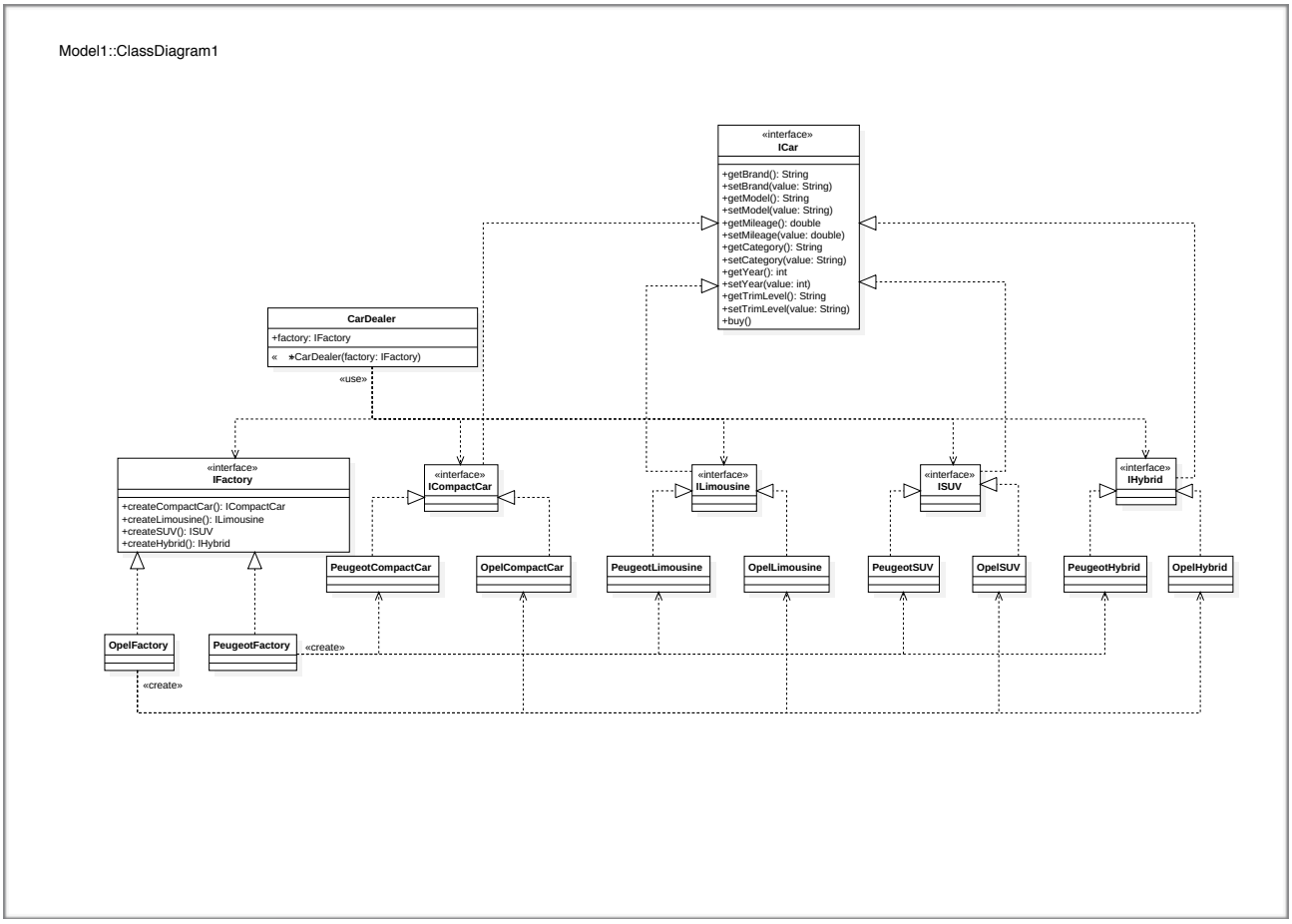
ANTHONY GUILLIER



Abstract Factory

Class : Design Pattern

Student : Anthony Guillier

01/04/2017

Group : 10

ANTHONY GUILLIER

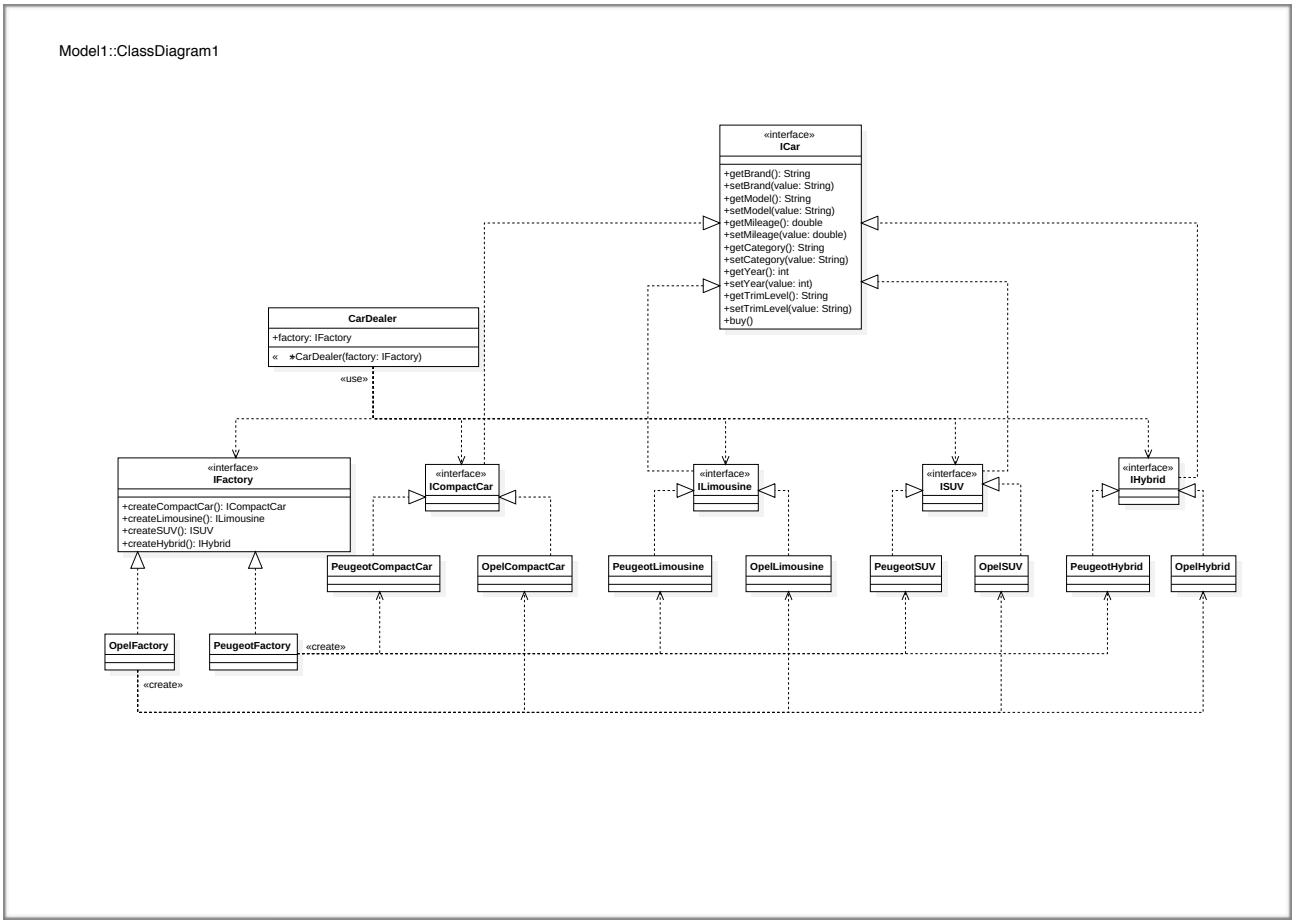
ABSTRACT FACTORY

Description of the application

. In the Netherlands, a Car Dealer usually offers cars of one brand (like Opel, Peugeot, Toyota, Kia, etc.), and this dealer sells the cars to its customers.
Each of those brands has its own factory, and they can all produce products like: compact car, limousine, SUV and hybrid. Define appropriate, different methods for each product. Beware: be very careful if you want those products extend a very global interface like ICar, because you might lose the feature *'family* of products'.
Write this application with several car factories, each producing products of its own brand. The car dealer gets an abstract factory in its constructor, and during its lifetime, it is totally unaware which brand it is.

ANTHONY GUILLIER

UML



Unit Tests

I run a test where I created 2 **Factories**: Opel & Peugeot. I also created 2 **CarDealer** linked with these factories without knowing which brand it is. I created all kinds of car with these factory and buy these to see what appear. Each factory create their own car !

```
IFactory opel = new OpelFactory();
IFactory peugeot = new PeugeotFactory();

CarDealer carDealer1 = new CarDealer(opel);
CarDealer carDealer2 = new CarDealer(peugeot);

ICompactCar cc1 = carDealer1.factory.createCompactCar("Astra", 10, "Sports Tourer", 2016);
ICompactCar cc2 = carDealer2.factory.createCompactCar("106", 10, "GTI", 2010);

IHybrid h1 = carDealer1.factory.createHybrid("Ampera", 100, "", 2017);
IHybrid h2 = carDealer2.factory.createHybrid("Hybrida", 10, "ECO", 2016);

ISUV suv1 = carDealer1.factory.createSUV("Mokka", 10, "X", 2016);
ISUV suv2 = carDealer2.factory.createSUV("SUVA", 10, "XXL", 2016);

ILimousine l1 = carDealer1.factory.createLimousine("Limousine", 10, "XL", 2016);
ILimousine l2 = carDealer2.factory.createLimousine("Limousine", 10, "Large", 2016);

cc1.buy();
h1.buy();
suv1.buy();
l1.buy();

cc2.buy();
h2.buy();
suv2.buy();
l2.buy();
```

```
Buy Opel Astra
Buy Opel Ampera
Buy Opel Mokka
Buy Opel Limousine
Buy Peugeot 106
Buy Peugeot Hybrida
Buy Peugeot SUVA
Buy Peugeot Limousine
```

Discussion about Reusability

Using this Pattern you only have to change *IFactory* with an other interface or create new concrete **Factory** and you have a whole different application doing something completely different but with the exact same pattern.

Discussion about Extensibility

The different **Factory** extends an *IFactory* interface. That means you can have as many **Factory** as you want. You can also create new different type of car that will extends *iCar* and be used by the **CarDealer**.

Discussion about Maintainability

Everything is separate in *interfaces*, so you can change prices and descriptions, without changing or creating a new application. The **Factory** is able to return any type of car instance it is requested for. It will be useful for making any kind of changes in car making process without even touching the classes using *IFactory*.