Design Patterns

CMSC 240 Software Systems Development

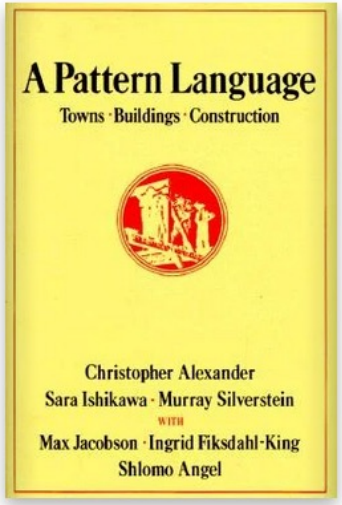
Today

- Storage Space
- Design Patterns
- Creational Patterns
  - Builder
- Structural Patterns
  - Adapter

RICHMOND

Design Patterns: What are they?

- Design patterns are typical solutions to commonly occurring problems in software design
- Pre-made blueprints that you customize to solve recurring design problems in your code
- Idea was initially applied to architecture of buildings and towns

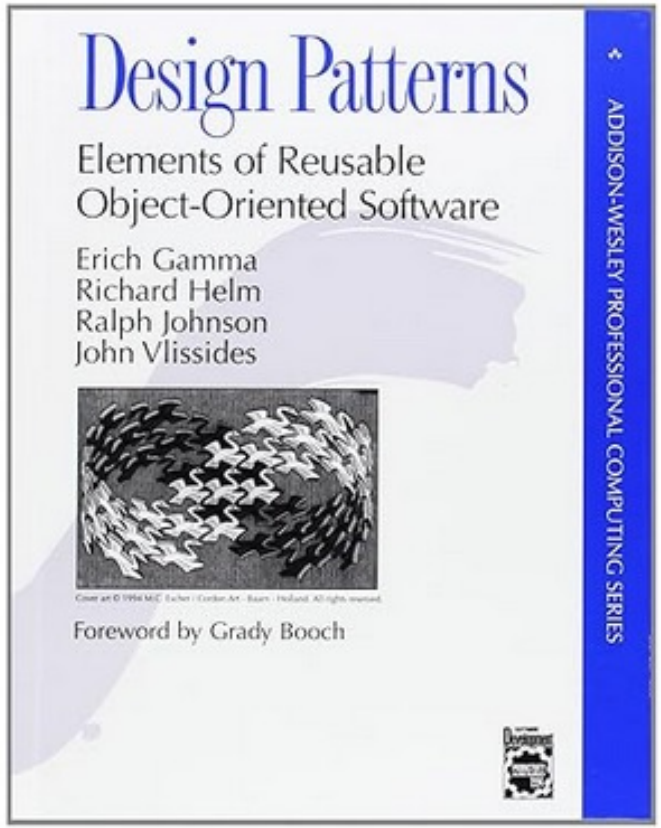


The elements of this language are entities called patterns. Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

-- Christopher Alexander, A Pattern Language

Design Patterns: What are they?

- Design patterns are a solution to a problem in a context
  - Help a designer get to the right design faster



Four Essential Parts

1. Pattern Name
   - Briefly describes the design problem provides a **common vocabulary** for software designers to use (e.g., *Builder*, *Singleton*, *Strategy*)
2. Problem
   - A description of the problem that the design pattern will solve
3. Solution
   - Describes what elements make up the design, their relationships and context
4. Consequences
   - What are the results and tradeoffs
   - Allows a comparison between different design patterns to see if there is a better fit

Classifying Design Patterns

Purpose: what a pattern does

1. **creational:** concerned with creation of objects

2. **structural:** related to composition of classes or objects

3. **behavioral:** related to interaction and distribution of responsibility

Classifying Design Patterns

Creational	Structural	Behavioral
Abstract Factory Builder Factory Method Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor Interpreter Template

Creational Patterns

- Purpose
  - **abstract the process of creating objects**
  - make a system unaware of how objects are created, composed, and represented
- What they do
  - encapsulate knowledge about which concrete classes a system uses (access created objects via interfaces)
  - hide how instances are created
- Provide flexibility with regards to
  - types of created objects
  - responsibility for creation
  - how and when objects are created

Creational Patterns

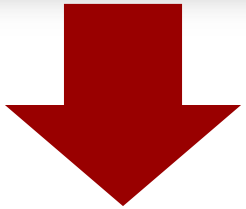
- Abstract Factory
- **Builder**
- Factory Method
- Prototype
- Singleton

Builder Pattern

1. Pattern Name
   - Builder
2. Problem
   - Complex objects often need detailed initialization, involving numerous fields
   - This initialization can result in large constructors with many parameters
   - Can lead to initialization steps being dispersed throughout various parts of the client code
3. Solution
   - Builder pattern suggests that you extract the object construction code out of its own class and move it to separate objects called builders
4. Consequences
   - Produce different types and representations using the same construction code
   - Isolates code for construction and representation
   - Construct complex objects step by step

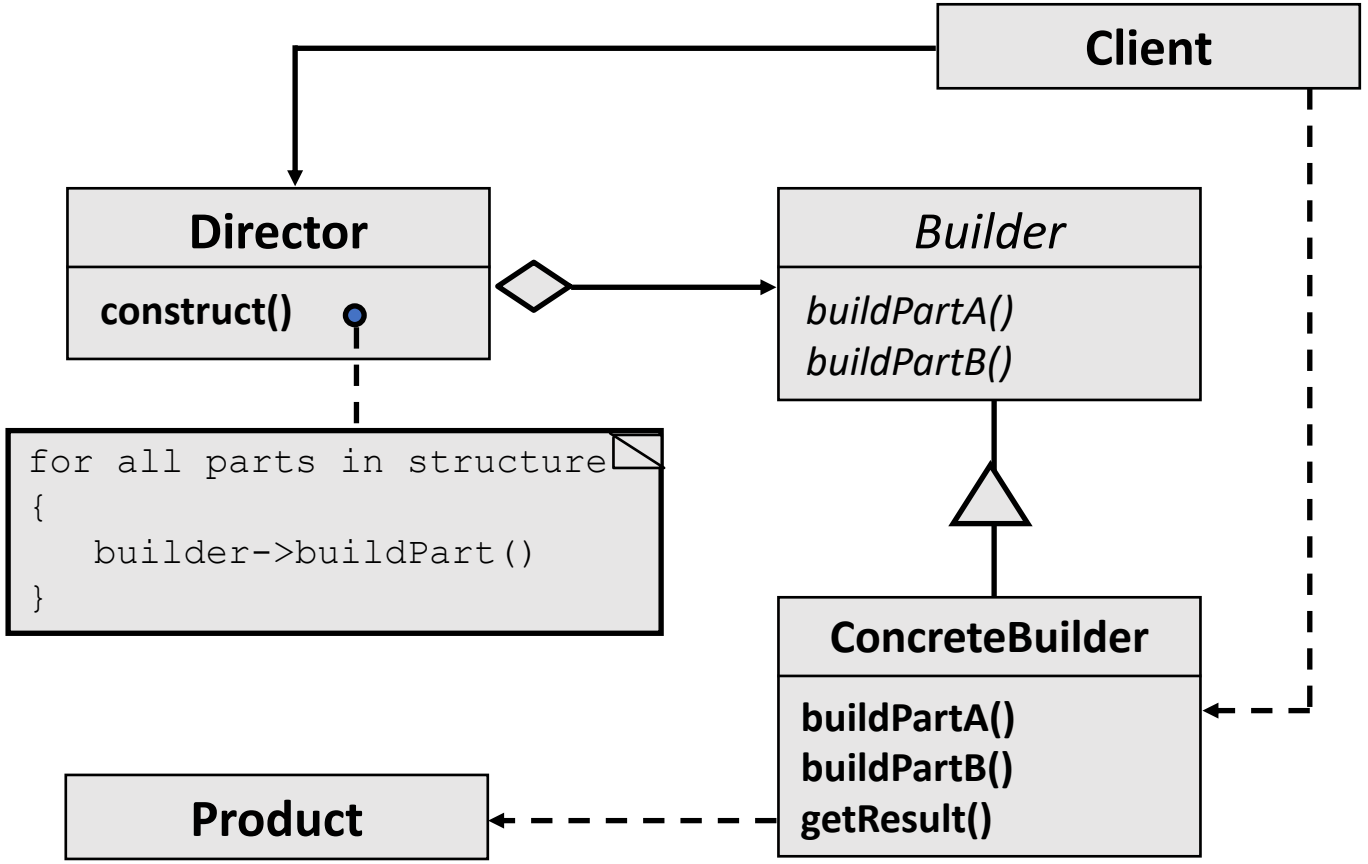
Builder Pattern

```
Car sportsCar(2, "V6 Engine", true, false, false);

Car sportsUtilityVehicle(5, "V8 Engine", true, true, true);
```



```
director.constructSportsCar(carBuilder);
Car sportsCar = carBuilder.getResult();

director.constructSUV(carBuilder);
Car sportsUtilityVehicle = carBuilder.getResult();
```

Builder Pattern



Structural Patterns

- Purpose
  - Explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient
- What they do
  - use inheritance to compose interfaces or implementations
  - describe ways to compose objects to realize new functionality

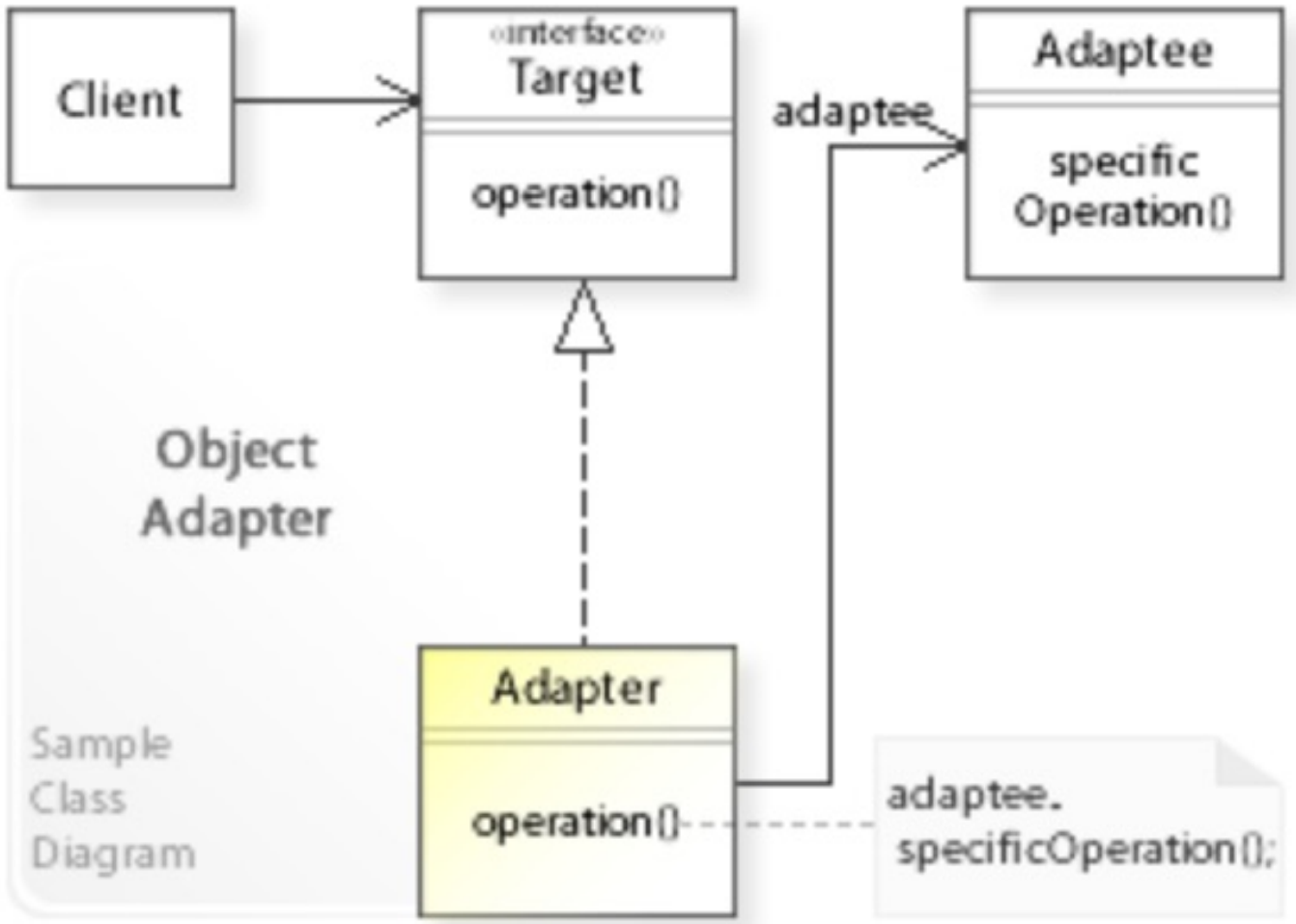
Structural Patterns

- **Adapter**
- Composite
- Proxy
- Flyweight
- Façade
- Bridge
- Decorator

Adapter Pattern

1. Pattern Name
   - Adapter
2. Problem
   - You want to use an existing class, and its interface does not match the one you need
3. Solution
   - Use an adapter to wrap one of the objects to hide the complexity of conversion happening behind the scenes
   - The wrapped object is not aware of the adapter
4. Consequences
   - Converts the interface of a class into another interface that clients expect

Adapter Pattern



CC Wikimedia

Behavioral Patterns

- Purpose
  - Improving communication and the assignment of responsibilities between objects
  - Deal with object interactions and how they distribute responsibilities
- What they do
  - Define the protocols or methods through which objects interact and communicate
  - Manage complex algorithms, relationships, and responsibilities among interacting objects
  - Reduce the tight coupling between objects, making a system more modular and easier to maintain or extend

Behavioral Patterns

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- **Observer**
- State
- **Strategy**
- Visitor
- Interpreter
- Template