

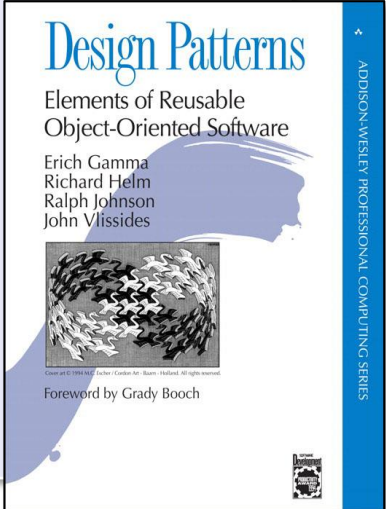
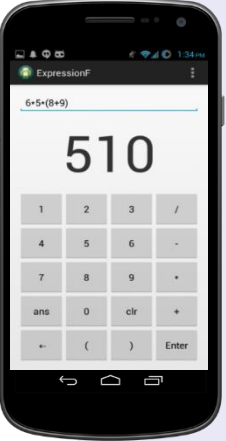
Overview of the Patterns Applied in the Expression Tree Processing App

Douglas C. Schmidt

Learning Objectives in This Lesson

- Recognize which GoF patterns the expression tree processing app uses.

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



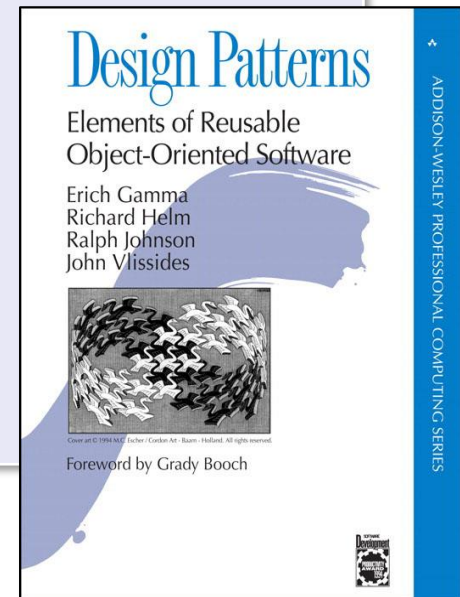
Douglas C. Schmidt

Lesson Introduction

Lesson Introduction

- The book *Design Patterns: Elements of Reusable Object-Oriented Software* (the so-called "Gang of Four" or "GoF" book) presents recurring solutions to common problems in software design in the form of 23 patterns.

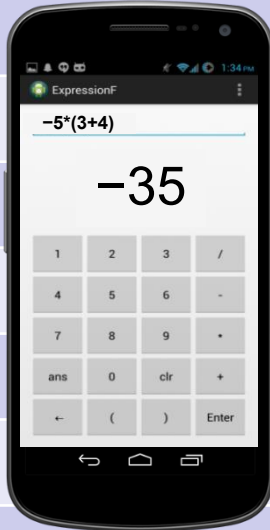
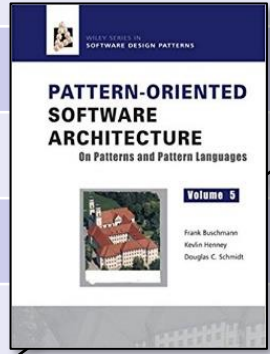
	Creational	Structural	Behavioral
Class	Factory Method	Adapter (class)	Interpreter Template Method
Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Façade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor



Douglas C. Schmidt

Design Problems & GoF Pattern Solutions

Design Problems & GoF Pattern Solutions

Design Problem		Pattern
Non-extensible & error-prone designs		Composite
Minimizing impact of variability		Bridge
Inflexible expression input processing		Interpreter
Inflexible interpreter output		Builder
Scattered request implementations		Command
Inflexible creation of variabilities		Factory Method
Inflexible expression tree traversal		Iterator
Obtrusive behavior changes		Strategy
Non-extensible tree operations		Visitor
Incorrect user request ordering		State
Non-extensible operating modes		Template Method
Minimizing global variable liabilities		Singleton

These patterns constitute a "pattern sequence" for the case study app.

See www.dre.vanderbilt.edu/~schmidt/POSA-tutorial.pdf

Design Problems & GoF Pattern Solutions

Design Problem

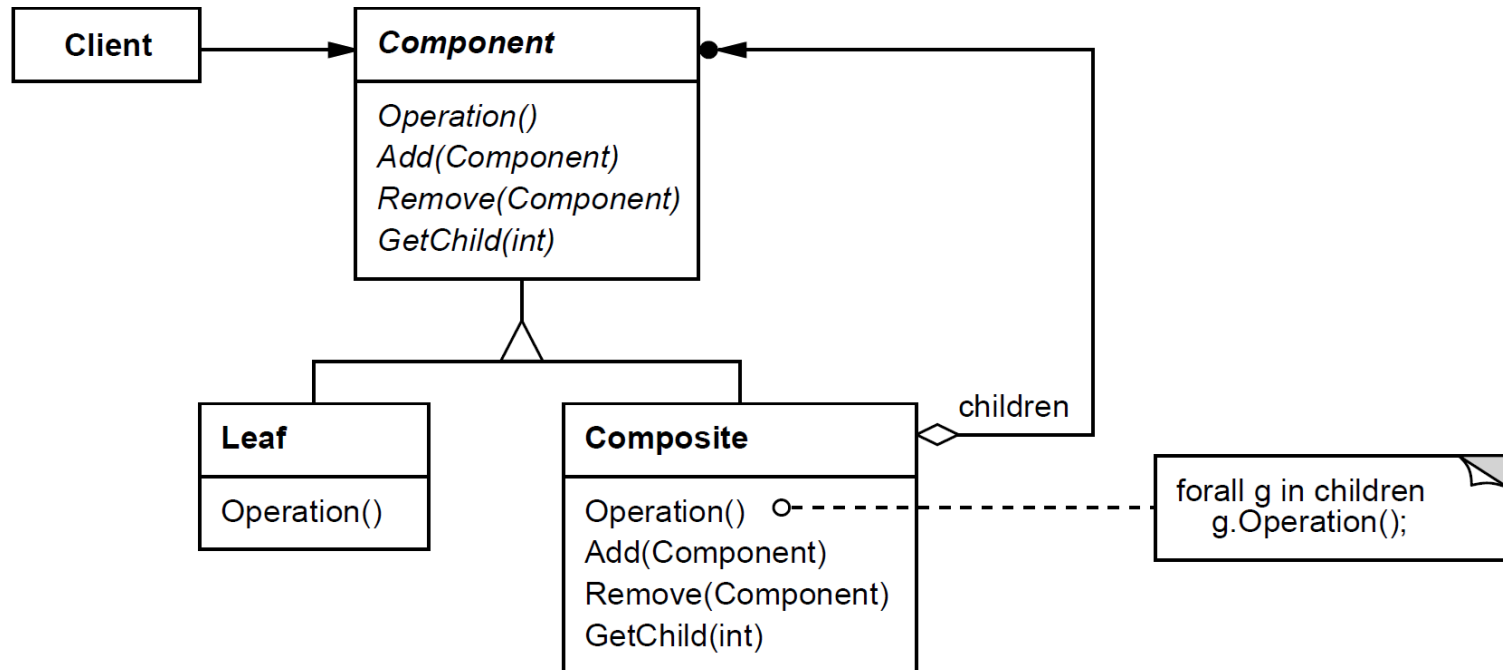
Pattern

Non-extensible & error-prone designs

Composite

Composite intent

- Treat individual objects & multiple, recursively-composed objects uniformly



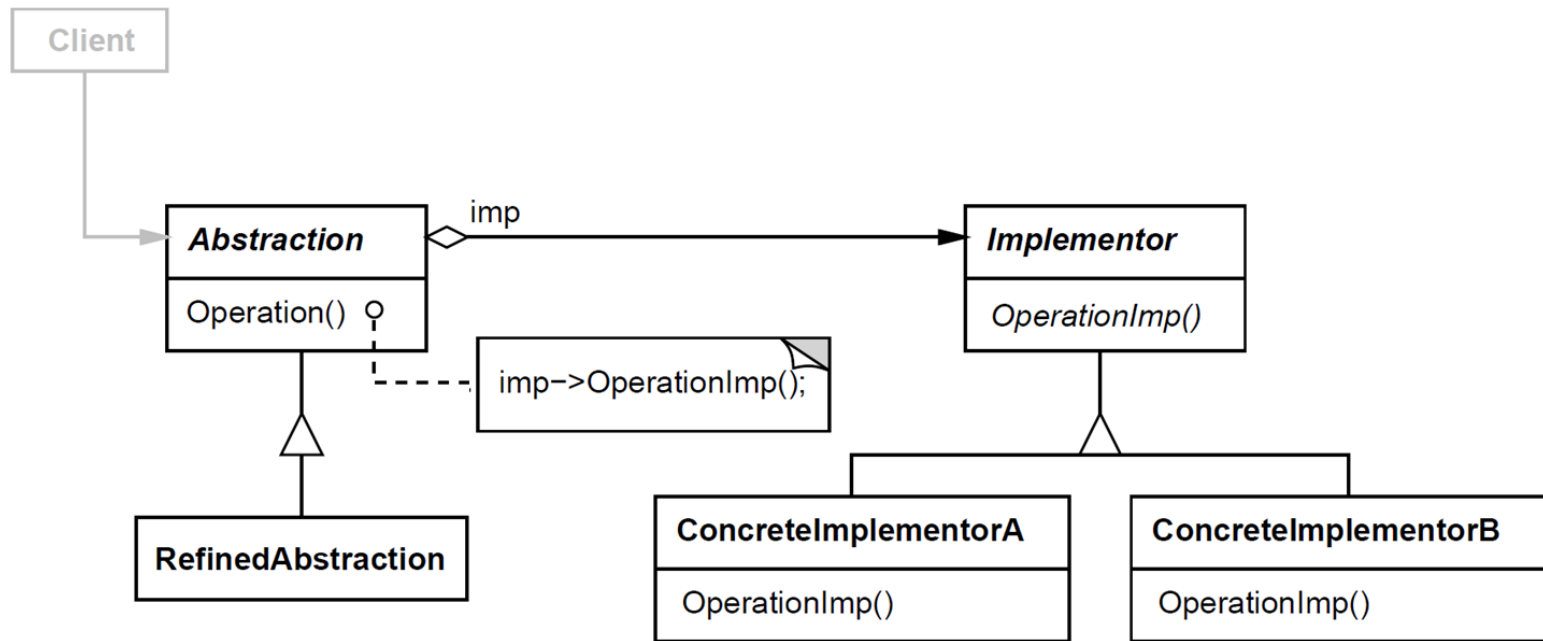
See en.wikipedia.org/wiki/Composite_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Minimizing impact of variability	Bridge

Bridge intent

- Separate an abstraction from its implementation(s) so the two can vary independently



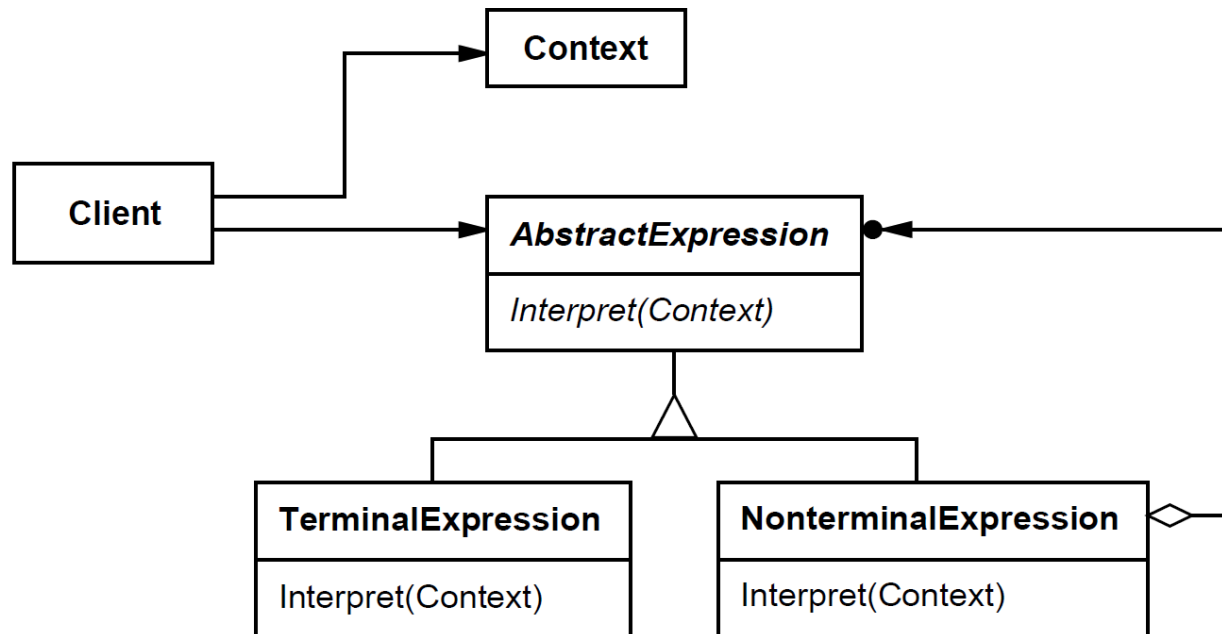
See en.wikipedia.org/wiki/Bridge_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Inflexible expression input processing	Interpreter

Interpreter intent

- Given a language, define a representation for its grammar, along with an interpreter that uses the representation to interpret sentences in the language



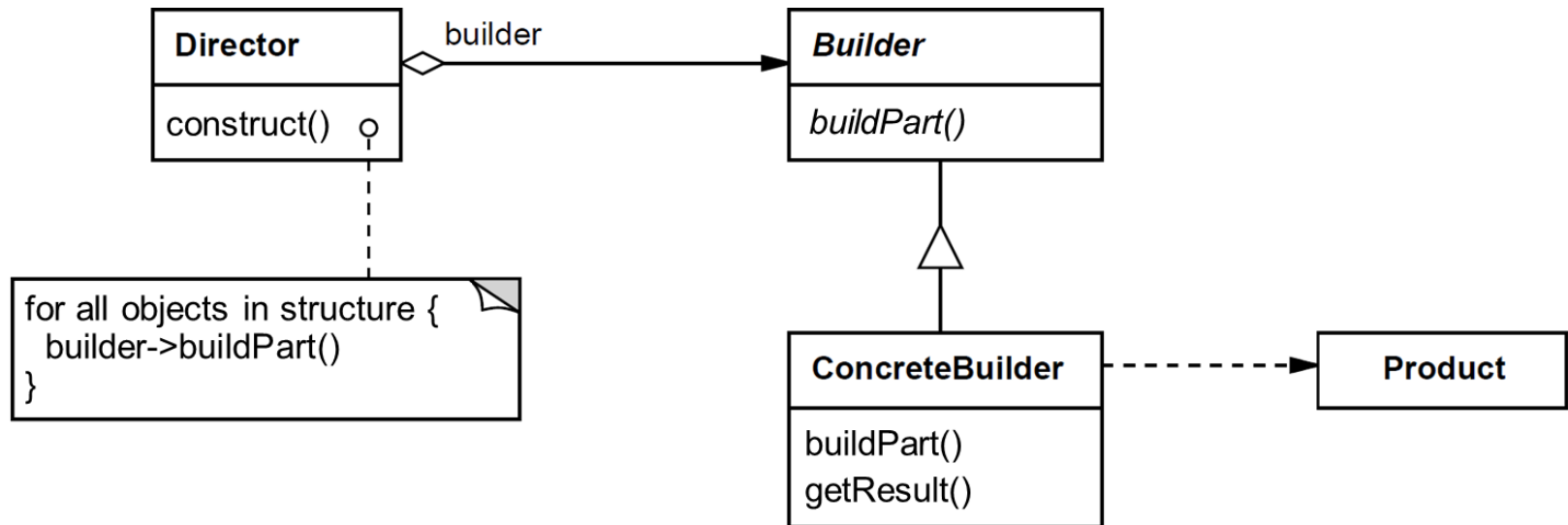
See en.wikipedia.org/wiki/Interpreter_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Inflexible interpreter output	Builder

Builder intent

- Separate the construction of a complex object from its representation



See en.wikipedia.org/wiki/Builder_pattern

Design Problems & GoF Pattern Solutions

Design Problem

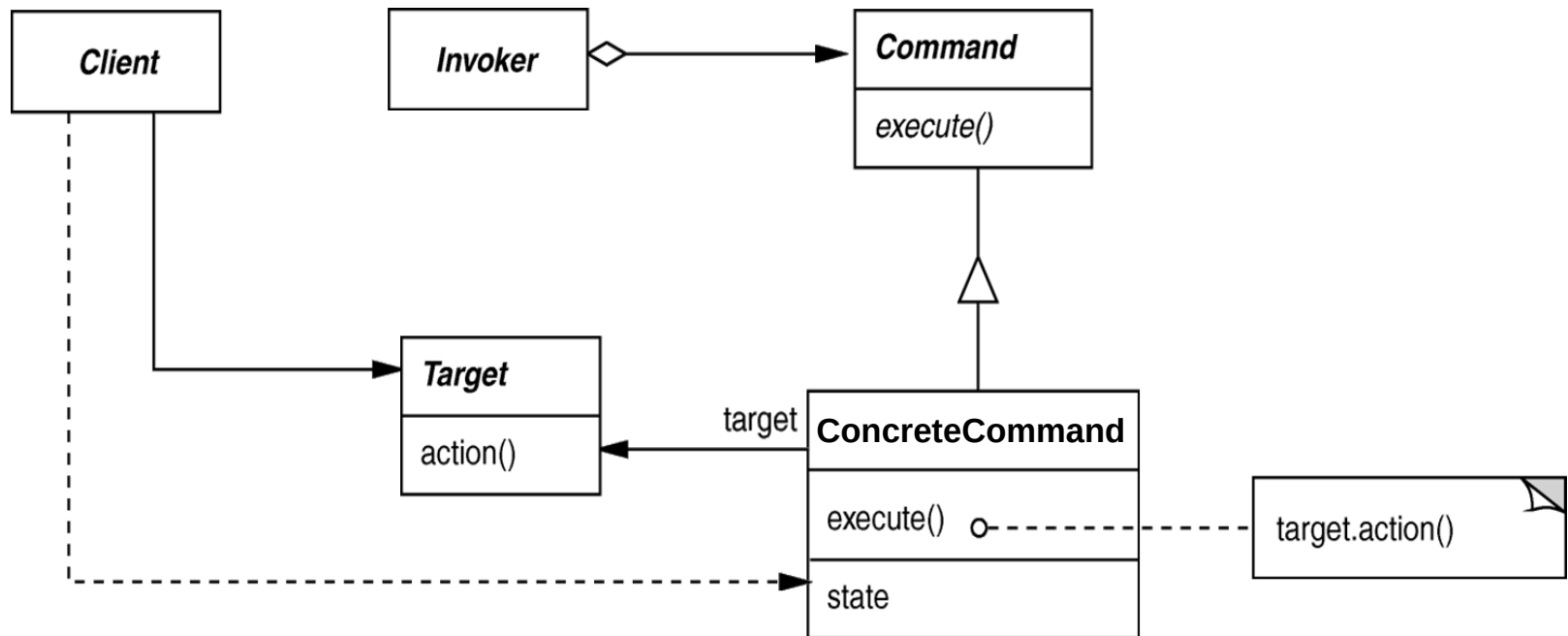
Scattered & fixed request implementations

Pattern

Command

Command intent

- Encapsulate the request for a service as an object



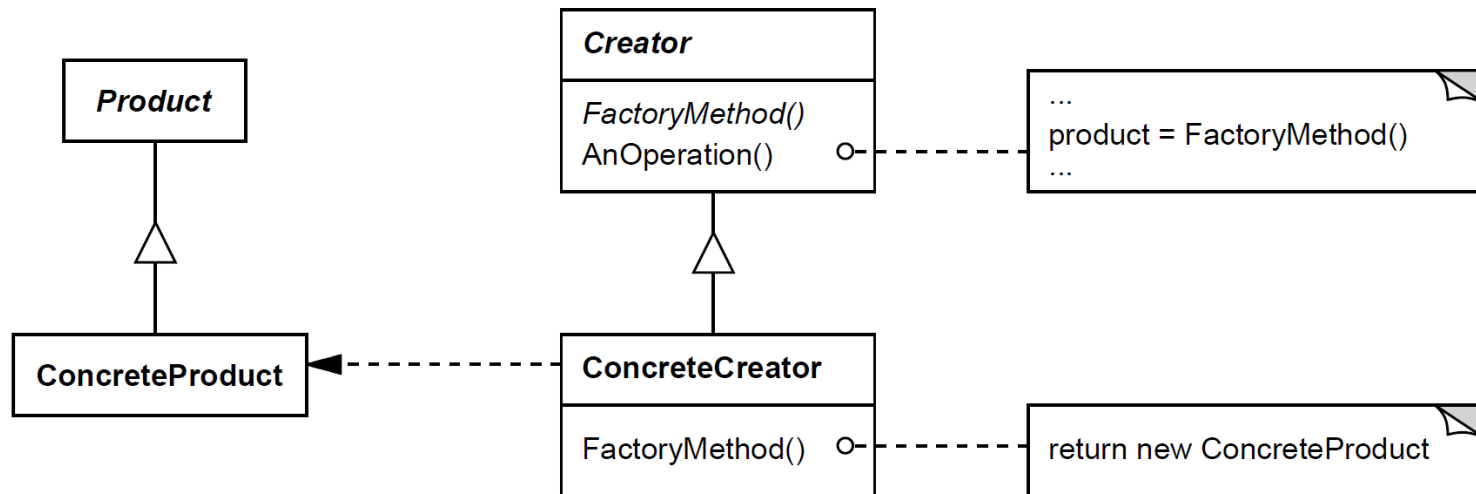
See en.wikipedia.org/wiki/Command_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Inflexible creation of variabilities	Factory Method

Factory Method intent

- Provide an interface for creating an object, but leave the choice of the object's concrete type to a subclass

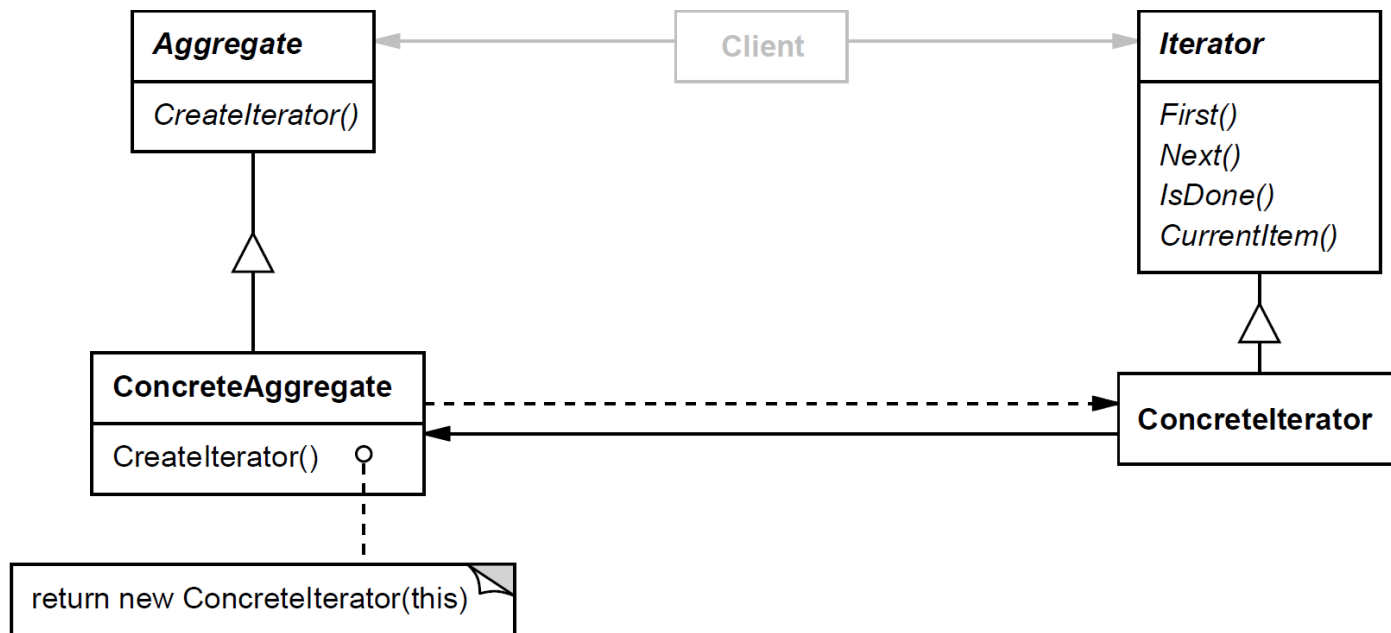


Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Inflexible expression tree traversal	Iterator

Iterator intent

- Access elements of an aggregate without exposing its representation

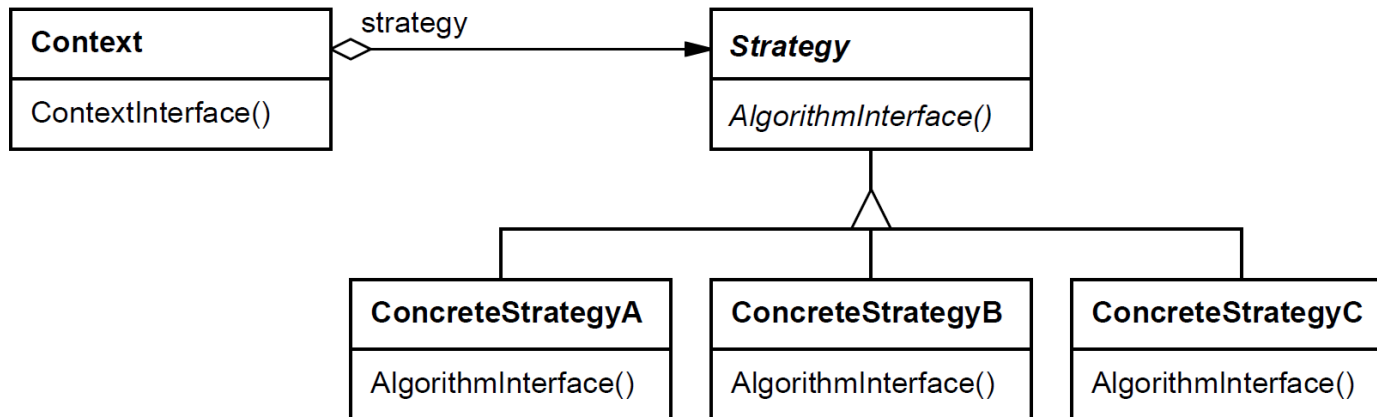


Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Obtrusive behavior changes	Strategy

Strategy intent

- Define a family of algorithms, encapsulate each one, & make them interchangeable to let clients & algorithms vary independently



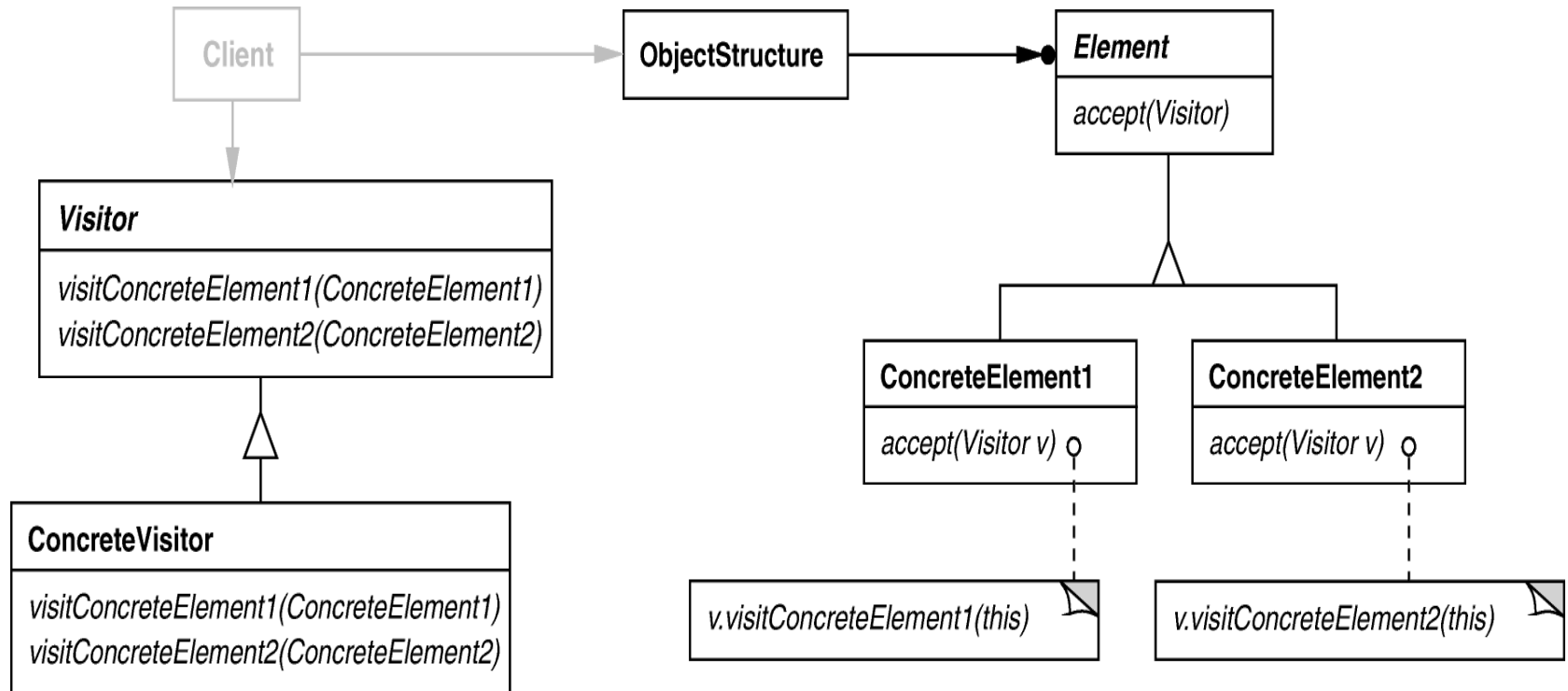
See en.wikipedia.org/wiki/Strategy_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Non-extensible tree operations	Visitor

Visitor intent

- Centralize operations on an object structure so that they can vary independently, but still behave polymorphically



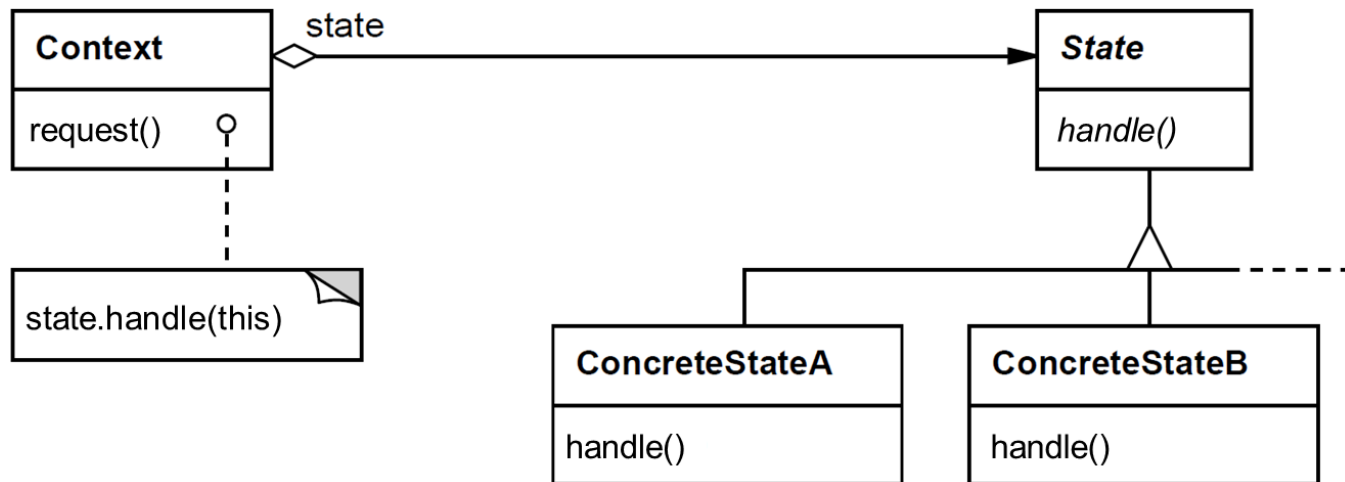
See en.wikipedia.org/wiki/Visitor_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Incorrect user request ordering	State

State intent

- Allow an object to alter its behavior when its internal state changes—the object will appear to change its class



See en.wikipedia.org/wiki/State_pattern

Design Problems & GoF Pattern Solutions

Design Problem

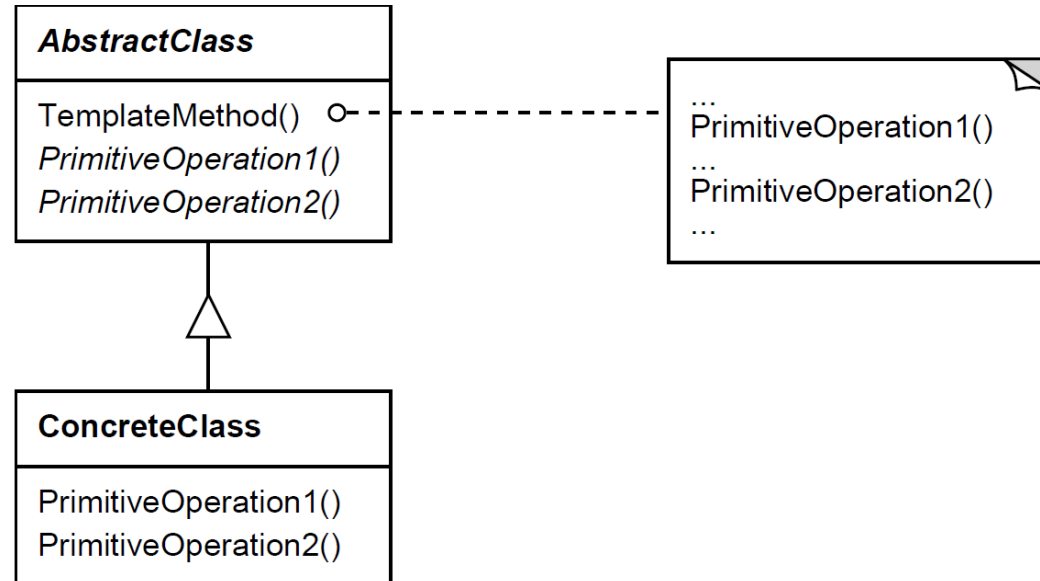
Non-extensible operating modes

Pattern

Template Method

Template Method intent

- Provide a skeleton of an algorithm in a method, deferring some steps to subclasses



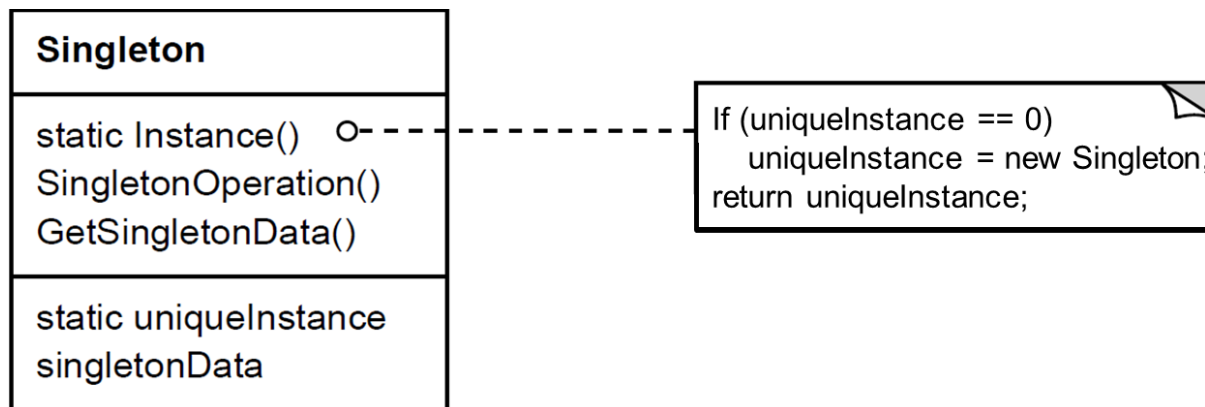
See en.wikipedia.org/wiki/Template_method_pattern

Design Problems & GoF Pattern Solutions

Design Problem	Pattern
Minimizing global variable liabilities	Singleton

Singleton intent

- Ensure a class only has one instance & provide a global point of access



See en.wikipedia.org/wiki/Singleton_pattern

Design Problems & GoF Pattern Solutions

Design Problem		Pattern
Non-extensible & error-prone designs		Composite
Minimizing impact of variability		Bridge
Inflexible expression input processing		Interpreter
Inflexible interpreter output		Builder
Scattered request implementations		Command
Inflexible creation of variabilities		Factory Method
Inflexible expression tree traversal		Iterator
Obtrusive behavior changes		Strategy
Non-extensible tree operations		Visitor
Incorrect user request ordering		State
Non-extensible operating modes		Template Method
Minimizing global variable liabilities		Singleton

Naturally, these patterns apply to more than expression tree processing apps!

