-Situação de Aprendizagem Senai-

**Estudante:** João Wictor Tessmann

Projeto Rede Ônibus 5.0

**Sobre o projeto:**

A empresa de transporte do município enfrenta um desafio significativo: a sobrecarga em algumas linhas de ônibus durante os horários de pico, o que causa grande desconforto aos passageiros. Para mitigar essa situação, é necessário aumentar a quantidade de ônibus nessas linhas. No entanto, o alto custo de investimento exige uma identificação precisa das necessidades.

Como um aluno dedicado e apaixonado por soluções inovadoras, aceitei o desafio de desenvolver um algoritmo que calcula a quantidade de passageiros em cada viagem. Este projeto visa aprimorar a identificação dessas necessidades, otimizando a gestão do transporte público. Com isso, busco não apenas reduzir a superlotação, mas também elevar a satisfação dos usuários, proporcionando uma experiência de viagem mais confortável e eficiente para todos. É um passo importante para transformar o transporte urbano e contribuir para uma cidade mais organizada e feliz.

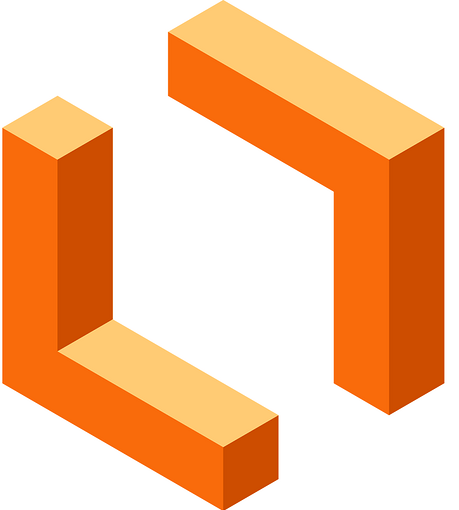
**Objetivo:** Desenvolver um algoritmo em Java que calcule com precisão o número de usuários que utilizam os ônibus de uma linha específica por viagem.

Os objetivos específicos incluem:

* Contabilizar entradas e saídas de passageiros em cada parada.
* Determinar o fluxo total de passageiros por viagem.
* Identificar as linhas com maior volume de passageiros para otimizar a alocação de recursos.

Este projeto fornecerá informações detalhadas para a prefeitura e a empresa de transporte, auxiliando na tomada de decisões para melhorar o serviço e a satisfação dos usuários.

**Tecnologias Utilizadas:**

**NetBeans**: IDE utilizada para o desenvolvimento do projeto.

**Java**: Linguagem de programação utilizada para desenvolver o algoritmo.

**Canva**: Ferramenta utilizada para a elaboração e criação de designs.

**LucidChart:** Ferramenta utilizada para a elaboração e criação de fluxogramas.

**Infraestrutura do Projeto:**

Para configurar e executar o projeto, siga os passos abaixo:

**Baixe o repositório**

Para utilizar o projeto, é necessário baixá-lo neste link: https://github.com/JoaoWictor97/ProjetoRedeOnibus-5.0

**Abra o Projeto no IDE**

Após clonar o repositório, abra a pasta “ProjetoRedeOnibus5.0” na IDE Apache NetBeans. Isso permitirá que você visualize o código-fonte do projeto e inicie o processo de desenvolvimento.

**Execute o Projeto**

Com o projeto aberto no IDE, execute o projeto através do menu "Run". O programa será iniciado e exibirá o menu de opções, permitindo que você explore as funcionalidades do sistema.

**Estrutura do Projeto:**

O projeto é organizado em várias classes, cada uma com uma responsabilidade específica:

**Classe Menu**

* **Descrição:** A classe Menu é o ponto inicial do sistema, responsável por controlar a interação com o usuário. Ela apresenta um menu com opções que permitem acessar as principais funcionalidades do programa.
* **Responsabilidades:**
  + - Exibir o menu de opções ao usuário.
    - Capturar as escolhas do usuário.
    - Chamar os métodos apropriados da classe Sistema para executar as ações solicitadas.

**Classe Sistema**

* **Descrição:** A classe Sistema reúne a lógica principal do programa, sendo responsável por controlar os dados das viagens, linhas e ônibus.
* **Responsabilidades:**
  + - Gerenciar e processar os dados de viagens, ônibus e linhas.
    - Métodos para calcular o número de passageiros, identificar linhas sobrecarregadas e outras funcionalidades essenciais do sistema.
    - Interagir com as classes Viagem, Ônibus e Linha para manipular os dados conforme necessário.

**Classe Viagem**

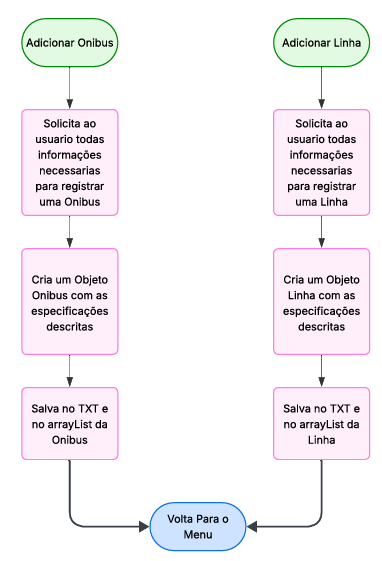
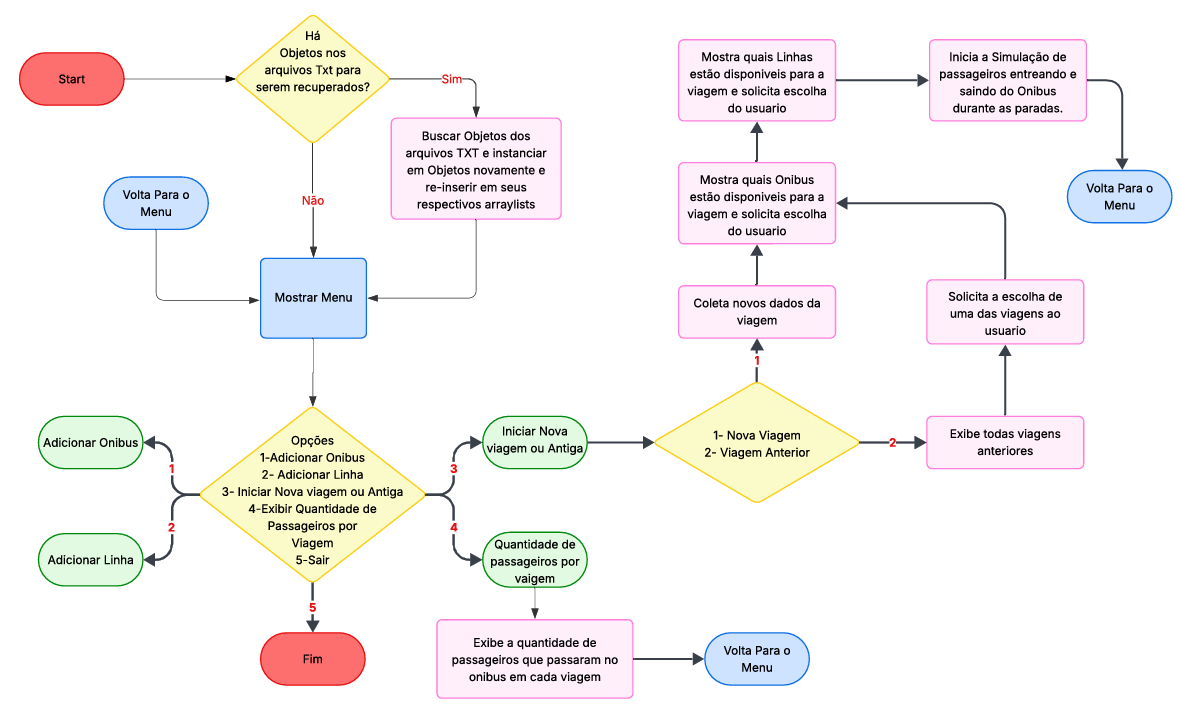
* **Descrição:** Representa uma viagem específica.
* **Responsabilidades:**
  + - Armazenar dados sobre a viagem, como número de passageiros e horários.
    - Métodos para acessar e manipular esses dados.

**Classe Linha**

* **Descrição**: Representa uma linha de ônibus.
* **Responsabilidades**:
  + - Armazenar dados sobre a linha, como nome da rota e paradas.
    - Métodos para acessar e manipular esses dados.

**Classe Ônibus**

* **Descrição**: Representa um ônibus específico.
* **Responsabilidades**:
  + - Armazenar informações sobre o ônibus, como placa e capacidade.
    - Métodos para acessar e manipular esses dados.

**Fluxograma:**

**Algoritmo:**

**Main.Java:**

**1 import java.util.ArrayList;**

**2 import java.util.Scanner;**

**3**

**4 //A classe principal que contém o método main**

**5 public class Main {**

**6**

**7 public static void *main*(String[] args) {**

**8 //Criando listas vazias para armazenar objetos do tipo Viagem, Onibus e Linha**

**9 ArrayList<Viagem> viagens = new ArrayList<>();**

**10 ArrayList<Onibus> onibus = new ArrayList<>();**

**11 ArrayList<Linha> linhas = new ArrayList<>();**

**12**

**13 //Criando um objeto do tipo Sistema, passando as listas como parâmetros**

**14 Sistema sistema = new Sistema(viagens, onibus, linhas);**

**15**

**16 //Criando um objeto Scanner para ler a entrada do usuário**

**17 Scanner scanner = new Scanner(System.*in*);**

**18**

**19 //Variável booleana para controlar o loop do menu**

**20 boolean iniciado = true;**

**21**

**22 //Laço de repetição que mantém o menu em execução até que o usuário escolha sair**

**23 while (iniciado) {**

**24 //Exibe o menu de opções para o usuário**

**25 System.*out*.println("Menu de Opções:");**

**26 System.*out*.println("1. Adicionar Ônibus");**

**27 System.*out*.println("2. Adicionar Linha");**

**28 System.*out*.println("3. Iniciar Viagem Nova ou Antiga");**

**29 System.*out*.println("4. Exibir Quantidade de Passageiros por Viagem");**

**30 System.*out*.println("5. Sair");**

**31 System.*out*.print("Escolha uma opção: ");**

**32**

**33 //Lê a escolha do usuário**

**34 String escolha = scanner.nextLine();**

**35**

**36 //Usa a escolha do usuário para saber qual ação seguir**

**37 switch (escolha) {**

**38 case "1":**

**39 //Se o usuário escolher "1", chama o método para adicionar um onibus**

**40 sistema.adicionarOnibus(scanner);**

**41 break;**

**42 case "2":**

**43 //Se o usuario escolher "2", chama o metodo para adicionar uma linha**

**44 sistema.adicionarLinha(scanner);**

**45 break;**

**46 case "3":**

**47 //Se o usuario escolher "3", chama o metodo para iniciar uma viagem**

**48 sistema.iniciarViagem(scanner);**

**49 break;**

**50 case "4":**

**51 //Se o usuario escolher "4", exibe as viagens e suas quantidades de passageiros**

**52 for (Viagem viagem : viagens) {**

**53 System.*out*.println("Viagem: " + viagem);**

**54 }**

**55 break;**

**56 case "5":**

**57 //Se o usuário escolher "5", sai do loop e encerra o programa**

**58 iniciado = false;**

**59 break;**

**60 default:**

**61 //Se o usuário digitar uma opção invalida, exibe uma mensagem**

**62 System.*out*.println("Opção inválida!");**

**63 }**

**64 }**

**65**

**66 //Fecha o scanner quando o programa termina**

**67 scanner.close();**

**68 }**

**69 }**

**70**

**Sistema.Java:**

1

2 import java.io.BufferedReader;

3 import java.io.FileReader;

4 import java.io.FileWriter;

5 import java.io.IOException;

6 import java.util.List;

7 import java.util.Random;

8 import java.util.Scanner;

9

10 public class **Sistema** {

11

12 //Listas que armazenam os Objetos de cada tipo

13 private List<Viagem> viagens;

14 private List<Onibus> onibus;

15 private List<Linha> linhas;

16

17 //Construtor da classe Sistema. Ele recebe três listas como parâmetros e as inicializa.

18 public **Sistema**(List<Viagem> viagens, List<Onibus> onibus, List<Linha> linhas) {

19 this.viagens = viagens;

20 this.onibus = onibus;

21 this.linhas = linhas;

22 carregarDados(); // Carregar dados ao iniciar o sistema

23 }

24

25 //Metodo para carregar os dados de Viagens, Onibus e Linhas ao iniciar o sistema

26 private void **carregarDados**() {

27 carregarViagens();

28 carregarOnibus();

29 carregarLinhas();

30 }

31

32 //Metodo para carregar as viagens ao iniciar o sistema a partir do TXT

33 private void **carregarViagens**() {

34 try (BufferedReader reader = new BufferedReader(new FileReader("viagens.txt"))) {

35 //Buffer para ler o arquivo "viagens.txt"

36 String linha;

37 while ((linha = reader.readLine()) != null) {

38 String[] dados = linha.split(", ");

39 String horarioSaida = dados[0].split(": ")[1];

40 String dataSaida = dados[1].split(": ")[1];

41 String dataChegada = dados[2].split(": ")[1];

42 String valorPassagem = dados[3].split(": ")[1];

43 int quantidadePassageiros = Integer.*parseInt*(dados[4].split(": ")[1]);

44

45 //Cria um Objeto com os dados recolhidos

46 Viagem viagem = new Viagem(horarioSaida, dataSaida, dataChegada, valorPassagem);

47

48 //Define a quantidade de passageiros que subiram na viagem

49 viagem.setQuantidadePassageiros(quantidadePassageiros);

50

51 //Adiciona o Objeto ao arrayList Onibus

52 viagens.add(viagem);

53 }

54 } catch (IOException e) {

55

56 //Caso o arquivo nao pode ser lido mostra mensagem de erro

57 System.*out*.println("Erro ao carregar os dados das viagens: " + e.getMessage());

58 }

59 }

60

61 //Metodo para carregar os onibus ao iniciar o sistema a partir do TXT

62 private void **carregarOnibus**() {

63 try (BufferedReader reader = new BufferedReader(new FileReader("onibus.txt"))) {

64 //Buffer para ler o arquivo "onibus.txt"

65 String linha;

66 while ((linha = reader.readLine()) != null) {

67 //Recolhe os dados do arquivo

68 String[] dados = linha.split(", ");

69 String placa = dados[0].split(": ")[1];

70 String marca = dados[1].split(": ")[1];

71 String modelo = dados[2].split(": ")[1];

72 String cor = dados[3].split(": ")[1];

73 int capacidade = Integer.*parseInt*(dados[4].split(": ")[1]);

74

75 //Cria um Objeto com os dados recolhidos

76 Onibus onibus = new Onibus(placa, marca, modelo, cor, capacidade);

77

78 //Adiciona o Objeto ao arrayList Onibus

79 this.onibus.add(onibus);

80 }

81 } catch (IOException e) {

82

83 //Caso o arquivo nao pode ser lido mostra mensagem de erro

84 System.*out*.println("Erro ao carregar os dados dos ônibus: " + e.getMessage());

85 }

86 }

87

88 //Metodo para carregar as linhas ao iniciar o sistema a partir do TXT

89 private void **carregarLinhas**() {

90 try (BufferedReader reader = new BufferedReader(new FileReader("linhas.txt"))) {

91 //Buffer para ler o arquivo "linhas.txt"

92 String linha;

93 while ((linha = reader.readLine()) != null) {

94 String[] dados = linha.split(", ");

95 int id = Integer.*parseInt*(dados[0].split(": ")[1]);

96 String nomeRota = dados[1].split(": ")[1];

97 int quantidadeParadas = Integer.*parseInt*(dados[2].split(": ")[1]);

98 String inicio = dados[3].split(": ")[1];

99 String fim = dados[4].split(": ")[1];

100

101 //Cria um Objeto com os dados recolhidos

102 Linha linhaObj = new Linha(id, nomeRota, quantidadeParadas, inicio, fim);

103

104 //Adiciona o Objeto ao arrayList Onibus

105 linhas.add(linhaObj);

106 }

107 } catch (IOException e) {

108

109 //Caso o arquivo nao pode ser lido mostra mensagem de erro

110 System.*out*.println("Erro ao carregar os dados das linhas: " + e.getMessage());

111 }

112 }

113

114 //Metodo para Adicionar uma nova viagem ao sistema

115 public void **adicionarViagem**(Scanner scanner) {

116 try {

117 //Solicitando ao usuario as informaçoes da viagem

118 System.*out*.print("Horário de saida: ");

119 String horarioSaida = scanner.nextLine().trim();

120 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

121 if (horarioSaida.isEmpty()) {

122 throw new IllegalArgumentException("Horário não pode ser vazio.");

123 }

124 System.*out*.print("Data de saída: ");

125 String dataSaida = scanner.nextLine().trim();

126 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

127 if (dataSaida.isEmpty()) {

128 throw new IllegalArgumentException("Data de saída não pode ser vazia.");

129 }

130 System.*out*.print("Data de chegada: ");

131 String dataChegada = scanner.nextLine().trim();

132 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

133 if (dataChegada.isEmpty()) {

134 throw new IllegalArgumentException("Data de chegada não pode ser vazia.");

135 }

136 System.*out*.print("Valor da passagem: ");

137 String valorPassagem = scanner.nextLine().trim();

138 valorPassagem = valorPassagem.replace(",", "."); // Substitui vírgula por ponto

139

140 //Cria um Objeto com as informaçoes que o usuario informou

141 Viagem novaViagem = new Viagem(horarioSaida, dataSaida, dataChegada, valorPassagem);

142

143 //adiciona o Objeto ao arrayList de Viagens

144 viagens.add(novaViagem);

145

146 // Salvar dados automaticamente apos adicionar uma viagem

147 salvarDados();

148

149 //correção de erros

150 } catch (NumberFormatException e) {

151 System.*err*.println("Erro: Valor da passagem deve ser um número.");

152 } catch (IllegalArgumentException e) {

153 System.*err*.println("Erro: " + e.getMessage());

154 }

155 }

156

157 //Metodo para Adicionar um novo Onibus ao sistema

158 public void **adicionarOnibus**(Scanner scanner) {

159 try {

160 //Solicitando ao usuario as informaçoes do Onibus

161 System.*out*.print("Placa do ônibus: ");

162 String placa = scanner.nextLine().trim();

163 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

164 if (placa.isEmpty()) {

165 throw new IllegalArgumentException("Placa não pode ser vazia.");

166 }

167 System.*out*.print("Marca do ônibus: ");

168 String marca = scanner.nextLine().trim();

169 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

170 if (marca.isEmpty()) {

171 throw new IllegalArgumentException("Marca não pode ser vazia.");

172 }

173 System.*out*.print("Modelo do ônibus: ");

174 String modelo = scanner.nextLine().trim();

175 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

176 if (modelo.isEmpty()) {

177 throw new IllegalArgumentException("Modelo não pode ser vazio.");

178 }

179 System.*out*.print("Cor do ônibus: ");

180 String cor = scanner.nextLine().trim();

181 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

182 if (cor.isEmpty()) {

183 throw new IllegalArgumentException("Cor não pode ser vazia.");

184 }

185 System.*out*.print("Capacidade do ônibus: ");

186 int capacidade = Integer.*parseInt*(scanner.nextLine().trim());

187 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

188 if (capacidade <= 0) {

189 throw new IllegalArgumentException("Capacidade deve ser um número positivo.");

190 }

191

192 //Cria um Objeto com as informaçoes que o usuario informou

193 onibus.add(new Onibus(placa, marca, modelo, cor, capacidade));

194

195 //Salvar dados automaticamente apos adicionar um onibus

196 salvarDados();

197

198 //Correção de erros

199 } catch (NumberFormatException e) {

200 System.*err*.println("Erro: Capacidade deve ser um número.");

201 } catch (IllegalArgumentException e) {

202 System.*err*.println("Erro: " + e.getMessage());

203 }

204 }

205

206 //Metodo para Adicionar uma nova Linha ao sistema

207 public void **adicionarLinha**(Scanner scanner) {

208 try {

209 //Solicitando ao usuario as informaçoes da Linha

210 System.*out*.print("ID da linha: ");

211 int id = Integer.*parseInt*(scanner.nextLine().trim());

212

213 System.*out*.print("Nome da rota: ");

214 String nomeRota = scanner.nextLine().trim();

215 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

216 if (nomeRota.isEmpty()) {

217 throw new IllegalArgumentException("Nome da rota não pode ser vazio.");

218 }

219 System.*out*.print("Quantidade de paradas: ");

220 int quantidadeParadas = Integer.*parseInt*(scanner.nextLine().trim());

221 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

222 if (quantidadeParadas <= 0) {

223 throw new IllegalArgumentException("Quantidade de paradas deve ser um número positivo.");

224 }

225 System.*out*.print("Início da Linha: ");

226 String inicio = scanner.nextLine().trim();

227 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

228 if (inicio.isEmpty()) {

229 throw new IllegalArgumentException("Início não pode ser vazio.");

230 }

231 System.*out*.print("Final da Linha: ");

232 String fim = scanner.nextLine().trim();

233 //Correção de erro, caso o usuario nao coloque a informação o sistema mostra mensagem de erro

234 if (fim.isEmpty()) {

235 throw new IllegalArgumentException("Final não pode ser vazio.");

236 }

237

238 //Cria um Objeto com as informaçoes que o usuario informou

239 linhas.add(new Linha(id, nomeRota, quantidadeParadas, inicio, fim));

240

241 // Salvar dados automaticamente apos adicionar uma linha

242 salvarDados();

243

244 //Correção de erros

245 } catch (NumberFormatException e) {

246 System.*err*.println("Erro: ID e Quantidade de paradas devem ser números.");

247 } catch (IllegalArgumentException e) {

248 System.*err*.println("Erro: " + e.getMessage());

249 }

250

251 }

252

253 //Metodo para salvar os dados nas listas em arquivo de texto

254 public void **salvarDados**() {

255 salvarListaEmArquivo("viagens.txt", viagens);

256 salvarListaEmArquivo("onibus.txt", onibus);

257 salvarListaEmArquivo("linhas.txt", linhas);

258 System.*out*.println("Dados salvos com sucesso!");

259 }

260

261 //Metodo que salva as informações nos arquivos usando <T>, que aceita qualquer tipo de lista

262 private <T> void **salvarListaEmArquivo**(String nomeArquivo, List<T> lista) {

263 try (FileWriter writer = new FileWriter(nomeArquivo)) {

264 for (T item : lista) { //Coleta cada informação e salva no arquivo

265 writer.write(item.toString() + "**\n**"); //Converte o Item em String para salvar

266 }

267 //Correção de erro, exibe mensagem de erro

268 } catch (IOException e) {

269 System.*out*.println("Erro ao salvar os dados: " + e.getMessage());

270 }

271 }

272

273 //Metodo para Iniciar Viagem, Solicitando ao usuario se ele quer uma nova viagem ou uma ja feita

274 public void **iniciarViagem**(Scanner scanner) {

275

276 //Exibe ao usuario as opções

277 System.*out*.println("1. Nova Viagem");

278 System.*out*.println("2. Viagem Anterior");

279

280 //Lê a opção que o usuario escolheu

281 int opcao = Integer.*parseInt*(scanner.nextLine());

282

283 //Verifica se o arrayList do Onibus e da Linha estão Vazio

284 if (onibus.isEmpty() || linhas.isEmpty()) {

285 System.*out*.println("Não há linhas ou ônibus suficientes.");

286 return;

287 }

288

289 //Usa a escolha do usuário para saber qual ação seguir

290 switch (opcao) {

291 case 1:

292 //Adicionar informações da Viagem

293 adicionarViagem(scanner);

294

295 //Mostrar quais onibus estão disponiveis

296 System.*out*.println("Ônibus disponíveis:");

297 for (int i = 0; i < onibus.size(); i++) {

298 System.*out*.println((i + 1) + ". " + onibus.get(i));

299 }

300 //Solicita ao usuario que escolha algum deles

301 System.*out*.print("Escolha o ônibus (número): ");

302 int onibusEscolhido = Integer.*parseInt*(scanner.nextLine()) - 1;

303

304 //Mostrar quais onibus estão disponiveis

305 System.*out*.println("Linhas disponíveis:");

306 for (int i = 0; i < linhas.size(); i++) {

307 System.*out*.println((i + 1) + ". " + linhas.get(i));

308 }

309 //Solicita ao usuario que escolha algum deles

310 System.*out*.print("Escolha a linha (número): ");

311 int linhaEscolhida = Integer.*parseInt*(scanner.nextLine()) - 1;

312

313 //Verificar se as escolhas existem

314 if (onibusEscolhido < 0 || onibusEscolhido >= onibus.size() || linhaEscolhida < 0 || linhaEscolhida >= linhas.size()) {

315 System.*out*.println("Escolha inválida.");

316 return;

317 }

318

319 //Atribui o Onibus e Linha escolhido a nova viagem

320 Viagem novaViagem = viagens.get(viagens.size() - 1);

321 novaViagem.setOnibus(onibus.get(onibusEscolhido));

322 novaViagem.setLinha(linhas.get(linhaEscolhida));

323

324 //Criando uma instancia da classe SimulacaoViagem e chamando o metodo correto

325 SimulacaoViagem simulacao = new SimulacaoViagem();

326

327 //Chama o metodo que simula os passageiros entrando e saindo

328 simulacao.simularPassageiros(novaViagem);

329 break;

330 case 2:

331 //Se o usuário escolher "2"

332 //Verifica se o Array Viagens esta vazio, se Vazio mostra mensagem

333 if (viagens.isEmpty()) {

334 System.*out*.println("Não há viagens feitas suficientes.");

335 return;

336 }

337 //Mostra as viagens ja feitas para ele escolher

338 System.*out*.println("Escolha a Viagem repetida:");

339 for (int i = 0; i < viagens.size(); i++) {

340 System.*out*.println((i + 1) + ". " + viagens.get(i));

341 }

342 //Solicita que o usuario escolha alguma delas

343 System.*out*.print("Escolha uma das viagens (número): ");

344 int viagemEscolhida = Integer.*parseInt*(scanner.nextLine()) - 1;

345

346 //Mostrar quais onibus estão disponiveis

347 System.*out*.println("Ônibus disponíveis:");

348 for (int i = 0; i < onibus.size(); i++) {

349 System.*out*.println((i + 1) + ". " + onibus.get(i));

350 }

351 //Solicita ao usuario que escolha algum deles

352 System.*out*.print("Escolha o ônibus (número): ");

353 int onibusEscolhido2 = Integer.*parseInt*(scanner.nextLine()) - 1;

354

355 //Mostrar quais linhas estão disponiveis

356 System.*out*.println("Linhas disponíveis:");

357 for (int i = 0; i < linhas.size(); i++) {

358 System.*out*.println((i + 1) + ". " + linhas.get(i));

359 }

360 //Solicita ao usuario que escolha alguma delas

361 System.*out*.print("Escolha a linha (número): ");

362 int linhaEscolhida2 = Integer.*parseInt*(scanner.nextLine()) - 1;

363

364 //Verificar se as escolhas existem

365 if (onibusEscolhido2 < 0 || onibusEscolhido2 >= onibus.size() || linhaEscolhida2 < 0 || linhaEscolhida2 >= linhas.size()) {

366 System.*out*.println("Escolha inválida.");

367 return;

368 }

369

370 //Atribui o Onibus e Linha escolhido a nova viagem

371 Viagem viagem = viagens.get(viagens.size() - 1);

372 viagem.setOnibus(onibus.get(onibusEscolhido2));

373 viagem.setLinha(linhas.get(linhaEscolhida2));

374

375 //Lê a escolha do usuário sobre qual viagem repetir

376 int ViagemEscolhida = Integer.*parseInt*(scanner.nextLine()) - 1;

377

378 //Verifica se a viagem escolhida existe

379 if (ViagemEscolhida < 0 || ViagemEscolhida >= viagens.size()) {

380 //Correção de erro para caso a viagem nao existir

381 System.*out*.println("Viagem inválida.");

382 return;

383 }

384

385 //Criando uma instancia da classe SimulacaoViagem e chamando o metodo correto

386 SimulacaoViagem simulacao2 = new SimulacaoViagem();

387

388 //Chama o metodo que simula os passageiros entrando e saindo

389 simulacao2.simularPassageiros(viagem);

390 break;

391 default:

392 //Correção de erro, caso a opção escolhida nao exista.

393 System.*out*.println("Opção inválida.");

394 }

395 }

396

397 //Metodo que simula a entrada e saida de passageiros

398 public class **SimulacaoViagem** {

399

400 //Atributos da classe para armazenar os totais

401 private int totalPassageirosEntraram = 0;

402 private int totalPassageirosNaoEntraram = 0;

403

404 //Metodo para simular a entrada e saída de passageiros

405 public void **simularPassageiros**(Viagem viagem) {

406 //Random para gerar numeros aleatorios de passageiros

407 Random random = new Random();

408 //Coleta a quantidade de paradas da Linha

409 int paradas = linhas.get(0).getQuantidadeParadas();

410 //Coleta a capacidade do Onibus

411 int capacidadeOnibus = onibus.get(0).getCapacidade();

412 int passageirosNoOnibus = 0;

413 int passageirosNaoEntraram = 0;

414

415 //Loop que simula cada parada

416 for (int i = 1; i <= paradas; i++) {

417 // Numero aleatorio de passageiros entrando

418 int passageirosEntrando = random.nextInt(10);

419 //Numero aleatorio de passageiros saindo

420 int passageirosSaindo = random.nextInt(passageirosNoOnibus + 1);

421

422 //Atualiza a quantidade de passageiros que estão no Onibus

423 passageirosNoOnibus -= passageirosSaindo;

424

425 //Verifica se a quantidade de passageiros entrando é maior que a capacidade do Onibus

426 if (passageirosNoOnibus + passageirosEntrando > capacidadeOnibus) {

427 //Calcula quantos passageiros não consiguiram entrar

428 passageirosNaoEntraram = (passageirosNoOnibus + passageirosEntrando) - capacidadeOnibus;

429

430 //Ajusta os passageiros que conseguiram entrar no Onibus

431 passageirosEntrando = capacidadeOnibus - passageirosNoOnibus;

432

433 //Onibus esta Cheio

434 passageirosNoOnibus = capacidadeOnibus;

435

436 //Atualizando o contador

437 totalPassageirosNaoEntraram += passageirosNaoEntraram;

438 totalPassageirosEntraram += passageirosEntrando;

439

440 //Exibe as informações da parada

441 System.*out*.println("Parada " + i + ": " + passageirosEntrando

442 + " passageiros entraram, " + passageirosSaindo

443 + " passageiros saíram. Total: " + passageirosNoOnibus);

444 System.*out*.println(passageirosNaoEntraram + " passageiros não conseguiram entrar.");

445 } else {

446 passageirosNoOnibus += passageirosEntrando;

447 //Atualizando o contador

448 totalPassageirosEntraram += passageirosEntrando;

449 System.*out*.println("Parada " + i + ": " + passageirosEntrando

450 + " passageiros entraram, " + passageirosSaindo

451 + " passageiros saíram. Total: " + passageirosNoOnibus);

452 }

453 }

454

455 //Finalizando a viagem com onibus vazio

456 if (passageirosNoOnibus > 0) {

457 System.*out*.println("Chegada final: Todos os " + passageirosNoOnibus + " passageiros saíram.");

458 passageirosNoOnibus = 0;

459 }

460

461 //Atualizar a quantidade de passageiros na viagem

462 viagem.setQuantidadePassageiros(totalPassageirosEntraram);

463 }

464

465 //Metodo para exibir o total de passageiros que entraram e não entraram

466 public void **exibirQuantidadeDePassageirosPorViagem**() {

467 System.*out*.println("Total de passageiros que entraram na viagem: " + totalPassageirosEntraram);

468 System.*out*.println("Total de passageiros que não conseguiram entrar: " + totalPassageirosNaoEntraram);

469 }

470 }

471 }

472

**Viagem.Java:**

1

2 public class **Viagem** {

3

4 //Variaveis da Classe Viagem

5 private String horarioSaida;

6 private String dataSaida;

7 private String dataChegada;

8 private String valorPassagem;

9 private int quantidadePassageiros;

10 private Onibus onibus;

11 private Linha linha;

12

13 //Construtor da Classe Viagem

14 public **Viagem**(String horarioSaida, String dataSaida, String dataChegada, String valorPassagem) {

15 this.horarioSaida = horarioSaida;

16 this.dataSaida = dataSaida;

17 this.dataChegada = dataChegada;

18 this.valorPassagem = valorPassagem;

19 this.quantidadePassageiros = 0;

20 }

21

22 public String **getHorario**() {

23 return horarioSaida;

24 }

25

26 public String **getDataSaida**() {

27 return dataSaida;

28 }

29

30 public String **getDataChegada**() {

31 return dataChegada;

32 }

33

34 public String **getValorPassagem**() {

35 return valorPassagem;

36 }

37

38 public int **getQuantidadePassageiros**() {

39 return quantidadePassageiros;

40 }

41

42 public void **setHorario**(String horario) {

43 this.horarioSaida = horario;

44 }

45

46 public void **setDataSaida**(String dataSaida) {

47 this.dataSaida = dataSaida;

48 }

49

50 public void **setDataChegada**(String dataChegada) {

51 this.dataChegada = dataChegada;

52 }

53

54 public void **setValorPassagem**(String valorPassagem) {

55 this.valorPassagem = valorPassagem;

56 }

57

58 public void **setQuantidadePassageiros**(int quantidadePassageiros) {

59 this.quantidadePassageiros = quantidadePassageiros;

60 }

61

62 public void **setOnibus**(Onibus onibus) {

63 this.onibus = onibus;

64 }

65

66 public void **setLinha**(Linha linha) {

67 this.linha = linha;

68 }

69

70 //Override evita erros

71 //String toString serve para facilitar a visualização das informações

72 @Override

73 public String **toString**() {

74 return "Horário: " + horarioSaida

75 + ", Data de Saída: " + dataSaida

76 + ", Data de Chegada: " + dataChegada

77 + ", Valor da Passagem: R$ " + valorPassagem

78 + ", Quantidade de Passageiros: " + quantidadePassageiros;

79 }

80 }

81

**Onibus.Java:  
  
1**

**2 public class Onibus {**

**3**

**4 //Variaveis da Classe Onibus**

**5 private String placa;**

**6 private String marca;**

**7 private String modelo;**

**8 private String cor;**

**9 private int capacidade;**

**10**

**11 //Construtor da Classe Onibus**

**12 public Onibus(String placa, String marca, String modelo, String cor, int capacidade) {**

**13 this.placa = placa;**

**14 this.marca = marca;**

**15 this.modelo = modelo;**

**16 this.cor = cor;**

**17 this.capacidade = capacidade;**

**18 }**

**19**

**20 public String getPlaca() {**

**21 return placa;**

**22 }**

**23**

**24 public String getMarca() {**

**25 return marca;**

**26 }**

**27**

**28 public String getModelo() {**

**29 return modelo;**

**30 }**

**31**

**32 public String getCor() {**

**33 return cor;**

**34 }**

**35**

**36 public int getCapacidade() {**

**37 return capacidade;**

**38 }**

**39**

**40 public void setPlaca(String placa) {**

**41 this.placa = placa;**

**42 }**

**43**

**44 public void setMarca(String marca) {**

**45 this.marca = marca;**

**46 }**

**47**

**48 public void setModelo(String modelo) {**

**49 this.modelo = modelo;**

**50 }**

**51**

**52 public void setCor(String cor) {**

**53 this.cor = cor;**

**54 }**

**55**

**56 public void setCapacidade(int capacidade) {**

**57 this.capacidade = capacidade;**

**58 }**

**59**

**60 //Override evita erros**

**61 //String toString serve para facilitar a visualização das informações**

**62 @Override**

**63 public String toString() {**

**64 return "Placa: " + placa**

**65 + ", Marca: " + marca**

**66 + ", Modelo: " + modelo**

**67 + ", Cor: " + cor**

**68 + ", Capacidade: " + capacidade;**

**69 }**

**70 }**

**71**

**Linha.Java:**

1

2 public class **Linha** {

3

4 //Variaveis da Classe Linha

5 private int id;

6 private String nomeRota;

7 private int quantidadeParadas;

8 private String inicio;

9 private String fim;

10

11 //Construtor da Classe Linha

12 public **Linha**(int id, String nomeRota, int quantidadeParadas, String inicio, String fim) {

13 this.id = id;

14 this.nomeRota = nomeRota;

15 this.quantidadeParadas = quantidadeParadas;

16 this.inicio = inicio;

17 this.fim = fim;

18 }

19

20 public int **getId**() {

21 return id;

22 }

23

24 public String **getNomeRota**() {

25 return nomeRota;

26 }

27

28 public int **getQuantidadeParadas**() {

29 return quantidadeParadas;

30 }

31

32 public String **getInicio**() {

33 return inicio;

34 }

35

36 public String **getFim**() {

37 return fim;

38 }

39

40 public void **setId**(int id) {

41 this.id = id;

42 }

43

44 public void **setNomeRota**(String nomeRota) {

45 this.nomeRota = nomeRota;

46 }

47

48 public void **setQuantidadeParadas**(int quantidadeParadas) {

49 this.quantidadeParadas = quantidadeParadas;

50 }

51

52 public void **setInicio**(String inicio) {

53 this.inicio = inicio;

54 }

55

56 public void **setFim**(String fim) {

57 this.fim = fim;

58 }

59

60 //Override evita erros

61 //String toString serve para facilitar a visualização das informações

62 @Override

63 public String **toString**() {

64 return "ID: " + id

65 + ", Nome da Rota: " + nomeRota

66 + ", Quantidade de Paradas: " + quantidadeParadas

67 + ", Início: " + inicio

68 + ", Final: " + fim;

69 }

70 }

71