Ch5. 객체지향 디자인

C++ Design Pattern (Basic)

Jong-Hyeok Park
akindo19@gmail.com

Design Pattern C++

- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Design Pattern C++

- **Creational Patterns**
  - Abstract Factory
  - **Builder**
  - Factory Method
  - Prototype
  - Singleton

Design Pattern C++

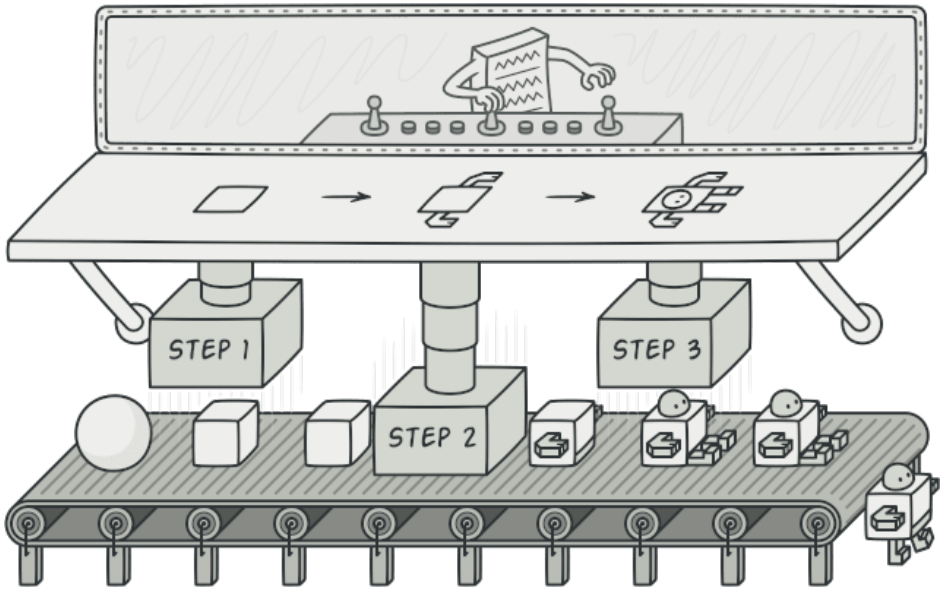
- **Structural Patterns**
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy

Design Pattern C++

- **Behavioral Patterns**
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor
  - Model-View-Controller (MVC)

Builder

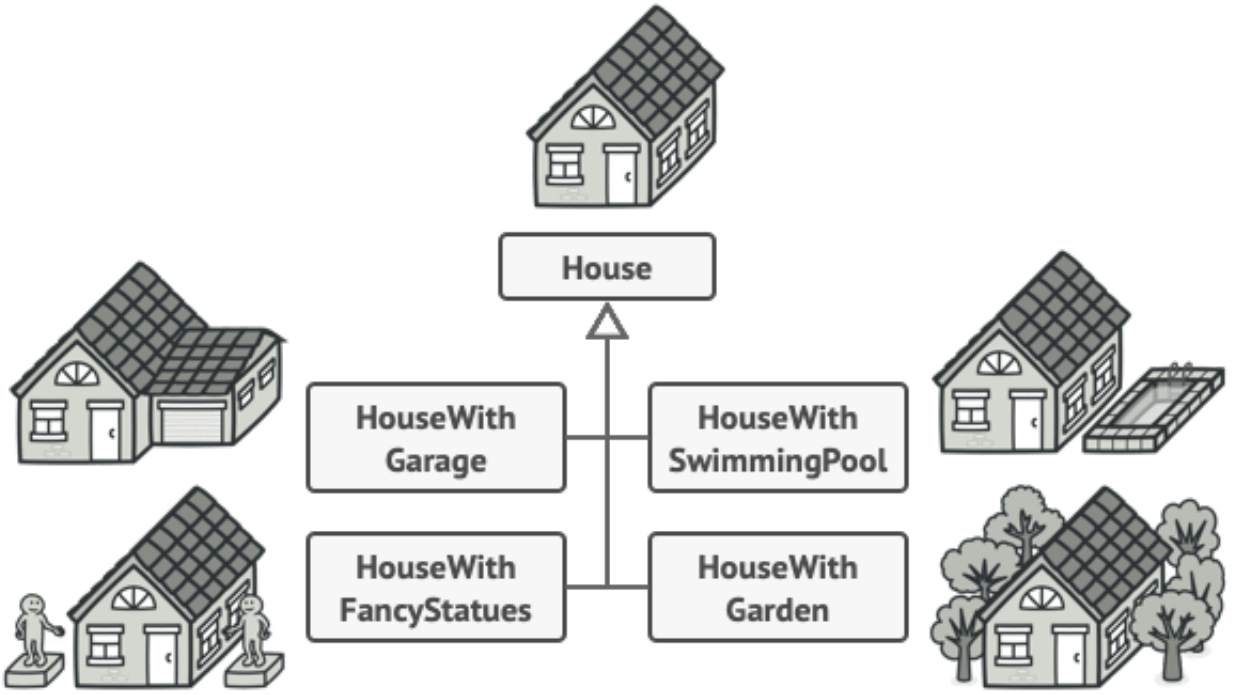
- **Description**
  - Let construct complex objects step by step
  - Allows to produce different types and representations of an object using the **same construction code**.



https://refactoring.guru/design-patterns/builder

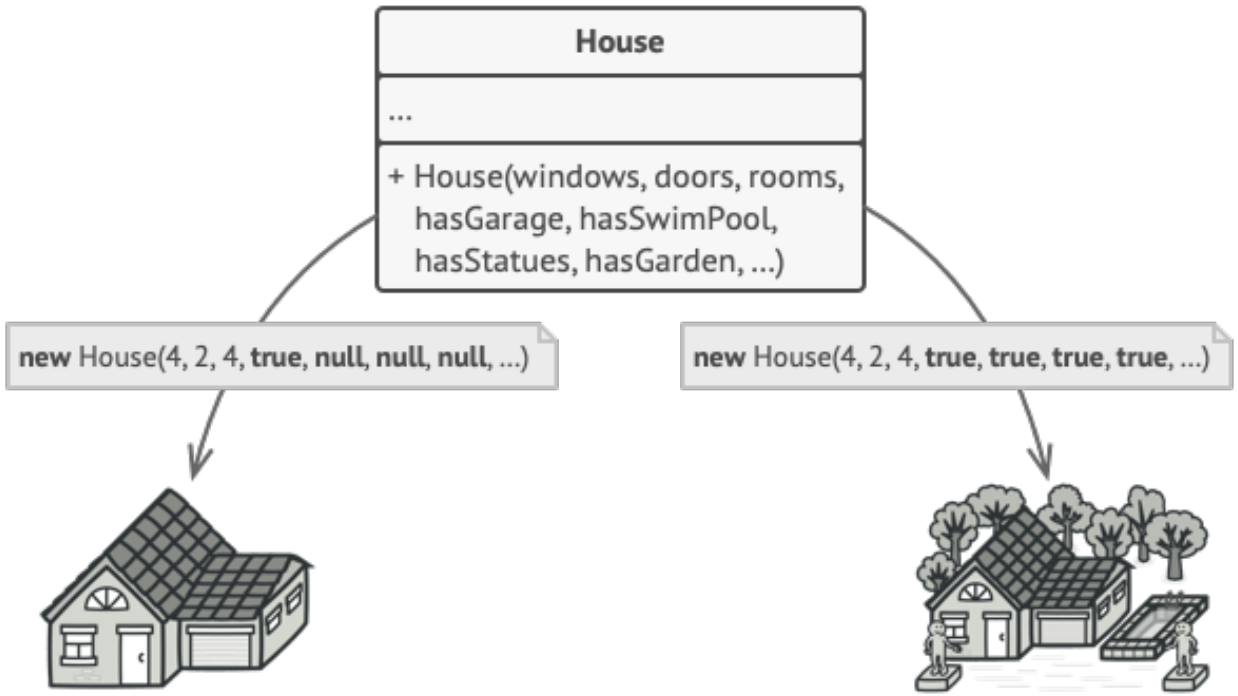
Builder

- **Problem**
  - Design more complex construcor with various properties
  - Simple solution → **subclasss**



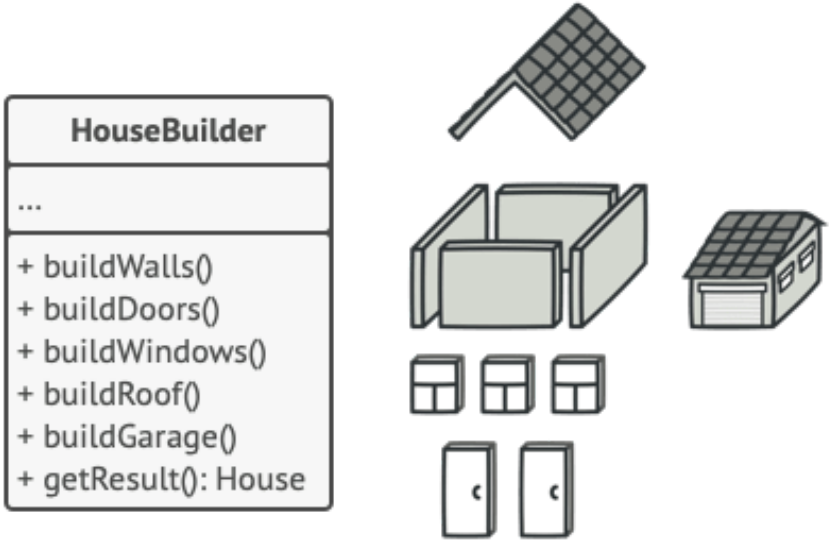
Builder

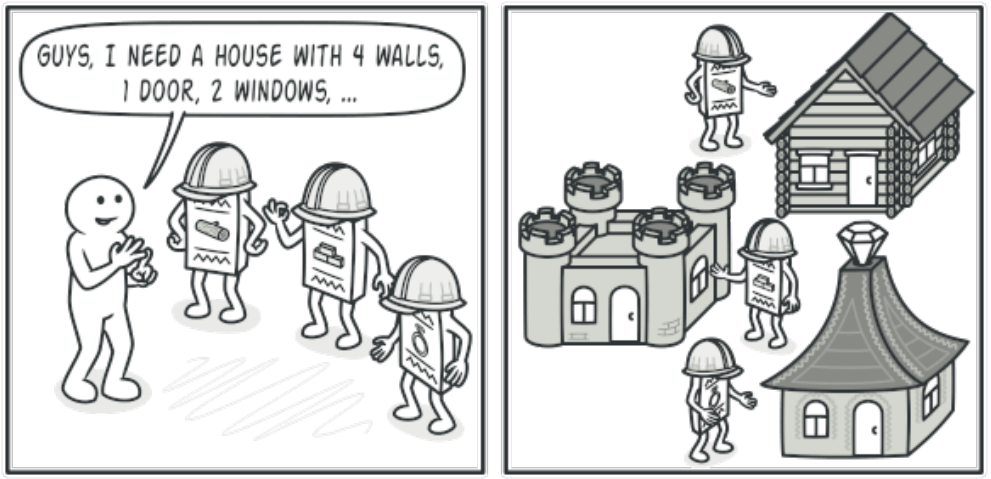
- **Problem**
  - Considerable number of subclasses (eventually).
  - Constructor calls are pretty **ugly**.
  - Most of the parameters will be **unused**,



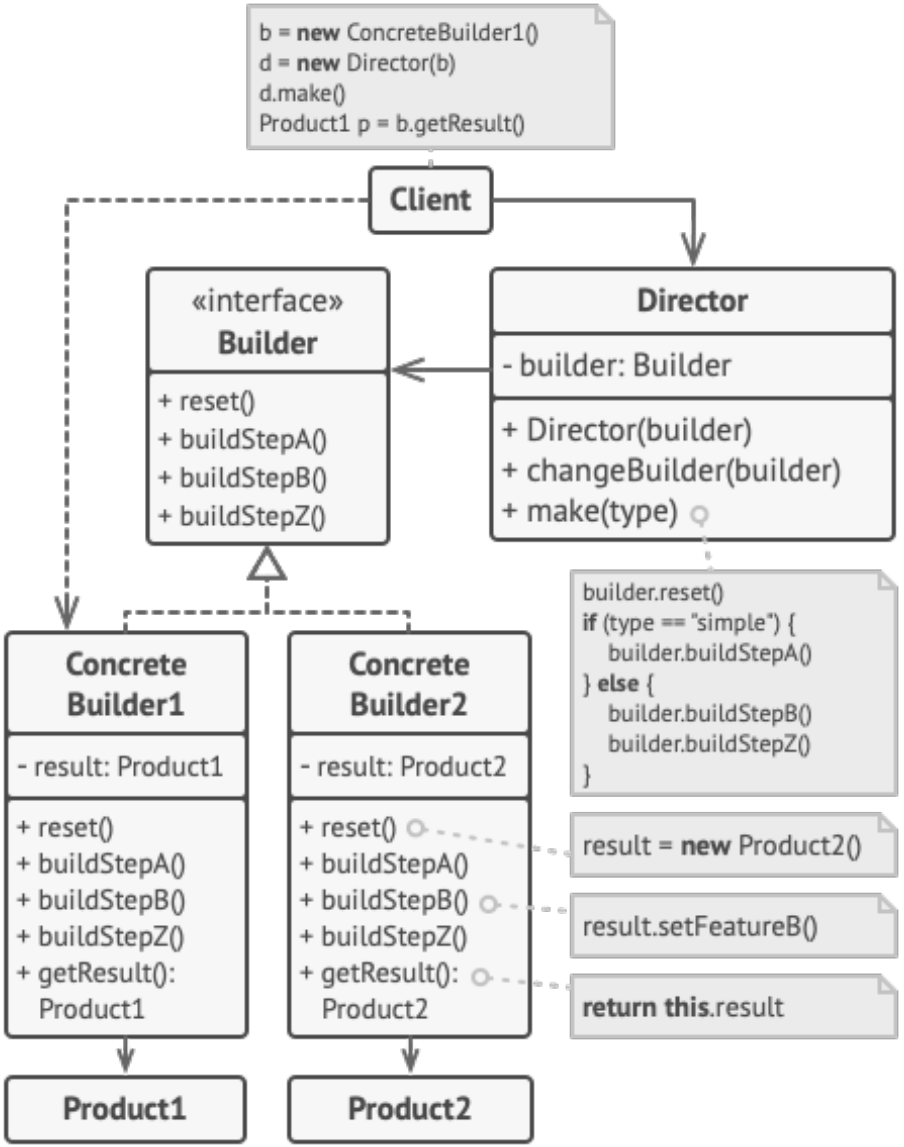
Builder

- **Solution**
  - Extract the object construction code out of its own class and move it to **separate objects** called ***builders***.
  - **Director:** defines the order in which to execute the building steps, while the builder provides the implementation for those steps.





Builder


b = new ConcreteBuilder1()
d = new Director(b)
d.make()
Product1 p = b.getResult()
Client
«interface»
Builder
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
Director
- builder: Builder
+ Director(builder)
+ changeBuilder(builder)
+ make(type)
builder.reset()
if (type == "simple") {
builder.buildStepA()
} else {
builder.buildStepB()
builder.buildStepZ()
}
Concrete
Builder1
- result: Product1
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product1
Concrete
Builder2
- result: Product2
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product2
result = new Product2()
result.setFeatureB()
return this.result
Product1
Product2

Builder

- **Pros**
  - Construct objects step-by-step
  - Reusable
  - *Single Responsibility Principle*

- **Cons**
  - Code Complexity
    : create multiple classes

References

[1] Marc Gregoire, 2018, Professional C++, 4th edition, WILEY

[2] https://refactoring.guru/design-patterns/cpp

[3] https://medium.com/must-know-computer-science/basic-design-patterns-in-c-39bd3d477a5c