Simulador de Tráfego em Malha Viária

Desenvolvimento de Sistemas Paralelos e Distribuídos

Professor: Fernando dos Santos
Alunas: Manoella Marques Felippe e Maria Eduarda Jonck

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA

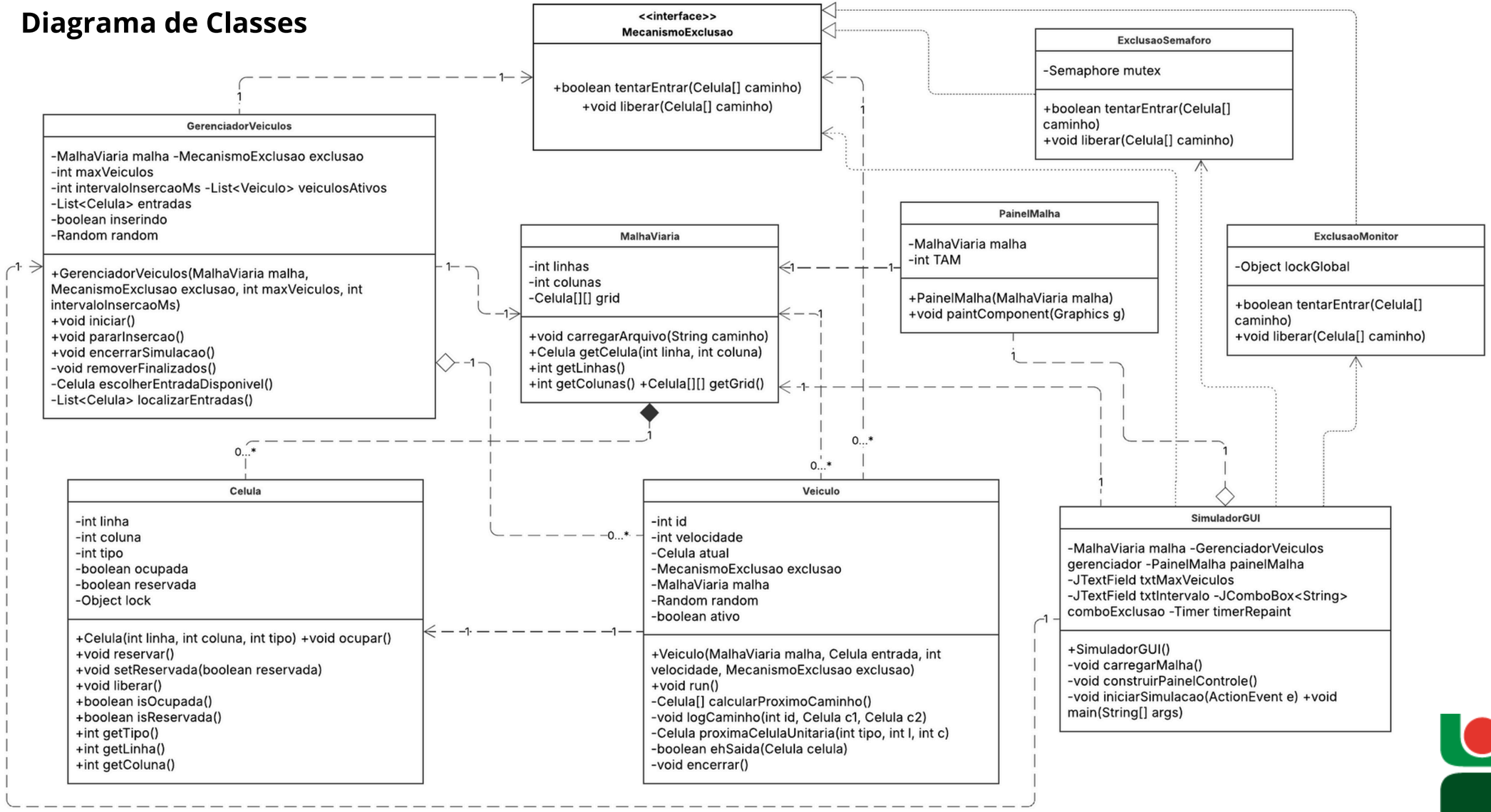
Arquitetura do sistema

Diagrama de Classes

- MalhaViaria: carrega e armazena a malha
- Celula: representa cada ponto da malha (tipo, ocupada, reservada)
- Veiculo: Thread que se move pela malha
- GerenciadorVeiculos: insere, controla e encerra veículos
- MecanismoExclusao: interface com implementações SemafaroExclusao e MonitorExclusao
- PainelMalha: renderiza graficamente a malha

Arquitetura do sistema

Diagrama de Classes



UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Padrões e Boas Práticas

Boas práticas

- Encapsulamento das regras de movimentação em **Veiculo:**
  - As regras para movimentação, como decidir o próximo caminho, lidar com cruzamentos, ocupar e liberar células, estão todas concentradas dentro da classe **Veiculo**.
- Interface para abstração da exclusão mútua:
  - A interface **MecanismoExclusao** define um contrato genérico para controlar o acesso exclusivo a células compartilhadas (ex: cruzamentos).
- Atualização visual feita com Swing thread-safe:
  - A visualização da malha (as "bolinhas" vermelhas) é feita pela classe PainelMalha com Swing. O repaint() é chamado para atualizar a tela.
  - **Thread-safe:**
    - Como a simulação roda com múltiplas threads (Veiculo), e o Swing não é thread-safe, qualquer alteração visual precisa ser feita no Event Dispatch Thread (EDT).
- Uso de **CopyOnWriteArrayList** para evitar **ConcurrentModificationException**:
  - Ao obter uma lista de veículos, onde a mesma pode ser modificada enquanto é percorrida por outra thread, ocorre o erro: **ConcurrentModificationException.** Neste caso, ao usar CopyOnWriteArrayList<*Veiculo*> —gera uma lista segura para acesso concorrente que cria uma cópia ao modificar.

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Padrões e Boas Práticas

Padrão de projeto: Strategy
Linguagem: Java

- O padrão **Strategy** permite definir uma família de algoritmos (neste caso, mecanismos de exclusão).
- **MecanismoExclusao**
  - É a interface comum que define as operações tentarEntrar() e liberar().
- **SemaforoExclusao e MonitorExclusao**
  - São as estratégias concretas, ou seja, implementações diferentes da mesma lógica de controle de exclusão mútua.
- **Veiculo e GerenciadorVeiculos**
  - Utilizam a interface (MecanismoExclusao), sem precisar saber se estão usando um monitor ou semáforo.

Dificuldades e Soluções

Dificuldades enfrentadas

- **Colisões**: ao chegar nos cruzamentos, os veículos colidiam um com outro, causando problemas na interface, geralmente ocorria quando um veículo estava mais rápido que o outro.

- **Oscilações**: em alguns momentos, nos cruzamentos, os veículos apresentavam oscilações na interface, isso se dava quando um veículo calculava o próximo caminho, porém não o ocupava.

- **Sem validação de caminho:** não verificava se o tipo da célula era de rua (diferente de 0) ou e o próximo caminho realmente existia.

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Dificuldades e Soluções

Soluções aplicadas

- **Colisões:** resolvido com verificação antes de ocupar, criando um controle de reserva da célula antes do veículo ocupar.
- **Oscilações:** resolvido evitando que a interface seja redesenhada o tempo todo enquanto o veículo está apenas aguardando a liberação da célula que estava ocupada.
- **Sem validação de caminho:** resolvido verificando se as células existem e são válidas para movimentação, usando aleatoriedade apenas se todas as rotas forem possíveis, se não forem válidas, escolhe a rota disponível.

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Ferramentas para Exclusão

Exclusão Semáforo:
A classe **ExclusaoSemaforo** usa um Semaphore para garantir que apenas um veículo de cada vez possa reservar e ocupar todas as células do cruzamento.

Funcionamento:
- mutex.acquire() → pede permissão.
- Verifica se todas as células do caminho estão livres.
- Reserva todas.
- mutex.release() → libera quando termina.

Exclusão Monitor:
A classe **ExclusaoMonitor** usa um bloco synchronized com um objeto de lock global (lockGlobal) para garantir exclusão mútua.

Funcionamento:
- Entra num bloco synchronized(lockGlobal).
- Verifica se as células do caminho estão livres.
- Se sim, reserva.
- Quando libera, também entra no bloco sincronizado.

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA