Relatório do Projeto: Simulador de Elevadores Inteligentes com Estruturas de Dados Próprias

Whuanderson de Sousa Porto Marinho
ICEV - Instituto de Ensino Superior
whuanderson.marinho@somosicev.com.br

André Nogueira Barbosa Dantas Teixeira
ICEV - Instituto de Ensino Superior
andre.teixeira@somosicev.com.br

Resumo

Este artigo apresenta a implementação de um **Simulador de Elevadores Inteligentes**, desenvolvido como trabalho final da disciplina de Estruturas de Dados. O projeto visa demonstrar o funcionamento de estruturas de dados fundamentais, como listas, filas e ponteiros, através da simulação da operação de múltiplos elevadores em edifícios com diversos andares. A proposta central foi a de utilizar **somente estruturas de dados próprias**, evitando coleções prontas da linguagem Java, garantindo o domínio total dos conceitos teóricos envolvidos. O sistema incorpora variáveis como tempo de viagem, filas de espera e priorização de usuários, além de aplicar heurísticas de controle para otimização de tempo e energia. A aplicação possui uma interface gráfica interativa desenvolvida em JavaFX, permitindo a configuração flexível de parâmetros e a visualização do movimento dos elevadores.

Palavras-chave: Estruturas de Dados, Java, JavaFX, Simulador de Elevador, Algoritmos de Controle, Filas, Listas.

---

1. Introdução

A disciplina de Estruturas de Dados é fundamental para a formação em Engenharia de Software, abordando conceitos essenciais para a organização e manipulação eficiente de dados. Com o intuito de solidificar o conhecimento prático desses conceitos, este projeto propõe o desenvolvimento de um simulador de elevadores. A complexidade do cenário de controle de elevadores oferece um ambiente rico para a aplicação de diversas estruturas de dados e algoritmos, permitindo a exploração de desafios como gerenciamento de filas de espera, otimização de rotas e priorização de requisições.

O objetivo principal deste projeto é implementar e demonstrar o funcionamento de diversas estruturas de dados fundamentais (listas, pilhas, filas), construindo-as do

zero, sem o uso de coleções prontas. Além disso, busca-se desenvolver um sistema que simule realisticamente a operação de elevadores, incorporando diferentes estratégias de controle e a interação com uma interface gráfica. O projeto visa, por fim, oferecer uma experiência prática no desenvolvimento e implementação passo a passo de um sistema complexo.

2. Solução Proposta e Detalhes da Implementação

O projeto "Estrutura_Dados" é um simulador de elevadores implementado em Java com interface gráfica utilizando JavaFX. A arquitetura do sistema é modular, organizada em pacotes para facilitar a manutenção e expansão. O código-fonte está localizado no diretório src/ do github https://github.com/Whuanderson/Estrutura_Dados.git[1].

2.1. Arquitetura e Componentes do Sistema

O projeto é estruturado nos seguintes pacotes Java, cada um com funções específicas:

- **Base**: Contém a modelagem dos objetos essenciais da simulação, como Elevador, Pessoa, Andar e Predio.
- **EstruturaDados**: Inclui as implementações próprias de estruturas de dados como Lista, Fila e Ponteiro, criados de forma geral, para poder chamar em várias instancias.
- **Simulacao**: Gerencia o fluxo da simulação e o controle principal, incluindo a classe Simulador e CentralDeControle.
- **Controle**: Onde as diferentes heurísticas para decisões inteligentes dos elevadores são implementadas.
- **Metricas**: Responsável por coletar e armazenar dados de desempenho da simulação, desde o tempo, consumo de energia, tempo e pessoas, entre muitos outros.
- **View**: Contém os arquivos relacionados à interface gráfica do usuário, desenvolvida com JavaFX.

2.2. Implementação das Estruturas de Dados

Conforme os requisitos do projeto, foram implementadas estruturas de dados próprias para garantir o domínio dos conceitos fundamentais:

- **Ponteiro.java**: Representa um nó fundamental para listas encadeadas, armazenando um elemento e uma referência para o próximo nó.

- **Lista.java**: Implementação de uma lista encadeada simples, com métodos para inserção no final (inserirFim), obtenção de elemento por posição (getElementoNaPosicao), e gerenciamento de inicio e fim.
- **Fila.java**: Implementação de uma fila FIFO (First-In, First-Out), com métodos para enfileirar, desenfileirar, verificar se estaVazia e obter o primeiro elemento. Inclui um método enfileirarPrioritario para adicionar elementos no início da fila, usado para usuários com prioridade.
- **FilaComPrioridade.java**: Utiliza duas instâncias de Fila (filaVip e filaNormal) para gerenciar prioridades de forma separada, desenfileirando primeiramente da fila VIP.

2.3. Lógica da Simulação

- **Pessoa.java**: Representa um usuário com atributos como ID, andar de origem e destino, e flags para indicar se é idoso ou cadeirante (determinando prioridade). Registra tempos de chegada, entrada/saída de fila e elevador para cálculo de métricas.
- **Andar.java**: Cada andar possui um número, uma fila de pessoas aguardando (pessoasAguardando), um painel de chamadas (painel) e uma lista de pessoas presentes no andar (pessoasPresentes). Gerencia a adição de pessoas à fila (com prioridade), o embarque de passageiros e o retorno de pessoas ao térreo após um tempo de permanência.
- **Elevador.java**: Define um elevador com ID, andar atual, capacidade máxima, direção (subindo), e uma lista de passageiros (passageiros). Gerencia o movimento, a adição/remoção de passageiros, pausas para desembarque e coleta de métricas.
- **Predio.java**: Agrega a lista de Andares e a CentralDeControle. Coordena a atualização de todos os andares e da central a cada minuto simulado.
- **CentralDeControle.java**: Coordenador principal dos elevadores, contendo a lista de Elevadores e Andares. Aplica a HeuristicaControle para tomar decisões e gerencia o processamento de tarefas dos elevadores, além de exibir métricas periodicamente.
- **Simulador.java**: Classe principal da simulação, responsável por configurar o ambiente (andares, elevadores, pessoas, velocidade), iniciar e controlar o loop de tempo, gerar pessoas dinamicamente (com variações para horários de pico), e imprimir o relatório final de métricas. Permite pausar, continuar e encerrar a simulação, e possui funcionalidade de salvar/carregar o estado.

2.4. Heurísticas de Controle

O controlador de elevadores pode aplicar uma das três heurísticas de funcionamento:

- **Modelo 1: Sem Otimização (HeuristicaSemOtimizacao.java)**: Atendimento simples por ordem de chegada, sem priorização ou ajustes dinâmicos. Os elevadores seguem sua lógica interna autônoma.
- **Modelo 2: Otimização por Tempo (HeuristicaTempo.java)**: Ajusta dinamicamente os ciclos dos elevadores para reduzir o tempo de espera da fila, agrupando destinos semelhantes e buscando o elevador mais adequado.
- **Modelo 3: Otimização por Energia (HeuristicaEnergia.java)**: Minimiza deslocamentos desnecessários e ajusta os ciclos de operação, buscando o elevador mais próximo e com capacidade disponível para economizar energia.

2.5. Interface Gráfica (JavaFX)

A interface gráfica do simulador é desenvolvida com JavaFX, iniciando pela classe App.java. O TelaInicialController.java gerencia os elementos da tela inicial, permitindo a configuração de parâmetros (número de andares, elevadores, pessoas, tempo de simulação), seleção da heurística, e o controle da simulação (iniciar, pausar, retomar, cancelar).

A visualização do prédio e dos elevadores é feita através do BuildingPane.java e DoorPane.java. O BuildingPane organiza visualmente os elevadores em uma grade, mostrando sua posição e quantidade de passageiros. O DoorPane adiciona uma animação simples para representar a abertura e fechamento das portas do elevador. As informações sobre pessoas em cada andar são exibidas ao lado da grade do prédio.

3. Análise dos Requisitos

A seguir, uma análise dos requisitos funcionais (RF) e não funcionais (RNF) e como o projeto os atende:

3.1. Requisitos Funcionais

- **RF1) O prédio deve possuir entre 5 e X andares e de 1 a N elevadores, todos controlados por uma central.**
    - **Status:** Atendido. O Simulador.java permite configurar andares e elevadores no método configurar. A CentralDeControle gerencia todos os elevadores.

- **RF2) Cada andar pode ter uma única fila de espera para todos os elevadores ou filas separadas por elevador.**
    - **Status:** Atendido (única fila por andar). Cada Andar possui uma Fila pessoasAguardando única para todas as chamadas daquele andar. A priorização é feita dentro desta fila.
- **RF3) Os painéis externos de chamada poderão ser configurados em três formatos: Painel numérico para digitar diretamente o andar desejado.**
    - **Status:** Atendido. O PainelElevador.java possui um método registrarChamada(int destino), que se assemelha a um botão de chamada geral.
- **RF4) O painel interno dos elevadores deverá ser definido de forma a complementar o painel externo escolhido.**
    - **Status:** Painel do elevador não tem painel interno de chamada, logo não é possível atender esse requisito.
- **RF5) O sistema deve simular o tempo de viagem de cada elevador, que varia com o andar e com o horário (com parâmetros distintos para horários de pico e fora de pico).**
    - **Status:** Atendido. O simulador avança o tempo em "minutos simulados", e a geração de pessoas varia em "horários de pico" (manhã e tarde). O Elevador calcula tempoDecorrido para métricas.
- **RF6) O controlador de elevadores deve aplicar uma das três heurísticas de funcionamento:**
    - **Status:** Atendido. As três heurísticas (HeuristicaSemOtimizacao, HeuristicaTempo, HeuristicaEnergia) são implementadas como classes distintas que estendem HeuristicaControle. O usuário pode selecioná-las na interface inicial.
- **RF7) Prioridades devem ser atribuídas a usuários cadeirantes e idosos, influenciando a ordem de atendimento.**
    - **Status:** Atendido. A classe Pessoa possui atributos idoso e cadeirante, e o método isPrioritaria(). A classe Fila tem o método enfileirarPrioritario(), que é utilizado pelo Andar para adicionar pessoas prioritárias ao início da fila de espera.

4.2. Requisitos Não Funcionais

- **RNF1) O sistema deve ser modular e permitir a fácil configuração de parâmetros como número de andares, elevadores, tempos de viagem, capacidade dos elevadores, e tipos de painéis.**
    - **Status:** Atendido. A arquitetura é modular, com pacotes bem definidos. Parâmetros como número de andares, elevadores e pessoas são configuráveis através da interface gráfica. A capacidade do elevador é um atributo da classe Elevador.
- **RNF2) Deve utilizar de forma eficiente estruturas de dados, como filas e listas, para o gerenciamento dos eventos e das chamadas.**
    - **Status:** Atendido. O projeto é construído em torno de implementações próprias de Lista, Fila e Ponteiro. Estas são utilizadas extensivamente para gerenciar passageiros em filas de espera nos andares, passageiros dentro dos elevadores, e a estrutura do prédio (lista de andares).
- **RNF3) A interface do simulador deverá apresentar logs e, se possível, uma visualização gráfica simples do estado do sistema.**
    - **Status:** Parcialmente Atendido. O sistema gera logs detalhados no console (System.out.println) sobre o movimento dos elevadores, embarque/desembarque de pessoas, e geração de passageiros. A interface gráfica mostra uma visualização simples do prédio e a posição dos elevadores, com a contagem de passageiros. No entanto, a interface mostrando o gráfico com os resultados das métricas ainda não foi implementada, sendo um requisito futuro.

5. Resultados e Discussão

O projeto implementa com sucesso um simulador funcional de elevadores, utilizando estruturas de dados customizadas, o que contribuiu significativamente para o aprendizado dos conceitos. As simulações podem ser configuradas com diferentes parâmetros e heurísticas, permitindo a análise de diversos cenários.

As métricas coletadas por elevador incluem energia total gasta, tempo total em movimentação, tempo total parado, número de viagens e número de pessoas transportadas. Um relatório final consolidado dessas métricas é exibido no console ao encerrar a simulação.

A principal limitação atual, mencionada no relatório do projeto, é a ausência da interface gráfica para a visualização dos resultados das métricas. Atualmente, esses dados são apresentados apenas no console.

6. Conclusão

O projeto "Estrutura_Dados" atingiu o objetivo de fornecer uma base prática para o estudo e compreensão das principais estruturas de dados em Java. A implementação de listas e filas próprias, bem como a aplicação de diferentes heurísticas de controle de elevadores, demonstram um bom entendimento dos conceitos abordados na disciplina. A organização modular do código facilita a leitura e manutenção. O projeto cumpriu os requisitos funcionais e não funcionais em completude, tendo adicionalmente desenvolvido o Front-end para a aplicação, sendo uma animação realmente complexa de ser desenvolvida em javaFX, apesar de alguns necessitar um desenvolvimento maior da interface gráfica, o projeto serve como uma experiência valiosa no desenvolvimento e na aplicação de estruturas de dados em um cenário de simulação real.

7. Trabalhos Futuros

As seguintes melhorias estão planejadas para o projeto:

- **Exportação Automática de Relatórios:** Geração de relatórios completos com gráficos finais integrados para análises externas. Este ponto abordaria a lacuna da visualização gráfica dos resultados.
- **Animação Avançada:** Detalhamento visual aprimorado, incluindo animação das portas dos elevadores e representação visual das filas de pessoas em cada andar.

8. Referencias Bibliográficas

[1] MARINHO, Whuanderson; NOGUEIRA, André. **Estrutura_Dados**. [S. l.]: GitHub, [2025]. Disponível em: https://github.com/Whuanderson/Estrutura_Dados.git. Acesso em: 17 jun. 2025.