceção "IllegalArgumentException"

Usando o mesmo princípio observado acima, esse outro método trata uma posição inválida.

```
40  private void throwsIfNotHasPilha(char posicao) {
41      if (!this.hasPilha(posicao)) {
42          System.out.println(x: "posição inválida");
43          throw new IllegalArgumentException(s: "posição inválida");
44      }
45  }
```

Figura 14 - Posição inválida

4. ORIENTAÇÕES DE USO

Logo ao abrir o programa, o usuário irá se deparar com quatro opções de funcionalidades disponíveis, que serão mostradas em um menu:

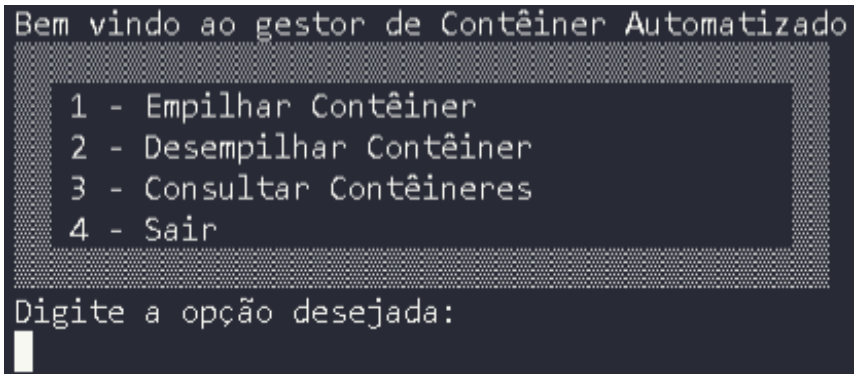

Figura 15 - Menu Principal do Aplicativo

O usuário deve inserir apenas o número referente a opção que deseja executar, não digitando outros números, letras ou quaisquer caracteres diferentes. Sendo escolhida a opção de número 1, empilhar contêiner, será necessário inserir os dados do contêiner que o usuário deseja cadastrar. Primeiro é necessário informar o tipo da carga:

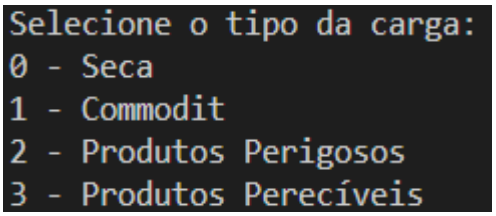

Figura 16 - Menu de Opções para Tipo de Carga

Após informar o tipo da carga, é preciso que seja escolhido o tipo da operação:

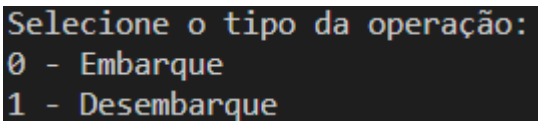

Figura 17 - Menu de Opções para Tipo de Operação

Em ambos casos, é necessário apenas escrever o número referente ao tipo que deseja escolher. Logo depois então, é preciso que seja informado o nome do proprietário do contêiner, o peso, em KG, da carga, e por último, a posição desejada, entre “A” e L”, para que o contêiner seja empilhado:

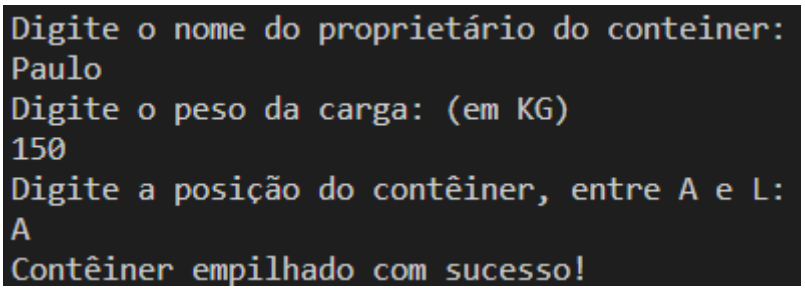

Figura 18 - Exemplo de Inserção de Dados

Para a inserção do peso da carga é necessário apenas escrever o valor do peso, caso o número seja decimal escrever o número dividido com ponto, como por exemplo: **40.5**. Assim como, ao digitar a posição deve se limitar a uma letra por vez e escolher apenas entre as letras A e L. Se todos os dados forem inseridos corretamente, além da existência de vaga na posição escolhida, o contêiner será empilhado com sucesso, e o programa retornará para o menu inicial. Caso contrário, o programa alertará o usuário sobre o tipo do erro, e irá pedir para que seja inserido o dado novamente.

Já na segunda opção, desempilhar contêiner, o programa irá requerer que o utilizador informe o identificador único do contêiner que ele deseja desempilhar:

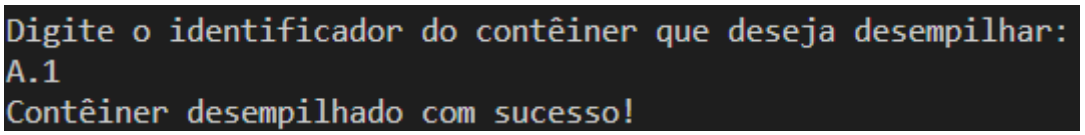

Figura 19 - Exemplo de Inserção de Identificador

O identificador sempre será será no formato **ch.n** (**ch**: letra A a L, e **n**: número de inserção naquela posição) e deve ser digitado assim para ser feito o desempilhamento. Caso em dúvida de qual o identificador correto, pode ser feito primeiro a consulta, como mostrado na Figura 20 (Página 14). Sendo um identificador válido, o programa irá desempilhar o contêiner com sucesso, se não, irá informar para o usuário que o identificador é inválido, retornando automaticamente para o menu inicial.

Caso seja escolhida a terceira opção, consultar contêineres, o usuário irá se deparar com um novo menu:

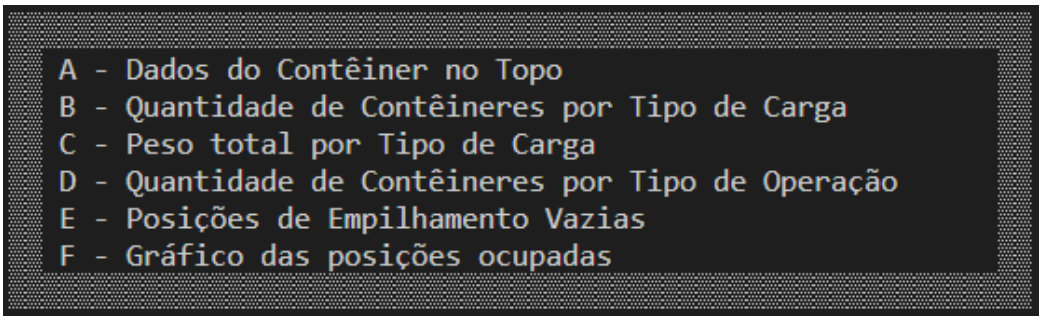

Figura 20 - Menu de Consultas do Aplicativo

Escolhendo a opção A, o usuário necessitará informar qual posição ele deseja consultar, respeitando a mesma regra de inserção já antes apresentada.:

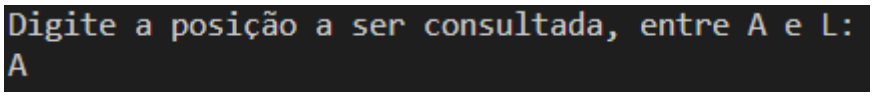

Figura 21 - Exemplo de Inserção para Consulta na Opção A

Caso haja contêiner na posição escolhida, será informado os dados sobre o contêiner encontrado no topo da posição:

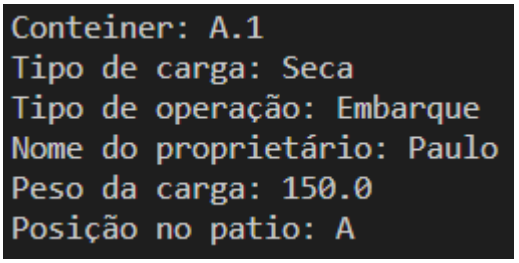

Figura 22 - Exemplo de retorno para Posição com Contêiner

Não havendo contêiner na posição escolhida pelo usuário, o aplicativo informará que a Posição está vazia::

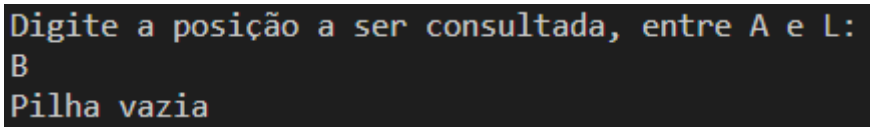

Figura 23 - Exemplo de retorno para Posição sem Contêiner

Optando pela opção B, o programa irá informar a quantidade de contêineres por tipo de carga:

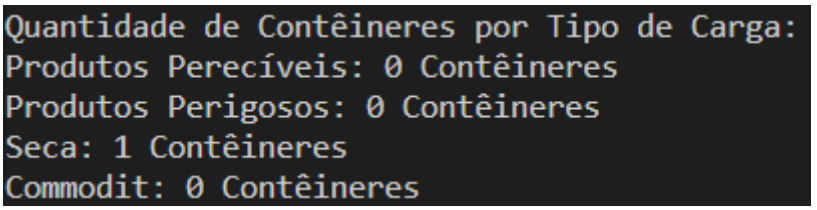

Figura 24 - Exemplo de retorno para Consulta na Opção B

A opção C irá trazer para o usuário o peso total por tipo de carga:

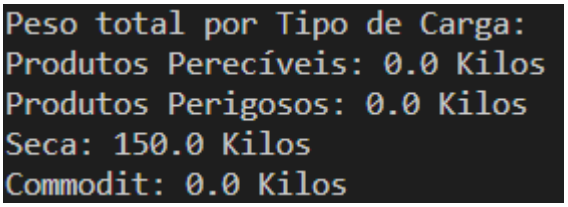

Figura 25 - Exemplo de retorno para Consulta na Opção C

A opção D informará a quantidade de contêineres por tipo de operação:

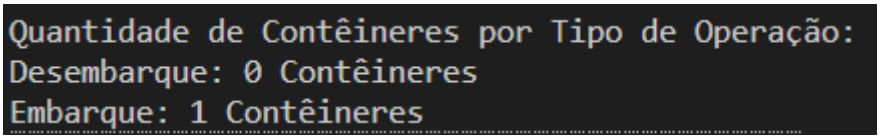

Figura 26 - Exemplo de retorno para Consulta na Opção D

A opção E irá mostrar as posições de empilhamento que ainda não contém contêineres:

Figura 27 - Exemplo de retorno para Consulta na Opção E

A opção F irá gerar um gráfico das posições que estão ocupadas:

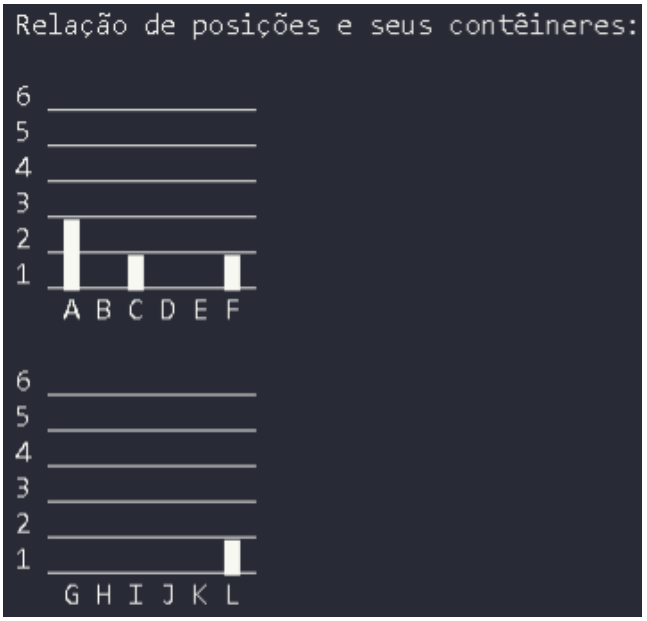

Figura 28 - Exemplo de retorno para Consulta na Opção F

E por fim, caso o usuário opte pela opção de número 4, o aplicativo será finalizado.

5. CONCLUSÕES

O projeto permitiu pela construção deste aplicativo um melhor entendimento de como funcionam as pilhas e como elas podem ser utilizadas na prática durante o desenvolvimento de um programa. Além de promover também, o uso de outras funcionalidades e propriedades do Java como o Switch Case e o Scanner que protagonizam papéis essenciais para o funcionamento do aplicativo, além do tratamento de exceções.

O aplicativo desenvolvido supre as exigências que lhe foram impostas, e possui uma interface fácil de ser utilizada pelos usuários. Para eventuais atualizações, primeiramente a opção de utilizar de um sistema de gerenciamento de dados como o MySQL para criar uma base de dados para ser acessada pelo programa na qual os dados dos contêineres seriam armazenados e não seriam perdidos caso o programa fosse fechado. Além disso, a implementação de outras funcionalidades como a consulta de dados de um contêiner específico, não apenas aquele no topo e a consulta de todos contêineres de uma posição de empilhamento.