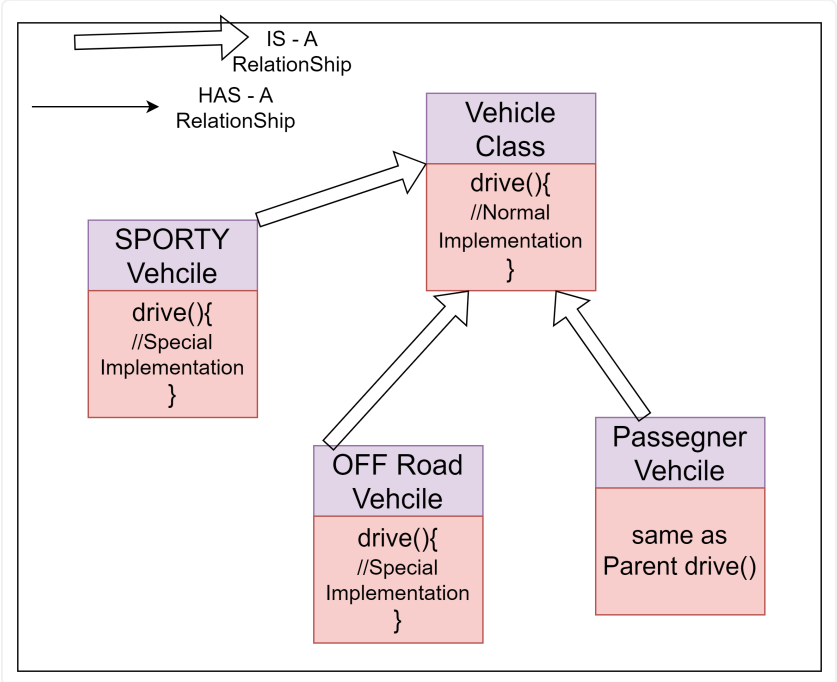
Strategy Design Pattern explanation.

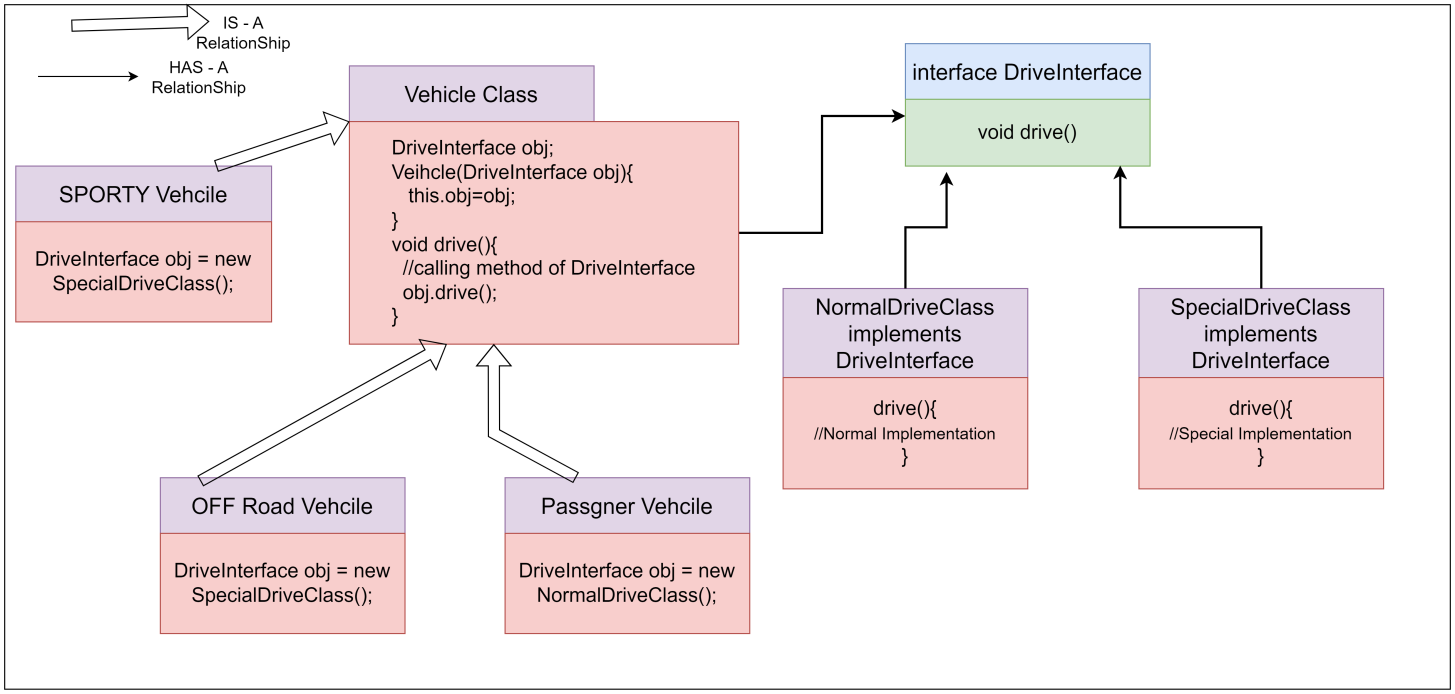
1. **What is Strategy Design Pattern ?.**

 IS - A Relation Ship
- >HAS - A RelationShip.

- There is a base class Vehicle and there is a method called drive() which have nprmal capability.
- Vehicle has multiple class like PassengerVehcile, OffRoadVehcile and SPORTY_Vehcile,
- OffRoadVehcile and SPORTY_Vehcile Given same special capabilty on child level to drive method.
- Code duplicay at child level for drive method.
- more special feature, brings more special implementation for overridien method at child level lead's to code duplicacy.
- Less Code Reusability.



To Solve that problem we are using **Strategy Design Pattern.**




```
package System_Design.DesignPatterns;

interface DriveInterface{
    void drive(String Strategy);
}

class NormalDriveStrategy implements DriveInterface{

    @Override
    public void drive(String Strategy) {
        System.out.println(Strategy);
    }
}
class SpecialDriveStrategy implements DriveInterface{

    @Override
    public void drive(String Strategy) {
        System.out.println(Strategy);
    }
}

class Vehicle{

    DriveInterface obj;
    String Strategy;
    Vehicle(DriveInterface obj,String Strategy){
        this.obj = obj;
        this.Strategy=Strategy;
    }
```

```
    public void drive(){
        obj.drive(Strategy);
    }
}

class OffRoadVehicle extends Vehicle{
    OffRoadVehicle(){super(new SpecialDriveStrategy(),"SpecialDriveStrategy OffRoad");}
}

class SPORTSVehicle extends Vehicle{
    SPORTSVehicle(){super(new SpecialDriveStrategy(),"SpecialDriveStrategy SPORTS");}
}

class PassegerVehicle extends Vehicle{
    PassegerVehicle(){super(new NormalDriveStrategy(),"NormalDriveStrategy Passenger");}
}

public class StrategyDesignPatterns {
    public static void main(String[] args){
        Vehicle vehicle = new SPORTSVehicle();
        vehicle.drive();
    }
}
//OUPUT:
SpecialDriveStrategy SPORTS

Process finished with exit code 0
```