**Design Patterns**

**Introduction:**

Design patterns are generalized solutions to repeated occurring problems in Software Design. We should think of design patterns not as a finished solution but a template to solve a more general problem. There are three main category of design patterns, Creational, Structural and Behavioral design patterns. Each of these categories has a set of specific design patterns that we will examine.

**Creational Design Patterns:**

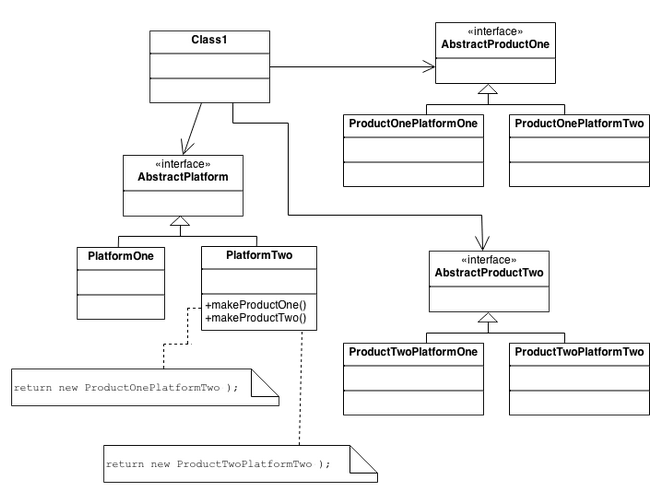
Creational patterns refer to designs that are specific to class instantiation and object creation. In this section we will take a look at the following creational design patterns:

* Abstract Factory
* Builder
* Factory Method
* Object Pool
* Prototype
* Singleton

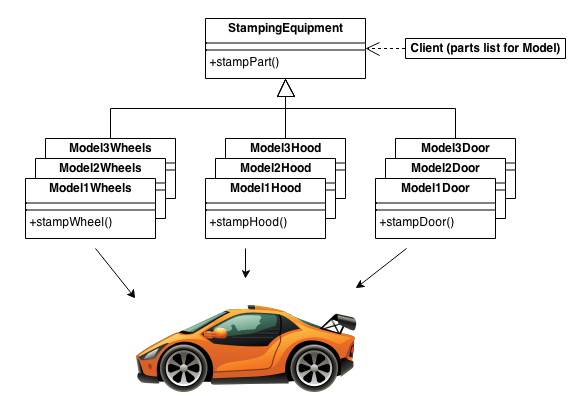
**Abstract Factory:**

The abstract factory design pattern is a pattern intended to provide an interface for creating related objects, without defining their concrete classes. It is typically good to use this design pattern when we are concerned about *portability*, e.g. across *different platform dependencies*, and encapsulating abstracted families of objects.

The abstract factory has factory methods used to create a concrete object part of the family of objects. Figure 1, shows an example of the general structure of an abstract factory design. A more specific example would be of a manufacture that is sheet metal stamping different parts of an automobile. For example, the wheels, the hood, the doors, etc. for different models of cars, which have different layouts of these general parts. We can see the design pattern visually in figure 2.



**Figure 1: An Example of a General Abstract Factory**



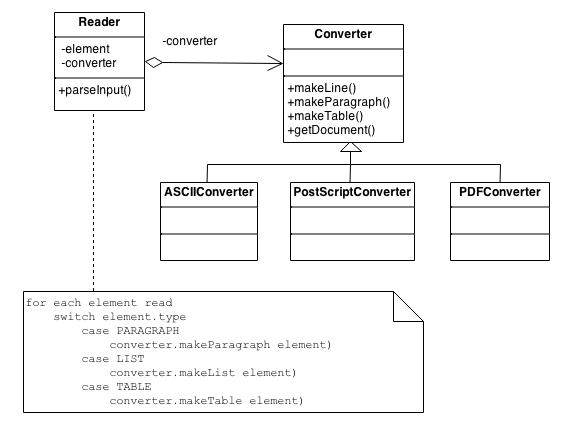
**Figure 2: A Specific Use-Case of an Abstract Factory Design Pattern**

**Builder:**

The builder design pattern is a creational pattern intended to separate the representation of an object from the construction of the object. This is so the pattern can use the same construction process for different representations of the object.

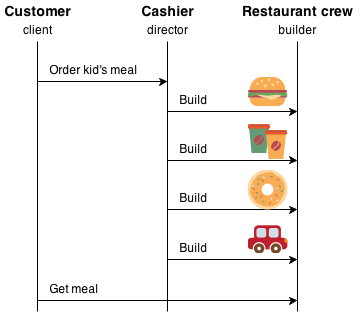
The “Director” calls the services of the “builder” to create parts of the complex object piece by piece. When the “director” is finished the client can retrieve the results of the complete object from the “builder”. This pattern allows for finer control over the construction process, which is unlike other creational patterns that construct the end product in on shot.

The structure of the builder design pattern can be seen below in figure 3, where the Reader object encapsulates the parsing of the common input. The Builder allows for many different possible polymorphic creations of many peculiar representations of targets.



**Figure 3: Structure of Builder Design Pattern**

An example that can highlight the usefulness of this pattern is, a fast-food restaurant that constructs different children’s meals. The children’s meals typically have a main item, side item, a drink, and a toy. Where we have a number of different main items, side items, drinks and toys. This example highlights the point that the creation process of the children’s meal is the same, however the contents in each children’s meal can vary significantly. Figure 4 displays a visual representation of this design pattern.



**Figure 4: Example of the Builder Design Pattern**

**Factory Method:**

**Structural Design Patterns:**

Structural patterns focus on object composition and interface design. In this section we will take a look at the following structural design patterns:

* Adapter
* Bridge
* Composite
* Decorator
* Façade
* Flyweight
* Private Class Data
* Proxy

**Behavioral Design Patterns:**

Behavior patterns focus mainly on the intercommunication between different objects. In this section we will take a look at the following behavioral design patterns:

* Chain of Responsibility
* Command
* Interpreter
* Iterator
* Mediator
* Memento
* Null Object
* Observer
* State
* Strategy
* Template Method
* Visitor

**References:**

1. <https://sourcemaking.com/design_patterns>
2. Design Patterns (Gang of four: Gamma, Helm, Johnson, Vlissides)
3. Data Structures and Problem Solving Using C++, 2nd-ed, Mark Weiss