# Abstract Factory Pattern

Preslav Gerchev 2487136  
Dimitar Vikentiev 2464020

# Introduction

In week 3 we had to develop a car selling shop application using the Abstract Factory Pattern.  
The following document will reflect on the project and on the pattern itself- positive and negative consequences, reusability, maintainability, extensibility.

# The Pattern

The formal definition for the pattern is as following**:**

**“Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”**

# Reusability

In terms of reusability the Abstract Factory Pattern does pretty well since it relies on interfaces and not concrete implementations. It can be easily reused throughout the whole application and helps for removing the *if-else* statements that might arise in the code. The IAbstractFactory interface can be reused as a template to create more and more factories and the only downside is if it has a long list of products- their implementation could take a while to code.

# Extensibility

In terms of extensibility the Abstract Factory pattern has both positives and negatives.

It allows for adding new factories as long as they implement the IAbstractCarPartsFactory interface. This is a big plus since it also provides consistency across all factories-we know for sure that they will produce similar products.

Working with interfaces also helps for fulfilling the Dependency Inversion Principle (something we should always strive for) – the client that is consuming the factory has no idea what kind of product he has – he is only aware that it is of a certain interface (an example would be if we had an ICar interface – the client doesn’t know if he’ll get a BMW or an Audi- he only knows that it will be of ICar type).

However if we wanted to add a new product to the IAbstractCarPartsFactory interface that would mean that we would have to implement it in *every* factory we have ever created- this might not be a problem if we have 2 or 3 factories but it is certainly a big deal if we have say 100 different factories.

# Maintainability

The Abstract Factory Pattern has both its positives and negatives when it comes to maintaining.

On one side the higher the abstraction is the easier it is to maintain – since we do not rely on anything concrete that means that the generic interface we use can’t change accidentally if we change something in its concrete implementations.

On the other hand if we were to change some of the generic interface that we work with (say the ICar interface example that was aforementioned gets a new extra method) that would mean that we would have to change every single implementation of it –which could be 1 or 100 classes.

# Extra

Because the Abstract Factory encapsulates within itself the task of creating classes the client is isolated from that process. The way clients work with the classes is through their generic interface that is exposed via the abstract factory.

The pattern also helps for interchanging the families of products in an easy and simple way –the factory creates the whole family at once – so all we have to do to change the family of products is to simply change the factory we’re using.

# Implementation

We have decided to alter the implementation a bit and rather than providing Cars from a factory we have decided to provide the parts for cars from the factory. We have a Car class that plays the role of the **client** and a constructor that takes an IAbstractCarPartsFactory object and from there all the different parts of the factory are created (in our case – engine, brand, speed, tyres and car color).

# What could be done

What we observed was that if we were to combine the Abstract Factory and the Strategy Pattern we would get the possibility of changing the car’s factory at runtime. If we had a property of the IAbstractCarPartsFactory type in our Car class all we’d had to do is do something like this:

*myCar.CarPartsFactory=new BMWCarPartsFactory();*

That would change the car’s factory and from there change the entire car’s structure (engine, speed, brand, color and tyres) since we would be delegating the responsibility of creating those to our factory.

# Summary

The abstract factory pattern finds its usage in applications where we have to create families of similar products. We have to be careful while implementing it because it could be difficult to maintain or extend during later review of the application.