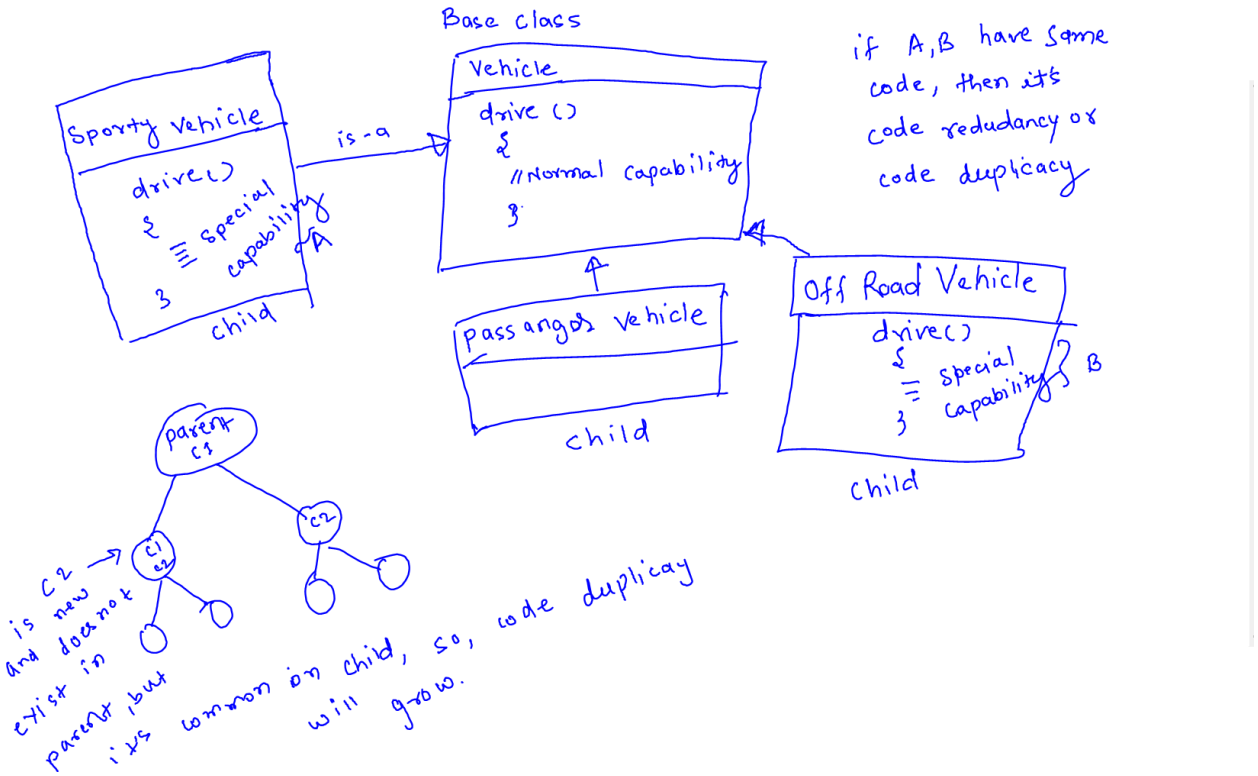
Strategy Design Pattern:

```
Class Vehicle
{
Drive()
{
 // Normal Capability
}
}
```

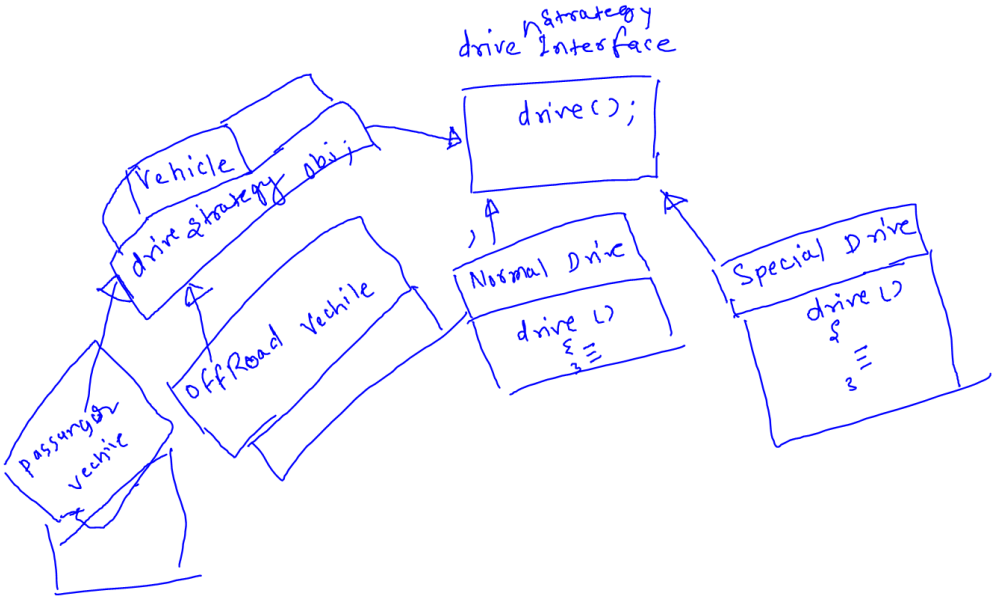


The fundamental of every design pattern is to separate out what changes over period of time from what remains constant. As you see in Strategy pattern, we are trying to separate out dependency of drive from the main class as much as possible by making various strategies to inject it dynamically based on client requirement so that in future if new requirement for drive comes, there will be minimal/no changes in existing code(Open closed Principle).

In Strategy Design pattern we create an interface and it's different implementation based on strategy !! And inject appropriate strategy in client class by creating constructor or any injection of your choice

Here, we won't define method in Base class, instead we will use in interface and child will define its own implementation.

Here we will do constructor injection. in vehicle class.



Without Strategy:

```
package withoutStrategy;

public class OffRoadVehicle extends Vehicle{
    @Override
    public void drive() {
        /* super.drive(); */
        System.out.println("special cabpability");

    }
}
```

```
package withoutStrategy;

public class PassengerVehicle extends Vehicle{
}
```

```java
package withoutStrategy;

public class SportyVehicle extends Vehicle {
    @Override
    public void drive()
    {
        System.out.println("special cabpability");
    }
}
```

```java
package withoutStrategy;

public class Vehicle {
    public  void drive()
    {
        System.out.println("dive capability");
    }
}
```

With Strategy :

```java
package withStrategey;

public interface DriveStrategey {
    public void drive();
}
```

```java
package withStrategey;

public class Main {
    public static void main(String[] args) {
        Vehicle vechObj = new OffRoadVehicle();
        vechObj.drive();
    }
}
```

```java
package withStrategey;
```

```java
public class NormalDriveStrategey implements
DriveStrategey{


    @Override
    public void drive() {
        System.out.println("Normal cap");
    }
}
```

```java
package withStrategey;

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

```java
package withStrategey;

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

```java
package withStrategey;

public class SportsDriveStrategey implements
DriveStrategey{

    @Override
    public void drive() {
        System.out.println("special cap");
    }
}
```