

## Análise

Lendo o texto, percebi que seria uma dificuldade como conseguir armazenar todas as distâncias entre cidades, já que elas estão em uma matriz e nunca programei ler arquivos de .csv que estão em uma matriz. Porém, para chegar nesta etapa, precisaria primeiro desenvolver as outras classes, como caminhões e itens.

Para conseguir me situar corretamente, grifei as partes do texto que seriam usadas para codificar, separando as cores de classes e atributos por cores diferentes do habitual.

A transportadora Dely tem sua frota composta por caminhões de portes distintos: um modelo de caminhão de porte pequeno transporta até 1 toneladas e possui o custo de R\$ 4,87 por km rodado; um caminhão de médio porte transporta até 4 toneladas e possui o custo de R\$ 11,92 por km rodado; e um caminhão de grande porte transporta até 10 toneladas e possui o custo de R\$ 27,44 por km rodado.

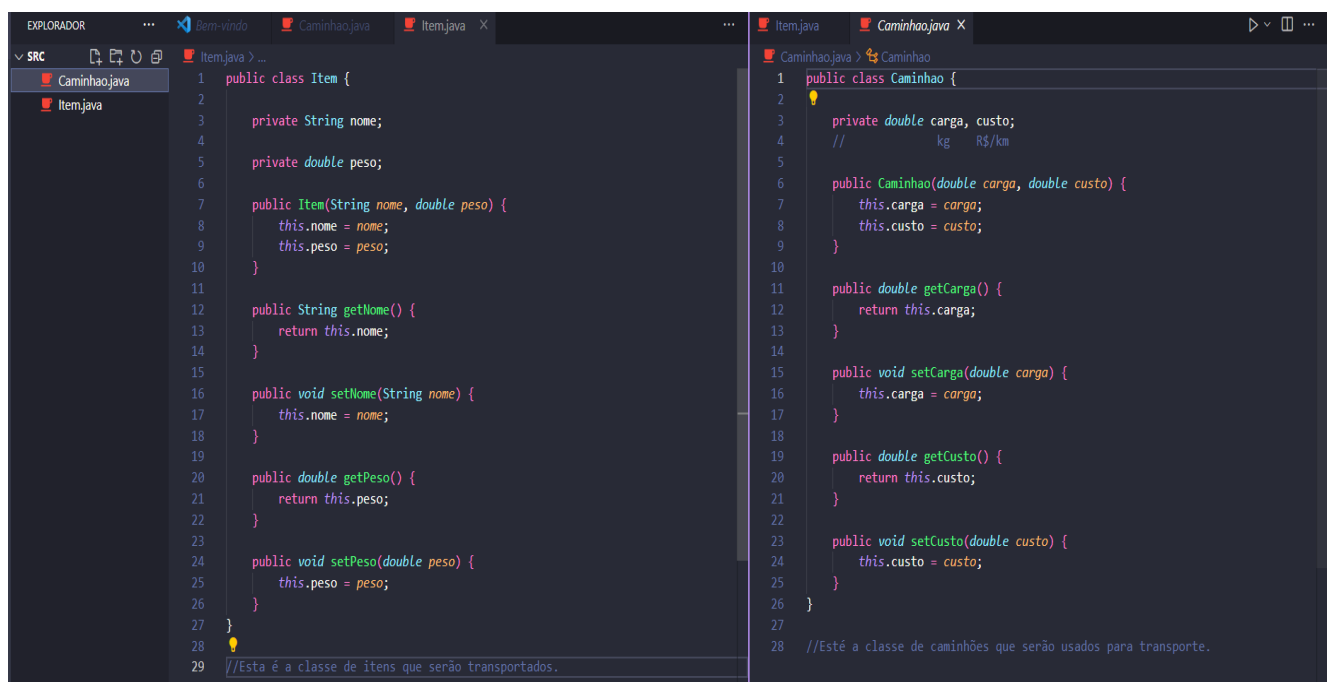
## Caminhão e Item

Comecei o código pela classe de caminhões, implementando como atributos:

- Carga - Fiquei indeciso por qual variável usar, pois os itens podem variar, exemplo do celular que possuía peso de ½ quilograma. Então, para não ter que fazer a conversão em um método mais para a frente, decidi colocar como *double* e atribuir 1000kg como 1 tonelada, por exemplo.
- Custo - Este foi mais simples, um *double* que seria baseado em R\$ por KM.

Finalizei a construção da classe o construtor, *getters* e *setters*.

Em seguida, como dito, segui para a classe de itens e implementei como atributos nome para identificação e peso, sem impedimentos nesta classe.

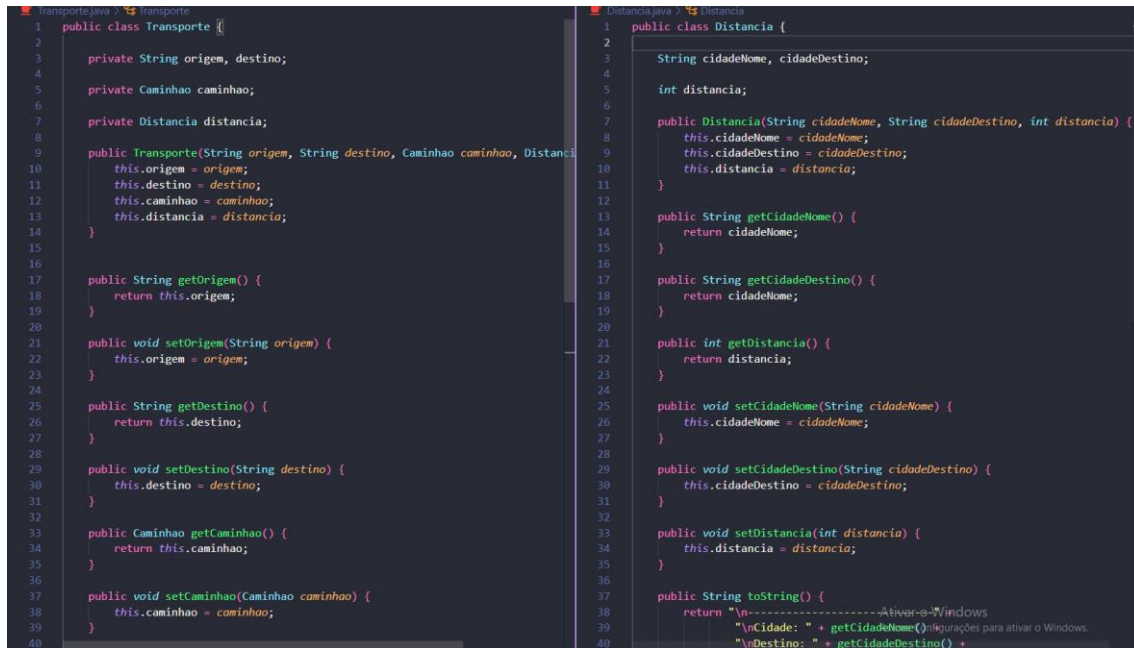


```
1 public class Item {
2
3     private String nome;
4
5     private double peso;
6
7     public Item(String nome, double peso) {
8         this.nome = nome;
9         this.peso = peso;
10    }
11
12    public String getNome() {
13        return this.nome;
14    }
15
16    public void setNome(String nome) {
17        this.nome = nome;
18    }
19
20    public double getPeso() {
21        return this.peso;
22    }
23
24    public void setPeso(double peso) {
25        this.peso = peso;
26    }
27 }
28
29 //Esta é a classe de itens que serão transportados.
```

```
1 public class Caminhao {
2
3     private double carga, custo;
4     //      kg      R$/km
5
6     public Caminhao(double carga, double custo) {
7         this.carga = carga;
8         this.custo = custo;
9     }
10
11    public double getCarga() {
12        return this.carga;
13    }
14
15    public void setCarga(double carga) {
16        this.carga = carga;
17    }
18
19    public double getCusto() {
20        return this.custo;
21    }
22
23    public void setCusto(double custo) {
24        this.custo = custo;
25    }
26 }
27
28 //Está a classe de caminhões que serão usados para transporte.
```

## Transporte

Nesta classe, inicialmente, implementei os atributos *string* origem e destino, sinalizando que seriam os nomes das cidades. Depois, pensei em introduzir uma nova classe só com as distâncias entre cidades e, com um método de leitura de arquivos na classe transporte, introduzi-las automaticamente.



```
1 public class Transporte {
2
3     private String origem, destino;
4
5     private Caminhao caminhao;
6
7     private Distancia distancia;
8
9     public Transporte(String origem, String destino, Caminhao caminhao, Distancia distancia) {
10         this.origem = origem;
11         this.destino = destino;
12         this.caminhao = caminhao;
13         this.distancia = distancia;
14     }
15
16
17     public String getOrigem() {
18         return this.origem;
19     }
20
21     public void setOrigem(String origem) {
22         this.origem = origem;
23     }
24
25     public String getDestino() {
26         return this.destino;
27     }
28
29     public void setDestino(String destino) {
30         this.destino = destino;
31     }
32
33     public Caminhao getCaminhao() {
34         return this.caminhao;
35     }
36
37     public void setCaminhao(Caminhao caminhao) {
38         this.caminhao = caminhao;
39     }
40 }
```

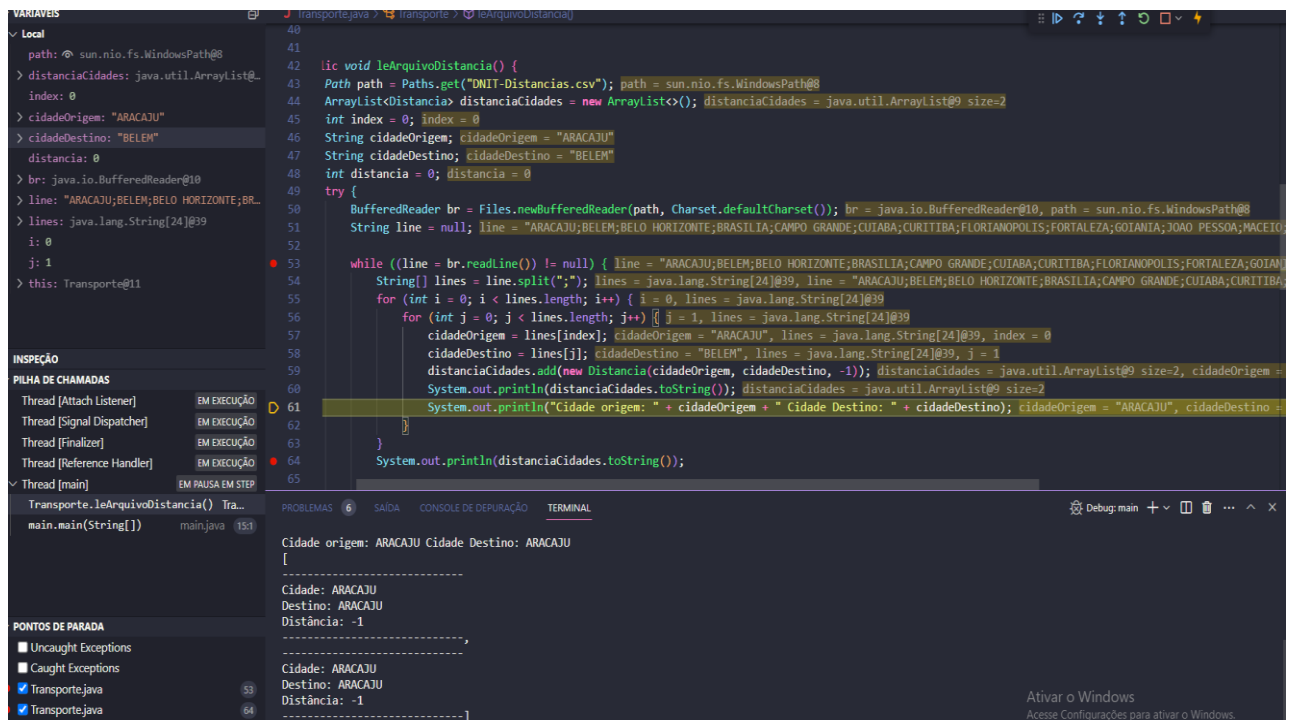
```
1 public class Distancia {
2
3     String cidadeNome, cidadeDestino;
4
5     int distancia;
6
7     public Distancia(String cidadeNome, String cidadeDestino, int distancia) {
8         this.cidadeNome = cidadeNome;
9         this.cidadeDestino = cidadeDestino;
10        this.distancia = distancia;
11    }
12
13    public String getCidadeNome() {
14        return cidadeNome;
15    }
16
17    public String getCidadeDestino() {
18        return cidadeDestino;
19    }
20
21    public int getDistancia() {
22        return distancia;
23    }
24
25    public void setCidadeNome(String cidadeNome) {
26        this.cidadeNome = cidadeNome;
27    }
28
29    public void setCidadeDestino(String cidadeDestino) {
30        this.cidadeDestino = cidadeDestino;
31    }
32
33    public void setDistancia(int distancia) {
34        this.distancia = distancia;
35    }
36
37    public String toString() {
38        return "\n-----Ativar o Windows\n"
39            + "\nCidade: " + getCidadeNome() + "\n"
40            + "\nDestino: " + getCidadeDestino() + "\n";
41    }
42 }
```

Passei algumas horas tentando implementar este método corretamente, pois como é uma matriz, não basta ler e atribuir o valor no próximo atributo, teria que ter uma estrutura de matriz também, como o .csv.

Não queria fazer com uma matriz pois achei que não seria prático, além de não ter muita familiaridade, tentei implementar como um *ArrayList*. Portanto, fiz um *ArrayList* da classe distância e, como não consegui pensar em uma maneira de pensar em como implementar a cidade origem, destino e distância juntos no mesmo objeto, comecei a pensar em como implementar apenas a primeira linha do .csv, que era a cidade origem e cidade destino no mesmo objeto.

Assim, pensei em implementar um *for(i)* e dentro um *for(j)*: o *for(i)* simbolizava a cidade origem e registrava todas as cidades destino que é o *for(j)*, passando um por um, até o fim da linha. Desta maneira, registraria todos os possíveis trajetos entre as cidades, faltando apenas a distância entre elas.

Por algum motivo, estava dando erro. Nos testes apareciam repetidas vezes as mesmas cidades de origem, no destino e, no *debug*, percebi que o nome das cidades destino eram corretos, mas no *print* da tela estava dando errado.



Na linha 57, a cidade é aracaju, enquanto na linha 58 a cidade é belem, porém, o *print* citava apenas a cidade aracaju tanto como origem, como destino. Fiquei um bom tempo neste problema e refiz o código diversas vezes.

Revisando o código, percebi o erro: o método *getDestino()* que estava na classe *Distancia* estava retornando a *cidadeNome* que é a origem, por isso estava dando erro no *toString()*. Após arrumar isso, o código teste fluiu normalmente registrando todos os possíveis trechos entre cidades:

```

cidade origem: RIO DE JANEIRO // cidade destino: BELEM
cidade origem: RIO DE JANEIRO // cidade destino: BELO HORIZONTE
cidade origem: RIO DE JANEIRO // cidade destino: BRASILIA
cidade origem: RIO DE JANEIRO // cidade destino: CAMPO GRANDE
cidade origem: RIO DE JANEIRO // cidade destino: CUIABA
cidade origem: RIO DE JANEIRO // cidade destino: CURITIBA
cidade origem: RIO DE JANEIRO // cidade destino: FLORIANOPOLIS
cidade origem: RIO DE JANEIRO // cidade destino: FORTALEZA
cidade origem: RIO DE JANEIRO // cidade destino: GOIANIA
cidade origem: RIO DE JANEIRO // cidade destino: JOAO PESSOA
cidade origem: RIO DE JANEIRO // cidade destino: MACEIO
cidade origem: RIO DE JANEIRO // cidade destino: MANAUS
cidade origem: RIO DE JANEIRO // cidade destino: NATAL
cidade origem: RIO DE JANEIRO // cidade destino: PORTO ALEGRE
cidade origem: RIO DE JANEIRO // cidade destino: PORTO VELHO
cidade origem: RIO DE JANEIRO // cidade destino: RECIFE
cidade origem: RIO DE JANEIRO // cidade destino: RIO BRANCO
cidade origem: RIO DE JANEIRO // cidade destino: RIO DE JANEIRO
cidade origem: RIO DE JANEIRO // cidade destino: SALVADOR
cidade origem: RIO DE JANEIRO // cidade destino: SAO LUIS
cidade origem: RIO DE JANEIRO // cidade destino: SAO PAULO
cidade origem: RIO DE JANEIRO // cidade destino: TERESINA
cidade origem: RIO DE JANEIRO // cidade destino: VITORIA

```

Agora, com os elementos de todos os possíveis trechos inseridos no *ArrayList*, comecei a pensar em maneiras de colocar a distância entre eles e, como os elementos estavam em ordem, bastava apenas um *for*, que acabava assim que chegasse ao final da linha, numerando a distância de cada elemento existente na lista.

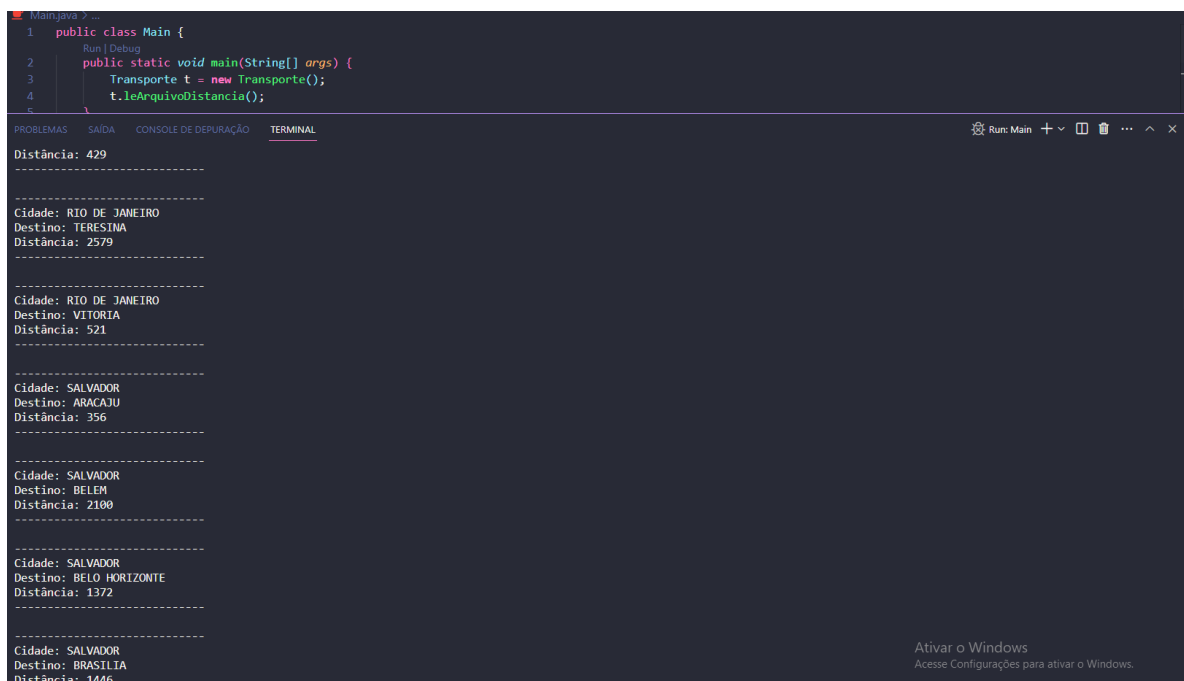
```
public void leArquivoDistancia() {
    Path path = Paths.get("DNIT-Distancias.csv");
    ArrayList<Distancia> distanciaCidades = new ArrayList<>();
    String cidadeOrigem;
    String cidadeDestino;
    int distancia;
    int index = 0;
    try {
        BufferedReader br = Files.newBufferedReader(path, Charset.defaultCharset());
        String line = null;

        while ((line = br.readLine()) != null) {
            String[] lines = line.split(";");
            for (int i = 0; i < lines.length; i++) {
                for (int j = 0; j < lines.length; j++) {
                    cidadeOrigem = lines[i];
                    cidadeDestino = lines[j];
                    distanciaCidades.add(new Distancia(cidadeOrigem, cidadeDestino, -1));
                    System.out.println("\ncidade origem: " + cidadeOrigem + " // cidade destino: " + cidadeDestino);
                }
            }

            for (int i = 0; i < lines.length; i++, index++) {
                distancia = Integer.parseInt(lines[i]);
                distanciaCidades.get(index).setDistancia(distancia);
            }

            break;
        }
    }
}
```

Ao testar o código, percebi que a linha não pulava para a outra, então pesquisei e aprendi que o *br.readLine()* é o que faz a linha ser “pulada”. Porém, testei implementando-o entre um *for* e outro e não funcionou, pois, a variável *line* torna-se *null* ao fim do *for(i)* e *for(j)*. Então, pensei em fazer outro *while*, após o fim do primeiro, com o novo *for* para determinar a distância. Testei e deu certo.



The screenshot shows an IDE with a Java class named `Main` containing a `main` method. The `main` method creates a `Transporte` object and calls `t.leArquivoDistancia()`. The terminal output displays the results of the program execution, showing city pairs and their distances.

```
1 public class Main {
2     public static void main(String[] args) {
3         Transporte t = new Transporte();
4         t.leArquivoDistancia();
5     }
6 }
```

Terminal Output:

```
Distância: 429
-----
Cidade: RIO DE JANEIRO
Destino: TERESINA
Distância: 2579
-----
Cidade: RIO DE JANEIRO
Destino: VITORIA
Distância: 521
-----
Cidade: SALVADOR
Destino: ARACAJU
Distância: 356
-----
Cidade: SALVADOR
Destino: BELEM
Distância: 2100
-----
Cidade: SALVADOR
Destino: BELO HORIZONTE
Distância: 1372
-----
Cidade: SALVADOR
Destino: BRASILIA
Distância: 1446
```

Para conseguir executar os testes, retirei temporariamente os parâmetros do método construtor da classe Transporte e coloquei um *forEach* com a chamada do método *toString()* de cada objeto de distância que havia no *ArrayList*, para checar se estavam corretos. Assim que os testes deram certo, implementei os parâmetros no construtor novamente e retirei o *forEach*.

```
Transporte.java > Transporte > leArquivoDistancia()
51     this.distancia = distancia;
52 }
53
54 public void leArquivoDistancia() {
55     Path path = Paths.get("DNIT-Distancias.csv");
56     ArrayList<Distancia> distanciaCidades = new ArrayList<>();
57     String cidadeOrigem;
58     String cidadeDestino;
59     int distancia;
60     int index = 0;
61     try {
62         BufferedReader br = Files.newBufferedReader(path, Charset.defaultCharset());
63         String line = null;
64
65         while ((line = br.readLine()) != null) {
66             String[] lines = line.split(";");
67             for (int i = 0; i < lines.length; i++) {
68                 for (int j = 0; j < lines.length; j++) {
69                     cidadeOrigem = lines[i];
70                     cidadeDestino = lines[j];
71                     distanciaCidades.add(new Distancia(cidadeOrigem, cidadeDestino, -1));
72                 }
73             }
74             break;
75         }
76
77         while ((line = br.readLine()) != null) {
78             String[] lines = line.split(";");
79             for (int i = 0; i < lines.length; i++, index++) {
80                 distancia = Integer.parseInt(lines[i]);
81                 distanciaCidades.get(index).setDistancia(distancia);
82             }
83         }
84     } catch (FileNotFoundException e) {
85         System.out.println(e);
86     } catch (IOException e) {
87         System.out.println(e);
88     }
89 }
```

## Frota e App

A classe Frota é responsável por obter todo o registro de transportes, além ser quem é instanciada para executar os programas do sistema. Ao mesmo tempo que codifiquei esta classe, lia novamente as funcionalidades requisitadas para implementar a execução corretamente.

Logo na primeira funcionalidade, percebi que precisaria instanciar a classe Transporte, que instanciaria a classe Distancia com atributos similares. Portanto, pensei na possibilidade de deletar a classe Distancia e fazer toda a tabela de distâncias entre cada cidade com um *ArrayList* chamado “trechos”, que sinalizaria que é a lista de trechos e distância entre cada cidade. Assim, o método que lê os arquivos da tabela de distância passou a ser da classe Frota.

Porém, estaria muito confuso, o parâmetro do construtor da classe Transporte passou a ter as cidades, carga e distância. Para registrar apenas a distância entre as cidades seria inútil ter o parâmetro carga, assim como para registrar um transporte não seria necessário retratar a distância. Dito isso, pensei na possibilidade de recriar a classe Distancia e separar novamente os atributos, pensei também em fazer uma herança com a classe Distancia sendo filha da classe Transporte, mas não haveria um nome e nem uma função para a classe mãe. Outrora, veio a ideia de fazer apenas um método de calcular a distância, mas seria desnecessário pois não há cálculo, então, caiu a ficha que apenas um atributo distância seria o suficiente.

## 1ª Funcionalidade

Indo por partes, o primeiro desafio desta funcionalidade é mostrar para o usuário quais trajetos há disponíveis. Como eu passei um bom tempo pensando sem conseguir achar uma maneira de implementar, resolvi ao mesmo tempo iniciar a classe App que será a responsável por fazer o sistema funcionar fornecendo informações ao usuário.

Como base, utilizei um código similar ao que eu utilizei em 4 trabalhos feitos no semestre passado na cadeira de POO (Programação Orientada a Objetos), no qual o professor Yamaguti nos forneceu este início de código com *exceptions* e modelos de apresentação ao usuário:

```
public void executa() {
    int opcao = -1;
    do {
        menu();
        boolean ok;
        do {
            ok = true;
            try {
                opcao = in.nextInt();
            } catch (InputMismatchException e1) {
                in.nextLine();
                ok = false;
                System.out.println("Tipo incorreto. Redigite.\n");
            } catch (Exception e2) {
                in.nextLine();
                ok = false;
                e2.printStackTrace();
                System.out.println("Redigite.\n");
            }
        } while (!ok);
        in.nextLine();

        switch (opcao) {
            case 0:
                break;
            case 1:
                consultaTrechosModalidade();
                break;
            case 2:
                cadastraTransporte();
                break;
            case 3:
                dadosEstatisticos();
                break;
            case 4:
                break;
            default:
                System.out.println("Opção inválida");
        }
    } while (opcao != 0);
}

public void menu() {
    System.out.println("\n=====");
    System.out.println("Bem-vindo ao sistema de transporte interestadual de cargas");
    System.out.println("[1] Consultar trechos disponíveis e modalidades de transporte");
    System.out.println("[2] Cadastrar novo transporte");
    System.out.println("[3] Exibir dados estatísticos");
    System.out.println("[4] Terminar o programa");
    System.out.println("=====");
    System.out.println("\nOpção desejada: ");
}
```

Implementando e testando a primeira função do código que é ler os arquivos e dizer ao usuário quais trechos estão disponíveis, aconteceu um erro:

```
72 public void consultaTrechosModalidade() {
73     System.out.println("Estes são os trechos disponíveis e a distância entre eles: ");
74     frota.consultaTrechos().stream().forEach(T -> System.out.println(T.toString())); frota = Frota@10
75     System.out.println("Digite o nome da cidade origem que deseja realizar seu transporte: ");
76     String cidadeOrigem = in.nextLine(); in = Scanner@13 "java.util.Scanner[delimiters=\p[\\java\\whitespace]+][position=3][match valid=true][need input=false][source=
PROBLEMAS (6) SAÍDA CONSOLE DE DEPURACÃO TERMINAL
[2] Cadastrar novo transporte
[3] Exibir dados estatísticos
[4] Terminar o programa
=====
Opção desejada:
1
Estes são os trechos disponíveis e a distância entre eles:
Digite o nome da cidade origem que deseja realizar seu transporte:
[]
```

Procurei o erro com o *debug* e descobri que o arquivo não estava sendo lido, testei criando um *arquivo.txt* aleatório e não estava sendo lido. No fim, reiniciei o vscode e deu certo, era um erro da IDE. Utilizei uma expressão lambda linha 74 que também aprendi em POO, para simplificar o código.

```
72 public void consultaTrechosModalidade() {
73     System.out.println("Estes são os trechos disponíveis e a distância entre eles: ");
74     frota.consultaTrechos().stream().forEach(T -> System.out.println(T.toString()));
75     System.out.println("Digite o nome da cidade origem que deseja realizar seu transporte: ");
76     String cidadeOrigem = in.nextLine();
77     System.out.println("Agora, digite o nome da cidade destino: ");
78     String cidadeDestino = in.nextLine();
79
80     cidadeOrigem = cidadeOrigem.toUpperCase();
81     cidadeDestino = cidadeDestino.toUpperCase();
82 }
```

PROBLEMAS SAÍDA CONSOLE DE DEBURAÇÃO TERMINAL

[3] Exibir dados estatísticos  
[4] Terminar o programa

=====

Opção desejada:  
1

Estes são os trechos disponíveis e a distância entre eles:

```
{ origem='ARACAJU', destino='ARACAJU', distancia='0', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='BELEM', distancia='2079', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='BELO HORIZONTE', distancia='1570', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='BRASILIA', distancia='1652', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='CAMPO GRANDE', distancia='2765', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='CUIABA', distancia='2775', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='CURITIBA', distancia='2595', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='FLORIANOPOLIS', distancia='2892', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='FORTALEZA', distancia='1183', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='GOIANIA', distancia='1848', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='JOAO PESSOA', distancia='611', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='MACEIO', distancia='294', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='MANAUS', distancia='5215', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='NATAL', distancia='788', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='PORTO ALEGRE', distancia='3296', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='PORTO VELHO', distancia='4230', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='RECIFE', distancia='501', carga='1.0', frota='[]' }
{ origem='ARACAJU', destino='RIO BRANCO', distancia='4763', carga='1.0', frota='[]' }
```

Seguindo para o próximo problema, seria necessário juntar o transporte com o caminhão. Fiquei um longo tempo pensando em como implementar isso. As ideias não encaixavam pois, não queria usar a má prática de instanciar um caminhão, sendo que havia a frota e o transporte como classes que estariam sendo utilizadas para referenciá-lo.

Tive muita dificuldade neste método, pedi ajuda para uma colega da faculdade que está cursando no mesmo semestre para ver se ela sabia de algo que poderia agregar no meu código, recebi a dica de transformar os atributos da classe Caminhao em uma classe Porte que seria *enum*. Implementei o código e agora os únicos atributos da classe Caminhao é a classe Porte e o número da carga.

```
Caminhao.java > Caminhao > Caminhao(Porte)
1 public class Caminhao {
2
3     private Porte porte;
4
5     //private double carga
6
7     public Caminhao(Porte porte) {
8         this.porte = porte;
9     }
10
11     public Porte getPorte() {
12         return porte;
13     }
14
15     public void setPorte(Porte porte) {
16         this.porte = porte;
17     }
18 }
```

```
Porte.java > Porte
1 public enum Porte {
2     PEQUENO(nome:"pequeno porte", preco:4.87, carga:1000), MEDIO(nome:"medio porte", p
3     private final String nome;
4     private final double preco;
5     private final int carga;
6
7
8     Porte (String nome, double preco, int carga) {
9         this.nome= nome;
10        this.preco=preco;
11        this.carga=carga;
12    }
13
14    public String getNome() {
15        return nome;
16    }
17
18    public double getPreco() {
19        return preco;
20    }
21
22    public int getCarga() {
23        return carga;
24    }
25 }
```

Após tirar uma pausa, voltei a tentar raciocinar para resolver este problema e cheguei à conclusão que deveria me basear em colocar as informações como parâmetro para classe Frota e o que ela não poderia fornecer, o transporte forneceria, apenas fiquei preocupado se conseguiria extrair informação da classe Caminhao que precisava. Cheguei a este código:

```

public void consultaTrechosModalidade() {
    System.out.println("Estes são os trechos disponíveis e a distância entre eles: ");
    frota.consultaTrechos().stream().forEach(r -> System.out.println(r.toString()));

    System.out.println("Digite o nome da cidade origem que deseja realizar seu transporte: ");
    String cidadeOrigem = in.nextLine();
    System.out.println("Agora, digite o nome da cidade destino: ");
    String cidadeDestino = in.nextLine();

    cidadeOrigem = cidadeOrigem.toUpperCase();
    cidadeDestino = cidadeDestino.toUpperCase();
    int distancia = frota.consultaDistancia(cidadeOrigem, cidadeDestino);

    if (distancia == -1) {
        System.out.println(
            "Não foi encontrada a distância entre estas cidades, verifique se ambas foram escritas corretamente, sem acentos.");
        return;
    }
    System.out.println(
        "Por fim, digite a sigla que corresponde o tamanho do caminhão em que deseja consultar o preço de transporte: ");
    System.out.println(
        "\nCaminhao de pequeno porte [P] + "\nCaminhao de medio porte [M]" + "\nCaminhao de grande porte [G]\n");
    char sigla = in.next().charAt(0);
    sigla = Character.toUpperCase(sigla);

    if (sigla == 'P') {
        Caminhao c = new Caminhao(Porte.PEQUENO);
        System.out.println("de " + cidadeOrigem + " para " + cidadeDestino + ", utilizando um caminhão de porte [" +
            sigla + "], a distância é de " + distancia + " km e o custo será de R$ " + frota.consultaTrechoModalidade(c, distancia));
    }

    else if (sigla == 'M') {
        Caminhao c = new Caminhao(Porte.MEDIO);
        System.out.println("de " + cidadeOrigem + " para " + cidadeDestino + ", utilizando um caminhão de porte [" +
            sigla + "], a distância é de " + distancia + " km e o custo será de R$ " + frota.consultaTrechoModalidade(c, distancia));
    }

    else if (sigla == 'G') {
        Caminhao c = new Caminhao(Porte.GRANDE);
        System.out.println("de " + cidadeOrigem + " para " + cidadeDestino + ", utilizando um caminhão de porte [" +
            sigla + "], a distância é de " + distancia + " km e o custo será de R$ " + frota.consultaTrechoModalidade(c, distancia));
    }

    else {
        System.out.println("Erro: certifique-se que escreveu a sigla corretamente.");
        return;
    }
}

```

Ao testá-lo, deu quase tudo certo, o único erro foi o valor que deu errado por algum motivo que iria tentar descobrir executando e depurando o programa.

```

(origem='VITORIA', destino='VITORIA', distancia='0', carga='1.0', frota=[])
Digite o nome da cidade origem que deseja realizar seu transporte:
porto alegre
Agora, digite o nome da cidade destino:
sao paulo
Por fim, digite a sigla que corresponde o tamanho do caminhão em que deseja consultar o preço de transporte:
Caminhao de pequeno porte [P]
Caminhao de medio porte [M]
Caminhao de grande porte [G]
G
de PORTO ALEGRE para SAO PAULO, utilizando um caminhão de porte [G], a distância é de 1109 km e o custo será de R$ 0.0

```

Depurando o programa, me encontrei no seguinte cenário:

```

59 public double calculaCusto() {
60     double custoP = 0, custoM = 0, custoG = 0; custoP = 0,000000, custoM = 0,000000, custoG = 0,000000
61     for (Caminhao caminhao : frota) { frota = ArrayList@14 size=1
62         if (caminhao.getPorte() == Porte.PEQUENO) {
63             custoP = custoP + (4.87 * distancia);
64         }
65         if (caminhao.getPorte() == Porte.MEDIO) {
66             custoM = custoM + (11.92 * distancia);
67         }
68         if (caminhao.getPorte() == Porte.GRANDE) {
69             custoG = custoG + 27.44 * distancia;
70         }
71     }
72     return custoP + custoM + custoG; custoP = 0,000000, custoM = 0,000000, custoG = 0,000000
73 }

```

Por algum motivo, mesmo executando o *if*, a conta encerra-se com tudo zerado.

```

custoG = custoG + 27.44 * distancia; custoG = 0,000000, distancia = 0
custoG = custoG;

```

Depois de 5 depurações diferentes para rever o código, descobri o erro: a distância não está sendo contabilizada. Para fazer a distância ser contabilizada, colocar um parâmetro distancia no método *calculaCusto()*, assim, imaginei que não precisaria ser a distancia que há no atributo da classe Transporte.



```

Digite o nome da cidade origem que deseja realizar seu transporte:
porto alegre
Agora, digite o nome da cidade destino:
rio de janeiro
Por fim, digite a sigla que corresponde o tamanho do caminhão em que deseja consultar o preço de transporte:

Caminhao de pequeno porte [P]
Caminhao de medio porte [M]
Caminhao de grande porte [G]
p
de PORTO ALEGRE para RIO DE JANEIRO, utilizando um caminhao de porte [P], a distância é de 1553km e o custo será de R$7563,11.

```

Felizmente, o teste deu certo e assim pude encerrar o desenvolvimento da 1ª funcionalidade, quero implementar as *exceptions* para o código não encerrar caso o usuário insira algo incorreto, mas deixarei isso para depois que acabar todas as funcionalidades, estou priorizando entregar todas as funcionalidades corretamente.

```

72     public void consultaTrechosModalidade() {
73         System.out.println("Estes são os trechos disponíveis e a distância entre eles: ");
74         System.out.println("-----");
75         frota.consultaTrechos().stream().forEach(T -> System.out.println(T.toString()));
76         System.out.println("-----");
77         System.out.println("Digite o nome da cidade origem que deseja realizar seu transporte: ");
78
79         String cidadeOrigem = in.nextLine();
80         System.out.println("Agora, digite o nome da cidade destino: ");
81         String cidadeDestino = in.nextLine();
82
83         cidadeOrigem = cidadeOrigem.toUpperCase();
84         cidadeDestino = cidadeDestino.toUpperCase();
85         int distancia = frota.consultaDistancia(cidadeOrigem, cidadeDestino);
86
87         if (distancia == -1) {
88             System.out.println(
89                 "Não foi encontrada a distância entre estas cidades, verifique se ambas foram escritas corretamente, sem acentos.");
90             return;
91         }
92         System.out.println(
93             "Por fim, digite a sigla que corresponde o tamanho do caminhão em que deseja consultar o preço de transporte: ");
94         System.out.printf(
95             "\nCaminhao de pequeno porte [P]" + "\nCaminhao de medio porte [M]"
96             + "\nCaminhao de grande porte [G]\n");
97         char sigla = in.next().charAt(0);
98         sigla = Character.toUpperCase(sigla);
99
100        if (sigla == 'P') {
101            Caminhao c = new Caminhao(Porte.PEQUENO);
102            System.out.printf(
103                "de %s para %s, utilizando um caminhao de porte [%c], a distância é de %dkm e o custo será de R$%.2f.",
104                cidadeOrigem, cidadeDestino, sigla, distancia, frota.consultaTrechoModalidade(c, distancia));
105        }
106
107        else if (sigla == 'M') {
108            Caminhao c = new Caminhao(Porte.MEDIO);
109            System.out.printf(
110                "de %s para %s, utilizando um caminhao de porte [%c], a distância é de %dkm e o custo será de R$%.2f.",
111                cidadeOrigem, cidadeDestino, sigla, distancia, frota.consultaTrechoModalidade(c, distancia));
112        }
113
114        else if (sigla == 'G') {
115            Caminhao c = new Caminhao(Porte.GRANDE);
116            System.out.printf(
117                "de %s para %s, utilizando um caminhao de porte [%c], a distância é de %dkm e o custo será de R$%.2f.",
118                cidadeOrigem, cidadeDestino, sigla, distancia, frota.consultaTrechoModalidade(c, distancia));
119        }
120
121        else {
122            System.out.println("Erro: certifique-se que escreveu a sigla corretamente.");
123            return;
124        }
125    }

```

## 2ª Funcionalidade

Para implementar esta funcionalidade, quis ir em partes assim como na 1ª. Me afastei do teclado e comecei a fazer um brainstorm de ideias e, assim que pensei em uma maneira em que eu fosse capaz de desenvolver, comentei no código para não esquecer-lo:

```

127         // for com cidades: cada um com origem, destino e itens de
128         // transporte. Quando for mais de um transporte, perguntar quantos itens serão
129         // deixados naquela viagem, a partir daí, subtrair
130         // o número de carga com outro calculo de peso de itens.
131     public void cadastraTransporte() {

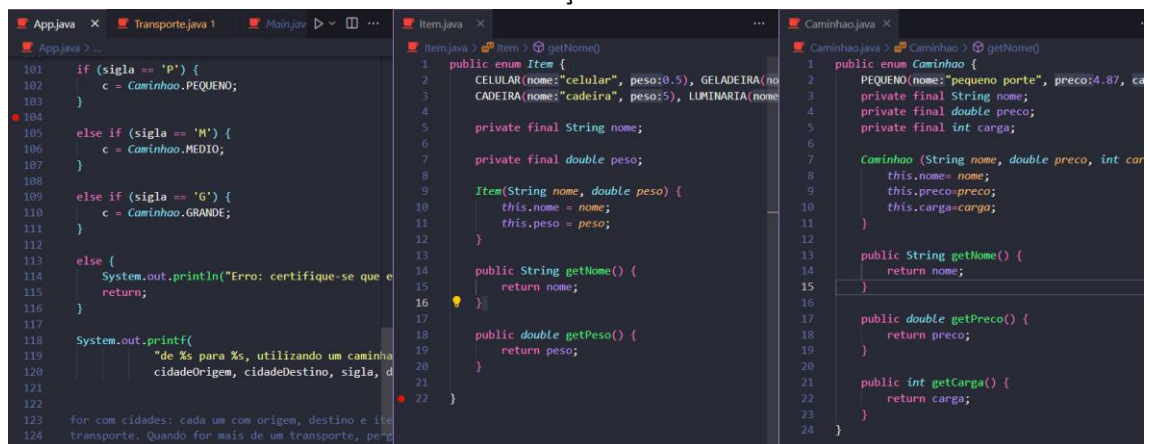
```

Detalhando o que pensei no comentário, permitia ao usuário dizer por quantas cidades seriam transportadas sua carga, assim, fazia um *for* até que *i* seja do mesmo tamanho que o *int cidades*. Neste *for*, haveria outro *for*, detalhando quantos itens diferentes o usuário iria transportar e, respectivamente, a quantia destes itens. Assim, consigo controlar a quantia do peso para calcular a carga e o custo através da quantia e porte de caminhões necessários. Além disso, caso o usuário deseje deixar certa quantia de carga em uma cidade para depois partir para outra, também conseguiria ser controlado.

Para o desenvolvimento deste método, pensei em duas coisas diferentes relacionados a *enum*:

- Transformar a classe *Caminhao* em *enum*, com os mesmos atributos da classe *Porte* e excluí-la. Pois, pensei que o caminhão não precisava adicionar nenhum parâmetro que pode variar, estes possíveis objetos já possuem parâmetros que são constantes (custo, carga limite), sempre variará em 3 tipos, então, não vale a pena criar a classe sempre com os mesmos parâmetros e para isso transformei-o em um *enum*.
- Transformar a classe *Item* em *enum*, basicamente pelos mesmos motivos da classe *Caminhao*, já possuem parâmetros constantes (peso) que variam apenas de acordo com o nome do item.

Assim ficou os *enums Caminhao* e *Item*, além da mudança que fiz no *App* para adaptar a mudança.



```
App.java
101 if (sigla == 'P') {
102     c = Caminhao.PEQUENO;
103 }
104
105 else if (sigla == 'M') {
106     c = Caminhao.MEDIO;
107 }
108
109 else if (sigla == 'G') {
110     c = Caminhao.GRANDE;
111 }
112
113 else {
114     System.out.println("Erro: certifique-se que o sigla é válida");
115     return;
116 }
117
118 System.out.printf(
119     "%d de %s para %s, utilizando um caminhão\n",
120     cidadeOrigem, cidadeDestino, sigla, d
121 );
122
123 for com cidades: cada um com origem, destino e listando o
124 transporte. Quando for mais de um transporte, perguntar se deseja continuar.
```

```
Item.java
1 public enum Item {
2     CELULAR(nome:"celular", peso:0.5), GELADEIRA(nome:"geladeira", peso:5),
3     CADEIRA(nome:"cadeira", peso:5), LUMINARIA(nome:"luminaria", peso:10);
4
5     private final String nome;
6     private final double peso;
7
8     Item(String nome, double peso) {
9         this.nome = nome;
10        this.peso = peso;
11    }
12
13    public String getNome() {
14        return nome;
15    }
16
17    public double getPeso() {
18        return peso;
19    }
20
21 }
22
```

```
Caminhao.java
1 public enum Caminhao {
2     PEQUENO(nome:"pequeno porte", preco:4.87, carga:1000),
3     MEDIO(nome:"medio porte", preco:8.74, carga:2000),
4     GRANDE(nome:"grande porte", preco:13.61, carga:3000);
5
6     private final String nome;
7     private final double preco;
8     private final int carga;
9
10    Caminhao(String nome, double preco, int carga) {
11        this.nome = nome;
12        this.preco = preco;
13        this.carga = carga;
14    }
15
16    public String getNome() {
17        return nome;
18    }
19
20    public double getPreco() {
21        return preco;
22    }
23
24    public int getCarga() {
25        return carga;
26    }
27 }
```

E assim ficou o “rascunho” do método *cadastraTransporte()*

```

public void cadastraTransporte() {
    System.out.println(
        "Digite a quantia de cidades que será transportada a carga que deseja(contabilizando a cidade em que ela sairá para transporte): ");
    int cidades = in.nextInt();
    in.nextLine();

    System.out.println("Qual a cidade de origem?");
    String origem = in.nextLine();
    String destino;
    int qtdItemDif;
    int qtdItem;
    String nomeItem;
    double carga = 0;
    Item celular = Item.CELULAR;
    Item geladeira = Item.GELADEIRA;
    Item freezer = Item.FREEZER;
    Item cadeira = Item.CADEIRA;
    Item luminaria = Item.LUMINARIA;
    Item lavadora = Item.LAVADORA;
    // Transporte t = new Transporte("", "", 0);

    for (int i = 1; i < cidades; i++) {
        System.out.println("Digite o destino do transporte que partirá da cidade " + origem + ":");
        destino = in.nextLine();
        // t.setOrigem(origem);
        // t.setDestino(destino);
        System.out.println("Você pode transportar os seguintes itens: ");
        System.out.printf("%s\n", "[Celular]\n[Geladeira]\n[Freezer]\n[Cadeira]\n[Luminaria]\n[Lavadora de roupa]");
        System.out.println("Neste trecho, quantos itens diferentes você deseja transportar?");
        qtdItemDif = in.nextInt();
        in.nextLine();
        for (int j = 1; j < qtdItemDif; j++) {
            System.out.println("Digite o nome do item (sem acentos) " + j + "/" + qtdItemDif + ": ");
            nomeItem = in.nextLine();
            nomeItem = nomeItem.toLowerCase();
            System.out.println("Qual a quantidade deste item que você deseja transportar?");
            qtdItem = in.nextInt();
            in.nextLine();
            if (nomeItem == celular.getNome()) {
                carga = carga + (celular.getPeso() * qtdItem);
            }
            if (nomeItem == geladeira.getNome()) {
                carga = carga + (geladeira.getPeso() * qtdItem);
            }
            if (nomeItem == freezer.getNome()) {
                carga = carga + (freezer.getPeso() * qtdItem);
            }
            if (nomeItem == cadeira.getNome()) {
                carga = carga + (cadeira.getPeso() * qtdItem);
            }
            if (nomeItem == luminaria.getNome()) {
                carga = carga + (luminaria.getPeso() * qtdItem);
            }
            if (nomeItem == lavadora.getNome()) {
                carga = carga + (lavadora.getPeso() * qtdItem);
            }
        }

        Transporte t = new Transporte(origem, destino, carga);
        frota.cadastaTransporte(t);
        System.out.printf("de %s para %s, a distância a ser percorrida é de %dkm, para transporte de produtos");
    }
}

```

O código ficou desta mesma maneira por algumas horas, sem nem mesmo testar. Pois, fiquei pensando em maneiras para conseguir contar a quantia de cada item, visto que, o exemplo de resposta fornecida no enunciado do exercício técnico mostrava e me preocupei com isso. Pensei em muitas possibilidades, como: adicionar um parâmetro que conta a quantidade de itens no transporte, fazer métodos novos em cada classe (frota e transporte), entre outros. Mas, o pensamento que mais fez sentido em minha cabeça, foi implementar um *ArrayList* de itens e, a cada vez que o usuário digita o tipo de item seguido da quantidade dele, eu acho uma maneira de adicionar esta mesma quantidade, só que de vezes, este mesmo item no *ArrayList*. Desta maneira, para contar a quantia de itens específicos, basta percorrer o a lista que obterei a informação.

Com este código, percebi que não preciso calcular a carga com todos aqueles *ifs*, pois, consigo calcular percorrendo o *ArrayList* também. Pensei em uma forma de simplificar e deixar este método menos “poluído” fazendo um *switch*.

Durante o desenvolvimento, fiquei estagnado devido ao pensamento de implementar para que consiga fazer com mais de 1 cidade de destino. Porém, este pensamento estava acabando com todas as possibilidades do meu código, então, comecei a implementar para cadastrar o transporte com apenas um destino, depois eu me preocuparia em destinos diferentes.

Notei que com a existência do *ArrayList* de itens, não seria necessário a existência do atributo *carga* no construtor do transporte, então pensei em removê-lo. Ainda estou em dúvidas se adiciono um método que calcula o peso da carga ou coloco o *ArrayList* como parâmetro e cálculo na frota.

Durante a implementação dos *enums*, tive muita dúvida pois havia esquecido as características dele, por exemplo, não sabia/não lembrava se era possível ter um *arraylist* de *enum*, havia esquecido até que podia instanciar um *enum*. Para dar uma revisada, pedi para um colega o material de *enum* do professor Yamaguti, que me ensinou isso ano passado e desta maneira consegui aprender a utilizar novamente. Para testar esta primeira tentativa, tentei fazer uma expressão lambda para poupar linhas e simplificar o código, mas não consegui (nomeei o *ArrayList* de itens como “carga” e deletei o *double* carga).

```
carga.stream().filter(i -> i == Item.CELULAR).count();
```

Portanto, fiz um *forEach* com *ifs* para implementar o contador de itens que eu desejava. O *forEach* com o contador de itens dentro do *ArrayList* me causou um problema que eu nunca havia visto antes no código: *java heap space error*. Ao pesquisar, entendi que é um erro de uso completo da memória. Eu já estava pensando que este método de colocar 300 itens de celular em um *ArrayList* estava sendo uma grande má prática, porém, não sabia que ia chegar a comprometer a memória. O momento em que lotou a memória foi após contar todos os itens, foi quando o programa chegava na contagem dos caminhões, que nem era tão grande.

Sendo assim, fiz de um modo diferente em que eu apenas adicionava 1 tipo de item no *ArrayList* e criei uma variável peso que multiplica o *qtdItem* com o peso dele. Assim, não precisaria adicionar vários elementos no *arraylist* e pude seguir com o desenvolvimento.

```
switch (opcao) {
    case 1:
        item = Item.CELULAR;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 2:
        item = Item.GELADEIRA;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 3:
        item = Item.FREEZER;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 4:
        item = Item.CADEIRA;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 5:
        item = Item.LUMINARIA;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 6:
        item = Item.LAVADORA;
        itens.add(item);
        peso += item.getPeso() * qtdItem;
        break;
    case 7:
        break;
    default:
        System.out.println("Opção inválida");
}
while (opcao != 7);
```

Passei muito tempo tentando fazer apenas o transporte para uma cidade, devido as preocupações em citar os itens que estão sendo transportados, no qual pensei que seria necessário citar a quantidade deles. Mas, após ler novamente o enunciado e perceber que não era necessário citar a quantia de cada item que está sendo transportado, foi só questão de tempo para implementar. As grandes dificuldades desta primeira parte de

testes, com certeza, foi a confusão que meu código deixou, tive que ir editando conforme testava e isso me deixou um pouco perdido. Porém, descansei e no outro dia com a cabeça leve consegui ter a calma e concentração que precisava para programar corretamente. O escopo do código ficou desta maneira:

```
126 public void cadastraTransporte() {
127     String origem;
128     String destino;
129     int qtdItem = 0;
130     int opcao = -1;
131     double peso = 0;
132     Item item;
133     ArrayList<Item> itens = new ArrayList();
134     Transporte t;
135
136     System.out.println(
137         "Digite a quantia de cidades que você deseja transportar seus itens: ");
138     int cidades = in.nextInt();
139     in.nextLine();
140
141     System.out.println("Qual a cidade de origem?");
142     origem = in.nextLine();
143     origem = origem.toUpperCase();
144
145     for (int i = 0; i < cidades; i++) {
146
147         System.out.println("\nDigite o destino do transporte que partirá da cidade " + origem + ":");
148         destino = in.nextLine();
149         destino = destino.toUpperCase();
150         do {
151             boolean ok;
152             do {
153                 System.out.println("\nSelecione o tipo de item que deseja transportar: ");
154                 System.out.printf(
155                     "[1] Celular\n[2] Geladeira\n[3] Freezer\n[4] Cadeira\n[5] Luminaria\n[6] Lavadora de roupa\n[7] Já selecionei todos os itens\n");
156                 ok = true;
157                 try {
158                     opcao = in.nextInt();
159                 } catch (InputMismatchException e1) { ...
160                 } catch (Exception e2) { ...
161             } if (opcao != 7) {
162                 qtdItem = 0;
163                 System.out.println("Quantas unidades deste item você deseja transportar?");
164                 try {
165                     qtdItem = in.nextInt();
166                 } catch (InputMismatchException e1) { ...
167                 } catch (Exception e2) { ...
168             }
169             } while (!ok);
170             in.nextLine();
171
172             switch (opcao) {
173                 case 1:
174                     item = Item.CELULAR;
175                     itens.add(item);
176                     peso += item.getPeso() * qtdItem;
177                     break;
178                 case 2:
179                     item = Item.GELADEIRA;
180                     itens.add(item);
181                     peso += item.getPeso() * qtdItem;
182                     break;
183                 case 3:
184                     item = Item.FREEZER;
185                     itens.add(item);
186                     peso += item.getPeso() * qtdItem;
187                     break;
188                 case 4:
189                     item = Item.CADEIRA;
190                     itens.add(item);
191                     peso += item.getPeso() * qtdItem;
192                     break;
193                 case 5:
194                     item = Item.LUMINARIA;
195                     itens.add(item);
196                     peso += item.getPeso() * qtdItem;
197                     break;
198                 case 6:
199                     item = Item.LAVADORA;
200                     itens.add(item);
201                     peso += item.getPeso() * qtdItem;
202                     break;
203                 case 7:
204                     break;
205                 default:
206                     System.out.println("Opção inválida");
207             }
208         } while (opcao != 7);
209
210         t = new Transporte(origem, destino);
211         frota.cadastaTransporte(t, itens, peso);
212         System.out.println("");
213         System.out.printf("\nde %s para %s, a distância a ser percorrida é de %dkm.\nPara o transporte dos produtos ",
214             origem, destino, frota.consultaDistancia(origem, destino));
215     }
216 }
```

de PORTO ALEGRE para FLITANOPOLIS, a distância a ser percorrida é de 476km.  
Para o transporte dos produtos geladeira, freezer, lavadora de roupa, será necessário utilizar

- [0] caminhões de pequeno porte
- [1] caminhões de médio porte
- [10] caminhões de grande porte

de forma a resultar no menor custo de transporte por km rodado. O valor total do transporte d

Este foi o pensamento que anotei para conseguir fazer o programa com mais destinos:

Para fazer com mais de 1 destino, pensei em algumas possibilidades, mas o maior problema era a questão da quantidade de itens. A variável *qtdItem* servia para apenas 1 item e eu precisaria armazenar a quantidade de todos os itens. Portanto, pesquisei se era possível, de alguma forma, adicionar algo que varie dentro do *enum* de item. Procurei apenas pelo motivo de não saber mais o que fazer, pois estava a um tempo pensando em alguma solução. Até que achei um site que explicava que sim, era possível. Assim como em uma classe, bastava implementar um atributo sem colocá-lo no construtor. Consegui implementar e agora possuo uma variável que me diga a quantidade de itens daquele tipo que haviam no meu transporte:

```

private int quantidade;

Item(String nome, double peso ) {
    this.nome = nome;
    this.peso = peso;
}

public String getName() {
    return nome;
}

public int getQuantidade() {
    return quantidade;
}

public void setQuantidade(int itemQuantidade) {
    this.quantidade = itemQuantidade;
}

```

Com isso, bastava eu implementar no *switch* do método *cadastraTransporte()* um *setQuantidade(qtdItem)*, assim eu teria armazenada a informação de quantos itens daquele tipo haviam no transporte. Também adicionei uma variável chamada *custoMedio* para controlar a média dos itens, pois, achei interessante que não apareceu na requisição da funcionalidade, mas apareceu no exemplo então me preocupei em deixar registrado também este número. No fim do programa, ele divide pelo tamanho do *ArrayList* de itens, assim sabendo a média.

```

switch (opcao) {
    case 1:
        item = Item.CELULAR;
        item.setQuantidade(qtdItem);
        itens.add(item);
        peso += item.getPeso() * item.getQuantidade();
        custoMedio += qtdItem;
        break;
}

```

Após a execução do *switch*, o usuário já registrou todos os itens em que deseja transportar, agora, é só armazenar as informações que obtive na classe *frota* e registrar os custos médios.

```

t = new Transporte(id, origem, destino);
frota.cadastraTransporte(t, itens, peso);
custoMedio = frota.transportePreco(t) / custoMedio;
totalCustoMedio += custoMedio;

```

Para imprimir na tela do usuário, utilizei o *printf* para conseguir colocar tudo na mesma linha. Entre os *printfs*, coloquei um *forEach* tanto de itens, quanto de caminhões que havia no transporte para retratar ao usuário quais tipos de cada um havia. Como foi dito nas páginas anteriores, tentei implementar a expressão lambda mas não deu certo, então tive que fazer do jeito que “polui” o código.

```

System.out.printf(
    "\nde %s para %s, a distância a ser percorrida é de %dkm.\nPara o transporte dos produtos ",
    origem, destino, frota.consultaDistancia(origem, destino));

String itAnt = null;
for (Item it : itens) {
    if (it.getNome() != itAnt) {
        System.out.printf("%s, ", it.getNome());
    }
    itAnt = it.getNome();
}

System.out.printf("será necessário utilizar ");

int qtdCpeq = 0, qtdCmed = 0, qtdCgran = 0;
for (Caminhao c : frota.qtdCaminhao(t)) {
    if (c.getNome() == "pequeno porte") {
        qtdCpeq += 1;
    }
    if (c.getNome() == "medio porte") {
        qtdCmed += 1;
    }
    if (c.getNome() == "grande porte") {
        qtdCgran += 1;
    }
}

System.out.printf(
    "\n[%d] caminhões de pequeno porte \n[%d] caminhões de médio porte \n[%d] caminhões de grande porte",
    qtdCpeq, qtdCmed, qtdCgran);
System.out.printf(
    "\nde forma a resultar no menor custo de transporte por km rodado. O custo do transporte destes itens é R$ %.2f e o custo unitário médio é R$%.2f",
    frota.transportePreco(t), custoMedio);

```

Resolvido tal problema, pensei em colocar todo o desenvolvimento que tinha feito até agora neste método dentro de um *if(qtdCidades == 0)* e, quando *qtdCidades != 0*, significa que haverá pelo menos 2 destinos diferentes. Coloquei esta possibilidade dentro de um *else*, que haveria outro *switch*, desta vez com itens que o usuário gostaria de descarregar. Ao selecionar o item em específico, ele dizia a quantidade que desejava retirar e esta quantidade ficaria armazenada no *qtdItem*. Pensando em uma forma de subtrair a quantidade de itens em cada transporte, implementei mais um método no *enum Item* que recebia um certo número e este número subtraía a quantidade de itens que obtinha.

```

public int removeQuantidade(int itensRemovidos) {
    if(this.quantidade > 0) {
        this.quantidade = this.quantidade - itensRemovidos;
        return itensRemovidos;
    } else {
        return 0;
    }
}

```

Com este método eu conseguia subtrair a quantidade de itens que iam ser descarregados na primeira parada. Caso a subtração resultasse um valor menor que 1 (0), o item era removido do *ArrayList*.

```

switch (opcao) {
    case 1:
        item = Item.CELULAR;
        item.removeQuantidade(qtdItem);
        if (item.getQuantidade() < 1) {
            itens.remove(item);
        }
        peso += item.getPeso() * item.getQuantidade();
        custoMedio += qtdItem;
        break;
}

```

Após este *switch*, o código fica similar ao código que fiz após o *switch* anterior, registrar as informações e dizer ao usuário.



Ao fim do *for*, o usuário já fez tudo que precisava fazer e agora é repassado a ele as informações totais do transporte:

```
double precoTotal = 0;
int distanciaTotal = 0;
for (Transporte sameID : frota.dadosEstatisticos()) {
    if (sameID.getId() == id) {
        precoTotal += frota.transportePreco(sameID);
        distanciaTotal += frota.consultaDistancia(sameID.getOrigem(), sameID.getDestino());
    }
}
System.out.printf("\n-----");
System.out.printf("\nO custo total para este transporte é R$%.2f", precoTotal);
System.out.printf("\nA distância total que este transporte percorrerá é %dkm", distanciaTotal);
System.out.printf("\nO custo unitário médio total é R$%.2f", totalCustoMedio / qtdCidade);
System.out.printf("\n-----");
```

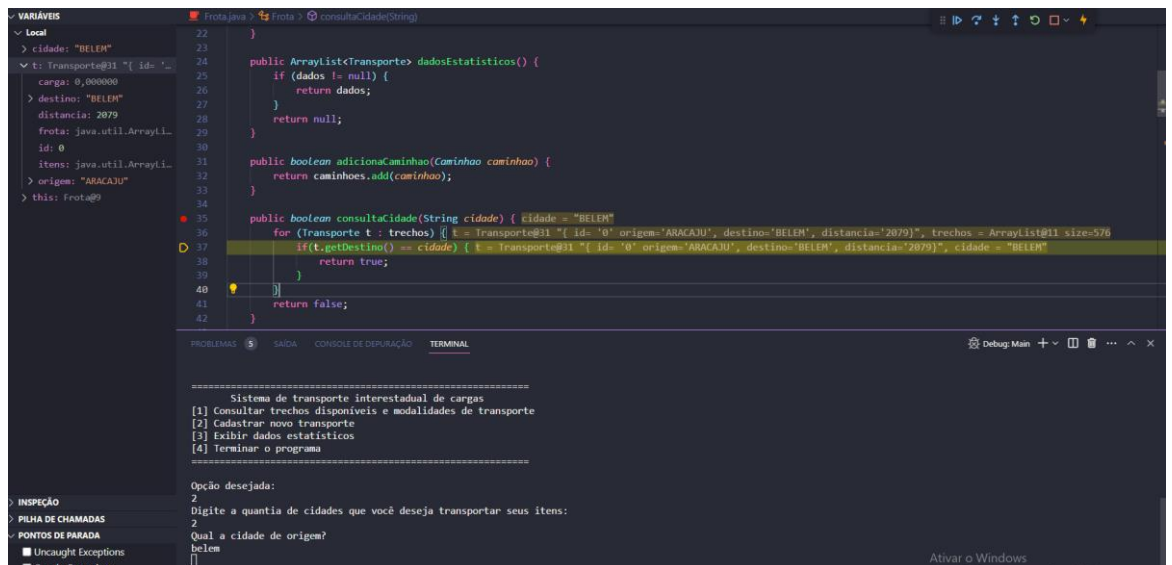
Agora que estou com o código feito, vou implementar o tratamento de exceções. Uso o mesmo modelo de tratamento de exceções em todos os *scanners* de *int* que estão no programa, que consiste em criar um *do* e dentro dele repetir a frase requisitada enquanto a resposta esperada não é digitada pelo usuário, juntamente com as *exceptions*. Tentei criar uma exceção que tratava caso o usuário digitasse um número menor que 1 na quantia de cidades.

```
1 import java.util.Scanner;
2
3 public class MenorQueUmException extends Exception {
4     Scanner in = new Scanner(System.in);
5     int n = in.nextInt();
6
7     MenorQueUmException(String msg) {
8         if(n < 1) {
9             System.out.println(msg);
10        }
11    }
12 }
13
```

Infelizmente, não consegui. Porém, implementei um *if* dentro do *do* que cobria esta possibilidade de inserção e deu certo.

```
boolean ok;
do {
    System.out.println(
        "Digite a quantia de cidades que você deseja transportar seus itens: ");
    ok = true;
    try {
        qtdCidade = in.nextInt();
    } catch (InputMismatchException e1) {
        in.nextLine();
        ok = false;
        System.out.println("Tipo incorreto. Redigite.\n");
    } catch (Exception e2) {
        in.nextLine();
        ok = false;
        e2.printStackTrace();
        System.out.println("Redigite.\n");
    }
    if (qtdCidade < 1) {
        ok = false;
        System.out.println("Não é possível transportar esta quantia de cidades.\n");
    }
} while (!ok);
in.nextLine();
```

Para fazer o usuário digitar a cidade da maneira correta, pensei em implementar um método *consultaCidade()* na frota que confere se há a existência da cidade em que o usuário digitou, faço isso dentro de um *do* que reinicia até ele escrever uma cidade que exista. Porém, me deparei com um problema:



```
22 }
23
24 public ArrayList<Transporte> dadosEstatisticos() {
25     if (dados != null) {
26         return dados;
27     }
28     return null;
29 }
30
31 public boolean adicionaCaminhao(Caminhao caminhao) {
32     return caminhoes.add(caminhao);
33 }
34
35 public boolean consultaCidade(String cidade) {
36     if (cidade == "BELEM")
37         for (Transporte t : trechos) {
38             if (t.getId() == cidade) {
39                 return true;
40             }
41         }
42     return false;
43 }
```

```
=====
Sistema de transporte interestadual de cargas
[1] Consultar trechos disponíveis e modalidades de transporte
[2] Cadastrar novo transporte
[3] Exibir dados estatísticos
[4] Terminar o programa
=====

Opção desejada:
2
Digite a quantidade de itens que deseja transportar seus itens:
2
Qual a cidade de origem?
belem
[]
```

Mesmo digitando a palavra, ele reiniciava o *do*. Pensei que poderia ser algo escrito errado no *if*, então troquei por um *.equals* e deu certo. Devido eu pedir 2 valores diferentes para usar no *switch*, ele também só recomeça depois que digita 2 valores. Mas creio que deixarei desta maneira, para mudá-lo não penso em nenhuma forma que não tenha que mexer em todo o código, por falta de tempo e não comprometer o código, não tentarei mudar isso.

Após colocar todos os tratamentos de exceções/*ifs*, o programa está completo. Se houver tempo após o término da 3ª funcionalidade, calcularei os custos unitários médios para checar se estão corretos, pois não fiz isso.

### 3ª funcionalidade

Estou pensando em implementar esta funcionalidade de um jeito similar a 2ª, colocar como variáveis tudo o que é necessário em questão de números e palavras, conseguindo-os através da frota.

Na implementação desta funcionalidade, pensei em fazer um *forEach* de transportes para pegar todos os transportes que estão registrados. Para os números mais simples como consultar custo através do trecho, não foi muito difícil, pois já havia sido feito para a 2ª funcionalidade, foi apenas uma questão de reutilização do método que está na classe Frota.

Porém, a grande dificuldade que tive nesta funcionalidade foram conseguir o total de itens transportados e a média deles. Pensei por muito tempo em como implementar mas não achei outro jeito além de atribuir mais um atributo na classe Transporte. Além disso, implementei mais um método na classe Frota que recebia o transporte que precisava ser lido. Consiste no seguinte: o transporte já obtém o custo e o item, através disso obtenho a quantidade desde item e, assim, consigo a média por produto. Adicionei-os em um *HashMap* de *String* e *Double*, desta maneira, era só implementar e colocá-los na tela do usuário. Eu achava que o *ArrayList* não poderia me ajudar neste quesito, então pesquisei e aprendi a usar o *HashMap*. Uma experiência muito boa, visto que nunca tinha utilizado ele para nada além de exemplos de aula.

```
public Map<String, Double> mediaItensTransportados(Transporte t) {  
  
    Map<String, Double> mediaItem = new HashMap<String, Double>();  
  
    for (Item item : t.getItens()) {  
        mediaItem.put(item.getNome(), t.calculaCusto(t.getDistancia()) / item.getQuantidade());  
    }  
  
    return mediaItem;  
}
```

Para conseguir a quantia total de caminhões, foi um pouco mais simples. Fiz um *ArrayList* e coloquei em ordem a quantia de caminhões de pequeno, médio e grande porte. Só tive o trabalho de cuidar para pegar o número do endereço correto. Este foi o final da minha programação no desafio, vou revisar, tentar deixar o código o mais bem visível possível e corrigir os erros.