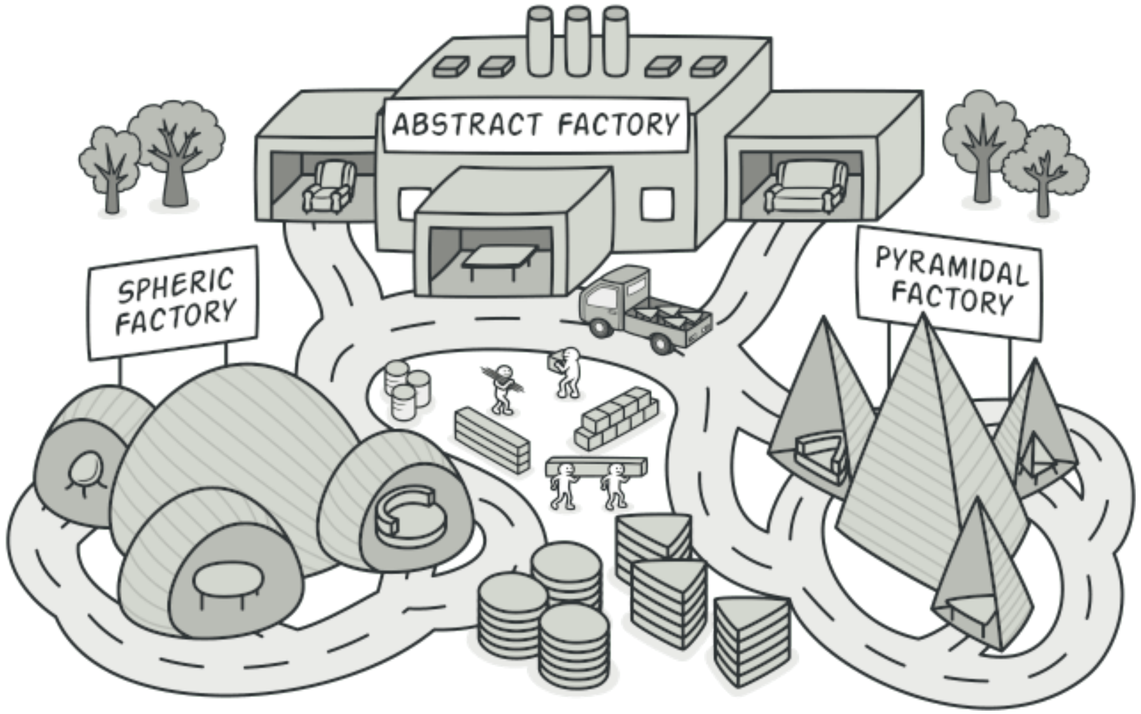
Information System Design

Lecture 6: Creational Design Patterns (Contd.)

Dr. Moustafa Alzantot

Abstract Factory

Abstract factory is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.



Factory

First, what's *a factory* ?

An object whose responsibilities are creating one or more objects for clients.

Why ?!!

The goal is to separate knowledge of '***how to use***' from knowledge of '***how to create***'.

- Clients should only only first part (how to use)
- Factory knows (how to create)

Abstract Factory

Factory example:

```
WidgetFactory* factory = new WidgetFactory()
Window w = factory->createWindow(..);
Scrollbar sc = factory->createScrollbar (..);
Button b = factory->createButton(..);
```

Abstract Factory

Applicability:

- Client is independent from the way products are created or represented.
- Products must be classified into families of products.
- Families are exclusive.
- Interface of a product must not depend on the family it belongs to.

Abstract Factory

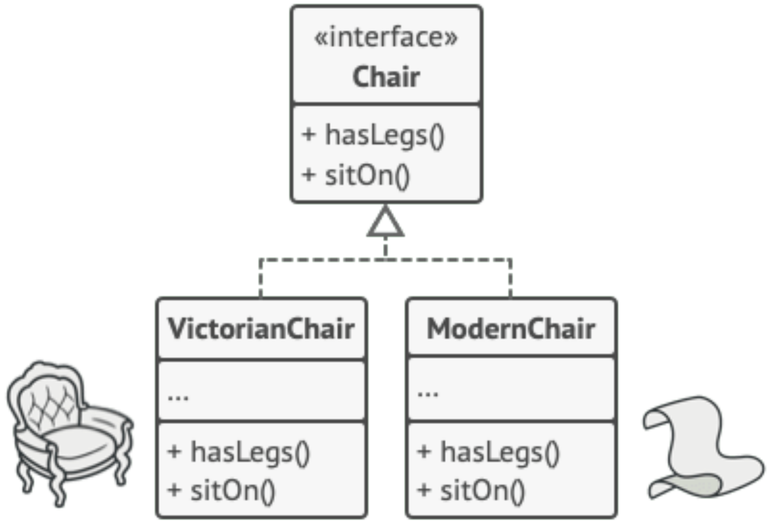
Example use cases:

- Creating GUI components for multiple platforms (e.g. Windows, Linux, Mac)
- Creating XML document components (e.g. Document, Node, Label, Attribute) for different parsers.

Abstract Factory

Solution:

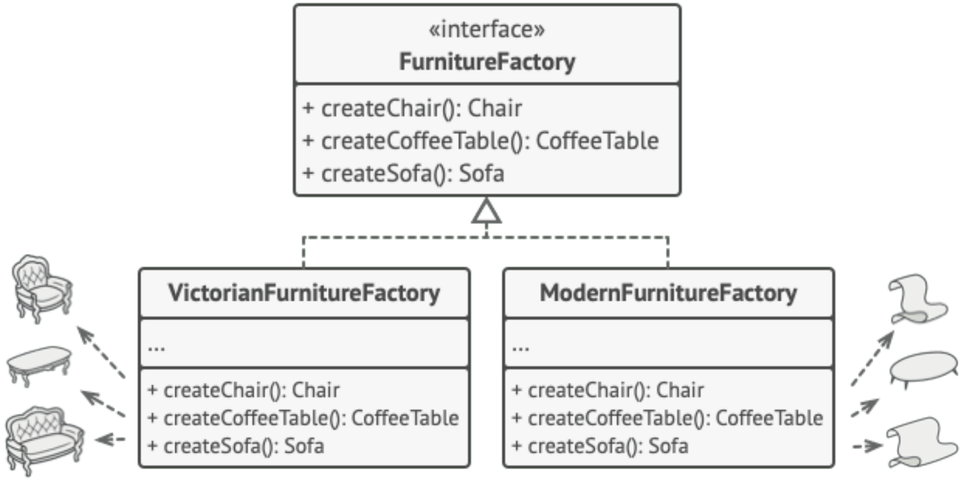
First step: create an interface for each product of the product family.



Abstract Factory

Solution:

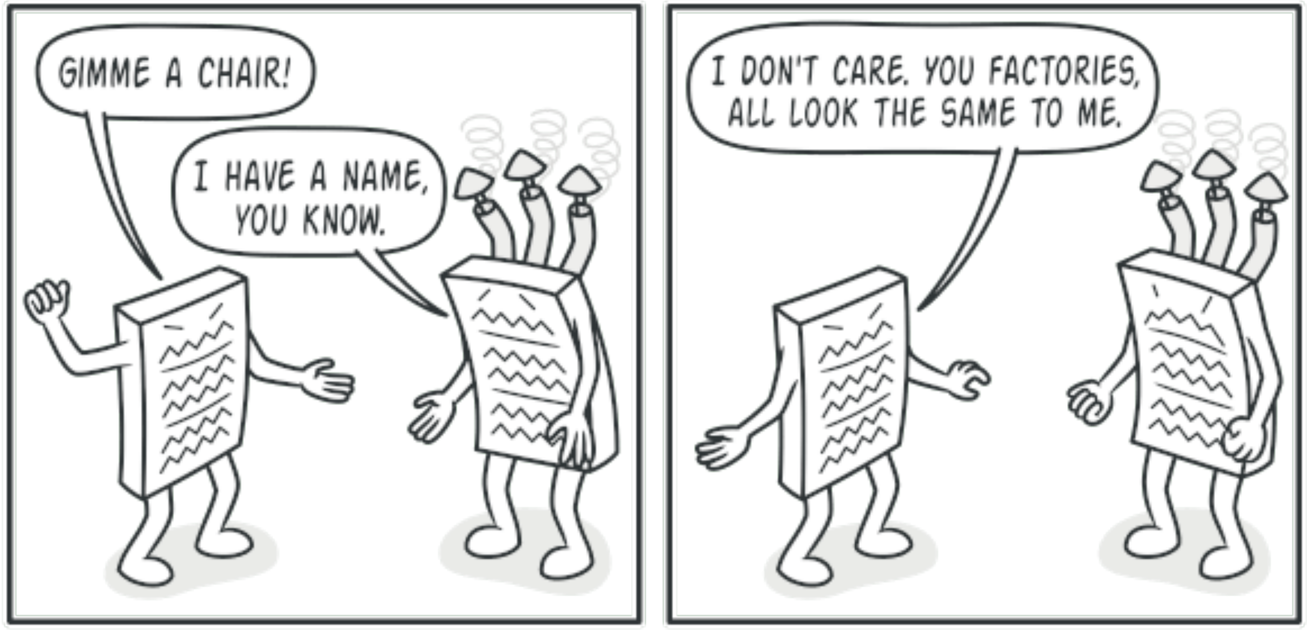
Then: declare an `AbstractFactory` whose interface lists the creation methods for all products that are part of the product family (e.g. `createChair`, `createSofa`, `createCoffeeTable`)



Abstract Factory

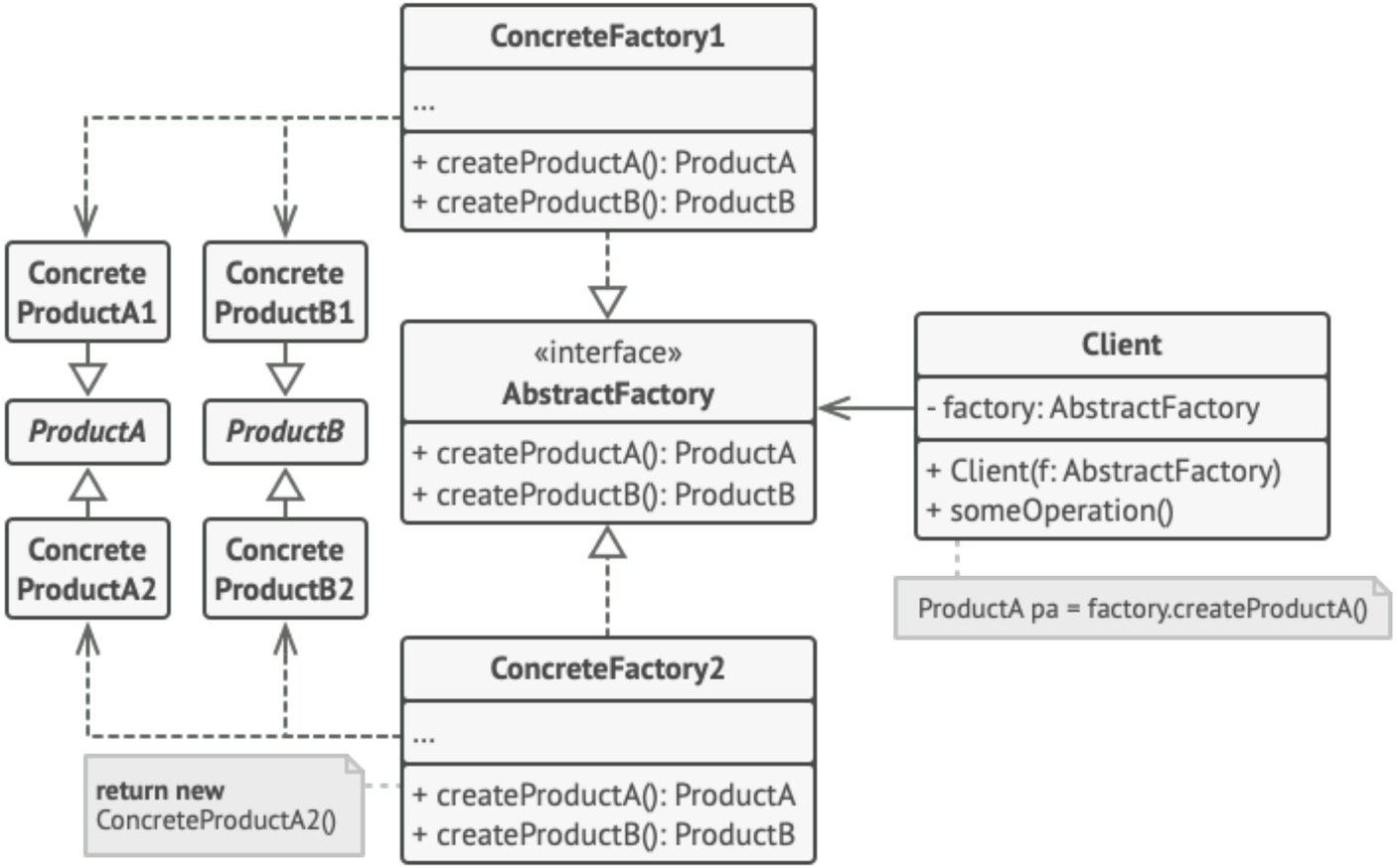
Solution:

Finally: enforce that client deals with both factories and products via their respective abstract interfaces.



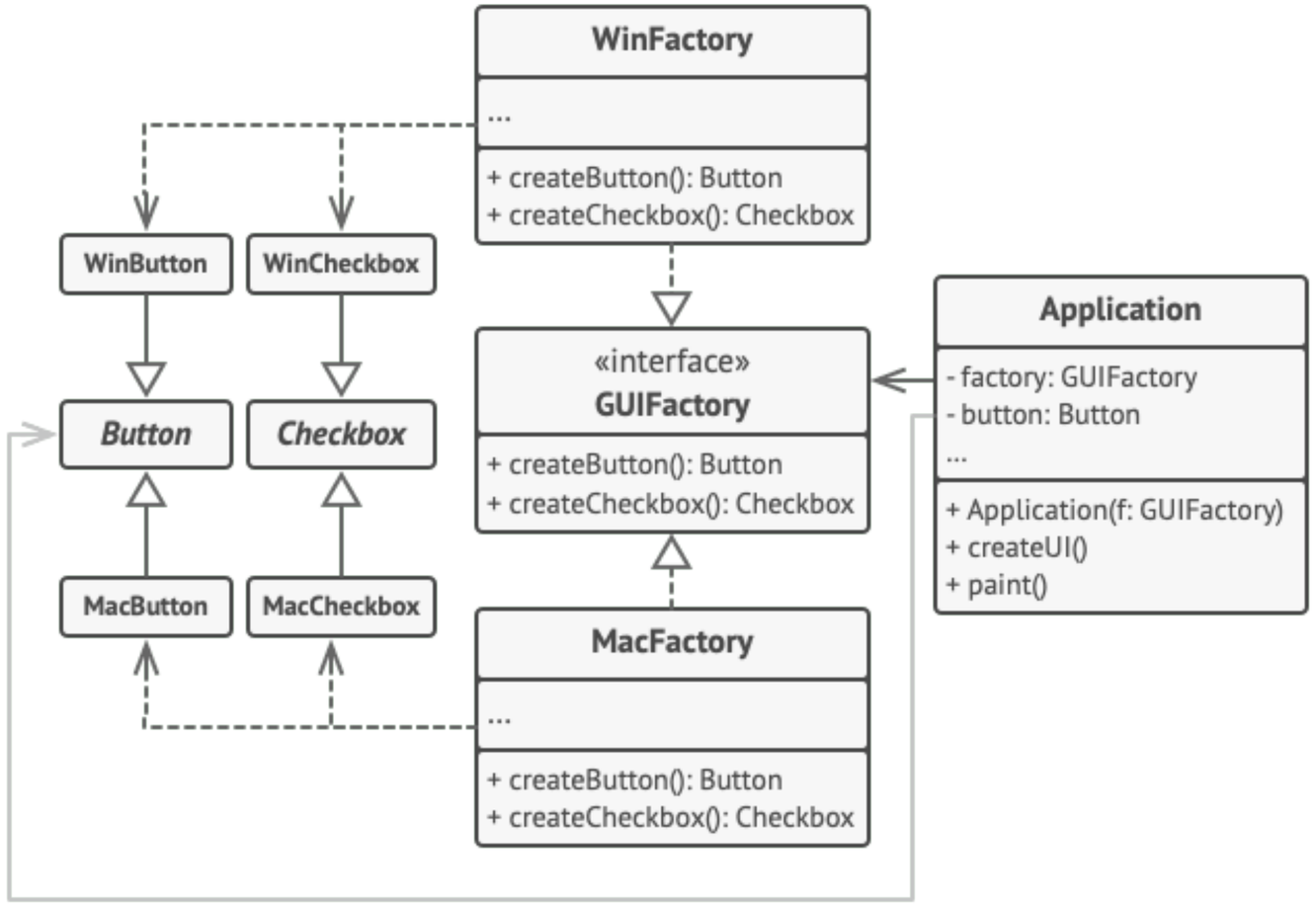
Abstract Factory

Complete Solution



Abstract Factory

Example:



Abstract Factory

Steps to implement:

- Map products into a set of distinct families.
- Declare abstract product interface for each product type. Make all concrete product classes implement these interfaces.
- Declare abstract factory interface with set of creation methods for all abstract products.
- Implement the set of concrete factory classes; one factory class per product family.
- Make client code relies only on abstract products and abstract factory interfaces. It doesn’t use the constructor of concrete products.

Abstract Factory

Advantages:

- Guarantees that products created from the factory are compatible with each other as long as they are generated via the same interface.
- Avoid tight coupling between concrete products and client code.
- Adheres ***Open/Closed principle:*** easy to add a new product family without changing the client code or other products.
- Adheres ***Single Responsibility Principle***: code for product creation is a single place.

Abstract Factory

Disadvantages:

- Hard to add a new product. Requires changing all product families concrete classes.
- Code may be come more complicated than it should be as it requires adding many new interfaces and classes to implement the pattern.

Builder Pattern

Builder patterns aims to “separate the construction of a complex object from its representation so that the same construction process can create different representations”.

It is used to construct a complex object step by step and the final step will return the object.

Builder Pattern

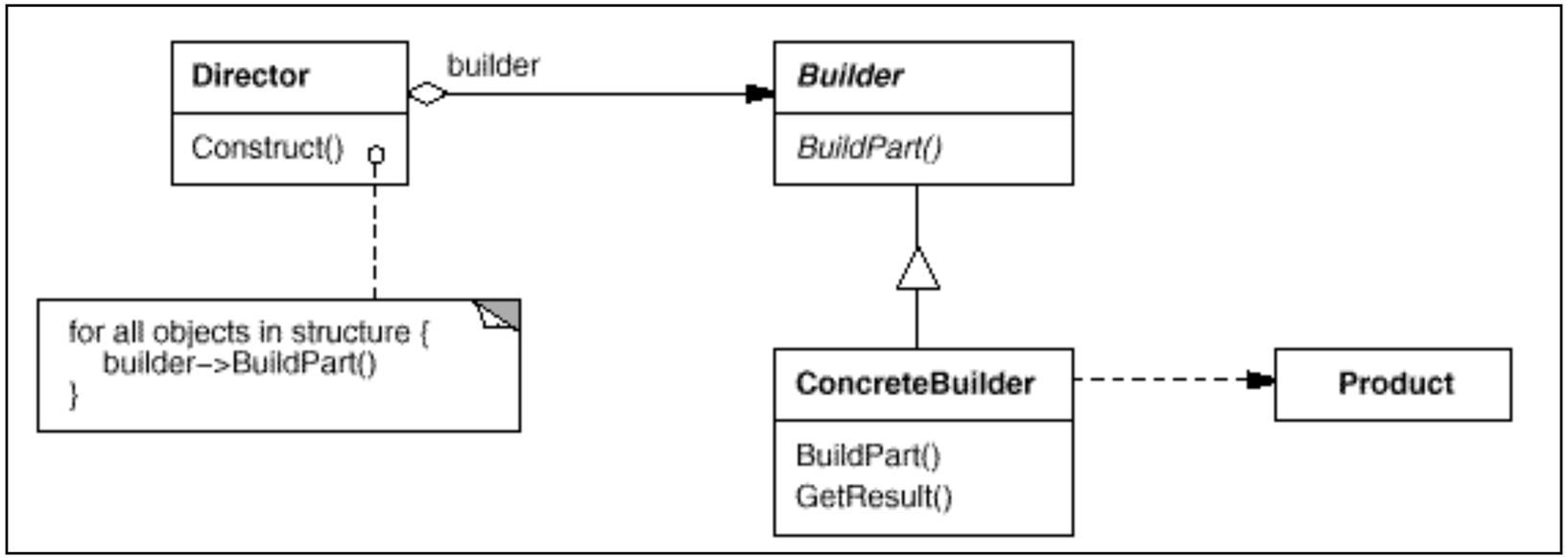
Example: Think of a car factory.

Boss tells workers (or robots) to build each part of a car.

Workers build each part and add them to the car being constructed.

Slide credit: https://www.cse.wustl.edu/~cdgill/courses/cse432_sp06/

Builder Pattern



Builder Pattern

`Builder:`
Abstract interface for creating parts of “**`Product`**” object.

`ConcreteBuilder:`

Constructs and assembles the parts of the product by implementing the builder interface.

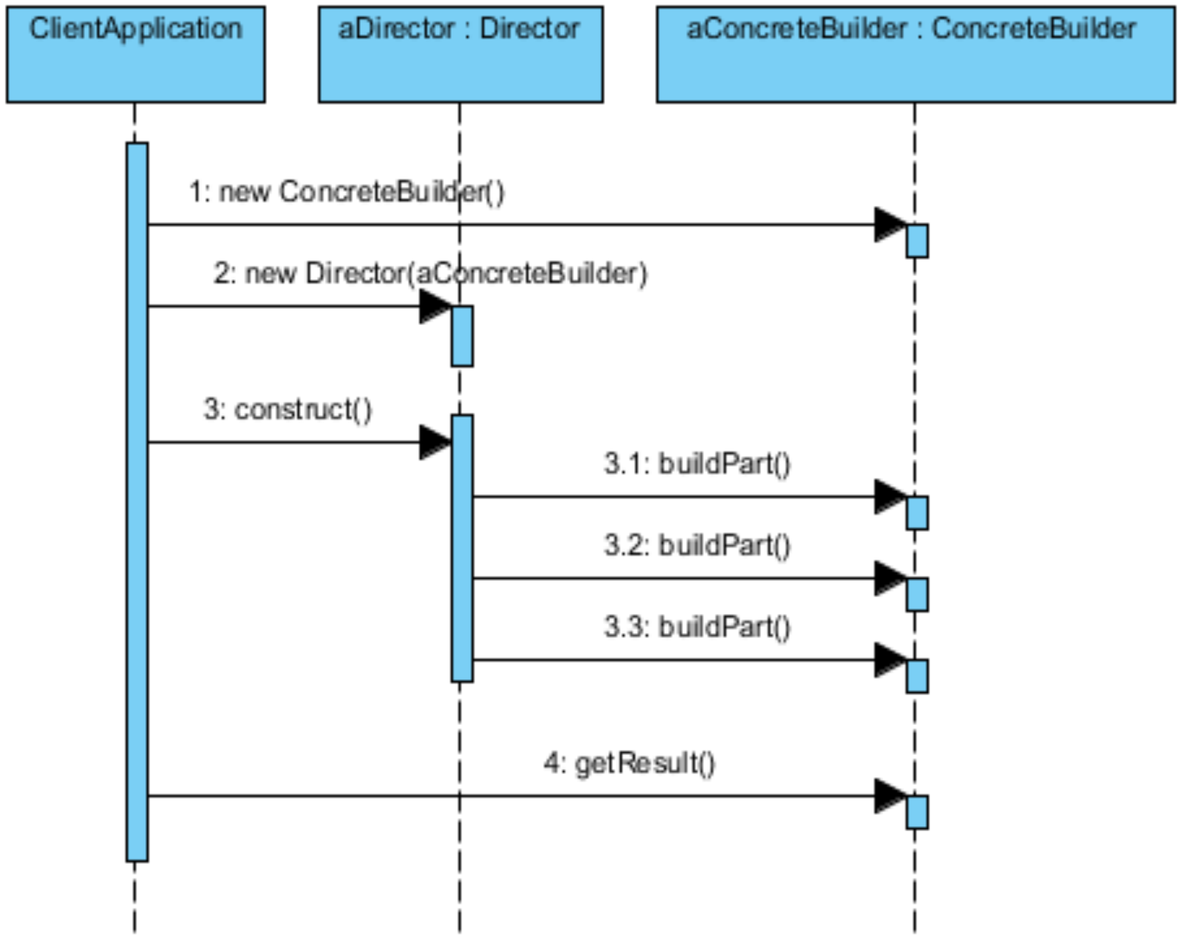
`Director`:

Constructs an object by using the **`Builder`** interface.

`Product`:

Represents the complex object under construction.

Builder Pattern : Interactions



Builder Pattern : Example

- **AerospaceEngineer**: director
- **AirplaneBuilder**: abstract builder
- **Airplane**: product
- Sample concrete builders:
  - CropDuster
  - FighterJet
  - Glider
  - Airliner

Builder Pattern

Advantages:

- You can construct objects step-by-step.
- Suitable for constructing complex objects.
- You can reuse the same construction code while building various representations of the products.
- Adheres to ***Single Responsibility principle***: you can isolate the code for construction from the business logic of the product.

Builder Pattern

Disadvantages:

- Increases code complexity since you need to create multiple classes.