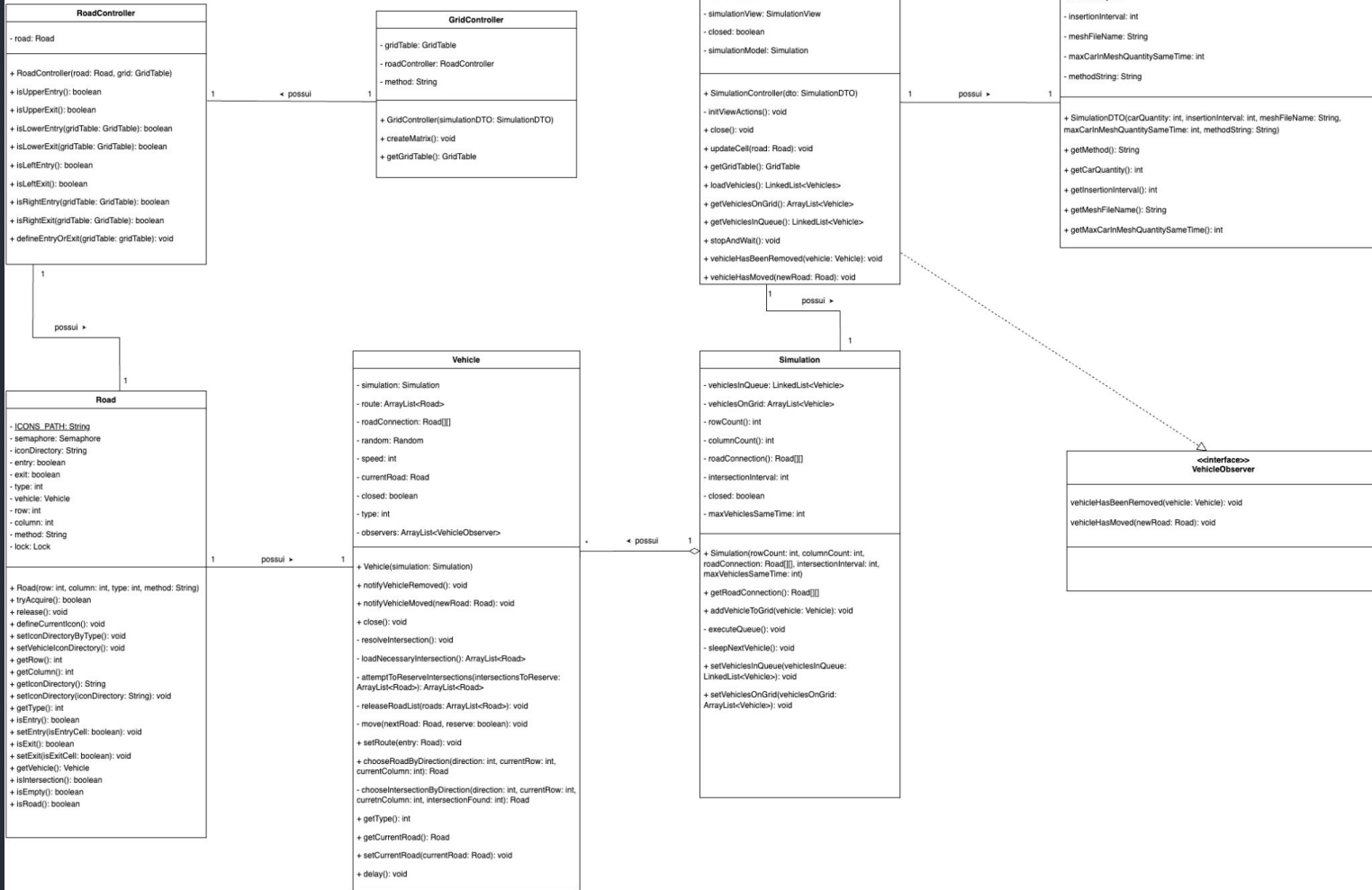


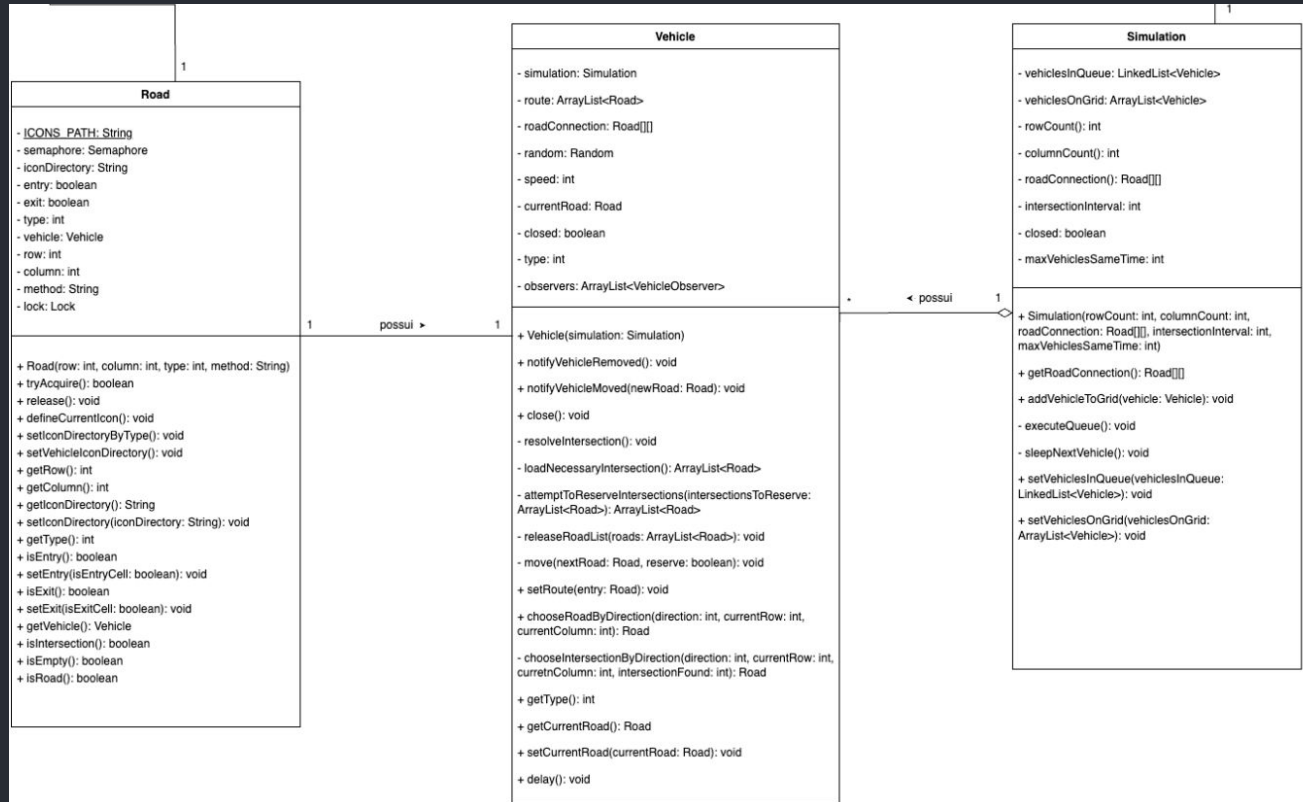
# Threads - 65DSD

Davi e Caue

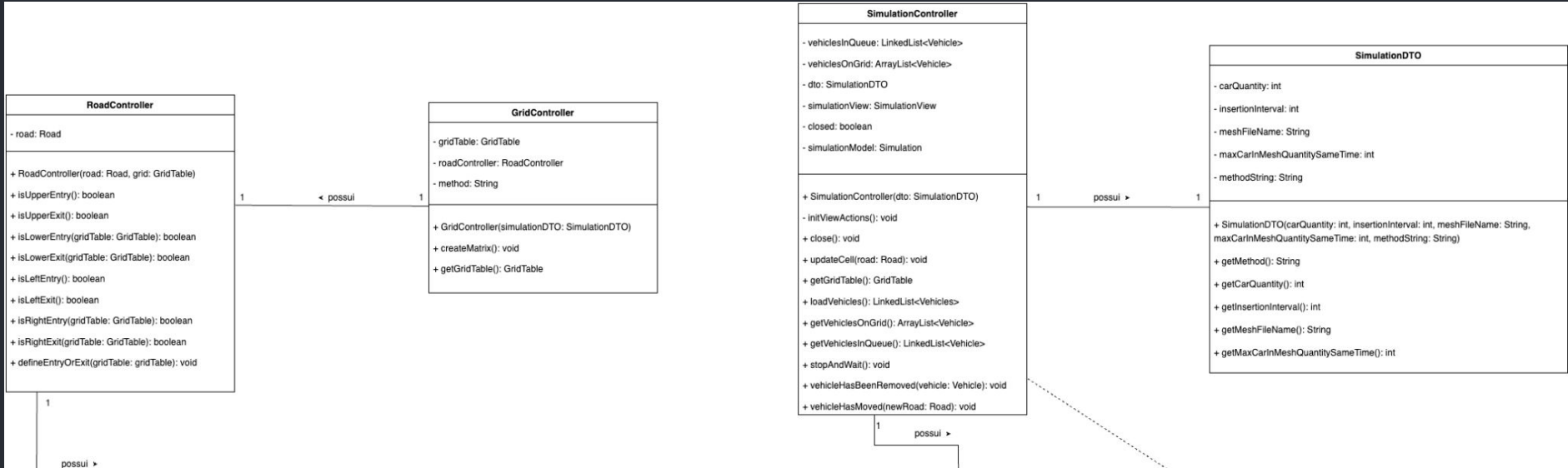
# Diagrama de Classes



# Diagrama de Classes:



# Diagrama de Classes:



# 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:

```
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:

```
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:

```
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

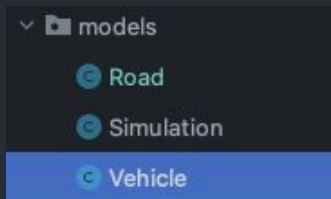
```
± marquescauee +1
@Override
public void run() {
    while (!this.closed) {
        while (!route.isEmpty()) {
            int nextPosition = 0;
            if (route.get(nextPosition).isIntersection()) {
                resolveIntersection();
            } else {
                Road road = this.route.get(nextPosition);
                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    Davi Lemes +1 \*

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

1 usage    Davi Lemes

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

1 usage    Davi Lemes \*

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

1 usage    Davi Lemes \*

```
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:

```
@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:



2 usages     marquescauee +1

```
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:



```
1 marquescauee
simulationView.addActionStopAndWaitBtn(new ActionListener(){
    1 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:



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

# Simulation Controller:



2 usages    + marquescauee +1 \*

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

usage    + Davi Lemes +1

```
public 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:



```
1 usage  👤 marquescauee +1
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:



```
1 usage  🧑 Davi Lemes +1 *
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;  
}
```