Simulador de Tráfego em Malha Viária

Bruno Trindade e Silva
Renan Ricardo Holler

Veículos

Threads

Cada veículo é uma thread independente, executando em paralelo e interagindo com o ambiente.

Fluxo e Exclusão Mútua

Os veículos respeitam o sentido do fluxo, evitando colisões com mecanismos de exclusão mútua.

Velocidades Variadas

Cada veículo possui uma velocidade aleatória, contribuindo para um comportamento mais realista.

Malha Viária

Carregamento

Malha lida a partir de arquivo de texto e desenhada usando o Java Swing.

Entradas e Saídas

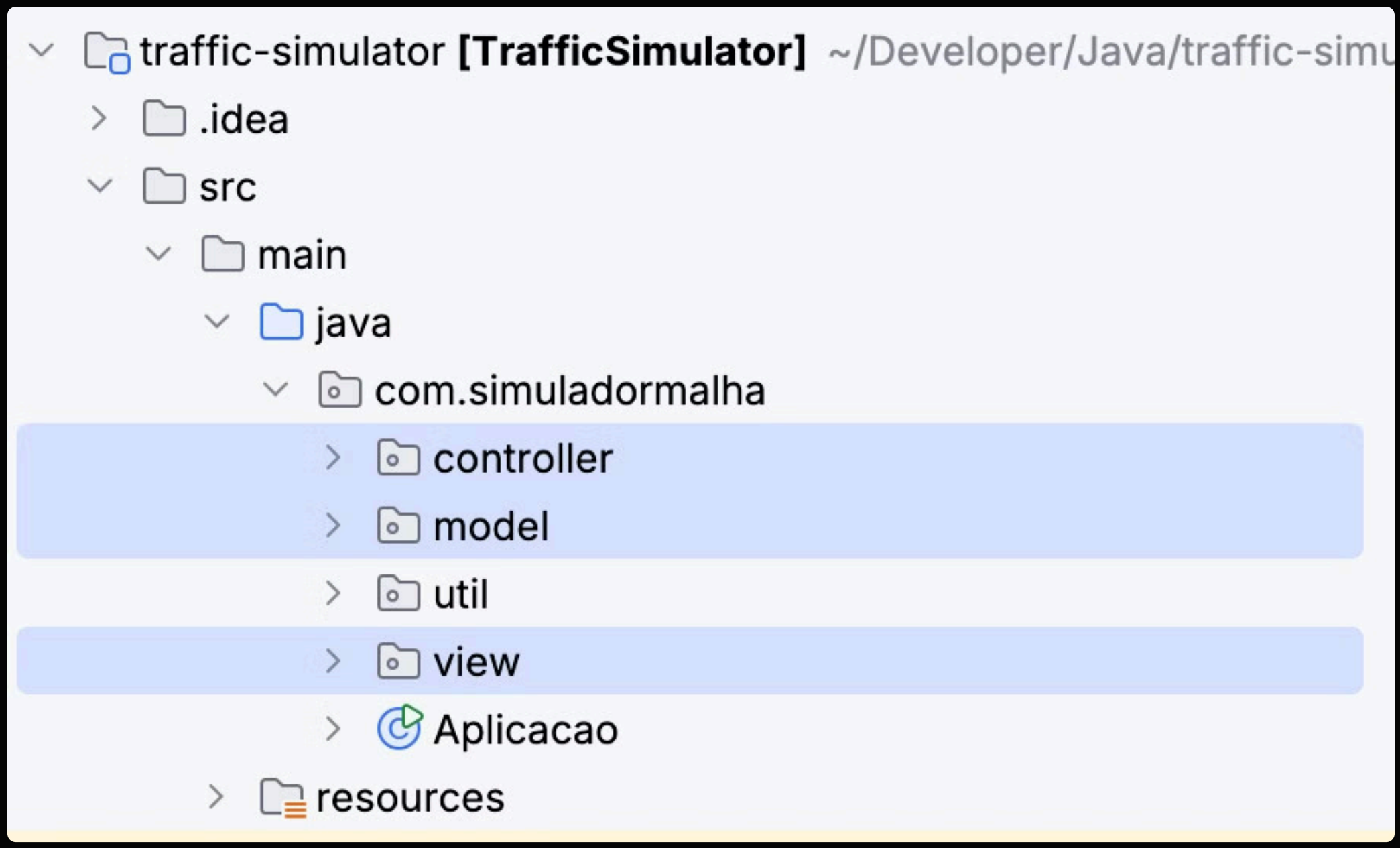
Definidas nas bordas da malha e visíveis para validações.

Disposição das Pistas

Pistas horizontais ou verticais, com duas faixas e sempre separadas.

MVC

Model	View	Controller
As classes MalhaViaria, Veiculo, Celula representam a parte **Model**. Elas lidam com a lógica do domínio, como a malha viária, a lógica de movimentação dos veículos, e as interações entre células.	A interface gráfica (InterfaceGrafica, PainelMalha) cuida da visualização do estado da malha viária e interações com o usuário. Essa parte exibe a malha e atualiza a interface enquanto os veículos se movem.	A classe ControllerSimulacao é responsável por controlar a simulação. Ela gerencia a lógica entre o **Model** (veículos, malha, etc.) e a **View** (interface gráfica). Define as estratégias e orquestra a simulação.



Factory

Centralização	Abstração	Randomização controlada
A VeiculoFactory concentra a lógica de criação de veículos. Isso facilita a manutenção, já que qualquer mudança no processo de criação afeta apenas a fábrica, sem alterar o resto do sistema.	Esconde os detalhes de como os veículos são inicializados (ex.: entrada e velocidade). O código que usa veículos não precisa saber como eles são criados, só precisa pedir para a fábrica.	A fábrica também centraliza a lógica de aleatoriedade (entrada e velocidade dos veículos), tornando o comportamento mais previsível e o código mais fácil de testar e modificar.

```java
package com.simuladormalha.util.factory;

import ...

public class VeiculoFactory {  3 usages  Bruno +1
    private MalhaViaria malha;  3 usages
    private ExclusaoMutuaStrategy exclusaoMutua;  3 usages
    private Random random;  3 usages

    public VeiculoFactory(MalhaViaria malha, ExclusaoMutuaStrategy exclusaoMutua) {  1 usage  Bruno
        this.malha = malha;
        this.exclusaoMutua = exclusaoMutua;
        this.random = new Random();
    }

    public Veiculo criarVeiculo() {  1 usage  Renan Holler +1
        List<Celula> entradas = malha.getPontosEntrada();
        Celula entrada = entradas.get(random.nextInt(entradas.size()));
        int velocidade = 200 + random.nextInt( bound: 100);

        return new Veiculo(malha, entrada.getLinha(), entrada.getColuna(), velocidade, exclusaoMutua);
    }

    public void setExclusaoMutua(ExclusaoMutuaStrategy exclusaoMutua) { this.exclusaoMutua = exclusaoMutua; }
}
```

Strategy

Centralização	Abstração	Flexibilidade
A interface ExclusaoMutuaStrategy define métodos para controlar a exclusão mútua. Isso permite centralizar a lógica de controle de acesso às células da malha, mantendo o código flexível.	O Strategy permite trocar a forma como o controle de acesso é feito. As classes MonitorStrategy e SemaforoStrategy implementam a mesma interface, mas usam abordagens diferentes (sincronização e semáforos, respectivamente).	A escolha da estratégia é dinâmica. Isso permite ajustar como os veículos reservam as células da malha sem alterar o código principal da simulação.

Interface

```
public interface ExclusaoMutuaStrategy {  15 usages  2
    boolean tentarReservar(List<Celula> caminho);
    void liberarCaminho(List<Celula> caminho);  4 us
    boolean isCaminhoLivre(List<Celula> celula);  1
}
```

Semáforo

```
public class SemaforoStrategy implements ExclusaoMutuaStrategy {  2 usages  Bruno
    private Semaphore[][] semaforos;  5 usages

    public SemaforoStrategy(MalhaViaria malha) {  1 usage  Bruno
        int linhas = malha.getLinhas();
        int colunas = malha.getColunas();
        semaforos = new Semaphore[linhas][colunas];

        for (int i = 0; i < linhas; i++) {
            for (int j = 0; j < colunas; j++) {
                semaforos[i][j] = new Semaphore( permits: 1);
            }
        }
    }

    @Override  1 usage  Bruno
    public synchronized boolean isCaminhoLivre(List<Celula> caminho) {
        boolean livre = true;
        for (Celula celula : caminho) {
            if (celula.estaReservada() && semaforos[celula.getLinha()][celula.getColuna()].availablePermits() == 0) {
                livre = false;
            }
        }
        return livre;
    }

    @Override  2 usages  Bruno
    public synchronized boolean tentarReservar(List<Celula> caminho) {
        for (Celula celula : caminho) {
            try {
                if (!semaforos[celula.getLinha()][celula.getColuna()].tryAcquire()) {
                    liberarCaminho(caminho);
                    return false;
                }
            } catch (Exception e) {
                liberarCaminho(caminho);
                return false;
            }
        }
        return true;
    }

    @Override  4 usages  Bruno
    public synchronized void liberarCaminho(List<Celula> caminho) {
        for (Celula celula : caminho) {
            semaforos[celula.getLinha()][celula.getColuna()].release();
        }
    }
}
```

Monitor

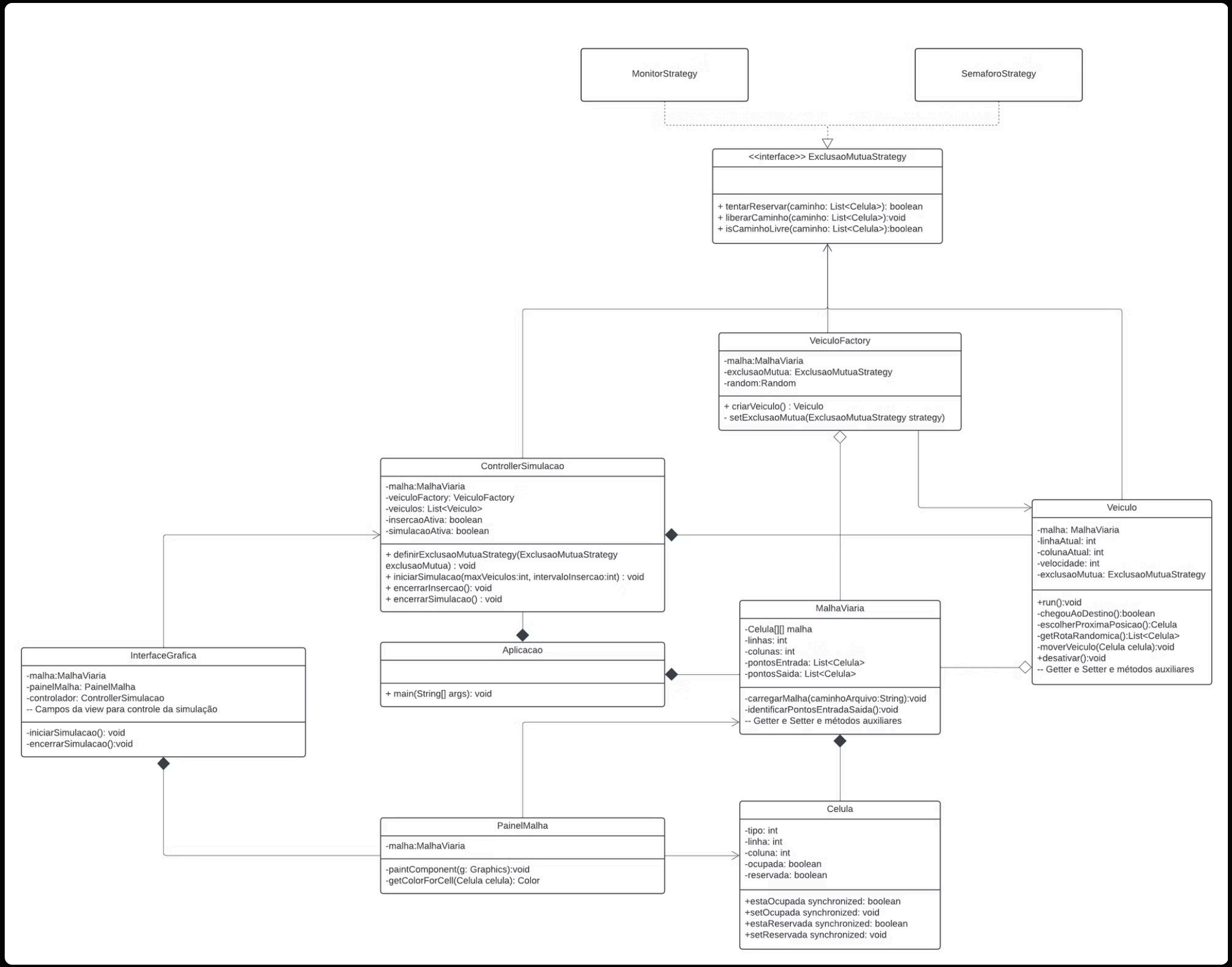
```
public class MonitorStrategy implements ExclusaoMutuaStrategy {  2 usages

    @Override  2 usages  Bruno
    public synchronized boolean tentarReservar(List<Celula> caminho) {
        for (Celula celula : caminho) {
            if (celula.estaOcupada() || celula.estaReservada()) {
                liberarCaminho(caminho);
                return false;
            }
        }
        for (Celula celula : caminho) {
            celula.setReservada(true);
        }
        return true;
    }

    @Override  4 usages  Bruno
    public synchronized void liberarCaminho(List<Celula> caminho) {
        for (Celula celula : caminho) {
            celula.setReservada(false);
        }
    }

    @Override  1 usage  Bruno
    public boolean isCaminhoLivre(List<Celula> caminho) {
        boolean livre = true;
        for (Celula celula : caminho) {
            if (celula.estaReservada()) {
                livre = false;
            }
        }
        return livre;
    }
}
```

Diagrama de Classes



Demonstração do Sistema