This UML Class Diagram represents an autonomous vehicle system that integrates vehicle control, environmental detection, and network communication. The system's goal is to maintain safe driving, handle emergencies, and communicate road and environmental data for decision making.

It consists of multiple interconnected classes: Vehicle, Driver, DetectionDevice, Obstacle, Sensor, ProximitySensor, Camera, ExternalCondition, NetworkElement, RoadPath, and Carrier. Each class is responsible for a specific aspect of the autonomous vehicle's functionality from navigation to obstacle detection and communication.

1. Vehicle

The Vehicle class is the central component of the system. It manages driving operations, navigation, and emergency responses.

- Attributes:
  - path: RoadPath — current route the vehicle follows.
  - speed: int — vehicle's current speed.
  - emergencyMode: bool = false — indicates whether emergency mode is active.
  - engineTemperature: float — temperature of the vehicle's engine.
  - transmissionForce: float — transmission power used.
  - tirePressure: float — pressure levels in the tires.
- Methods:
  - turnLeft(float degrees): void / turnRight(float degrees): void — adjust vehicle direction.
  - activateEmergencyMode(string disaster): void — enables emergency driving behavior.
  - changeTrajectory(RoadPath object): void — adjusts driving path.

    - followRoad(RoadPath path): void — follows given path.
    - updateNetwork(NetworkElement antenna): RoadPath — communicates with the network.
    - break(): void — stops the vehicle.

2. Driver

Represents the user interface and manual control of the vehicle.

- Attributes:
    - field: type — defines driver configuration.
- Methods:
    - activateEmergencyMode(): bool — activates the emergency mode.
    - activateAutonomousDriving(): bool — switches to autonomous mode.

This class acts as the interface between the human operator and the automated system.

3. DetectionDevice

Handles obstacle scanning and environmental awareness. It integrates data from sensors to detect and assess dangers.

- Attributes:
    - scannedObstacle: Obstacle — stores the currently detected obstacle.
- Methods:
    - getScannedObstacle(): Obstacle — returns detected obstacle.
    - isDanger(Obstacle scannedObstacle): bool — checks if obstacle poses a risk.

- communicateDanger(): Obstacle — sends warning data to the vehicle.

4. Obstacle

Represents objects or hazards detected by sensors.

- Attributes:
  - width: float, height: float, distance: float — define obstacle size and proximity.
- Methods:
  - getWidth(): float, getHeight(): float, getDistance(): float — return obstacle dimensions.

5. Sensor (Abstract Class)

A base class for all sensors in the system.

- Attributes:
  - field: type — sensor-specific attribute.
- Methods:
  - method(type): type — represents general sensor behavior.

6. ProximitySensor (Inherits from Sensor)

A specialized sensor used for detecting distance to objects.

- Attributes:
  - distanceToObject: float — measures how far an object is.
- Methods:
  - detectSurroundings(): Obstacle — returns detected nearby obstacle.

7. Camera (Inherits from Sensor)

Captures visual data of the environment.

- Attributes:
  - field: type — defines camera-specific configuration.
- Methods:
  - detectSurroundings(): Obstacle — identifies obstacles visually.

8. ExternalCondition

Monitors weather and visibility affecting driving.

- Attributes:
  - weatherCondition: string
  - visibility: float
  - weather: string
- Methods:
  - getWeather(): string — returns current weather.
  - getWeatherReading(weather): void — updates conditions.
  - getVisibility(): float / setVisibility(float): void — manage visibility metrics.

9. NetworkElement

Manages communication between the vehicle and network infrastructure.

- Attributes:
  - carrierInfo: Carrier

    - updatedRoad: RoadPath
- Methods:
    - communicatePath(RoadPath path): RoadPath — shares path info.
    - communicatePath(RoadPath path, string emergency): RoadPath — includes emergency data.
    - getCarrierInfo(): Carrier
    - getRoadInfo(): RoadPath

This class ensures the vehicle receives live road and emergency updates.

10. RoadPath

Stores road and path information.

- Attributes:
    - leftMarkDist: float, rightMarkDist: float, miles: float — describe lane markings and length.
- Methods:
    - getMarkDist(float l, float r): float — retrieves distance between lane marks.
    - getMiles(): float — returns road length.

11. Carrier

Represents external communication services, such as satellite or network providers.

- Attributes:
    - currentRoad: RoadPath
    - disasters: string[] — list of road hazards.

- Methods:
    - providePath(Location origin, Location destiny): RoadPath
    - communicateDisaster(Location currentLocation, string disaster): void

It enables GPS and disaster communication between external systems and the vehicle.

12. IMU Sensor

The IMUSensor (Inertial Measurement Unit Sensor) tracks the vehicle's motion and orientation, crucial for stability control and navigation accuracy.

- Attributes:
    - acceleration: Vector3D — Measures linear acceleration in three dimensions.
    - angularRate: Vector3D — Measures the rate of rotation around each axis.
- Methods:
    - detectSurroundings(): Obstacle — Uses motion data to detect potential hazards based on movement changes (e.g., skidding or collisions).
    - getVehicleAltitude(): Vector3D — Returns the vehicle's spatial orientation or height level.

13. TirePressure Sensor

Monitors tire pressure to ensure safety and performance. It's vital for maintaining traction and avoiding blowouts.

- Attributes:
    - currentPressure: float — The current tire pressure reading.
    - lowPressureThreshold: float — The minimum safe tire pressure value.
- Methods:
    - detectSurroundings(): Obstacle — May detect abnormal pressure or surface issues related to obstacles.
    - isPressureLow(): bool — Returns true if the current pressure is below the safe threshold.

This UML diagram provides a detailed model of an autonomous driving system, showcasing how the vehicle interacts with sensors, networks, and drivers to ensure safety, communication, and adaptability.
The design supports both normal driving and emergency mode, demonstrating strong modularity and clear separation of responsibilities across components.