**Creational Patterns**

# Creational Patterns

Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

Problem with basic object creation:

* could result in design problems or
* added complexity to the design

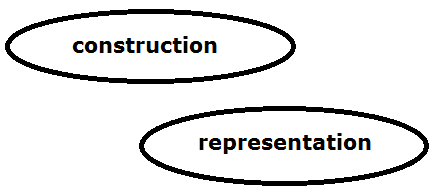
# Abstract Factory

Creates an instance of several families of classes

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

# Builder

Builder Creational Pattern is used to separate the construction of a complex object from its representation so that the same construction process can create different objects representations.



Builder Creational Pattern

## Problem:

We want to construct a complex object, however we do not want to have a complex constructor member or one that would need many arguments.

## Solution:

Define an intermediate object whose member functions define the desired object part by part before the object is available to the client. Builder Pattern lets us defer the construction of the object until all the options for creation have been specified.

Builder design pattern describes how to solve such problems:

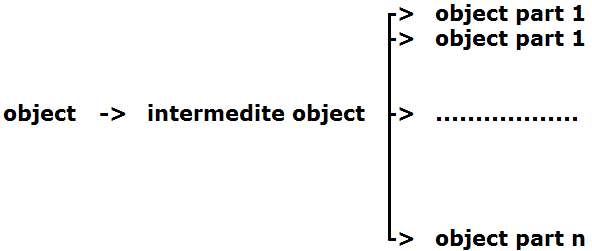
* Encapsulate creating and assembling the parts of a complex object in a separate Builder object.
* A class delegates object creation to a Builder object instead of creating the objects directly.

## Advantages

* Allows you to vary a product’s internal representation
* Encapsulates code for construction and representation
* Provides control over steps of construction process

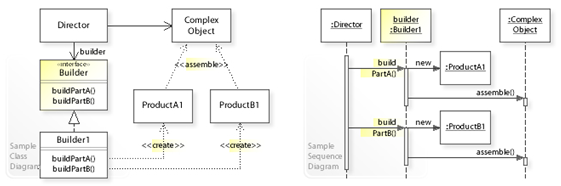
## Disadvantages

* Requires creating a separate ConcreteBuilder for each different type of product
* Requires the builder classes to be mutable
* Data members of class aren't guaranteed to be initialized
* Dependency injection may be less supported

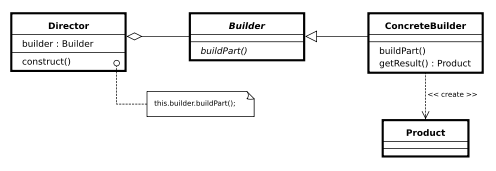


Builder Creational Patterns

## Structure



UML class and sequence diagram for the Builder design pattern



Class diagram

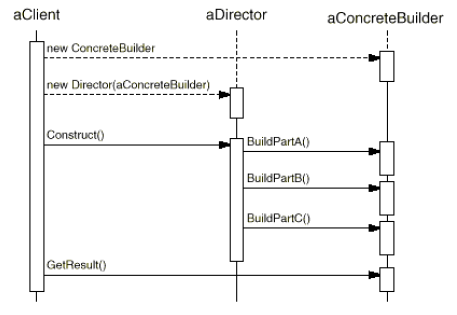
**Builder:** Abstract interface for creating objects (product)

**ConcreteBuilder:** Provides implementation for Builder. It is an object able to construct other objects. Constructs and assembles parts to build the objects

**Director:** constructs an object using the Builder interface

**Product:** represents the complex object under construction. ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled

Includes classes that define the constituent parts, including interfaces for assembling the parts into the final result

****

Interaction Diagram

## Example:



# Factory

A utility class that creates an instance of a class from a family of derived classes.

Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

The factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.

This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

**Also known as Virtual Constructor**

# Prototype

# Singleton

# References

<https://sourcemaking.com/design_patterns/creational_patterns>

<https://en.wikipedia.org/wiki/Software_design_pattern>

# END