Design Pattern Term Project

Muhammet Uçar - 20180808085

Emine Ece Bayramer – 20180808020

Statement of Work

Our project is a shop that produces a car and adds and develops new features to the cars produced. We build and sell cars in a simpler and more efficient way. We can produce cars of different brands. The CarStore class provides this process. There are currently 3 brands: Toyota, Mercedes and Volvo. The features that are common to cars and must be present in all of them are created in the Main body. On top of that, the car is customized by selecting its features according to the brand and model. It is mandatory to have in cars, but features with different types are selected. For example, every car has a gear system, but these systems may be different. While one car has an automatic transmission system, another car may have a 5-speed brake system. Or the brake systems may be different in each car. It may have abs or asr braking system.

The same is true for fuel types and engine cylinder types. Gasoline, diesel and electric options are available. The car can have only one of these options.

Different add-ons can also be added to the selected car. There are features that are not available in the car but can be added later. For example carplay, sensors, glass roof etc. Accordingly, the cost of the car changes. The features of the car are also updated according to the plugin. In this way, brand-specific cars can be produced and features can be added. We made it understandable with our design.

Design Patterns

Common features that should be in every car are taken from a single class (class main body: wheels, door number, seat number). Features that exist in each car but have different types were made using the strategy pattern. Interfaces were created for common features, and the variants belonging to that feature were created as concrete classes. For example, there are the "Gear" interface and the automatic gear, gear 5, and gear 6 classes that implement it. Each car draws from that class whatever type of feature it has.

Since our feature here is mutable, we have chosen the strategy pattern here because the features can be interchanged.

Selected add-ons to the car were created using the decorator pattern. The customer can use the class of the feature they want to add to the car, and the car is updated. We chose the Decorator Pattern here because we add new features and new responsibilities to our objects dynamically.

The production of cars and their differentiation according to brands were created with the Factory Method Pattern.

We used the Factory Method Pattern when producing cars because subclasses decide the features of the cars.

Design principles we use in our Project :

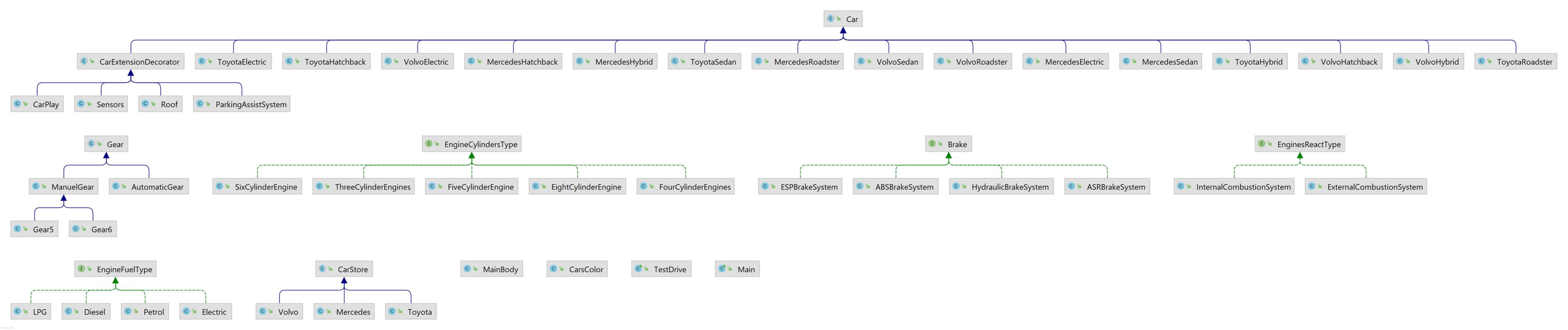
1.) Design Principle: Program to an interface, not an implementation.
2.) Design Principle: Favor composition over inheritance
3.) Design Principle: Classes should be open for extension, but closed for modifi cation.
- Our goal is to allow classes to be easily extended to incorporate new behavior without modifying existing code
- Start with a Car and "decorate" it with the car feature at runtime

4.) Design Principle: Depend upon abstractions. Do not depend upon concrete classes.

UML Class Diagram

Implementation

We implemented this project in java language.


Car
CarExtensionDecorator
ToyotaElectric
ToyotaHatchback
VolvoElectric
MercedesHatchback
MercedesHybrid
ToyotaSedan
MercedesRoadster
VolvoSedan
VolvoRoadster
MercedesElectric
MercedesSedan
ToyotaHybrid
VolvoHatchback
VolvoHybrid
ToyotaRoadster
CarPlay
Sensors
Roof
ParkingAssistSystem
Gear
ManuelGear
AutomaticGear
Gear5
Gear6
EngineCylindersType
SixCylinderEngine
ThreeCylinderEngines
FiveCylinderEngine
EightCylinderEngine
FourCylinderEngines
Brake
ESPBrakeSystem
ABSBrakeSystem
HydraulicBrakeSystem
ASRBrakeSystem
EnginesReactType
InternalCombustionSystem
ExternalCombustionSystem
EngineFuelType
LPG
Diesel
Petrol
Electric
CarStore
Volvo
Mercedes
Toyota
MainBody
CarsColor
TestDrive
Main


Car
cost() double
displayCar() void
performBrake() void
performEngineCylindersType() void
performEngineFuelType() void
performEnginesReactType() void
performGear() void
prepare() void
sale() void
brake Brake
engineCapacity int
engineCylindersType EngineCylindersType
engineFuelType EngineFuelType
enginePower int
enginesReactType EnginesReactType
gear Gear
mainBody MainBody
name String
CarExtensionDecorator
name String
ToyotaRoadster
cost() double
VolvoHybrid
cost() double
VolvoHatchback
cost() double
ToyotaHybrid
cost() double
MercedesSedan
cost() double
display() void
MercedesElectric
cost() double
VolvoRoadster
cost() double
VolvoSedan
cost() double
MercedesRoadster
cost() double
ToyotaSedan
cost() double
MercedesHybrid
cost() double
MercedesHatchback
cost() double
VolvoElectric
cost() double
ToyotaHatchback
cost() double
ToyotaElectric
cost() double
ParkingAssistSystem
car Car
cost() double
name String
CarPlay
car Car
cost() double
name String
Roof
car Car
cost() double
name String
Sensors
car Car
cost() double
name String
CarStore
mainBody MainBody
mainBody2 MainBody
mainBody3 MainBody
mainBody4 MainBody
absBrakeSystem Brake
espBrakeSystem Brake
asrBrakeSystem Brake
hydraulicBrakeSystem Brake
automaticGear Gear
gear6 Gear
gear5 Gear
fuelTypeElectric EngineFuelType
fuelTypePetrol EngineFuelType
fuelTypeDiesel EngineFuelType
eightCylinderEngine EngineCylindersType
fiveCylinderEngine EngineCylindersType
sixCylindersType EngineCylindersType
externalCombustionSystem EnginesReactType
internalCombustionSystem EnginesReactType
createCar(String) Car
orderCar(String) Car
Toyota
createCar(String) Car
Mercedes
createCar(String) Car
Volvo
createCar(String) Car
Gear
clutch boolean
gear() void
ManuelGear
gear() void
toString() String
AutomaticGear
gear() void
toString() String
Gear6
gear() void
Gear5
gear() void
MainBody
toString() String
accelerator boolean
airbag boolean
carStereo String
doorNumber int
gasTank int
numberOfSeat int
numberOfWheels int
seatBelt boolean
trunk boolean
CarsColor
Black String
Red String
Green String
Yellow String
Blue String
Magenta String
Cyan String
White String
EngineCylindersType
cylindersType() void
FourCylinderEngines
cylindersType() void
SixCylinderEngine
cylindersType() void
EightCylinderEngine
cylindersType() void
FiveCylinderEngine
cylindersType() void
ThreeCylinderEngines
cylindersType() void
Brake
brake() void
HydraulicBrakeSystem
brake() void
ESPBrakeSystem
brake() void
ASRBrakeSystem
brake() void
ABSBrakeSystem
brake() void
EngineFuelType
fuelType() void
Electric
fuelType() void
LPG
fuelType() void
Petrol
fuelType() void
Diesel
fuelType() void
EnginesReactType
reactType() void
ExternalCombustionSystem
reactType() void
InternalCombustionSystem
reactType() void
Main
main(String[]) void
TestDrive
main(String[]) void