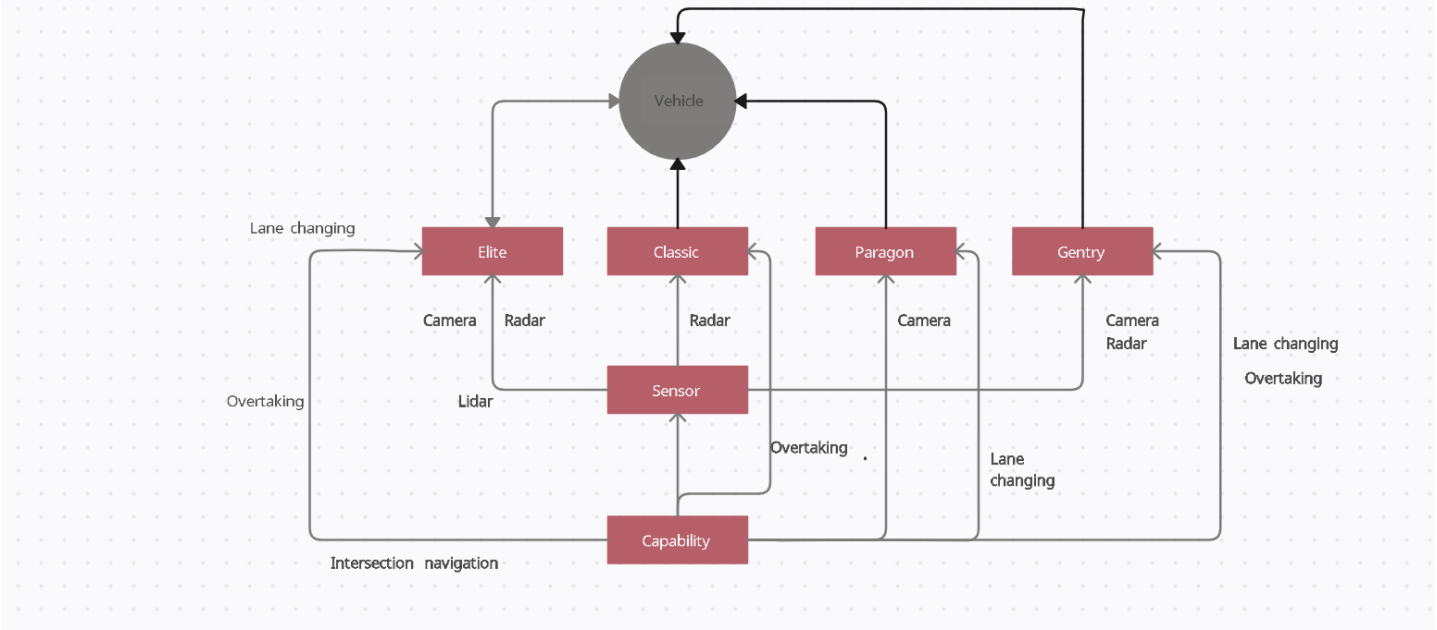
VAS VEHICLES AND THEIR CAPABILITIES

1. The UML diagram is shown below . It consists of four classes: Vehicle, Sensor, Capability and Configuration. Vehicle is an abstract class that represents a generic vehicle with a name and a list of capabilities. Sensor is an enum class that defines the possible sensor types: CAMERA, RADAR and LiDAR. Capability is an enum class that defines the possible vehicle functionalities: LANE_CHANGING, OVERTAKING and INTERSECTION_NAVIGATION. Configuration is a class that encapsulates the vehicle model and the sensor hardware selected by the buyer.



2. Code implementation for the same is shown below in python. It uses inheritance to define four subclasses of Vehicle: Elite, Classic, Paragon and Gentry. Each subclass overrides the **init** method to set its name and capabilities according to its sensor hardware. The code also defines a factory function create_vehicle that takes a Configuration object as input and returns a Vehicle object with the specified capabilities based on the supplied configuration.

VAS VEHICLES AND THEIR CAPABILITIES

Main.py

```
from enum import Enum


class Sensor(Enum):
    CAMERA = 1
    RADAR = 2
    LiDAR = 3

class Capability(Enum):
    LANE_CHANGING = 1
    OVERTAKING = 2
    INTERSECTION_NAVIGATION = 3

class Vehicle:
    def __init__(self, name):
        self.name = name
        self.capabilities = []

    def add_capability(self, capability):
        self.capabilities.append(capability)

    def has_capability(self, capability):
        return capability in self.capabilities

class Elite(Vehicle):
    def __init__(self, sensors):
        super().__init__("Elite")
        if Sensor.CAMERA in sensors:
            self.add_capability(Capability.LANE_CHANGING)
        if Sensor.RADAR in sensors:
            self.add_capability(Capability.OVERTAKING)
        if Sensor.LiDAR in sensors:
            self.add_capability(Capability.INTERSECTION_NAVIGATION)

class Classic(Vehicle):
```

```
    def __init__(self, sensors):
        super().__init__("Classic")
        if Sensor.RADAR in sensors:
            self.add_capability(Capability.OVERTAKING)

class Paragon(Vehicle):
    def __init__(self, sensors):
        super().__init__("Paragon")
        if Sensor.CAMERA in sensors:
            self.add_capability(Capability.LANE_CHANGING)

class Gentry(Vehicle):
    def __init__(self, sensors):
        super().__init__("Gentry")
        if Sensor.CAMERA in sensors:
            self.add_capability(Capability.LANE_CHANGING)
        if Sensor.RADAR in sensors:
            self.add_capability(Capability.OVERTAKING)

class Configuration:
    def __init__(self, model, sensors):
        self.model = model
        self.sensors = sensors

def create_vehicle(config):
     if config.model == "Elite":
        return Elite(config.sensors)
    elif config.model == "Classic":
        return Classic(config.sensors)
    elif config.model == "Paragon":
        return Paragon(config.sensors)
    elif config.model == "Gentry":
        return Gentry(config.sensors)

config = Configuration("Elite", [Sensor.CAMERA, Sensor.RADAR])
vehicle = create_vehicle(config)
```

Test.py

```
import unittest
from main import *

class TestCreateVehicle(unittest.TestCase):

    def test_create_elite_with_camera_and_radar(self):
        config = Configuration("Elite", [Sensor.CAMERA, Sensor.RADAR])
        vehicle = create_vehicle(config)
        self.assertEqual(vehicle.name, "Elite")
        self.assertTrue(vehicle.has_capability(Capability.LANE_CHANGING))
        self.assertTrue(vehicle.has_capability(Capability.OVERTAKING))
        self.assertFalse(vehicle.has_capability(Capability.INTERSECTION_NAVIGATION))



if __name__ == "__main__":
    unittest.main()
```