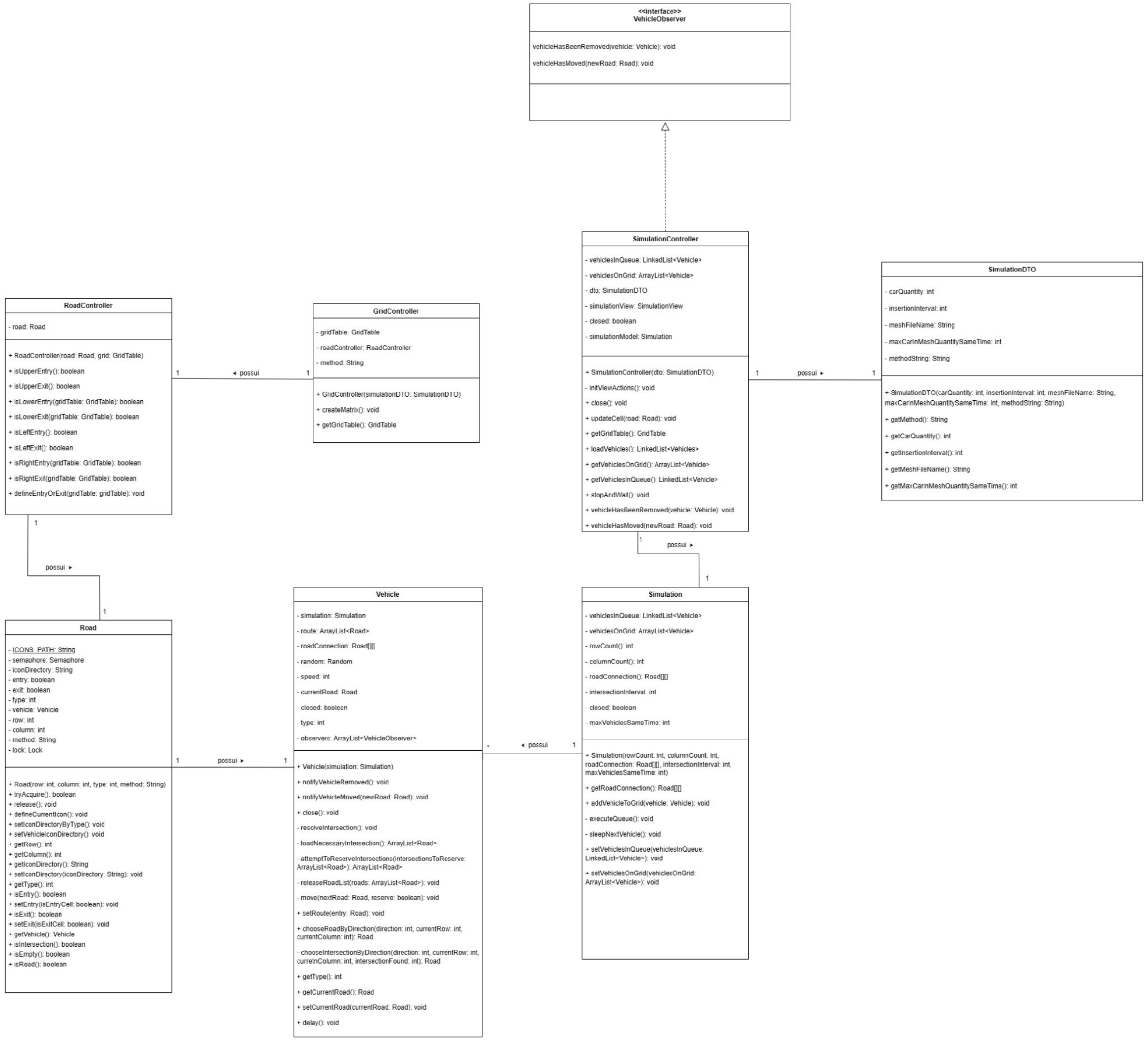
Threads - 65DSD

Davi e Cauê

Diagrama de Classes



<<interface>>
VehicleObserver
vehicleHasBeenRemoved(vehicle: Vehicle): void
vehicleHasMoved(newRoad: Road): void
SimulationController
- vehiclesInQueue: LinkedList<Vehicle>
- vehiclesOnGrid: ArrayList<Vehicle>
- dto: SimulationDTO
- simulationView: SimulationView
- closed: boolean
- simulationModel: Simulation
+ SimulationController(dto: SimulationDTO)
- initViewActions(): void
+ close(): void
+ updateCell(road: Road): void
+ getGridTable(): GridTable
+ loadVehicles(): LinkedList<Vehicles>
+ getVehiclesOnGrid(): ArrayList<Vehicle>
+ getVehiclesInQueue(): LinkedList<Vehicle>
+ stopAndWait(): void
+ vehicleHasBeenRemoved(vehicle: Vehicle): void
+ vehicleHasMoved(newRoad: Road): void
1
possui ▸
1
SimulationDTO
- carQuantity: int
- insertionInterval: int
- meshFileName: String
- maxCarInMeshQuantitySameTime: int
- methodString: String
+ SimulationDTO(carQuantity: int, insertionInterval: int, meshFileName: String, maxCarInMeshQuantitySameTime: int, methodString: String)
+ getMethod(): String
+ getCarQuantity(): int
+ getInsertionInterval(): int
+ getMeshFileName(): String
+ getMaxCarInMeshQuantitySameTime(): int
RoadController
- road: Road
+ RoadController(road: Road, grid: GridTable)
+ isUpperEntry(): boolean
+ isUpperExit(): boolean
+ isLowerEntry(gridTable: GridTable): boolean
+ isLowerExit(gridTable: GridTable): boolean
+ isLeftEntry(): boolean
+ isLeftExit(): boolean
+ isRightEntry(gridTable: GridTable): boolean
+ isRightExit(gridTable: GridTable): boolean
+ defineEntryOrExit(gridTable: gridTable): void
1
◂ possui
1
GridController
- gridTable: GridTable
- roadController: RoadController
- method: String
+ GridController(simulationDTO: SimulationDTO)
+ createMatrix(): void
+ getGridTable(): GridTable
1
possui ▸
1
1
possui ▸
1
Road
- ICONS_PATH: String
- semaphore: Semaphore
- iconDirectory: String
- entry: boolean
- exit: boolean
- type: int
- vehicle: Vehicle
- row: int
- column: int
- method: String
- lock: Lock
+ Road(row: int, column: int, type: int, method: String)
+ tryAcquire(): boolean
+ release(): void
+ defineCurrentIcon(): void
+ setIconDirectoryByType(): void
+ setVehicleIconDirectory(): void
+ getRow(): int
+ getColumn(): int
+ getIconDirectory(): String
+ setIconDirectory(iconDirectory: String): void
+ getType(): int
+ isEntry(): boolean
+ setEntry(isEntryCell: boolean): void
+ isExit(): boolean
+ setExit(isExitCell: boolean): void
+ getVehicle(): Vehicle
+ isIntersection(): boolean
+ isEmpty(): boolean
+ isRoad(): boolean
1
possui ▸
1
Vehicle
- simulation: Simulation
- route: ArrayList<Road>
- roadConnection: Road[][]
- random: Random
- speed: int
- currentRoad: Road
- closed: boolean
- type: int
- observers: ArrayList<VehicleObserver>
+ Vehicle(simulation: Simulation)
+ notifyVehicleRemoved(): void
+ notifyVehicleMoved(newRoad: Road): void
+ close(): void
- resolveIntersection(): void
- loadNecessaryIntersection(): ArrayList<Road>
- attemptToReserveIntersections(intersectionsToReserve: ArrayList<Road>): ArrayList<Road>
- releaseRoadList(roads: ArrayList<Road>): void
- move(nextRoad: Road, reserve: boolean): void
+ setRoute(entry: Road): void
+ chooseRoadByDirection(direction: int, currentRow: int, currentColumn: int): Road
- chooseIntersectionByDirection(direction: int, currentRow: int, curretnColumn: int, intersectionFound: int): Road
+ getType(): int
+ getCurrentRoad(): Road
+ setCurrentRoad(currentRoad: Road): void
+ delay(): void
*
◂ possui
1
Simulation
- vehiclesInQueue: LinkedList<Vehicle>
- vehiclesOnGrid: ArrayList<Vehicle>
- rowCount(): int
- columnCount(): int
- roadConnection(): Road[][]
- intersectionInterval: int
- closed: boolean
- maxVehiclesSameTime: int
+ Simulation(rowCount: int, columnCount: int, roadConnection: Road[][], intersectionInterval: int, maxVehiclesSameTime: int)
+ getRoadConnection(): Road[][]
+ addVehicleToGrid(vehicle: Vehicle): void
- executeQueue(): void
- sleepNextVehicle(): void
+ setVehiclesInQueue(vehiclesInQueue: LinkedList<Vehicle>): void
+ setVehiclesOnGrid(vehiclesOnGrid: ArrayList<Vehicle>): void

D

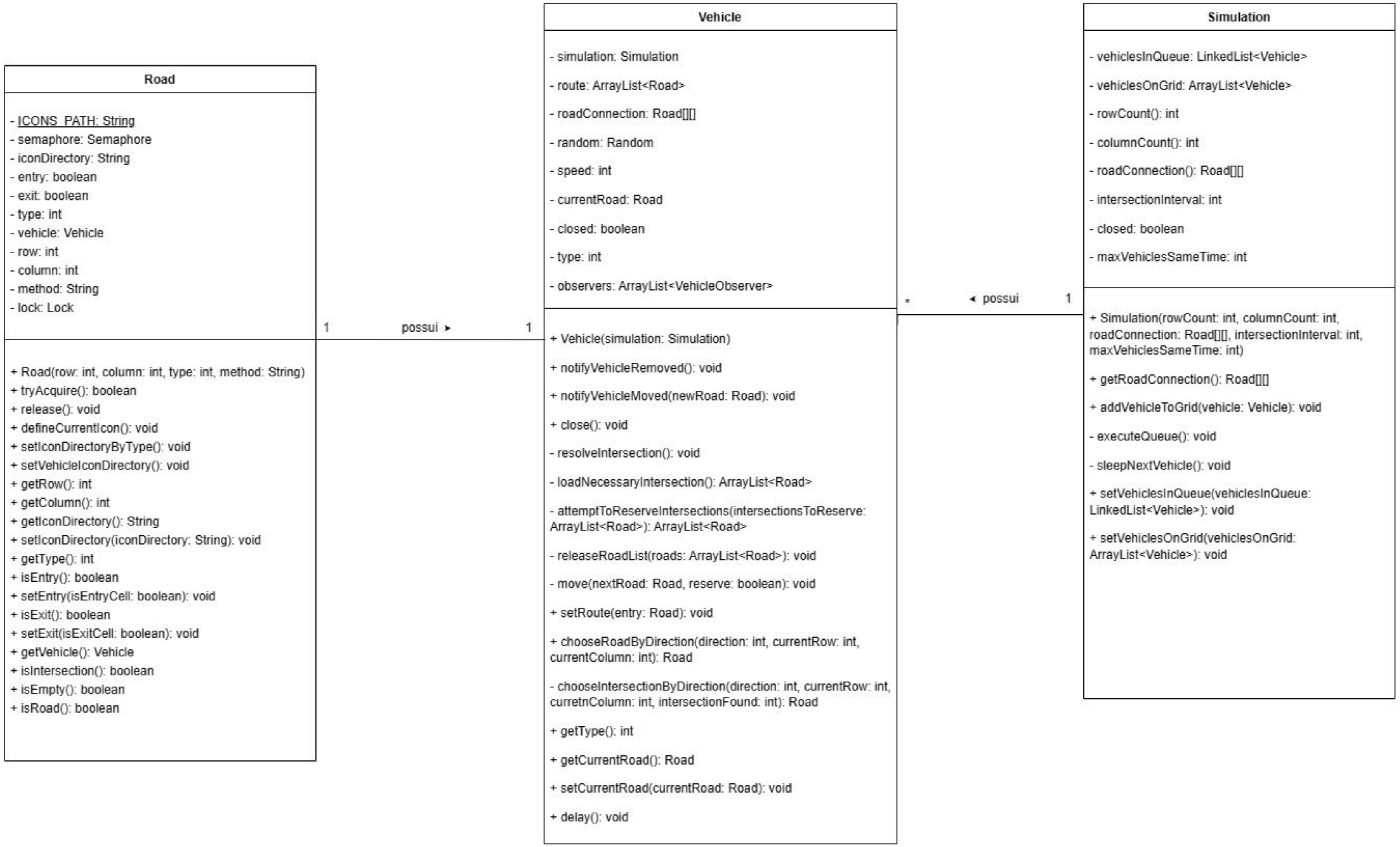
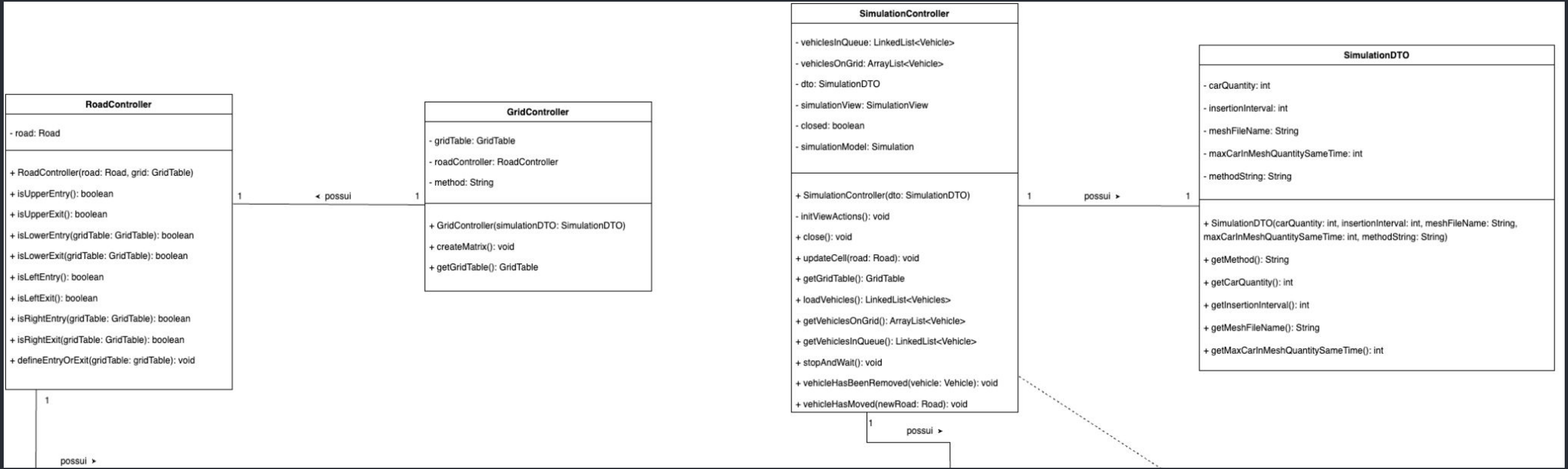
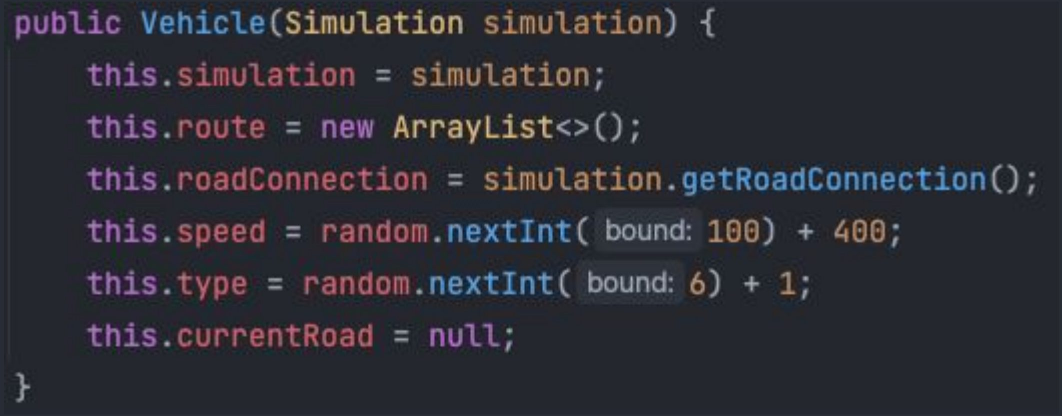

Road
- ICONS_PATH: String
- semaphore: Semaphore
- iconDirectory: String
- entry: boolean
- exit: boolean
- type: int
- vehicle: Vehicle
- row: int
- column: int
- method: String
- lock: Lock
+ Road(row: int, column: int, type: int, method: String)
+ tryAcquire(): boolean
+ release(): void
+ defineCurrentIcon(): void
+ setIconDirectoryByType(): void
+ setVehicleIconDirectory(): void
+ getRow(): int
+ getColumn(): int
+ getIconDirectory(): String
+ setIconDirectory(iconDirectory: String): void
+ getType(): int
+ isEntry(): boolean
+ setEntry(isEntryCell: boolean): void
+ isExit(): boolean
+ setExit(isExitCell: boolean): void
+ getVehicle(): Vehicle
+ isIntersection(): boolean
+ isEmpty(): boolean
+ isRoad(): boolean
1
possui ➤
1
Vehicle
- simulation: Simulation
- route: ArrayList<Road>
- roadConnection: Road[][]
- random: Random
- speed: int
- currentRoad: Road
- closed: boolean
- type: int
- observers: ArrayList<VehicleObserver>
+ Vehicle(simulation: Simulation)
+ notifyVehicleRemoved(): void
+ notifyVehicleMoved(newRoad: Road): void
+ close(): void
- resolveIntersection(): void
- loadNecessaryIntersection(): ArrayList<Road>
- attemptToReserveIntersections(intersectionsToReserve: ArrayList<Road>): ArrayList<Road>
- releaseRoadList(roads: ArrayList<Road>): void
- move(nextRoad: Road, reserve: boolean): void
+ setRoute(entry: Road): void
+ chooseRoadByDirection(direction: int, currentRow: int, currentColumn: int): Road
- chooseIntersectionByDirection(direction: int, currentRow: int, curretnColumn: int, intersectionFound: int): Road
+ getType(): int
+ getCurrentRoad(): Road
+ setCurrentRoad(currentRoad: Road): void
+ delay(): void
*
◄ possui
1
Simulation
- vehiclesInQueue: LinkedList<Vehicle>
- vehiclesOnGrid: ArrayList<Vehicle>
- rowCount(): int
- columnCount(): int
- roadConnection(): Road[][]
- intersectionInterval: int
- closed: boolean
- maxVehiclesSameTime: int
+ Simulation(rowCount: int, columnCount: int, roadConnection: Road[][], intersectionInterval: int, maxVehiclesSameTime: int)
+ getRoadConnection(): Road[][]
+ addVehicleToGrid(vehicle: Vehicle): void
- executeQueue(): void
- sleepNextVehicle(): void
+ setVehiclesInQueue(vehiclesInQueue: LinkedList<Vehicle>): void
+ setVehiclesOnGrid(vehiclesOnGrid: ArrayList<Vehicle>): void

Diagrama de Classes:


RoadController
- road: Road
+ RoadController(road: Road, grid: GridTable)
+ isUpperEntry(): boolean
+ isUpperExit(): boolean
+ isLowerEntry(gridTable: GridTable): boolean
+ isLowerExit(gridTable: GridTable): boolean
+ isLeftEntry(): boolean
+ isLeftExit(): boolean
+ isRightEntry(gridTable: GridTable): boolean
+ isRightExit(gridTable: GridTable): boolean
+ defineEntryOrExit(gridTable: gridTable): void
1
◂ possui
1
GridController
- gridTable: GridTable
- roadController: RoadController
- method: String
+ GridController(simulationDTO: SimulationDTO)
+ createMatrix(): void
+ getGridTable(): GridTable
SimulationController
- vehiclesInQueue: LinkedList<Vehicle>
- vehiclesOnGrid: ArrayList<Vehicle>
- dto: SimulationDTO
- simulationView: SimulationView
- closed: boolean
- simulationModel: Simulation
+ SimulationController(dto: SimulationDTO)
- initViewActions(): void
+ close(): void
+ updateCell(road: Road): void
+ getGridTable(): GridTable
+ loadVehicles(): LinkedList<Vehicles>
+ getVehiclesOnGrid(): ArrayList<Vehicle>
+ getVehiclesInQueue(): LinkedList<Vehicle>
+ stopAndWait(): void
+ vehicleHasBeenRemoved(vehicle: Vehicle): void
+ vehicleHasMoved(newRoad: Road): void
1
possui ▸
1
SimulationDTO
- carQuantity: int
- insertionInterval: int
- meshFileName: String
- maxCarInMeshQuantitySameTime: int
- methodString: String
+ SimulationDTO(carQuantity: int, insertionInterval: int, meshFileName: String, maxCarInMeshQuantitySameTime: int, methodString: String)
+ getMethod(): String
+ getCarQuantity(): int
+ getInsertionInterval(): int
+ getMeshFileName(): String
+ getMaxCarInMeshQuantitySameTime(): int
1
possui ▸
1
possui ▸

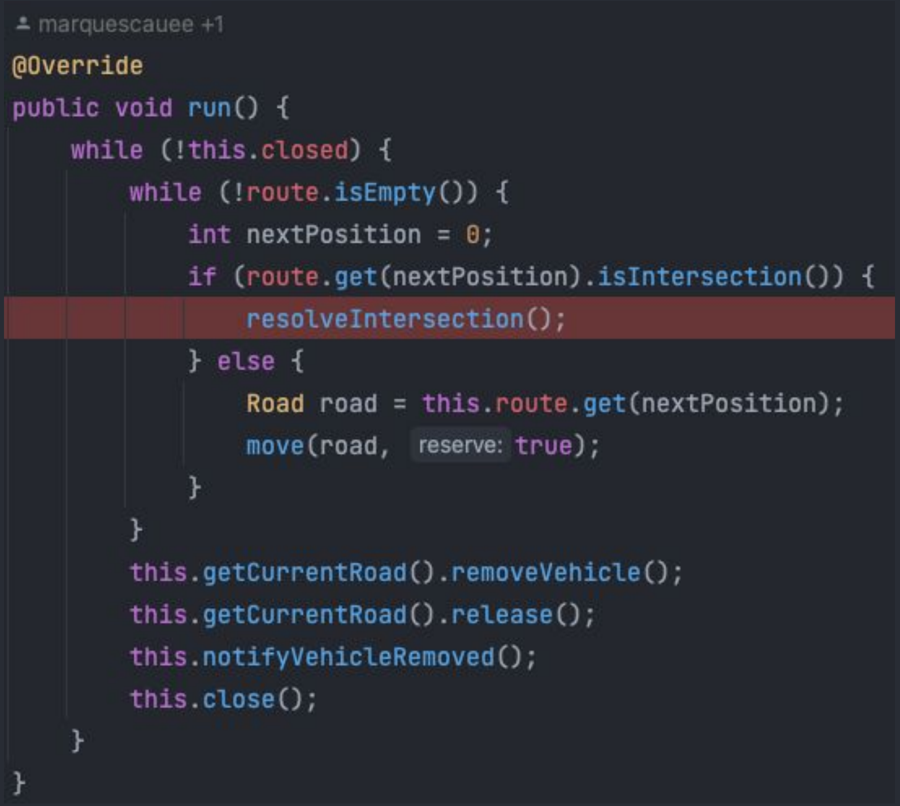
Model de Veículo

Veículos - Construtor:



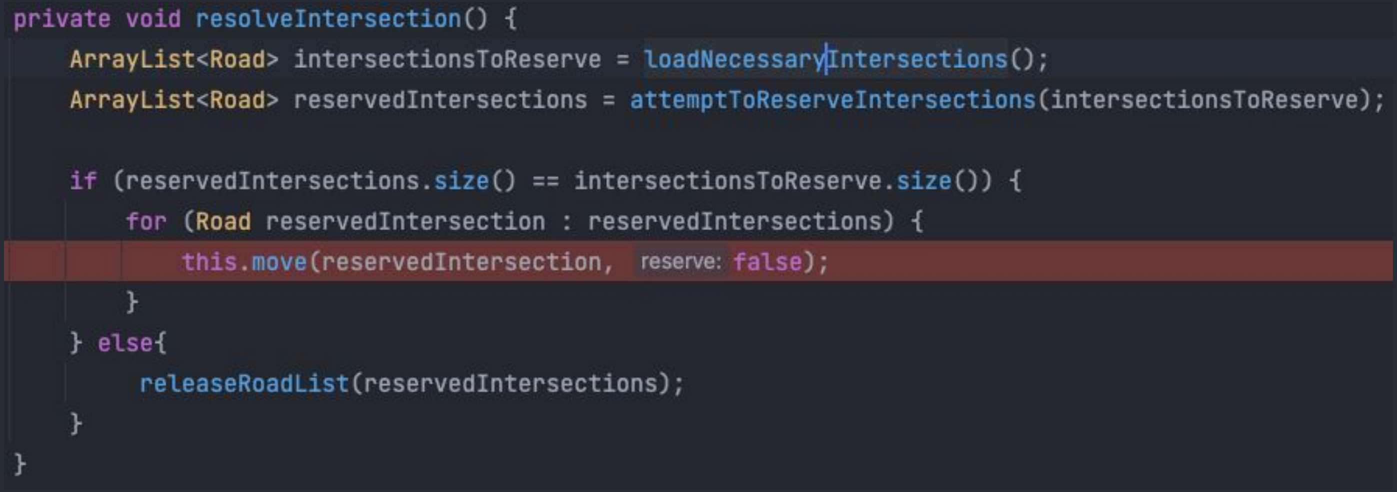
```
public Vehicle(Simulation simulation) {
    this.simulation = simulation;
    this.route = new ArrayList<>();
    this.roadConnection = simulation.getRoadConnection();
    this.speed = random.nextInt( bound: 100) + 400;
    this.type = random.nextInt( bound: 6) + 1;
    this.currentRoad = null;
}
```

Veículos - Run:


marquescauee +1

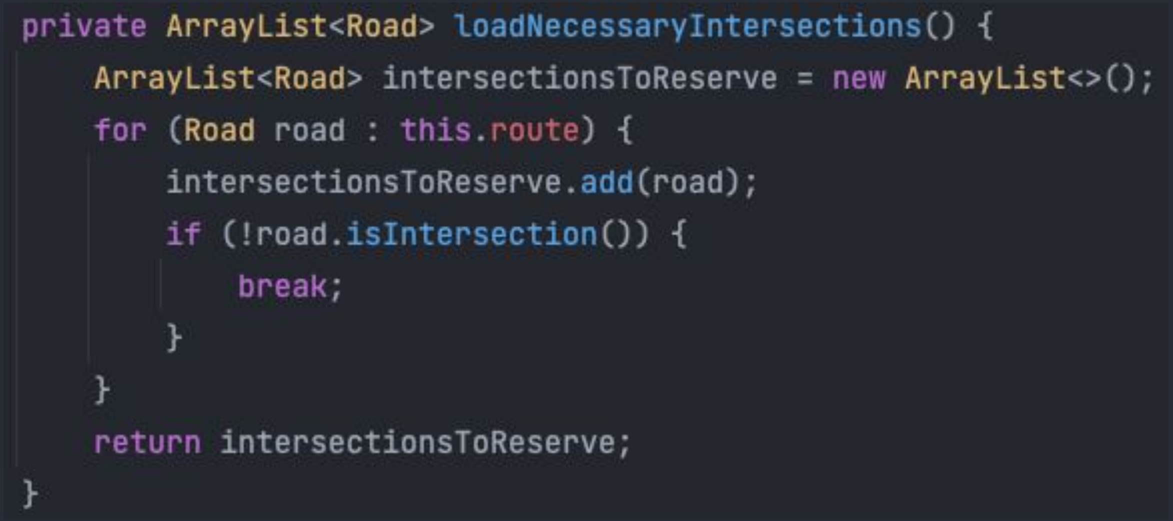
```
@Override
public void run() {
    while (!this.closed) {
        while (!route.isEmpty()) {
            int nextPosition = 0;
            if (route.get(nextPosition).isIntersection()) {
                resolveIntersection();
            } else {
                Road road = this.route.get(nextPosition);
                move(road,  reserve: true);
            }
        }
        this.getCurrentRoad().removeVehicle();
        this.getCurrentRoad().release();
        this.notifyVehicleRemoved();
        this.close();
    }
}
```

Veículos - Resolvendo Interseções:



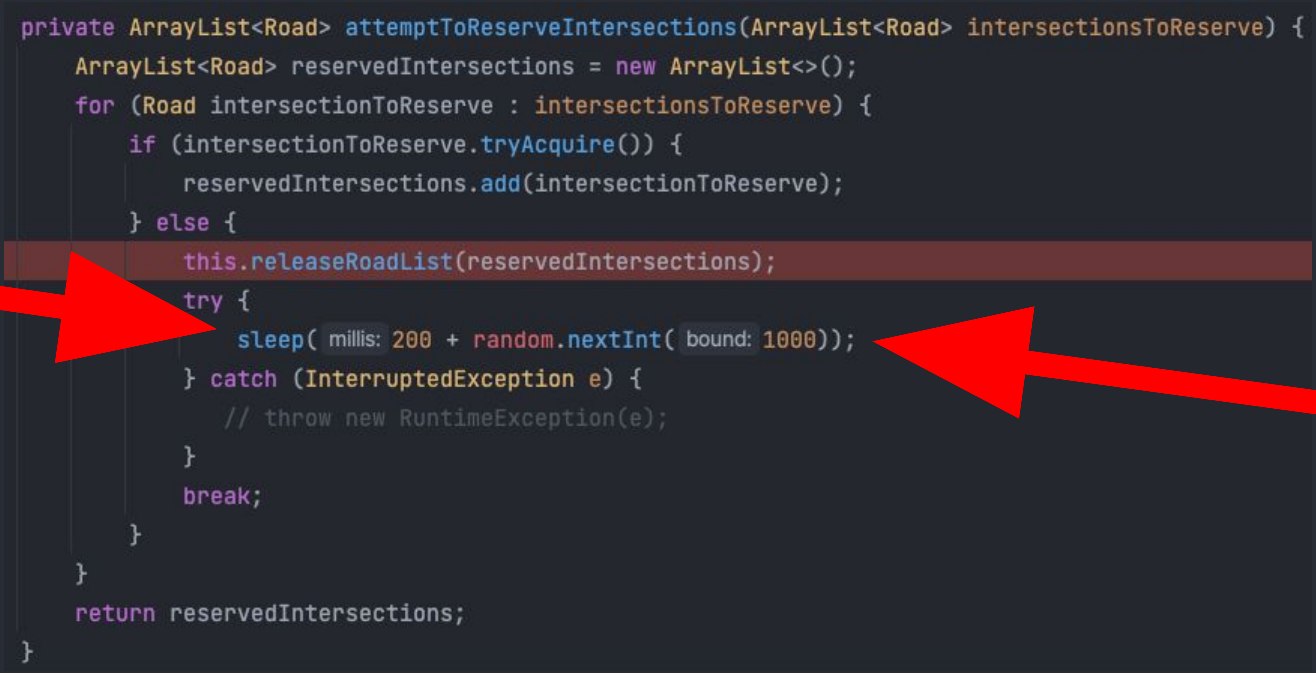
```java
private void resolveIntersection() {
    ArrayList<Road> intersectionsToReserve = loadNecessaryIntersections();
    ArrayList<Road> reservedIntersections = attemptToReserveIntersections(intersectionsToReserve);

    if (reservedIntersections.size() == intersectionsToReserve.size()) {
        for (Road reservedIntersection : reservedIntersections) {
            this.move(reservedIntersection,  reserve: false);
        }
    } else{
         releaseRoadList(reservedIntersections);
    }
}
```

Veículos - Resolvendo Interseções:



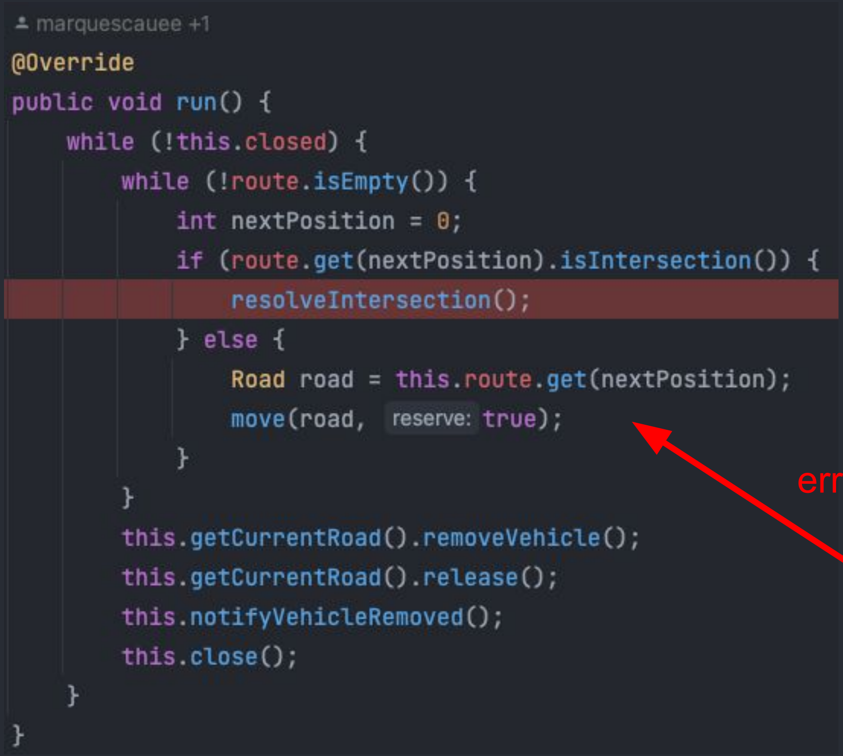
```java
private ArrayList<Road> loadNecessaryIntersections() {
    ArrayList<Road> intersectionsToReserve = new ArrayList<>();
    for (Road road : this.route) {
        intersectionsToReserve.add(road);
        if (!road.isIntersection()) {
            break;
        }
    }
    return intersectionsToReserve;
}
```

Veículos - Resolvendo Interseções:



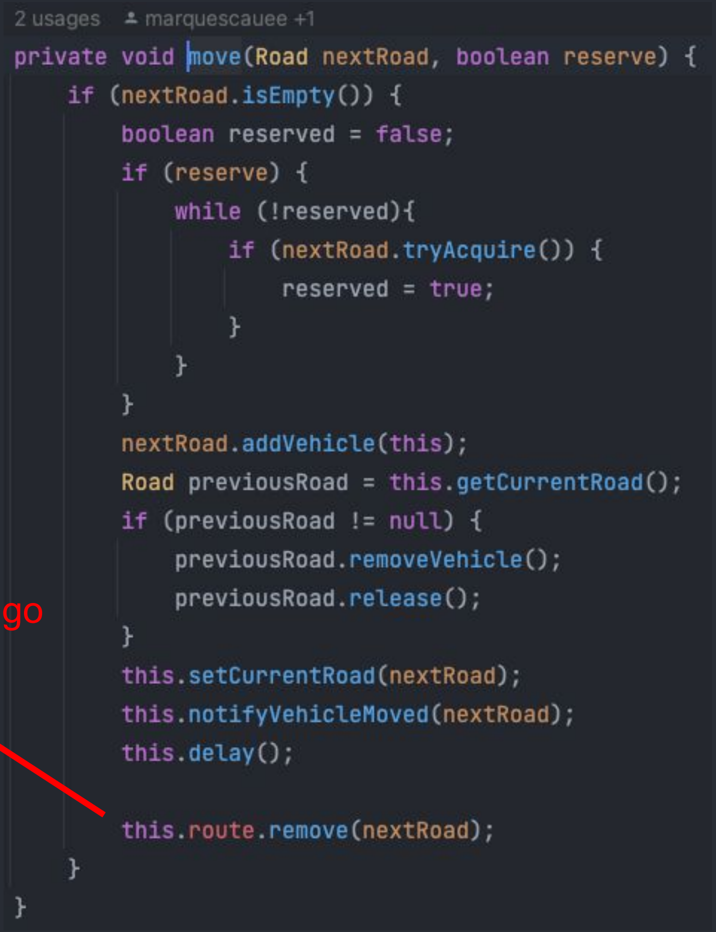
```java
private ArrayList<Road> attemptToReserveIntersections(ArrayList<Road> intersectionsToReserve) {
    ArrayList<Road> reservedIntersections = new ArrayList<>();
    for (Road intersectionToReserve : intersectionsToReserve) {
        if (intersectionToReserve.tryAcquire()) {
            reservedIntersections.add(intersectionToReserve);
        } else {
            this.releaseRoadList(reservedIntersections);
            try {
                sleep( millis: 200 + random.nextInt( bound: 1000));
            } catch (InterruptedException e) {
              // throw new RuntimeException(e);
            }
            break;
        }
    }
    return reservedIntersections;
}
```

Veículos - Move



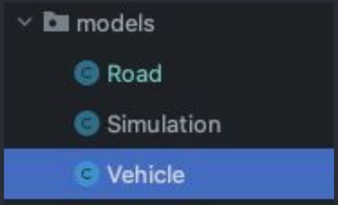
```
marquescauee +1
@Override
public void run() {
    while (!this.closed) {
        while (!route.isEmpty()) {
            int nextPosition = 0;
            if (route.get(nextPosition).isIntersection()) {
                resolveIntersection();
            } else {
                Road road = this.route.get(nextPosition);
                move(road, reserve: true);
            }
        }
        this.getCurrentRoad().removeVehicle();
        this.getCurrentRoad().release();
        this.notifyVehicleRemoved();
        this.close();
    }
}
```

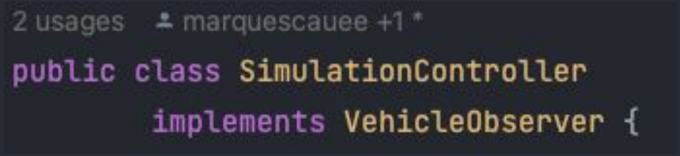
erro antigo



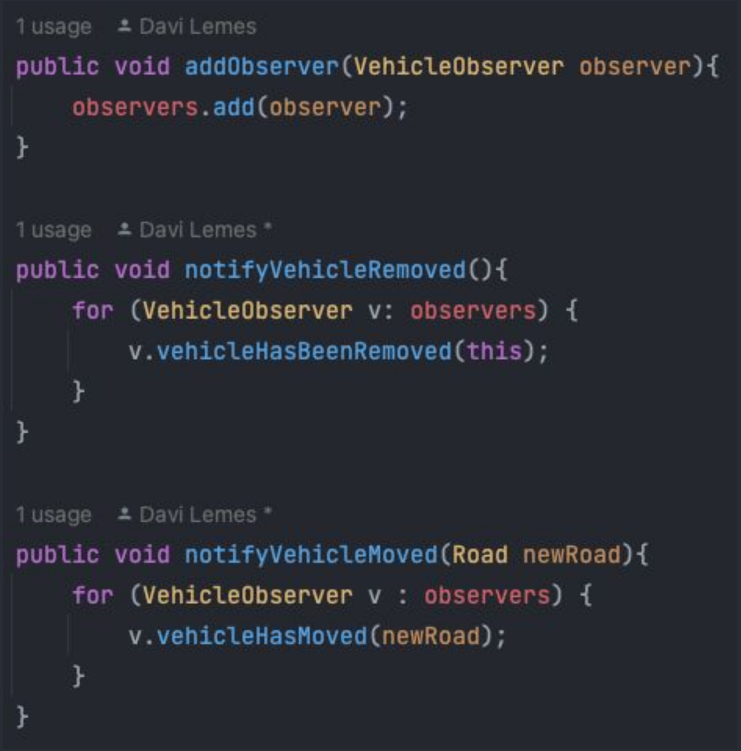
```
2 usages   marquescauee +1
private void move(Road nextRoad, boolean reserve) {
    if (nextRoad.isEmpty()) {
        boolean reserved = false;
        if (reserve) {
            while (!reserved){
                if (nextRoad.tryAcquire()) {
                    reserved = true;
                }
            }
        }
        nextRoad.addVehicle(this);
        Road previousRoad = this.getCurrentRoad();
        if (previousRoad != null) {
            previousRoad.removeVehicle();
            previousRoad.release();
        }
        this.setCurrentRoad(nextRoad);
        this.notifyVehicleMoved(nextRoad);
        this.delay();

        this.route.remove(nextRoad);
    }
}
```

Veículos - Observer:




2 usages   marquescauee +1 *

```java
public class SimulationController
        implements VehicleObserver {
```


1 usage   Davi Lemes

```java
public void addObserver(VehicleObserver observer){
    observers.add(observer);
}
```

1 usage   Davi Lemes *

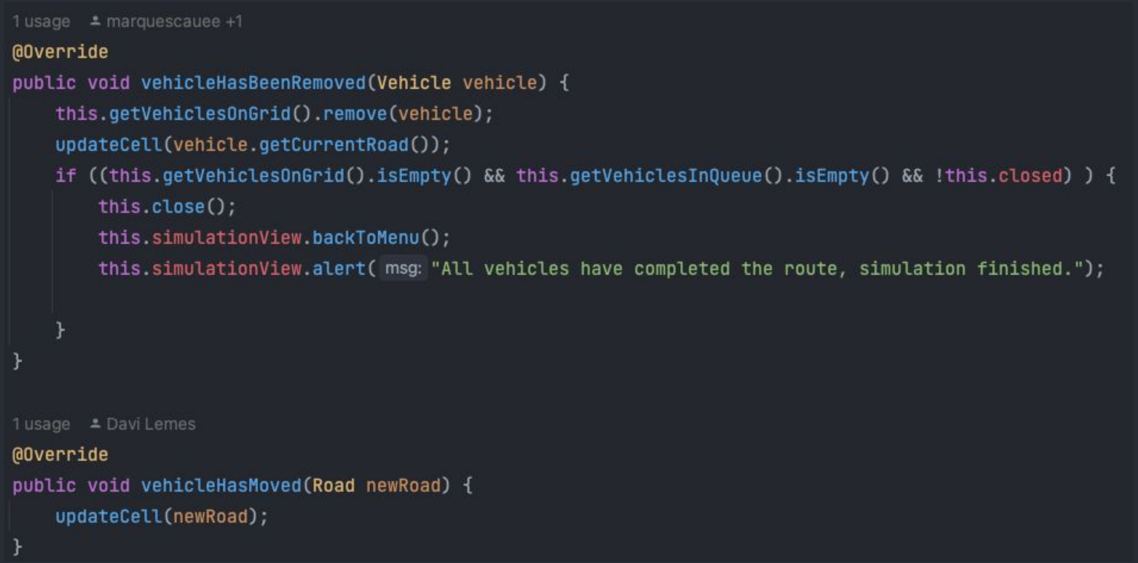
```java
public void notifyVehicleRemoved(){
    for (VehicleObserver v: observers) {
        v.vehicleHasBeenRemoved(this);
    }
}
```

1 usage   Davi Lemes *

```java
public void notifyVehicleMoved(Road newRoad){
    for (VehicleObserver v : observers) {
        v.vehicleHasMoved(newRoad);
    }
}
```

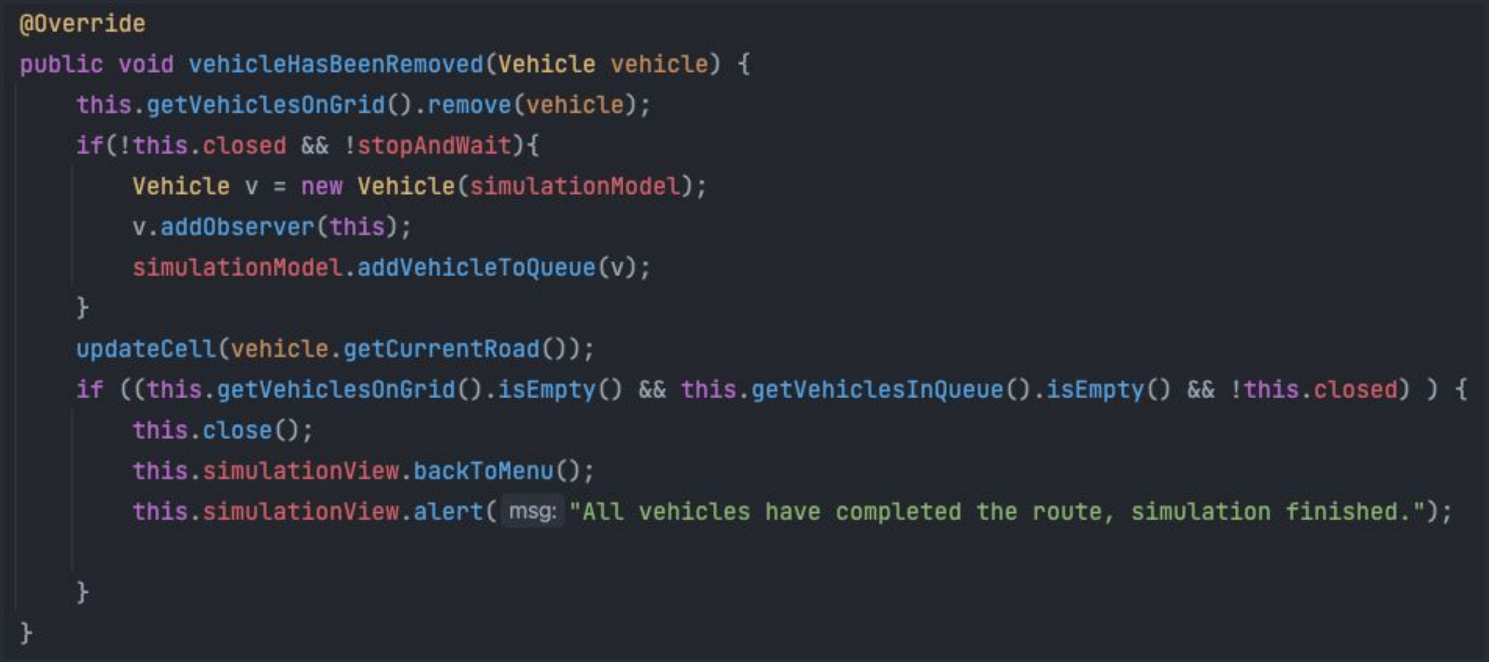
Simulation Controller

Simulation Controller Observer - Primeira Versão:



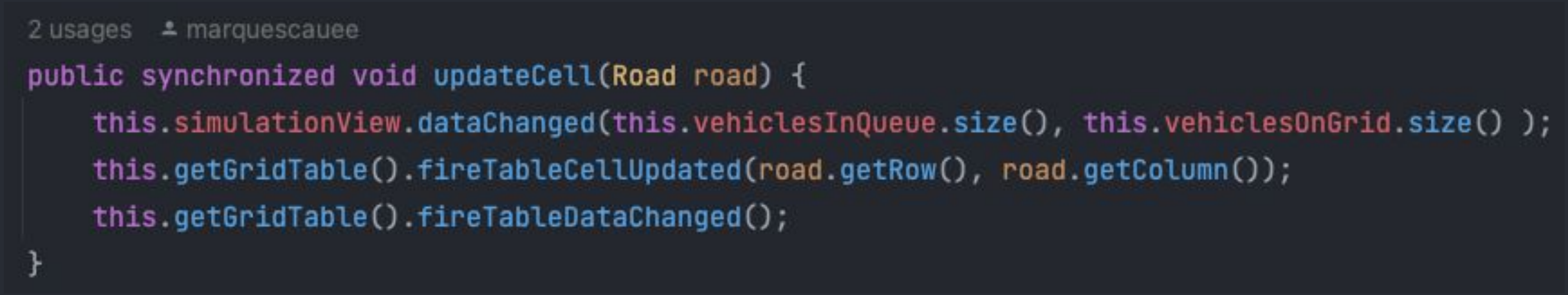
```
1 usage  marquescauee +1
@Override
public void vehicleHasBeenRemoved(Vehicle vehicle) {
    this.getVehiclesOnGrid().remove(vehicle);
    updateCell(vehicle.getCurrentRoad());
    if ((this.getVehiclesOnGrid().isEmpty() && this.getVehiclesInQueue().isEmpty() && !this.closed) ) {
        this.close();
        this.simulationView.backToMenu();
        this.simulationView.alert( msg: "All vehicles have completed the route, simulation finished.");

    }
}

1 usage  Davi Lemes
@Override
public void vehicleHasMoved(Road newRoad) {
    updateCell(newRoad);
}
```

Simulation Controller Observer - Versão Final:



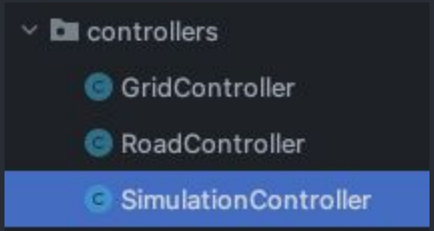
```java
@Override
public void vehicleHasBeenRemoved(Vehicle vehicle) {
    this.getVehiclesOnGrid().remove(vehicle);
    if(!this.closed && !stopAndWait){
        Vehicle v = new Vehicle(simulationModel);
        v.addObserver(this);
        simulationModel.addVehicleToQueue(v);
    }
    updateCell(vehicle.getCurrentRoad());
    if ((this.getVehiclesOnGrid().isEmpty() && this.getVehiclesInQueue().isEmpty() && !this.closed) ) {
        this.close();
        this.simulationView.backToMenu();
        this.simulationView.alert( msg: "All vehicles have completed the route, simulation finished.");

    }
}
```

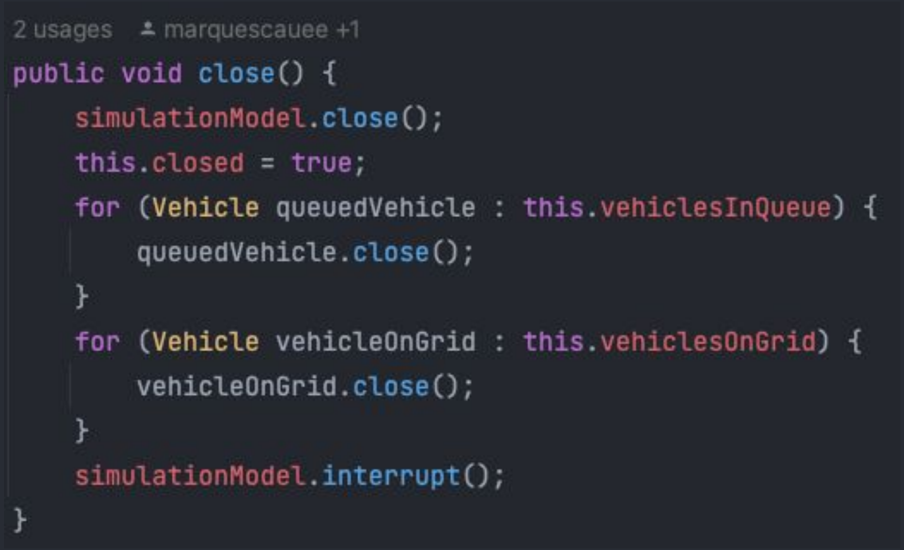
Simulation Controller - Comunicação com a View:


2 usages marquescauee

```
public synchronized void updateCell(Road road) {
    this.simulationView.dataChanged(this.vehiclesInQueue.size(), this.vehiclesOnGrid.size() );
    this.getGridTable().fireTableCellUpdated(road.getRow(), road.getColumn());
    this.getGridTable().fireTableDataChanged();
}
```

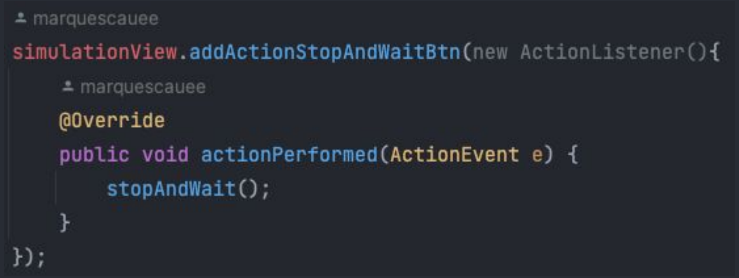
Simulation Controller - Close:

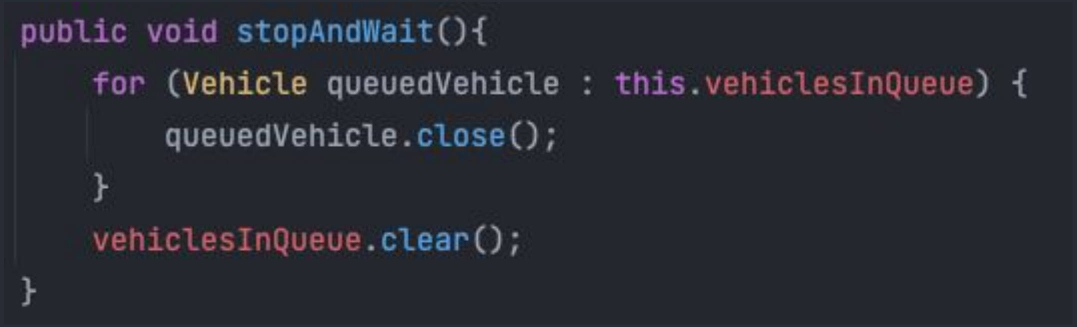




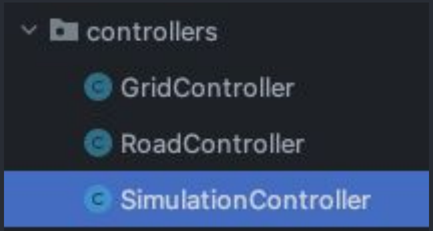
```
public void close() {
    simulationModel.close();
    this.closed = true;
    for (Vehicle queuedVehicle : this.vehiclesInQueue) {
        queuedVehicle.close();
    }
    for (Vehicle vehicleOnGrid : this.vehiclesOnGrid) {
        vehicleOnGrid.close();
    }
    simulationModel.interrupt();
}
```

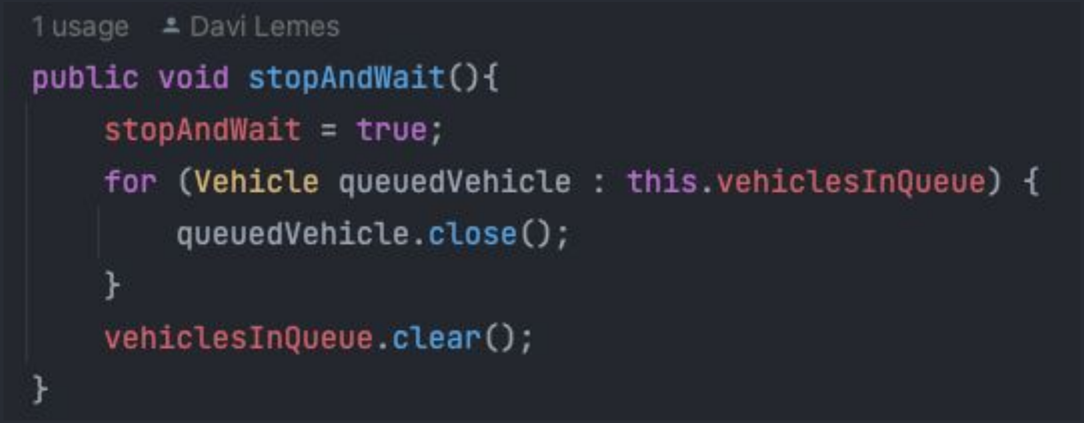
Simulation Controller - Stop and Wait - Primeira Versão:


```
marquescauee
simulationView.addActionStopAndWaitBtn(new ActionListener(){
    marquescauee
    @Override
    public void actionPerformed(ActionEvent e) {
        stopAndWait();
    }
});
```


```
public void stopAndWait(){
    for (Vehicle queuedVehicle : this.vehiclesInQueue) {
        queuedVehicle.close();
    }
    vehiclesInQueue.clear();
}
```

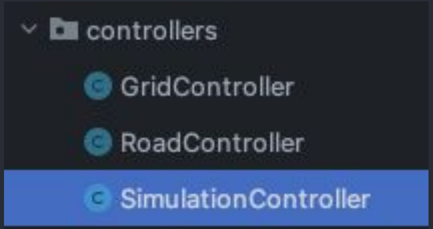
Simulation Controller - Stop and Wait - Versão Final:

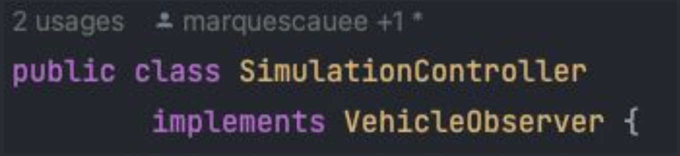

controllers
GridController
RoadController
SimulationController


1 usage   Davi Lemes

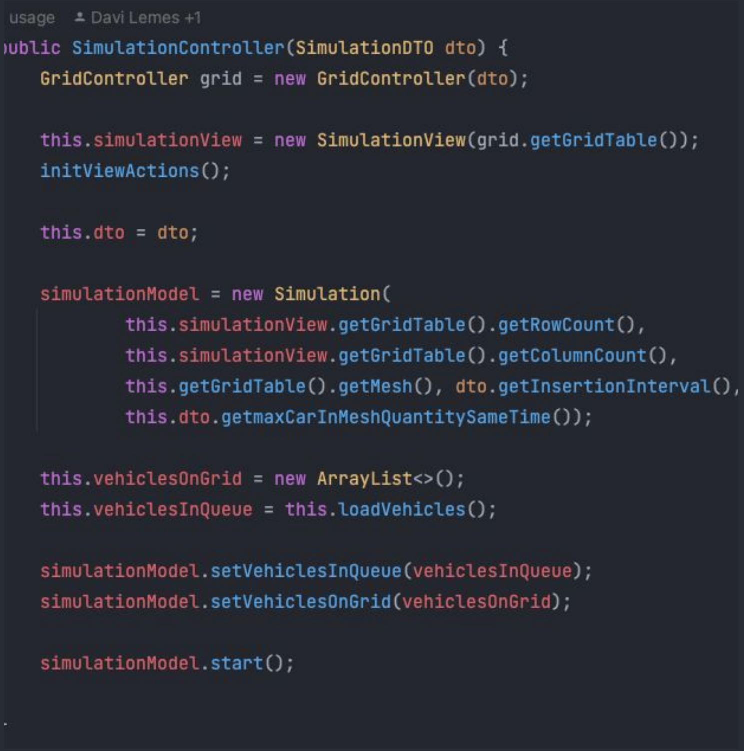
```
public void stopAndWait(){
    stopAndWait = true;
    for (Vehicle queuedVehicle : this.vehiclesInQueue) {
        queuedVehicle.close();
    }
    vehiclesInQueue.clear();
}
```

Simulation Controller:


controllers
GridController
RoadController
SimulationController

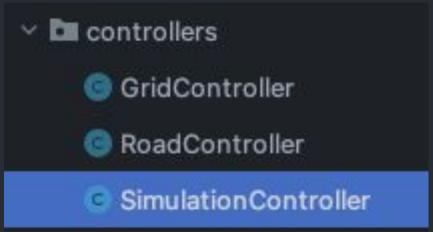

2 usages · marquescauee +1 *

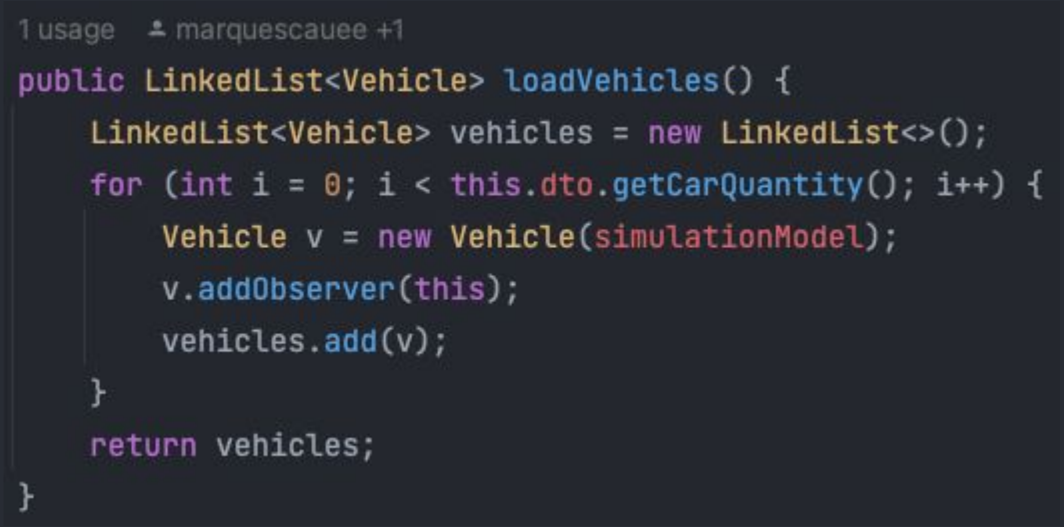
```
public class SimulationController
        implements VehicleObserver {
```


usage · Davi Lemes +1

```
ublic SimulationController(SimulationDTO dto) {
    GridController grid = new GridController(dto);

    this.simulationView = new SimulationView(grid.getGridTable());
    initViewActions();

    this.dto = dto;

    simulationModel = new Simulation(
            this.simulationView.getGridTable().getRowCount(),
            this.simulationView.getGridTable().getColumnCount(),
            this.getGridTable().getMesh(), dto.getInsertionInterval(),
            this.dto.getmaxCarInMeshQuantitySameTime());

    this.vehiclesOnGrid = new ArrayList<>();
    this.vehiclesInQueue = this.loadVehicles();

    simulationModel.setVehiclesInQueue(vehiclesInQueue);
    simulationModel.setVehiclesOnGrid(vehiclesOnGrid);

    simulationModel.start();
```

Simulation Controller - Criação dos Veículos:

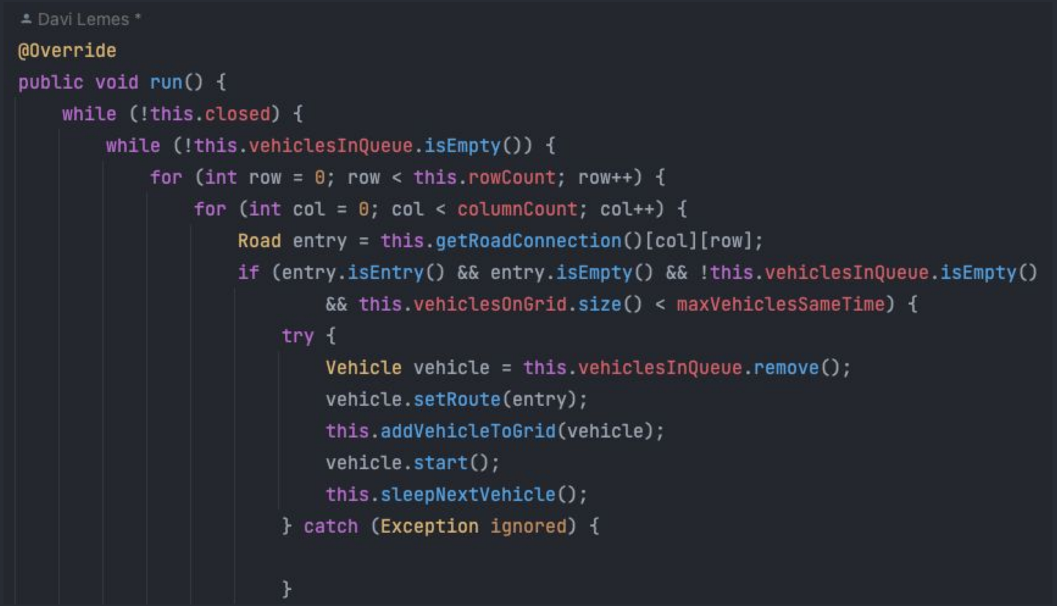




```java
public LinkedList<Vehicle> loadVehicles() {
    LinkedList<Vehicle> vehicles = new LinkedList<>();
    for (int i = 0; i < this.dto.getCarQuantity(); i++) {
        Vehicle v = new Vehicle(simulationModel);
        v.addObserver(this);
        vehicles.add(v);
    }
    return vehicles;
}
```

Model de Simulation

Simulation:

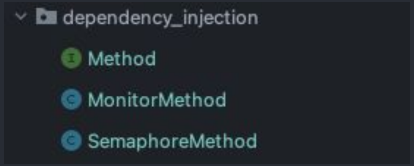

Davi Lemes *

```
@Override
public void run() {
    while (!this.closed) {
        while (!this.vehiclesInQueue.isEmpty()) {
            for (int row = 0; row < this.rowCount; row++) {
                for (int col = 0; col < columnCount; col++) {
                    Road entry = this.getRoadConnection()[col][row];
                    if (entry.isEntry() && entry.isEmpty() && !this.vehiclesInQueue.isEmpty()
                            && this.vehiclesOnGrid.size() < maxVehiclesSameTime) {
                        try {
                            Vehicle vehicle = this.vehiclesInQueue.remove();
                            vehicle.setRoute(entry);
                            this.addVehicleToGrid(vehicle);
                            vehicle.start();
                            this.sleepNextVehicle();
                        } catch (Exception ignored) {

                        }
```

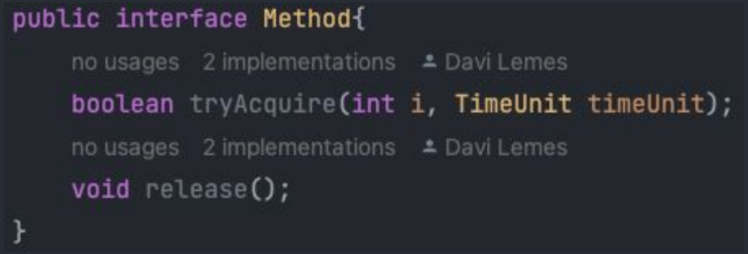
Semáforo vs Monitor

Semáforo vs Monitor: ~~Injeção de Dependência~~

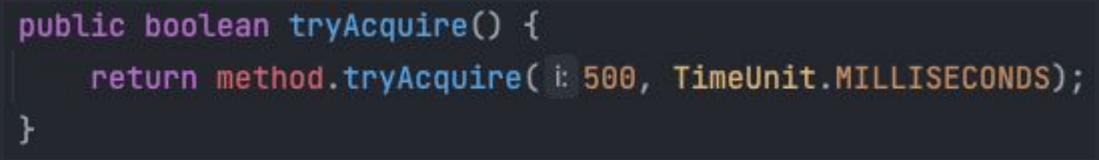
Primeira Tentativa:



```
dependency_injection
    Method
    MonitorMethod
    SemaphoreMethod
```

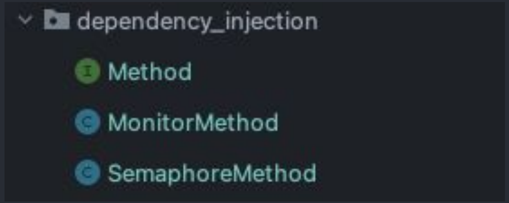


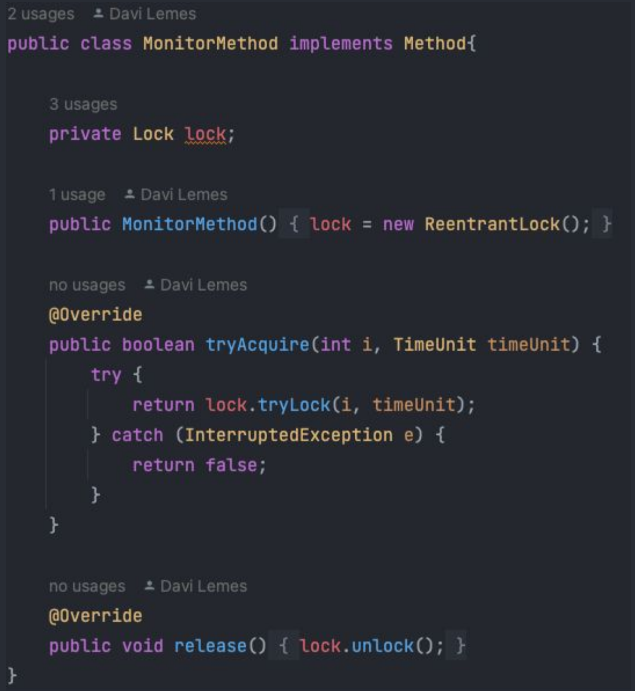
```
public interface Method{
    no usages   2 implementations   Davi Lemes
    boolean tryAcquire(int i, TimeUnit timeUnit);
    no usages   2 implementations   Davi Lemes
    void release();
}
```

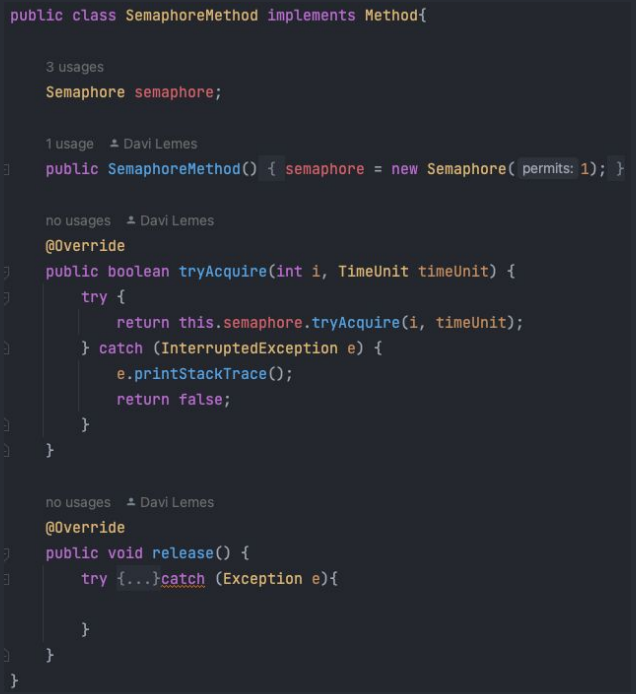


```
public boolean tryAcquire() {
    return method.tryAcquire( i: 500, TimeUnit.MILLISECONDS);
}
```

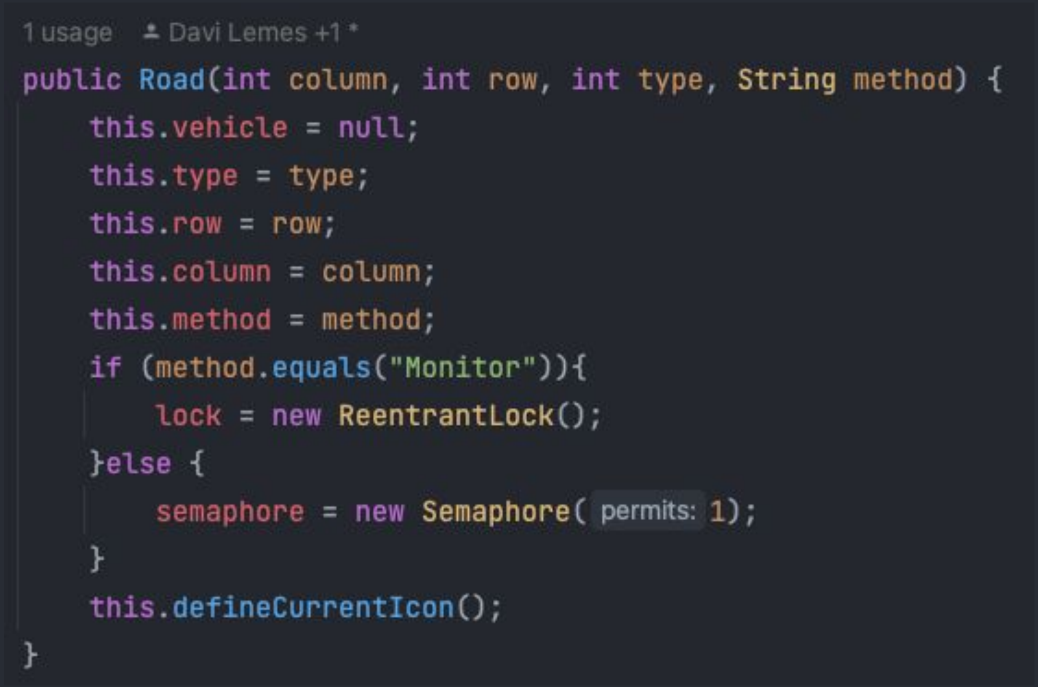
Semáforo vs Monitor: ~~Injeção de Dependência~~


```
dependency_injection
    Method
    MonitorMethod
    SemaphoreMethod
```


```
2 usages  Davi Lemes
public class MonitorMethod implements Method{

    3 usages
    private Lock lock;

    1 usage  Davi Lemes
    public MonitorMethod() { lock = new ReentrantLock(); }

    no usages  Davi Lemes
    @Override
    public boolean tryAcquire(int i, TimeUnit timeUnit) {
        try {
            return lock.tryLock(i, timeUnit);
        } catch (InterruptedException e) {
            return false;
        }
    }

    no usages  Davi Lemes
    @Override
    public void release() { lock.unlock(); }
}
```

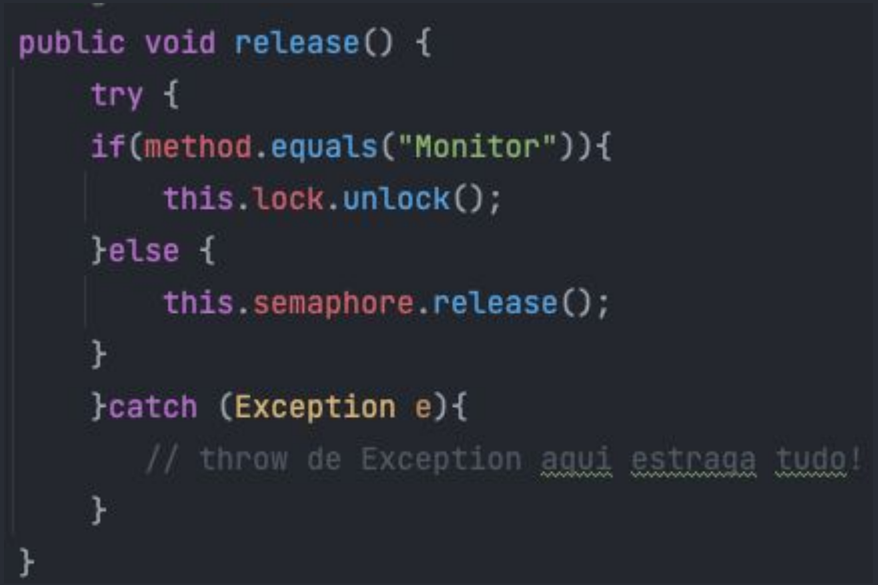

```
public class SemaphoreMethod implements Method{

    3 usages
    Semaphore semaphore;

    1 usage  Davi Lemes
    public SemaphoreMethod() { semaphore = new Semaphore( permits: 1); }

    no usages  Davi Lemes
    @Override
    public boolean tryAcquire(int i, TimeUnit timeUnit) {
        try {
            return this.semaphore.tryAcquire(i, timeUnit);
        } catch (InterruptedException e) {
            e.printStackTrace();
            return false;
        }
    }

    no usages  Davi Lemes
    @Override
    public void release() {
        try {...}catch (Exception e){

        }
    }
}
```

Semáforo vs Monitor:



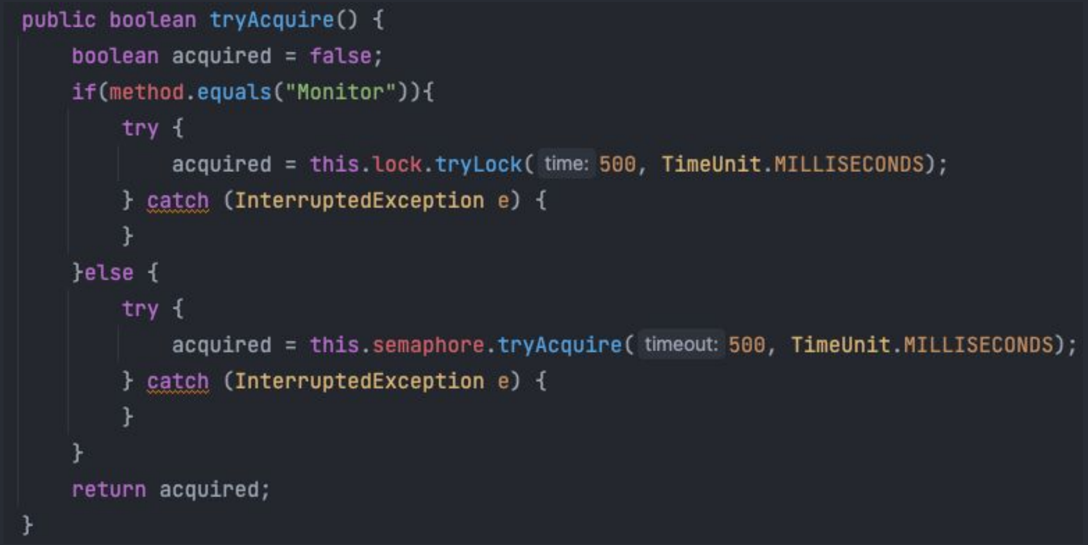
```java
public Road(int column, int row, int type, String method) {
    this.vehicle = null;
    this.type = type;
    this.row = row;
    this.column = column;
    this.method = method;
    if (method.equals("Monitor")){
        lock = new ReentrantLock();
    }else {
        semaphore = new Semaphore( permits: 1);
    }
    this.defineCurrentIcon();
}
```

Semáforo vs Monitor:



```
public void release() {
    try {
    if(method.equals("Monitor")){
        this.lock.unlock();
    }else {
        this.semaphore.release();
    }
    }catch (Exception e){
       // throw de Exception aqui estraga tudo!
    }
}
```

Semáforo vs Monitor:



```
public boolean tryAcquire() {
    boolean acquired = false;
    if(method.equals("Monitor")){
        try {
            acquired = this.lock.tryLock( time: 500, TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
        }
    }else {
        try {
            acquired = this.semaphore.tryAcquire( timeout: 500, TimeUnit.MILLISECONDS);
        } catch (InterruptedException e) {
        }
    }
    return acquired;
}
```