# Documento Técnico — Simulador de Elevadores Inteligentes

## 1. Introdução

Este documento técnico descreve a modelagem, as estruturas de dados utilizadas e os principais algoritmos implementados no simulador de elevadores inteligentes. O sistema simula o funcionamento de um prédio com múltiplos elevadores, gerenciando passageiros com diferentes perfis (idosos, cadeirantes, peso), filas de espera e controle centralizado para otimização do atendimento.

---

## 2. Modelagem do Sistema

O sistema é composto por várias classes que representam os elementos essenciais: elevadores, passageiros, filas de espera e o simulador principal que orquestra a simulação. Abaixo estão as principais classes e seus papéis.

---

## 3. Classes e Estruturas de Dados

### 3.1. Classe Passageiro

- **Descrição:**
  Representa um passageiro do sistema, incluindo atributos essenciais para definir prioridades e destino dentro do prédio.

- **Atributos principais:**

    - id (int): Identificador único do passageiro.

    - peso (int): Peso do passageiro em kg.

    - idade (int): Idade do passageiro, usada para determinar prioridade (idosos).

- cadeirante (boolean): Indica se é cadeirante, conferindo prioridade especial.
- andarOrigem (int): Andar onde o passageiro aguarda o elevador.
- andarDestino (int): Andar para o qual o passageiro deseja ir.
- embaracado (boolean): Estado que indica se o passageiro está embarcado no elevador.

- **Métodos principais:**
  - Getters para os atributos.
  - Método para marcar o passageiro como embarcado/desembaracado.
  - toString() para exibir informações do passageiro, incluindo prioridade.

---

### 3.2. Classe Elevador

- **Descrição:**
  Representa um elevador, controlando seus passageiros, capacidade máxima, andar atual, consumo de energia e histórico de movimentação.

- **Estrutura de dados:**
  - Usa um array fixo para armazenar passageiros embarcados, evitando o uso de estruturas prontas do Java.
  - Variáveis primitivas para controle de peso, quantidade de passageiros, andares percorridos e energia consumida.

- **Principais métodos:**
  - podeEmbarcar(Passageiro p): Verifica se o passageiro pode ser embarcado respeitando limites de peso e lotação.
  - embarcar(Passageiro p): Embarca o passageiro, atualizando peso e quantidade.

- desembarcarPassageirosNoAndar(): Remove passageiros cujo andar destino é o atual, atualizando peso e lista.

- moverParaAndar(int destino): Move o elevador, atualizando energia e andares percorridos.

- Métodos getters para obter estado do elevador.

- toString() para exibir estado atual e passageiros.

---

### 3.3. Classe FilaPassageiros

- **Descrição:**
  Implementa uma fila circular para gerenciar a ordem dos passageiros que aguardam o elevador em um determinado andar.

- **Estrutura de dados:**

  - Array fixo para armazenar os passageiros da fila.

  - Controle manual dos índices inicio e fim para o funcionamento circular da fila.

  - Variável tamanho para indicar quantos passageiros estão na fila.

- **Principais métodos:**

  - enfileirar(Passageiro p): Adiciona um passageiro ao fim da fila, se houver capacidade.

  - desenfileirar(): Remove e retorna o passageiro do início da fila.

  - estaVazia(): Verifica se a fila está vazia.

  - tamanho(): Retorna a quantidade atual de passageiros na fila.

  - toString(): Representação textual da fila com todos os passageiros.

---

### 3.4. Classe SimuladorElevadores

- **Descrição:**
  Classe principal responsável por controlar toda a simulação, integrando elevadores, passageiros, filas e regras de operação.

- **Estrutura de dados:**
  - Arrays fixos para os elevadores do sistema.
  - Filas de passageiros para cada andar, representadas por instâncias da FilaPassageiros.
  - Variáveis para controle de tempo, energia total consumida, histórico de eventos e estatísticas da simulação.

- **Principais métodos:**
  - iniciarSimulacao(): Gerencia o fluxo de tempo da simulação, gera passageiros, aciona elevadores e atualiza estados.
  - atualizarEstado(): Atualiza o estado dos elevadores e filas a cada passo da simulação.
  - exibirStatus(): Exibe o status atual, incluindo andares, passageiros e consumo de energia.
  - Métodos para aplicar heurísticas de prioridade no atendimento e controle de filas.
  - Métodos auxiliares para cálculos e geração de relatórios finais.

---

# 4. Algoritmos Implementados

### 4.1. Controle de Embarque e Desembarque

- **Embarque:**
  O elevador verifica se o passageiro cabe dentro dos limites de peso e capacidade. Se sim, o passageiro é adicionado ao array interno do elevador, e o peso total é atualizado.

- **Desembarque:**
  O elevador verifica os passageiros cujo andar destino coincide com o andar atual. Esses passageiros são removidos da lista, o peso é descontado, e o espaço interno liberado.

### 4.2. Movimentação dos Elevadores

- O elevador se move para o andar destino indicado pela heurística ou pelo sistema de controle.

- A distância percorrida é calculada pela diferença entre o andar atual e o destino.

- A energia consumida é atualizada multiplicando a distância percorrida por uma constante fixa (exemplo: 10 unidades por andar).

### 4.3. Filas Circulares para Passageiros

- Cada andar possui uma fila circular que armazena os passageiros que aguardam atendimento.

- Essa estrutura garante que o tempo e o espaço para filas sejam controlados, evitando estouros e mantendo ordem de chegada.

### 4.4. Heurística e Prioridades

- Passageiros idosos e cadeirantes recebem prioridade para embarque.

- O sistema pode ordenar filas e controlar embarques respeitando esses critérios.

- O sistema evita exceder limites de peso e lotação para segurança e eficiência.

---

## 5. Considerações Finais

O sistema foi projetado para funcionar sem o uso de estruturas prontas do Java, utilizando arrays fixos e controle manual de índices para as filas e listas de

passageiros. Isso aumenta a transparência do código e permite maior controle sobre a simulação.

A modelagem das classes permite escalabilidade para futuros aprimoramentos, como inclusão de diferentes heurísticas, painéis de controle variados e registro detalhado de consumo energético.