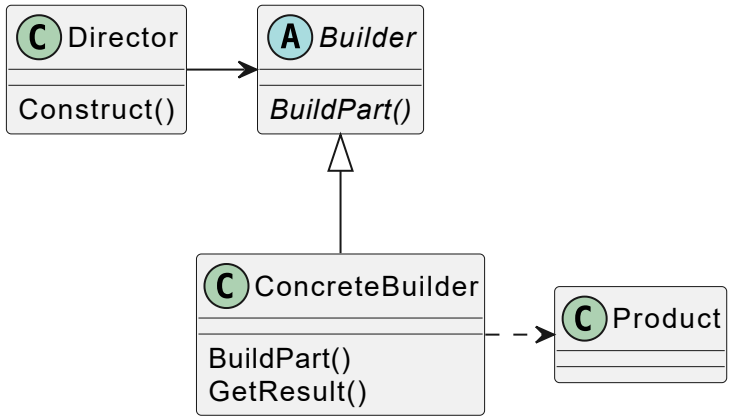
Budowniczy (Builder)

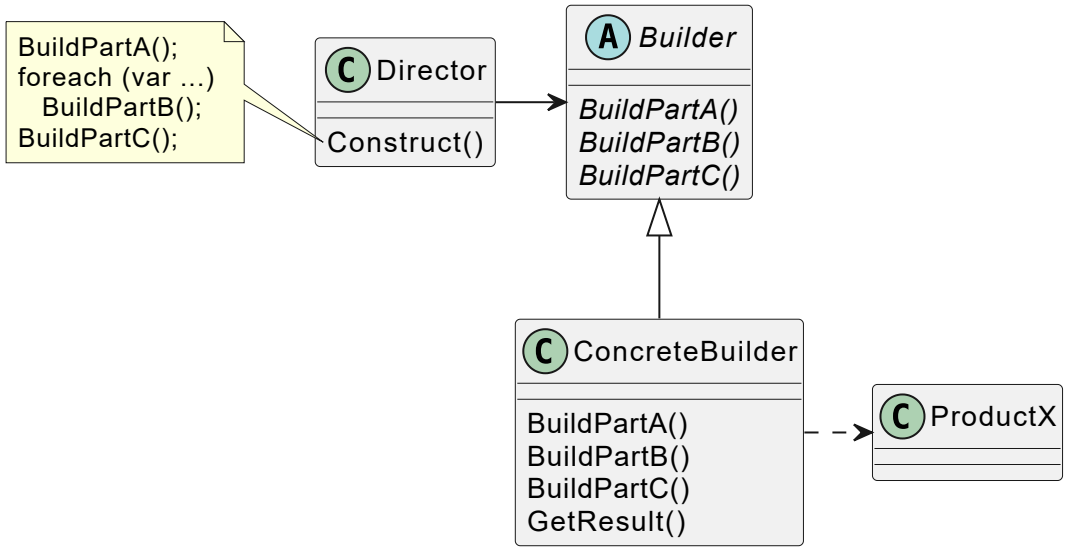
Builder

- Separate the construction of a complex object from its representation so that the same construction process can create different representations



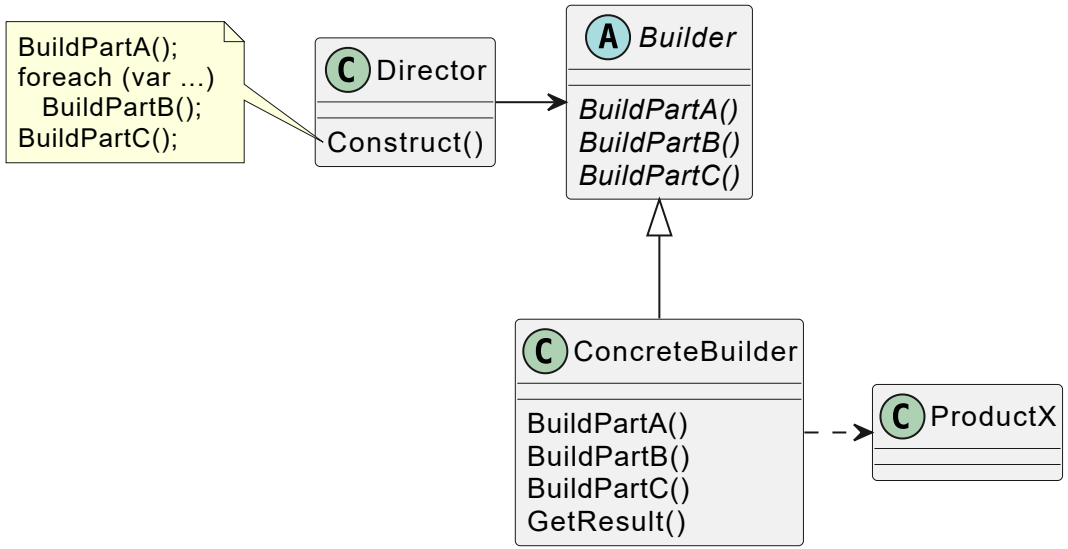
Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations



Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations



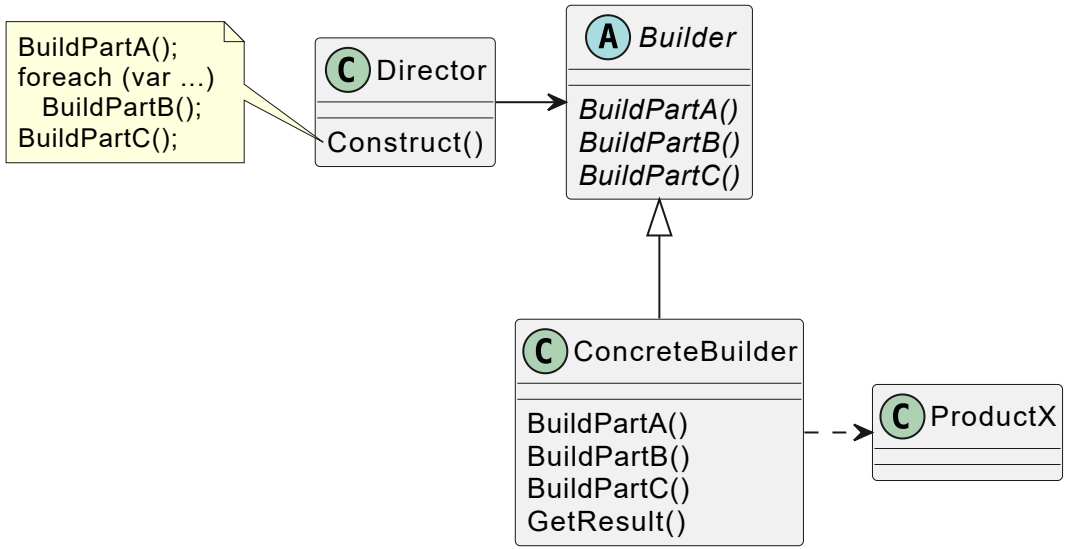

```
var builder = new ConcreteBuilder();
var director = new Director(builder);
director.Construct();
ProductX product = builder.GetResult();
```

Degraded (modern) approach inlines **`Director`**:

```
var builder = new ConcreteBuilder();
builder.BuildPartA();
foreach (var ...)
   builder.BuildPartB();
builder.BuildPartC();
ProductX product = builder.GetResult();
```

Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations




```
var builder = new ConcreteBuilder();
var director = new Director(builder);
director.Construct();
ProductX product = builder.GetResult();
```

Degraded (modern) approach inlines **`Director`**:

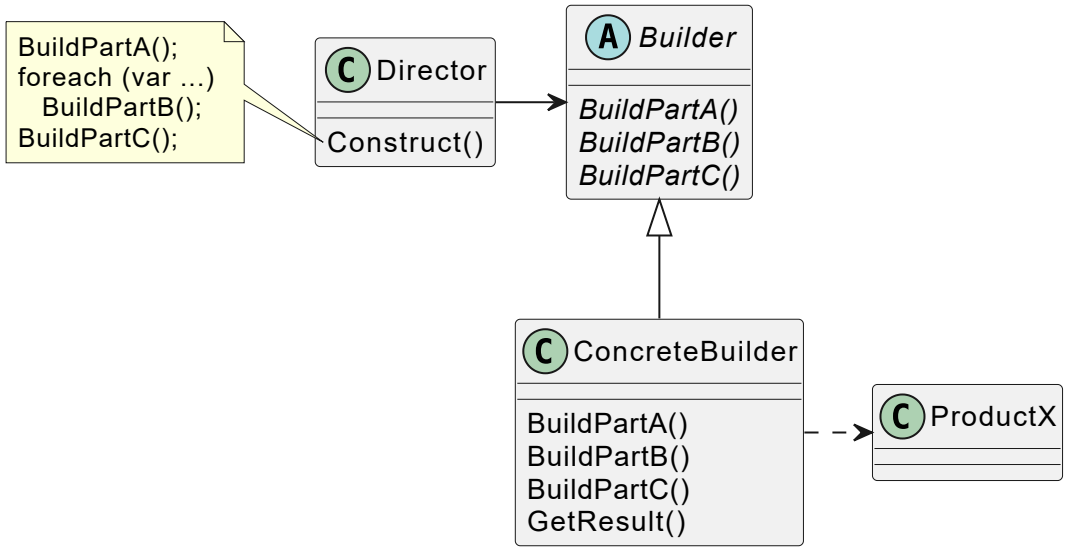
```
var builder = new ConcreteBuilder();
builder.BuildPartA();
foreach (var ...)
   builder.BuildPartB();
builder.BuildPartC();
ProductX product = builder.GetResult();
```

Hmm...🤔 Two aspects:

- gives the fine grained control over the construction process - **`Construct`** may vary, call some **`Build...`** methods or not etc. Step by step.
- concrete product will be created - note there is no *product abstration* (products do not have to belond to same class hierarchy)

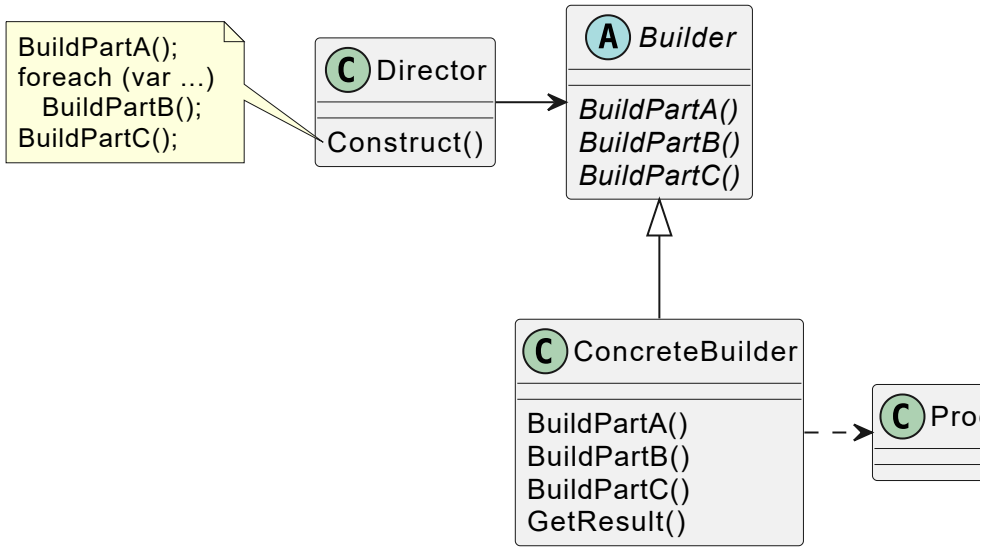
Builder

Yes, *Director* "construct" method is kind of
Template Method for specifying Builder's steps

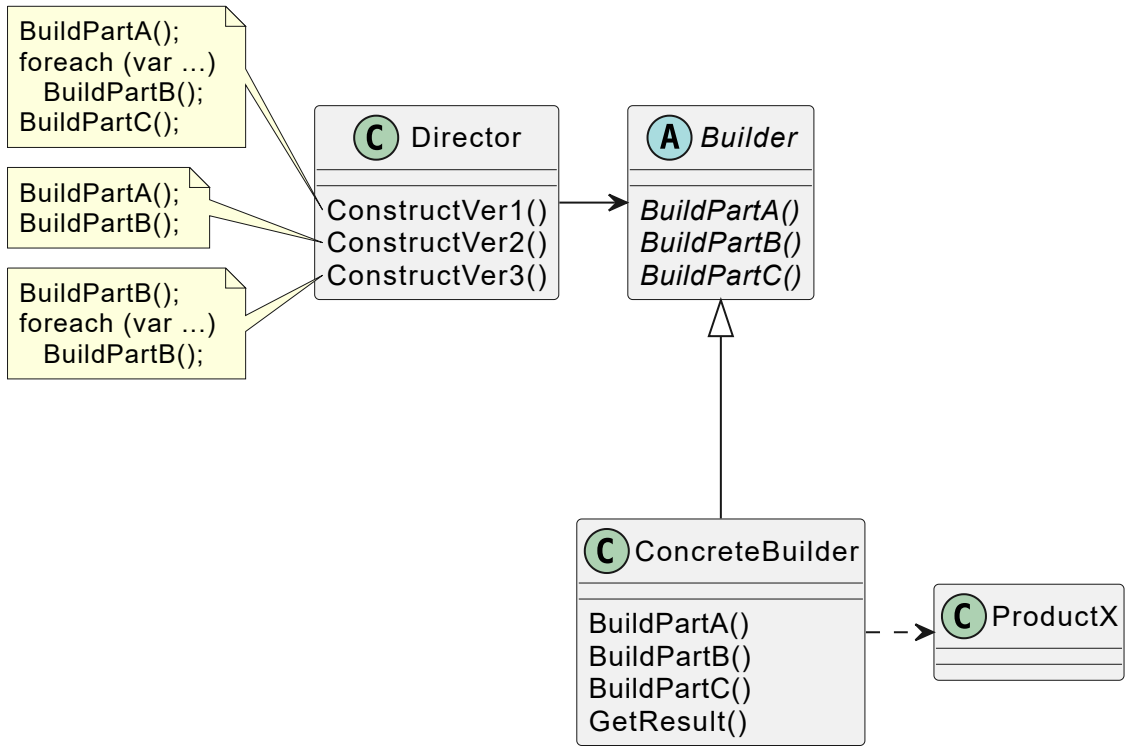


Builder

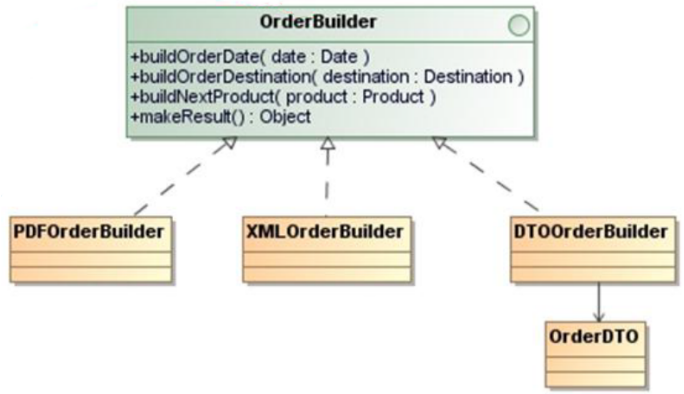
Yes, *Director* "construct" method is kind of *Template Method* for specifying Builder's steps



Multiple "constructs"s in *Director* makes it very similar to *Abstract Factory* but here we have a "build-steps" approach promoted

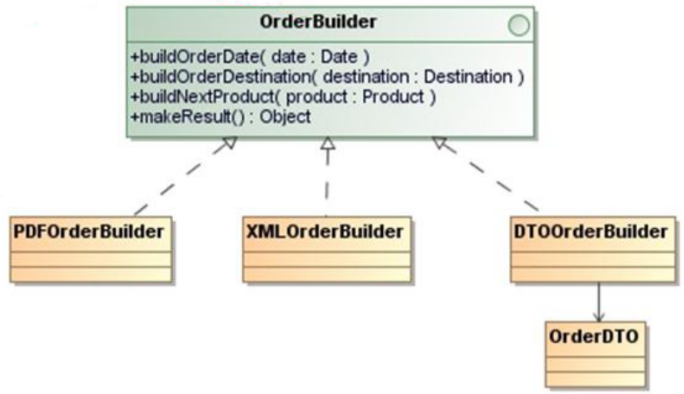


Builder - example




```
public class Order
{
   private DateTime _date;
   private List<Product> _products;
   private Destination _destination;
   object Export(OrderBuilder builder)
   {
      builder.BuildOrderDate(_date);
      builder.BuildOrderDestination(_destination);
      foreach (var product in _products)
      {
         builder.BuildNextProduct(product);
      }
      return builder.GetResult();
   }
}
```

Builder - example




```
public class Order
{
   private DateTime _date;
   private List<Product> _products;
   private Destination _destination;
   object Export(OrderBuilder builder)
   {
      builder.BuildOrderDate(_date);
      builder.BuildOrderDestination(_destination);
      foreach (var product in _products)
      {
         builder.BuildNextProduct(product);
      }
      return builder.GetResult();
   }
}
```

- *Builder* class interface must be general enough to allow the construction of products for all kinds of concrete builders

Builder

- lets you construct complex objects step by step
  - instead of, for example, monstrous constructor with lots of parameters (including many boolean flags)
- often leads to Fluent APIs for building
- may be nice to create *Composite* instances
- we can combine it with *Bridge* where Builder is an implementation, and *Director* is an Abstraction (so we have various concrete builders **and** directors)