Design Patterns in C# Made Simple

Module 7 Encapsulating Construction Rules with the Builder Pattern

Zoran Horvat
CEO at Coding Helmet

http://codinghelmet.com
zh@codinghelmet.com
zoranh75

Advancing from Factories to Builders

- Abstract Factory and Factory Method patterns
  - Introduce subtle coupling between consumer and concrete factory
- Drawbacks of coupling
  - Consumer needs to know which concrete product it is consuming
- Example: Constructing a database connection string
  - Client may need to know concrete database provider

Pros and Cons of Factories

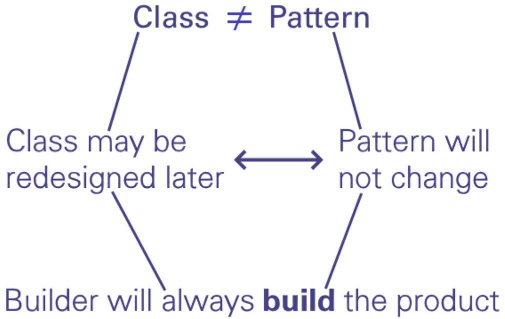
- Factory patterns fail in complex cases
  - They do not support variation well
- The issue with factories
  - Their signature is fixed
- Alternative to factory patterns
  - Builder can offer multiple methods to handle variation

```
class ConnectionStringBuilder
{

}
```

Warning

Using pattern name in class name


Class ≠ Pattern
Class may be redesigned later
Pattern will not change
Builder will always **build** the product

Factory vs. Builder

Input validation

- Builder’s methods can validate arguments
- Factory can validate arguments, too

Varying construction process

- Factory method’s signature cannot vary
- It would be several factories instead of only one

Summary

Advancing from Factory Method to Builder

Builder object is inconsistent by design

- Incomplete builder throws from Build()

Builder is mutable by design

- Even in immutable design, builders would be mutable
- Builder is collecting components
- This requires mutable internal representation

Summary

- Builder’s methods are self-referential
  - In violation of command-query separation principle
  - Method should either return a result or mutate state
  - It’s useful to return the modified builder
  - Makes it easy to chain calls to a builder
- Builder vs. Factory Method
  - Factory’s argument becomes builder’s component
  - Validate components separately, or postpone validation
  - Postponed validation offers better flexibility
  - Expose CanBuild() method alongside Build()

Summary

Supporting components with alternatives

- A component may be constructed in several ways
- Builder can offer method overloads for all cases

Supporting optional components

- Builder can supply reasonable defaults (substitutes)
- It can handle legitimate missing components
- Builder helps make the process more flexible

Next module:

Constructing Complex Objects
with the Builder Pattern