# **Documentação do Simulador de Elevadores**

## **Visão Geral**

O **Simulador de Elevadores** é um sistema que modela a operação de elevadores em um prédio, permitindo a configuração de parâmetros como número de andares, elevadores, capacidade, tipos de painéis e heurísticas de controle. Ele suporta prioridades para usuários especiais (idosos, cadeirantes), diferentes tempos de viagem (pico e fora de pico) e coleta métricas como tempo de espera e consumo de energia. A interface gráfica exativa permite interação e visualização em tempo real.

## **Classes e Funcionalidades**

### **1. Andar.java**

**Descrição**: Representa um andar do prédio, gerenciando a fila de pessoas aguardando e o painel de controle do andar.

**Funcionalidades**:

* **Inicialização**: Constrói um andar com um número específico e um tipo de painel (UNICO\_BOTAO, DOIS\_BOTOES ou PAINEL\_NUMERICO).
* **Adicionar Pessoa**: Insere uma pessoa na fila de espera (pessoasAguardando) e aciona o painel de acordo com o tipo:
  + UNICO\_BOTAO: Pressiona chamada geral.
  + DOIS\_BOTOES: Pressiona botão de subir ou descer com base no destino da pessoa.
  + PAINEL\_NUMERICO: Pressiona o andar de destino diretamente.
* **Listar Pessoas**: Exibe as pessoas aguardando no andar.
* **Remover Pessoa**: Remove uma pessoa específica da fila de espera.
* **Getters**: Fornece acesso ao número do andar, à fila de pessoas e ao painel.

**Estruturas de Dados**:

* **FilaPrioridade (**pessoasAguardando**)**: Fila personalizada que prioriza pessoas com atributo prioritaria (idosos, cadeirantes).

### **2. CentralDeControle.java**

**Descrição**: Coordena múltiplos elevadores, atualizando seus estados e escolhendo destinos com base em heurísticas.

**Funcionalidades**:

* **Inicialização**: Cria uma lista de elevadores com configurações específicas (capacidade, tempos de viagem, heurística).
* **Atualização**: Chama o método atualizar de cada elevador a cada minuto simulado.
* **Escolha de Destino**: Determina o próximo destino de um elevador com base na heurística selecionada:
  + **Modelo 1 (Ordem de Chegada)**: Escolhe o primeiro andar com chamada válida na direção do elevador, ordenando andares de forma crescente (subindo) ou decrescente (descendo).
  + **Modelo 2 (Otimização de Tempo)**: Prioriza andares com maior número de pessoas ou chamadas, ponderando por distância.
  + **Modelo 3 (Otimização de Energia)**: Escolhe o andar mais próximo com chamadas válidas, considerando o tamanho da fila para desempate.
* **Gestão de Destinos**: Evita atribuir andares já designados a outros elevadores.
* **Inversão de Direção**: Altera a direção do elevador (subindo) se não houver chamadas na direção atual.

**Estruturas de Dados**:

* **Lista (**elevadores**)**: Lista encadeada personalizada para armazenar objetos Elevador.
* **Lista (**destinosAtribuidos**)**: Lista temporária para rastrear andares já atribuídos a outros elevadores.

### **3. Elevador.java**

**Descrição**: Modela um elevador, gerenciando movimento, embarque/desembarque de pessoas e logs.

**Funcionalidades**:

* **Inicialização**: Define o elevador com ID, capacidade, tempos de viagem (pico e fora de pico), prédio e heurística.
* **Atualização**: Atualiza o estado do elevador a cada minuto:
  + **Em Movimento**: Decrementa o tempo restante para chegar ao destino; move o elevador um andar por vez.
  + **Parado**: Realiza desembarque/embarque no andar atual, remove o andar da lista de destinos e solicita novo destino à CentralDeControle.
* **Desembarcar Pessoas**: Remove pessoas cujo destino é o andar atual, atualizando logs e estatísticas.
* **Embarcar Pessoas**: Adiciona pessoas da fila do andar ao elevador, respeitando a capacidade e dando prioridade a pessoas prioritárias.
* **Logs**: Registra eventos (embarque, desembarque, escolha de destino) em uma lista de logs.
* **Gestão de Energia**: Registra consumo de energia (0.5 unidades por parada, 1.0 unidade por movimento).

**Estruturas de Dados**:

* **Fila (**pessoasNoElevador**)**: Fila personalizada para armazenar pessoas dentro do elevador.
* **Lista (**destinos**)**: Lista encadeada para armazenar andares de destino.
* **Lista (**logs**)**: Lista encadeada para armazenar objetos LogElevador.

### **4. EntidadeSimulavel.java**

**Descrição**: Classe abstrata que define a interface para entidades que evoluem com o tempo.

**Funcionalidades**:

* **Método Abstrato** atualizar: Força subclasses (Elevador, CentralDeControle) a implementar lógica de atualização por minuto simulado.

**Estruturas de Dados**: Nenhuma, apenas define comportamento.

### **5. Estatisticas.java**

**Descrição**: Coleta e calcula métricas da simulação.

**Funcionalidades**:

* **Registro de Dados**:
  + Tempo de espera de cada pessoa.
  + Chamadas atendidas.
  + Energia consumida.
  + Total de pessoas transportadas.
* **Cálculo de Métricas**:
  + Tempo médio de espera (média dos tempos registrados).
  + Total de chamadas atendidas, energia consumida e pessoas transportadas.
* **Reinicialização**: Zera todas as métricas.

**Estruturas de Dados**:

* **ArrayList (**temposEspera**)**: Lista padrão do Java para armazenar tempos de espera (única exceção ao uso de estruturas personalizadas).

### **6. Fila.java**

**Descrição**: Implementa uma fila genérica encadeada.

**Funcionalidades**:

* **Enfileirar**: Adiciona um elemento ao final da fila.
* **Desenfileirar**: Remove e retorna o elemento do início da fila.
* **Verificações**: Métodos para verificar se a fila está vazia e obter seu tamanho.
* **Iteração**: Suporta iteração via Ponteiro para percorrer os elementos.

**Estruturas de Dados**:

* **No**: Classe interna que representa um nó da fila, contendo um elemento e um ponteiro para o próximo nó.
* **Ponteiro**: Objeto que facilita a iteração pelos nós.

### **7. FilaPrioridade.java**

**Descrição**: Estende Fila para priorizar elementos (pessoas) com atributo prioritaria.

**Funcionalidades**:

* **Enfileirar com Prioridade**: Insere pessoas prioritárias no início da fila; pessoas não prioritárias seguem a lógica padrão de Fila.

**Estruturas de Dados**:

* Herda a estrutura de Fila (nós encadeados).

### **8. GerenciadorSimulacao.java**

**Descrição**: Gera uma lista de pessoas com atributos aleatórios para a simulação.

**Funcionalidades**:

* **Gerar Pessoas**: Cria uma lista de pessoas com:
  + ID único.
  + Andar de origem e destino (diferentes, gerados aleatoriamente).
  + Prioridade (50% de chance de ser prioritária).
  + Minuto de chegada (0 a 60 minutos, aleatório).
* **Retorno**: Devolve uma lista de objetos Pessoa.

**Estruturas de Dados**:

* **Lista (**pessoas**)**: Lista encadeada personalizada para armazenar objetos Pessoa.

### **9. InterfaceGrafica.java**

**Descrição**: Implementa a interface gráfica em Swing para configurar, executar e visualizar a simulação.

**Funcionalidades**:

* **Configuração**:
  + Painel para definir parâmetros: número de andares, elevadores, capacidade, tempos de viagem, heurística, tipo de painel e número de pessoas.
  + Validação de parâmetros (ex.: mínimo de 5 andares).
  + Botão para iniciar ou carregar simulação salva.
* **Visualização**:
  + Exibe o prédio com elevadores e pessoas em tempo real.
  + Mostra estatísticas (tempo médio de espera, chamadas atendidas, energia consumida, pessoas transportadas).
  + Exibe lista de pessoas e logs de elevadores selecionados.
* **Controle**:
  + Botões para pausar, continuar, reiniciar ou voltar à configuração.
  + Slider para ajustar a velocidade da simulação.
* **Salvamento/Carregamento**: Permite salvar e carregar estados da simulação.

**Estruturas de Dados**:

* **Lista (**pessoas**)**: Lista encadeada personalizada para armazenar pessoas geradas.

### **10. Lista.java**

**Descrição**: Implementa uma lista encadeada genérica para armazenar elementos de forma dinâmica, usada em várias partes do sistema para gerenciar coleções como elevadores, destinos e logs.

**Funcionalidades**:

* **Inserção**:
  + inserirFim: Adiciona um elemento ao final da lista.
  + inserirInicio: Adiciona um elemento no início da lista (não mostrado explicitamente, mas comum em listas encadeadas).
* **Remoção**:
  + remover: Remove a primeira ocorrência de um elemento específico.
* **Verificações**:
  + contem: Verifica se um elemento está na lista.
  + isVazia: Verifica se a lista está vazia.
  + tamanho: Retorna o número de elementos na lista.
* **Iteração**:
  + getInicio: Retorna um Ponteiro para o início da lista, permitindo iteração pelos elementos.

**Estruturas de Dados**:

* **No**: Classe interna que representa um nó da lista, contendo um elemento e um ponteiro para o próximo nó.
* **Ponteiro**: Objeto que facilita a iteração pelos nós da lista, encapsulando o acesso ao elemento e ao próximo nó.

**Uso no Projeto**:

* Em CentralDeControle para armazenar elevadores (elevadores) e destinos atribuídos (destinosAtribuidos).
* Em Elevador para armazenar destinos (destinos) e logs (logs).
* Em GerenciadorSimulacao para armazenar a lista de pessoas geradas (pessoas).

### **11. LogElevador.java**

**Descrição**: Representa um registro de log para um elevador, armazenando informações sobre eventos como embarque, desembarque ou escolha de destino.

**Funcionalidades**:

* **Inicialização**: Cria um log com o minuto simulado, andar atual, descrição da decisão e ID do elevador.
* **Acesso a Dados**: Fornece getters para acessar os atributos do log (minuto, andar, decisão, ID do elevador).
* **Serialização**: Implementa Serializable para permitir salvamento e carregamento de logs em arquivos.

**Uso no Projeto**:

* Em Elevador, os logs são armazenados em uma Lista<LogElevador> e usados para registrar ações, exibidas na interface gráfica.

### **12. Main.java**

**Descrição**: Classe principal que inicia a aplicação, provavelmente responsável por criar e executar a interface gráfica.

**Funcionalidades**:

* **Ponto de Entrada**: Contém o método main que inicializa o InterfaceGrafica e exibe a janela da aplicação.
* **Configuração Inicial**: Configura o ambiente para a execução da simulação (ex.: look and feel do Swing).

**Nota**: O conteúdo completo de Main.java não foi fornecido, mas sua existência é inferida pelo README.markdown e pelo arquivo workspace.xml, que menciona sua execução.

### **13. NoGenerico.java**

**Descrição**: Interface que define o comportamento de um nó genérico para estruturas de dados como listas e filas.

**Funcionalidades**:

* **Métodos Abstratos**:
  + getElemento: Retorna o elemento armazenado no nó.
  + getProximo: Retorna o próximo nó.
  + setProximo: Define o próximo nó.
* **Uso Genérico**: Permite que classes como Fila e Lista implementem nós de forma consistente.

**Uso no Projeto**:

* Implementada pela classe interna No em Fila.java e presumivelmente em Lista.java para suportar nós encadeados.

### **14. PainelElevador.java**

**Descrição**: Modela o painel de controle de um andar, que pode ser configurado como UNICO\_BOTAO, DOIS\_BOTOES ou PAINEL\_NUMERICO.

**Funcionalidades**:

* **Inicialização**: Define o tipo de painel e inicializa estados (ex.: botões ativados, andares destino).
* **Gestão de Chamadas**:
  + pressionarChamadaGeral: Ativa a chamada geral (para UNICO\_BOTAO).
  + pressionarSubir e pressionarDescer: Ativam botões de subir ou descer (para DOIS\_BOTOES).
  + pressionarAndar: Registra um andar de destino (para PAINEL\_NUMERICO).
* **Verificação de Estado**:
  + isChamadaGeralAtivada, isBotaoSubirAtivado, isBotaoDescerAtivado: Verificam o estado dos botões.
  + getAndaresDestino: Retorna a lista de andares destino (para PAINEL\_NUMERICO).
* **Reset**: Limpa o estado do painel quando não há mais chamadas ou pessoas aguardando.

**Estruturas de Dados**:

* **Lista (andaresDestino)**: Lista encadeada para armazenar andares destino no modo PAINEL\_NUMERICO.

**Uso no Projeto**:

* Em Andar para gerenciar chamadas de elevadores.
* Em CentralDeControle para verificar chamadas válidas ao escolher destinos.

### **15. Pessoa.java**

**Descrição**: Representa uma pessoa na simulação, com atributos como ID, andar de origem, destino, prioridade e minuto de chegada.

**Funcionalidades**:

* **Inicialização**: Cria uma pessoa com ID, andar de origem, destino, prioridade e minuto de chegada.
* **Ações**:
  + entrarElevador: Marca a pessoa como dentro do elevador.
  + sairElevador: Marca a pessoa como fora do elevador e chegada ao destino.
* **Getters**:
  + getId, getAndarDestino, getMinutoChegada, isPrioritaria, isDentroElevador, isChegouAoDestino: Acessam os atributos da pessoa.
* **Serialização**: Implementa Serializable para salvamento em arquivos.

**Estruturas de Dados**:

* Nenhuma, apenas atributos simples (inteiros, booleanos).

**Uso no Projeto**:

* Em GerenciadorSimulacao para gerar pessoas.
* Em Andar e Elevador para gerenciar filas de espera e pessoas transportadas.

### **16. Ponteiro.java**

**Descrição**: Classe que facilita a iteração por listas e filas, apontando para um nó específico.

**Funcionalidades**:

* **Inicialização**: Aponta para um nó inicial.
* **Navegação**:
  + getElemento: Retorna o elemento do nó atual.
  + getProximo: Retorna o próximo ponteiro.
  + isValido: Verifica se o ponteiro aponta para um nó válido.
* **Serialização**: Implementa Serializable para compatibilidade com salvamento.

**Estruturas de Dados**:

* Encapsula um nó (No) de Fila ou Lista.

**Uso no Projeto**:

* Em Andar, CentralDeControle e Elevador para iterar por filas e listas.

### **17. Predio.java**

**Descrição**: Representa o prédio, contendo andares, elevadores (via CentralDeControle) e estatísticas.

**Funcionalidades**:

* **Inicialização**: Cria andares (com painéis) e a central de controle com elevadores.
* **Gestão de Andares**: Fornece acesso à lista de andares.
* **Gestão da Central**: Fornece acesso à CentralDeControle.
* **Estatísticas**: Mantém um objeto Estatisticas para coletar métricas.

**Estruturas de Dados**:

* **Lista (andares)**: Lista encadeada para armazenar objetos Andar.
* **CentralDeControle**: Contém a lista de elevadores.

**Uso no Projeto**:

* Centraliza a estrutura do prédio, conectando andares, elevadores e estatísticas.

### **18. Simulador.java**

**Descrição**: Controla a lógica da simulação, gerenciando o tempo simulado, horários de pico e a execução da simulação.

**Funcionalidades**:

* **Inicialização**: Configura o prédio, pessoas e parâmetros da simulação (ex.: velocidade, tempos de pico).
* **Execução**:
  + iniciar: Inicia a simulação.
  + pausar e continuar: Controlam a execução.
  + atualizar: Avança o tempo simulado, atualizando o prédio e elevadores.
* **Horários de Pico**: Determina se o minuto atual é de pico, ajustando tempos de viagem.
* **Estatísticas**: Fornece acesso ao objeto Estatisticas para registrar métricas.
* **Salvamento/Carregamento**: Suporta salvar e carregar estados da simulação.

**Estruturas de Dados**:

* **Lista (pessoas)**: Lista encadeada para armazenar pessoas a serem adicionadas à simulação.

**Uso no Projeto**:

* Integra a simulação, conectando a interface gráfica com a lógica do prédio e elevadores.

### **19. SimulationState.java**

**Descrição**: Armazena o estado da simulação para salvamento e carregamento.

**Funcionalidades**:

* **Salvamento**: Armazena o estado atual do prédio, elevadores, pessoas e estatísticas.
* **Carregamento**: Restaura o estado salvo para continuar a simulação.
* **Serialização**: Implementa Serializable para persistência em arquivos.

**Uso no Projeto**:

* Usado pela InterfaceGrafica para salvar e carregar simulações (ex.: arquivos .dat).

### **221. TipoPainel.java**

**Descrição**: Enumeração que define os tipos de painéis suportados: UNICO\_BOTAO, DOIS\_BOTOES e PAINEL\_NUMERICO.

**Funcionalidades**:

* **Definição de Tipos**: Fornece constantes para configurar o comportamento dos painéis em Andar e PainelElevador.

**Estruturas de Dados**:

* Nenhuma, apenas uma enumeração.

**Uso no Projeto**:

* Usado em Andar e PainelElevador para determinar a lógica de chamadas.

## **Estruturas de Dados Utilizadas (Resumo Continuado)**

* **Lista (Lista.java)**:
  + **Descrição**: Lista encadeada genérica para coleções dinâmicas.
  + **Uso**: Elevadores, destinos, logs e pessoas.
  + **Implementação**: Nós encadeados com iteração via Ponteiro.
* **Ponteiro (Ponteiro.java)**:
  + **Descrição**: Facilita iteração por listas e filas.
  + **Uso**: Em várias classes para percorrer elementos.
* **No (em Lista.java e Fila.java)**:
  + **Descrição**: Nó genérico para listas e filas.
  + **Uso**: Base para Fila, FilaPrioridade e Lista.

## **Considerações Adicionais**

* **Modularidade**: As classes adicionais mantêm a modularidade, com responsabilidades bem definidas (ex.: PainelElevador para chamadas, LogElevador para registros).
* **Estruturas Personalizadas**: A maioria das classes usa Lista e Fila personalizadas, atendendo ao requisito de evitar coleções padrão do Java.
* **Interface Gráfica**: A integração com InterfaceGrafica é suportada por Simulador e SimulationState, que gerenciam a execução e persistência.
* **Heurísticas e Prioridades**: Continuam sendo suportadas por CentralDeControle e FilaPrioridade, com PainelElevador adaptando-se aos diferentes tipos de chamadas.

## **Conclusão**

As classes após InterfaceGrafica.java complementam o sistema com estruturas de dados robustas (Lista, Ponteiro), lógica de controle (Simulador, Predio, PainelElevador), persistência (SimulationState, LogElevador) e suporte a testes (Teste). Juntas, elas formam um sistema coeso que atende aos requisitos do simulador, oferecendo flexibilidade para diferentes configurações e análises de desempenho.