1. Classification of Design Patterns

12 January 2024 21:46

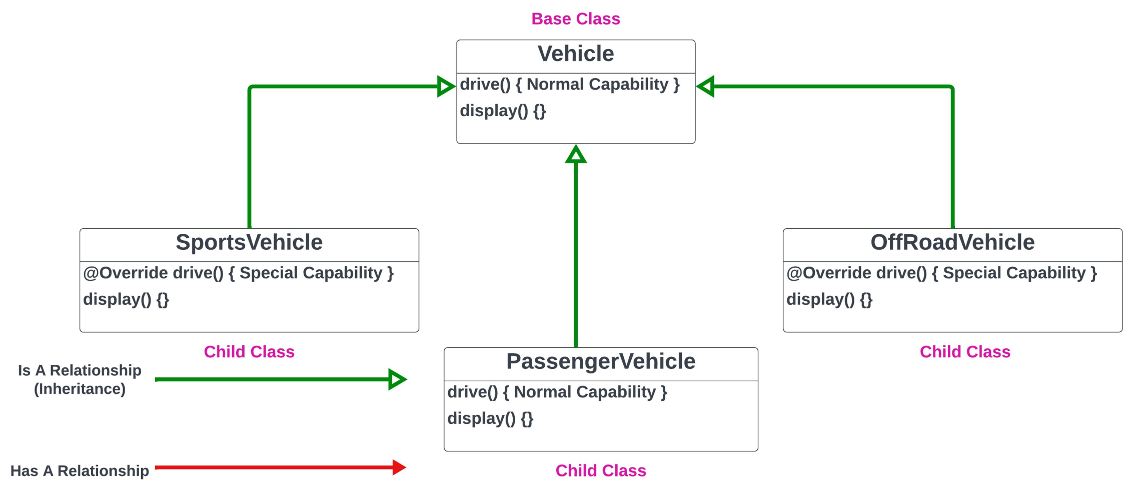
- **Creational Design Patterns**
  - **Abstract Factory**
  - **Builder**
  - **Factory Method**
  - **Prototype**
  - **Singleton**
- **Structural Design Patterns**
  - **Adapter**
  - **Composite**
  - **Bridge**
  - **Decorator**
  - **Façade**
  - **Flyweight**
  - **Proxy**
- **Behavioral Design Patterns**
  - **Chain of Responsibility**
  - **Interpreter**
  - **Iterator**
  - **Mediator**
  - **Memento**
  - **Observer**
  - **State**
  - **Strategy**
  - **Template Method**
  - **Visitor**

2. Strategy Design Pattern

12 January 2024 22:39

Pre-requisites:

1. We have a Vehicle Base class which have drive and display methods
2. **SportsVehicle** and **OffRoadVehicle** classes extending Vehicle class need Special Drive Capabilities so overrides drive method
3. **PassengerVehicle** class needs Normal Drive Capability so does not override drive method



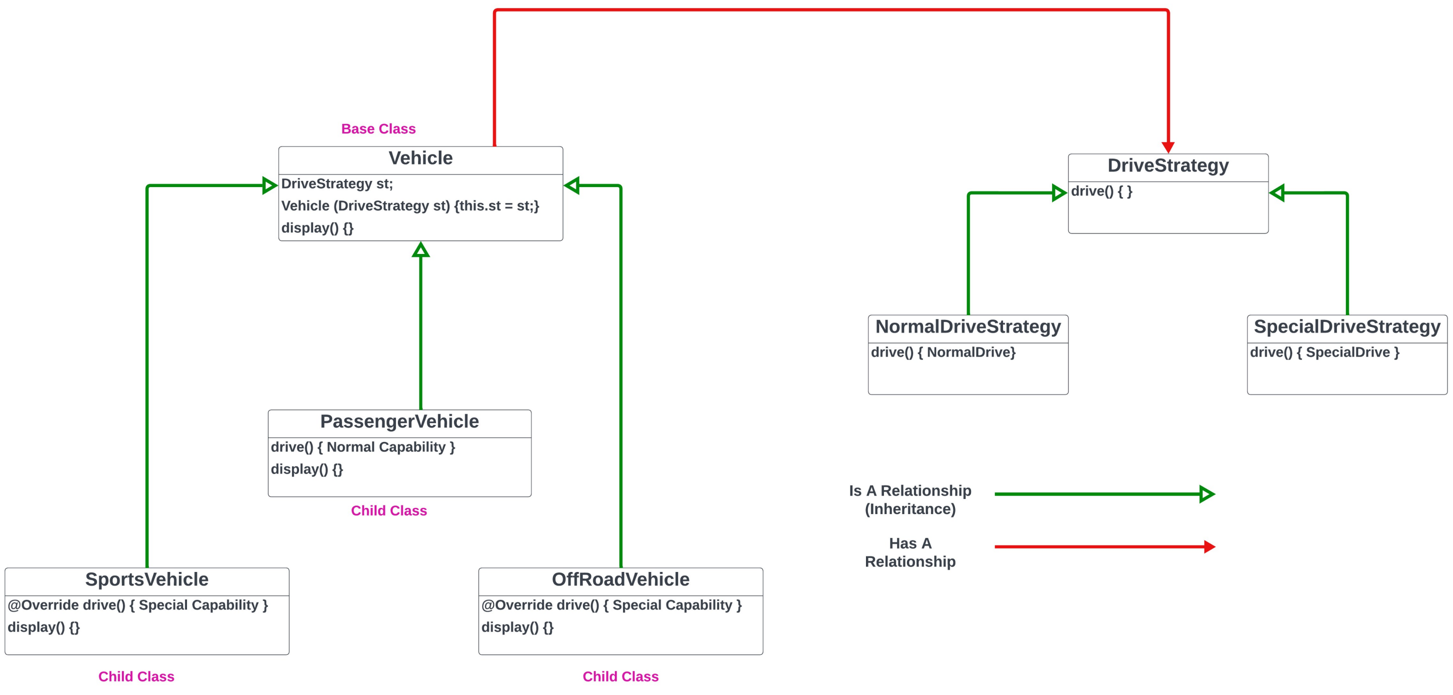
Base Class
Vehicle
drive() { Normal Capability }
display() {}
SportsVehicle
@Override drive() { Special Capability }
display() {}
Child Class
OffRoadVehicle
@Override drive() { Special Capability }
display() {}
Child Class
PassengerVehicle
drive() { Normal Capability }
display() {}
Child Class
Is A Relationship (Inheritance)
Has A Relationship

Problem:

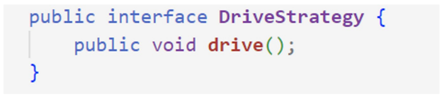
1. Here both **SportsVehicle** and **OffRoadVehicle** class needs Special Drive capabilities and their functionality is different from Base class functionality so may result into duplication of code.
2. This duplication of code may increase when we need additional TypesVehicle classes

Solution:

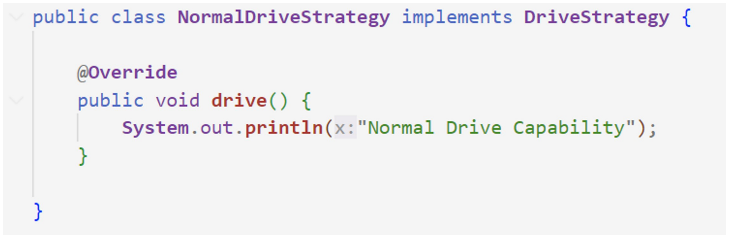
1. This can be resolved using Strategy Design Pattern
2. Create a **DriveStrategy** Interface with concrete classes implementing the same as **NormalDriveStrategy** and **SpecialDriveStrategy**
3. In Vehicle class, use a variable of **DriveStrategy**
4. Now the individual classes have the responsibility to pass the Strategy to **Vehicle** class



Strategy Designs Implementation



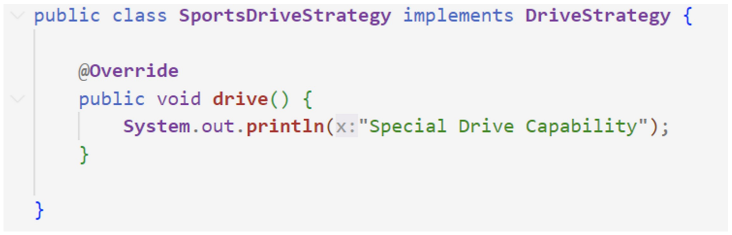
```java
public interface DriveStrategy {
    public void drive();
}
```



```java
public class NormalDriveStrategy implements DriveStrategy {

    @Override
    public void drive() {
        System.out.println(x:"Normal Drive Capability");
    }

}
```



```java
public class SportsDriveStrategy implements DriveStrategy {

    @Override
    public void drive() {
        System.out.println(x:"Special Drive Capability");
    }

}
```

Vehicle Classes Modification to use the above Strategy

```java
public class Vehicle {

    private DriveStrategy strategy;

    public Vehicle(DriveStrategy strategy) {
        this.strategy = strategy;
    }

    public void drive() {
        strategy.drive();
    }

    public DriveStrategy getDriveStrategy() {
        return this.strategy;
    }
}
```

```java
public class OffRoadVehicle extends Vehicle {
    public OffRoadVehicle() {
        super(new SportsDriveStrategy());
    }
}
```

```java
public class PassengerVehicle extends Vehicle {
    public PassengerVehicle() {
        super(new NormalDriveStrategy());
    }
}
```

```java
public class SportsVehicle extends Vehicle {
    public SportsVehicle() {
        super(new SportsDriveStrategy());
    }
}
```