Gerenciador de Elevadores Concorrentes

ALUNOS:

- Michael Silva
- Carlos Vinícius

- Disciplina: Paradigmas de Programação
- Professor: Sidney Nogueira

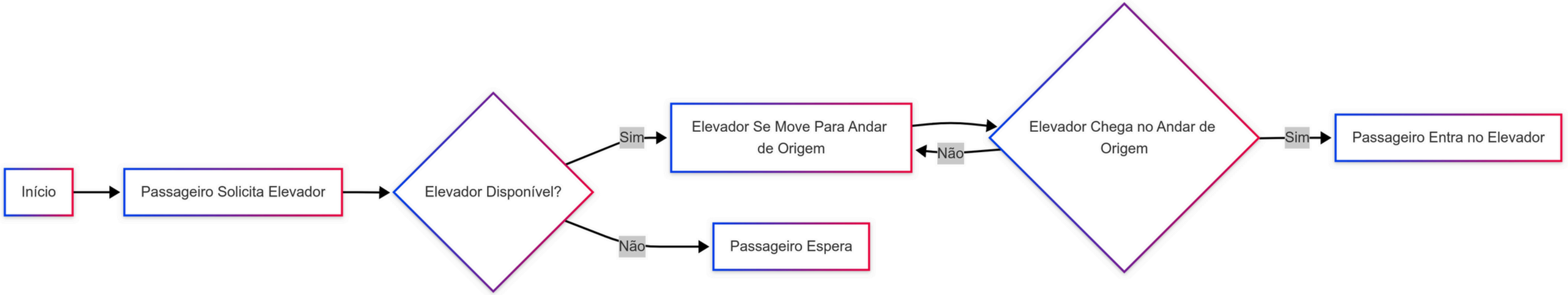
Objetivo do Projeto

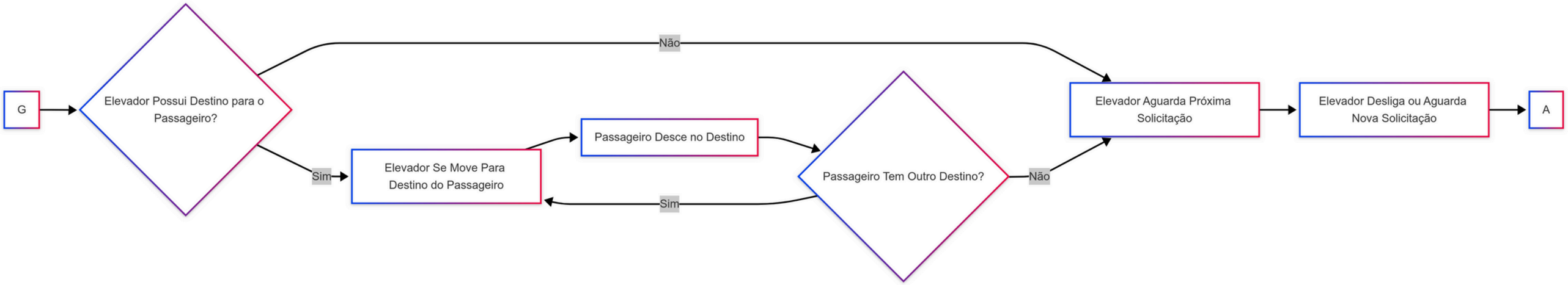
- O objetivo deste projeto é simular o gerenciamento de elevadores em um prédio, onde múltiplos passageiros podem solicitar o elevador e ser atendidos de forma concorrente.

- A implementação deve lidar com a concorrência de maneira eficiente utilizando threads e sincronização.

Funcionamento Geral do Sistema

- O sistema simula elevadores em um prédio, onde passageiros solicitam o elevador de um andar para outro.
  - O elevador se move entre andares e gerencia as solicitações de forma concorrente, atendendo passageiros de maneira justa e eficiente.


Início
Passageiro Solicita Elevador
Elevador Disponível?
Sim
Não
Elevador Se Move Para Andar de Origem
Passageiro Espera
Não
Elevador Chega no Andar de Origem
Sim
Passageiro Entra no Elevador


G
Elevador Possui Destino para o Passageiro?
Não
Sim
Elevador Se Move Para Destino do Passageiro
Passageiro Desce no Destino
Passageiro Tem Outro Destino?
Sim
Não
Elevador Aguarda Próxima Solicitação
Elevador Desliga ou Aguarda Nova Solicitação
A

Concorrência no Sistema

- Múltiplos passageiros podem solicitar o elevador ao mesmo tempo, o que cria uma situação concorrente.
- O elevador deve ser acessado de forma controlada para evitar problemas como race conditions e starvation.
- A sincronização é feita utilizando ReentrantLock para garantir que apenas um passageiro entre ou saia do elevador por vez.

Classes e Funcionalidades Principais

Classe **Elevador:**

- Gerencia o movimento do elevador e o embarque/desembarque de passageiros.
- Controla o movimento de andar para andar com o método moverElevador()
- Sincroniza o acesso aos recursos compartilhados com ReentrantLock

Classe **Passageiro**:

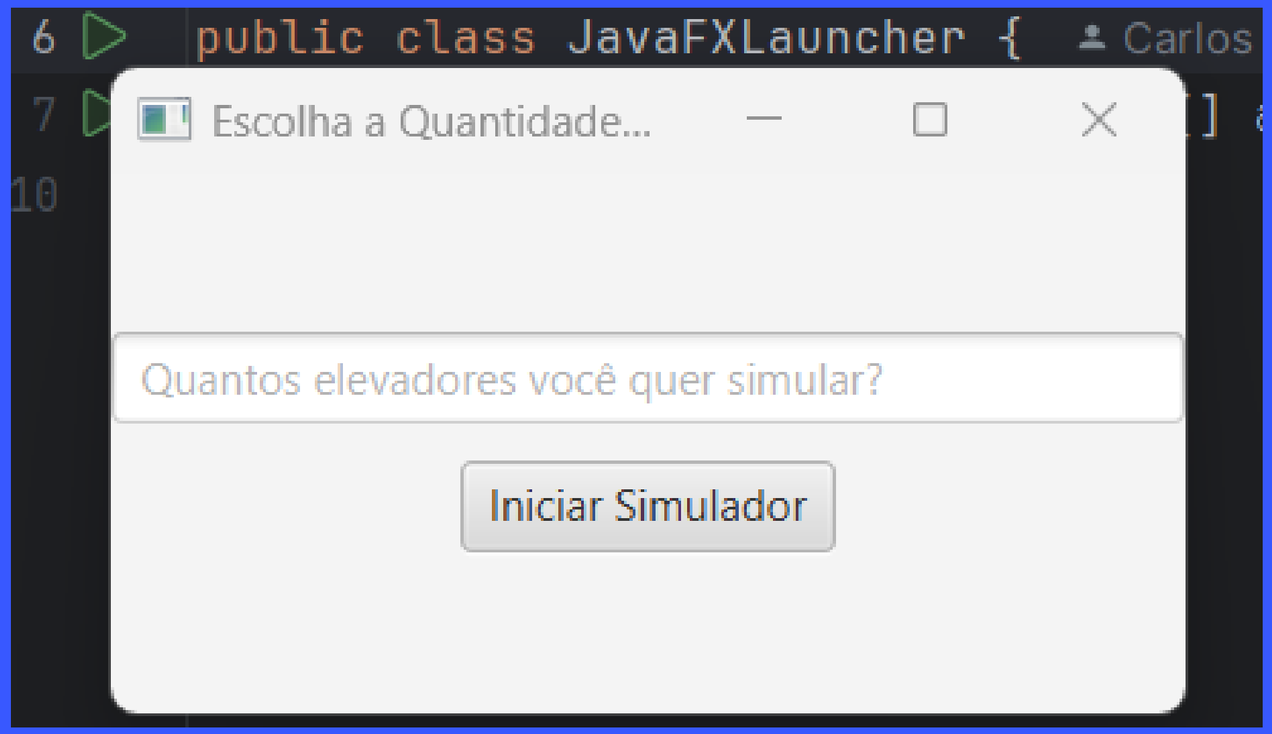
- Representa os passageiros, incluindo o andar de origem e destino.
- Possui métodos para obter e definir o nome e os andares de origem e destino

Controle de Concorrência com ReentrantLock

- Classe Elevador utiliza o método lock da classe ReentrantLock para garantir que apenas um passageiro entre ou saia do elevador ao mesmo tempo.

```
public void solicitarElevador(Passageiro passageiro) {
    lock.lock();
    try {
        passageirosEsperando.add(passageiro);
    } finally {
        lock.unlock();
    }
}
```

Demonstração do Sistema



Propriedades de Concorrência

- Liveness (Viva):
- O que é? Garante que o sistema nunca "trave" e que todas as tarefas sejam eventualmente realizadas.
- Como funciona?: O elevador sempre atende aos passageiros, não importa quantos ou quão rápido eles entrem ou saiam.

- Starvation (Fome):
- O que é? Quando uma tarefa nunca tem chance de ser executada.
- Como evitamos?: O sistema é projetado para garantir que todos os passageiros sejam atendidos, sem deixar ninguém esperando por tempo indefinido.

Divisão de Tarefas

Michael Silva:

- Implementação da classe Elevador, sincronização, controle de concorrência.

Carlos Silva:

- Implementação da classe Passageiro, testes de funcionalidade e documentação.

Desafios e Soluções

- Desafio: Sincronizar o acesso aos recursos compartilhados sem causar deadlock ou race conditions.
- Solução: Utilização de ReentrantLock para garantir o controle exclusivo de acesso a listas de passageiros.
- Desafio: Garantir que a movimentação do elevador fosse feita de forma eficiente e justa.
- Solução: Implementação do controle de destino do elevador, considerando as prioridades dos passageiros.

Testes Realizados

- Testes com diferentes números de elevadores e passageiros para verificar o comportamento concorrente.
- Testes realizados com até 10 passageiros e 3 elevadores para garantir que o sistema não gerasse problemas de sincronização.
- Resultados: O sistema manteve a performance e não houve deadlocks.

Conclusão e Futuras Melhorias

- Conclusão: O projeto simula o gerenciamento de elevadores de forma eficiente e concorrente, com controle adequado das threads.

- Futuras Melhorias:
  - Adicionar mais elevadores ou tipos de recursos compartilhados.
  - Implementar um algoritmo de priorização para passageiros com urgência.

Obrigado!