Design Patterns

1. Creational Patterns

Creational Patterns

- They are all about about class instantiation
- They are:
  - Singleton
  - Simple Factory
  - Factory Method
  - Abstract Factory
  - Builder
  - Prototype

Singleton

A particular class should have only one instance. We will use only that instance whenever we are in need.

Singleton - Practice

- Can you think of some real world examples?
- Did you encounter any examples in the code you've seen so far?

Design a logger class and use this pattern.

Use it:

- when you need to manage a shared resource
- when information flows only one way (don't use it to control business logic)
- retrieve and store information on external files

Simple Factory

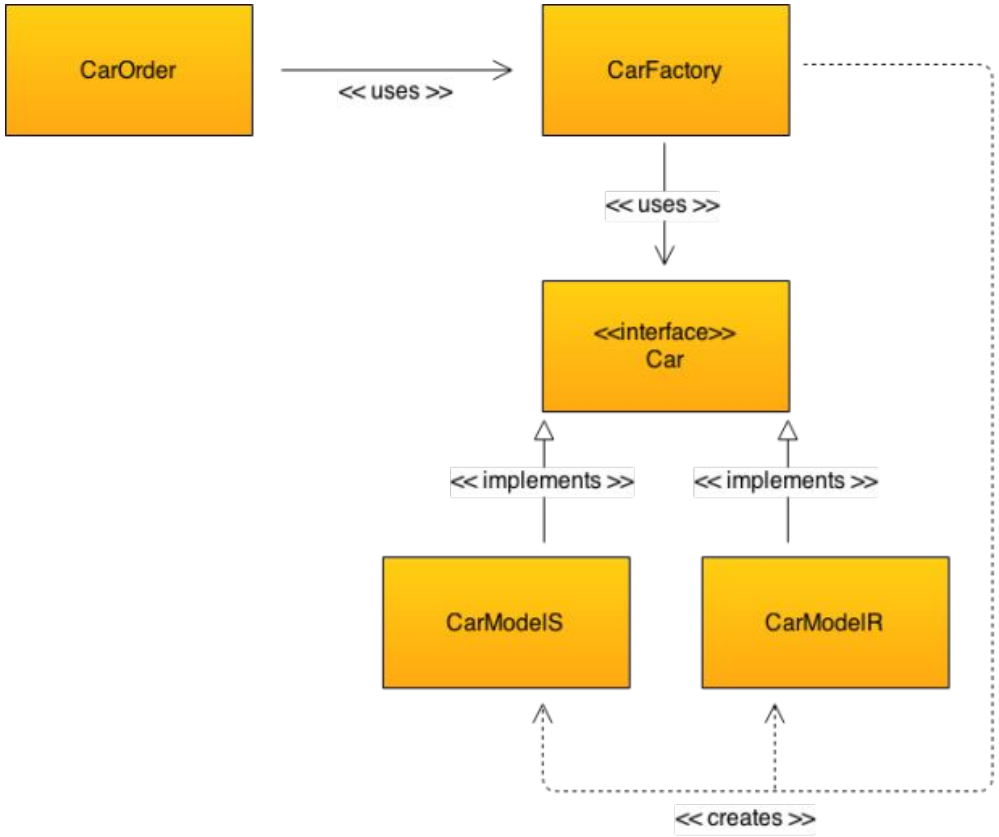
Generates an instance for a client without exposing any instantiation logic to the client.

Simple Factory - Practice

Suppose that you are required to develop a system that accepts orders for cars. The logos are presented in the image.

Draw a class diagram for the system.

Simple Factory - UML



Factory Method

Define an interface for creating an object, but let the subclasses decide which class to instantiate.

Factory Method

Use the Factory method if the business requirements are more than just product creation.

To be precise, if you want to control product creation steps and want to control every step and those steps are customized the Simple Factory pattern is not enough.

Factory Method - Practice

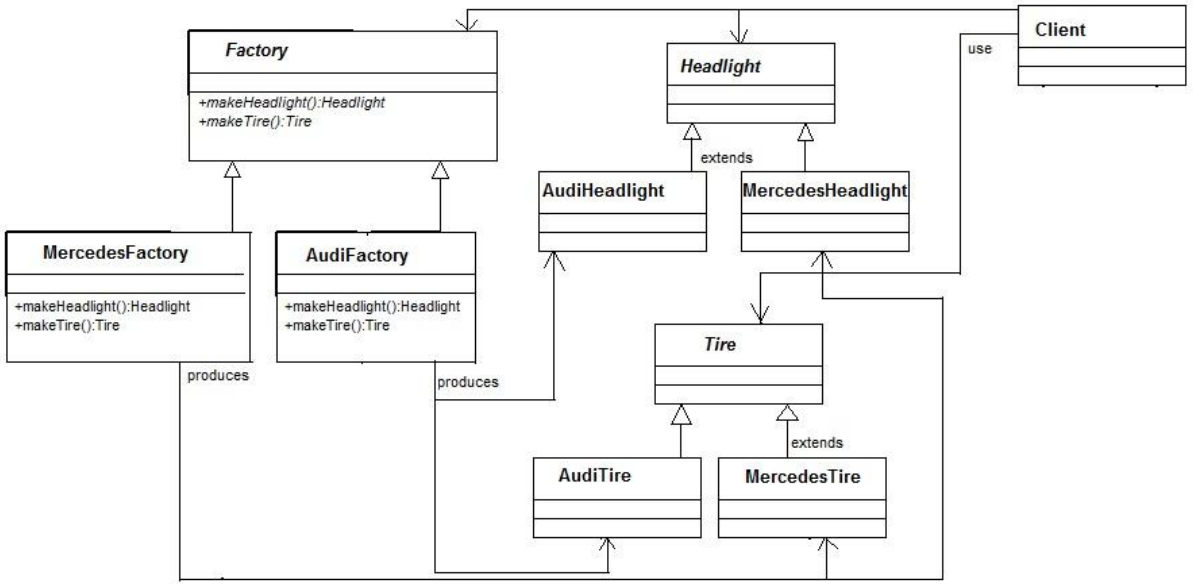
As you know there are different types of car a client might request. Some might want a bigger engine, electrical windows, a particular fuel, etc. This might result in an extra cost.

You need to extend the system for this.

Abstract Factory

Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Abstract Factory - UML



Builder

Is used when the creation of an object is a multi step process. Helps in the process of creating complex objects.

Builder - What do we want?

```
Car
        .Builder
        .setEngineCapacity("1.6")
        .setColor("red")
        .setFuelType("gasoline")
        .isHybrid(true)
        .build();
```

How do we do it?

Builder

Let’s analyse what we did…

- We added a **builder class** that contains all of the fields that exist on the class itself
- We added **methods** that help us build parts of the object
- We have a **build** method that knows how to use our plan to build a new object

Prototype

Create object based on an existing object through cloning. Allows you to create a copy of an existing object and modify it to your needs, instead of going through the trouble of creating an object from scratch and setting it up.

Prototype - how to?

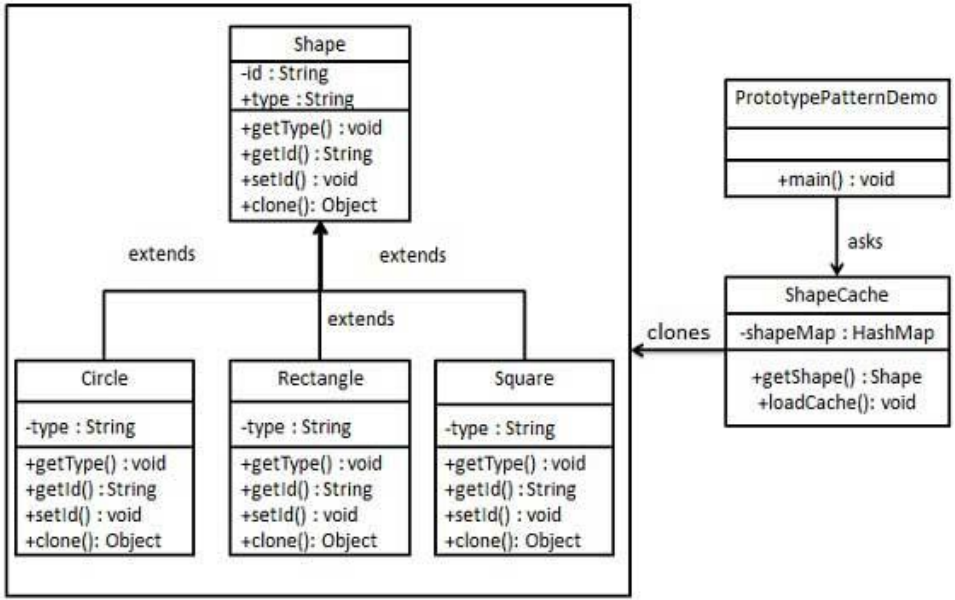
The **classes participating** to the Prototype Pattern are:

- *Client* - creates a new object by asking a prototype to clone itself
- *Prototype* - declares an interface for cloning itself
- *ConcretePrototype* - implements the operation for cloning itself

Process:

1. Create a class to get concrete classes and store them in a hashtable/map
2. The Client asks for a new object of that class and sends the request. Based on the map we stored the class will know how to handle the cloning through the **clone()** method, making a new instance of the concrete class wanted

Prototype in action



Prototype - Practice

Remember the CarFactory?

Try to implement using this pattern.

Thank you

For any questions you can contact me here: **tincu.alecsandra@gmail.com**