CSSE 374: Introduction to Gang of Four Design Patterns

Shawn Bohner

Office: Moench Room F212

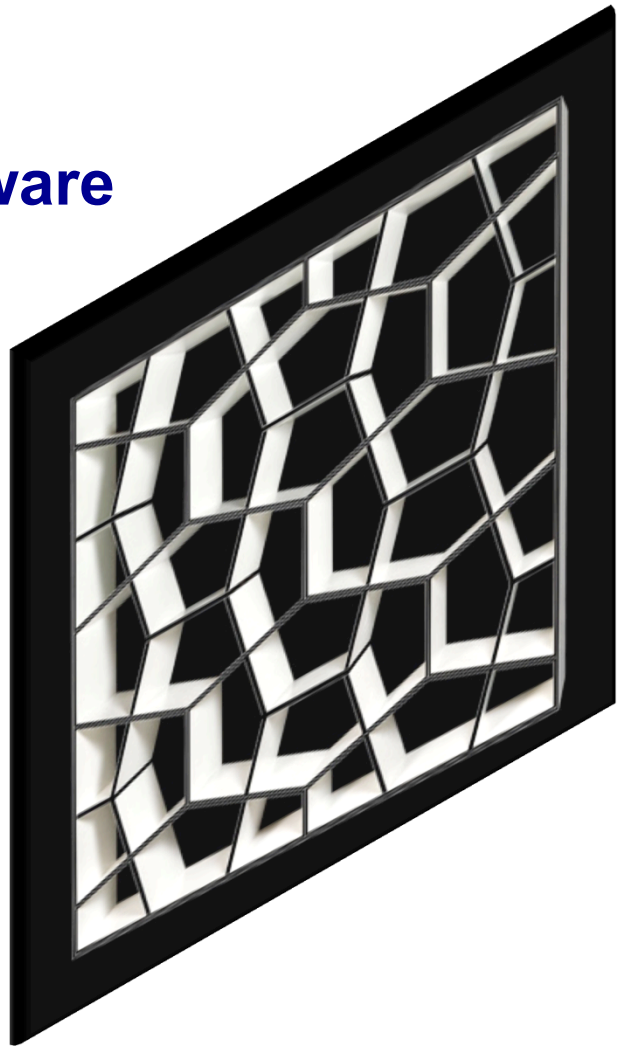
Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Learning Outcomes: Patterns, Tradeoffs

Identify criteria for the design of a software system and select patterns, create frameworks, and partition software to satisfy the inherent trade-offs.



- Introduce Gang of Four Concepts
- Describe and use GoF Patterns
  - Adapter
  - Factory
  - Singleton
  - Strategy
- Design Studio with Team 2.2

So, why bother to learn design patterns?

- Think for 15 seconds…
- Turn to a neighbor and discuss it for a minute

Gang of Four (GoF)

- Ralph Johnson, Richard Helm, Erich Gamma, and John Vlissides (left to right)

Gang of Four Design Patterns

Behavioral

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

Creational

- ✓ Factory Method
- Abstract Factory
- Builder
- Prototype
- ✓ Singleton

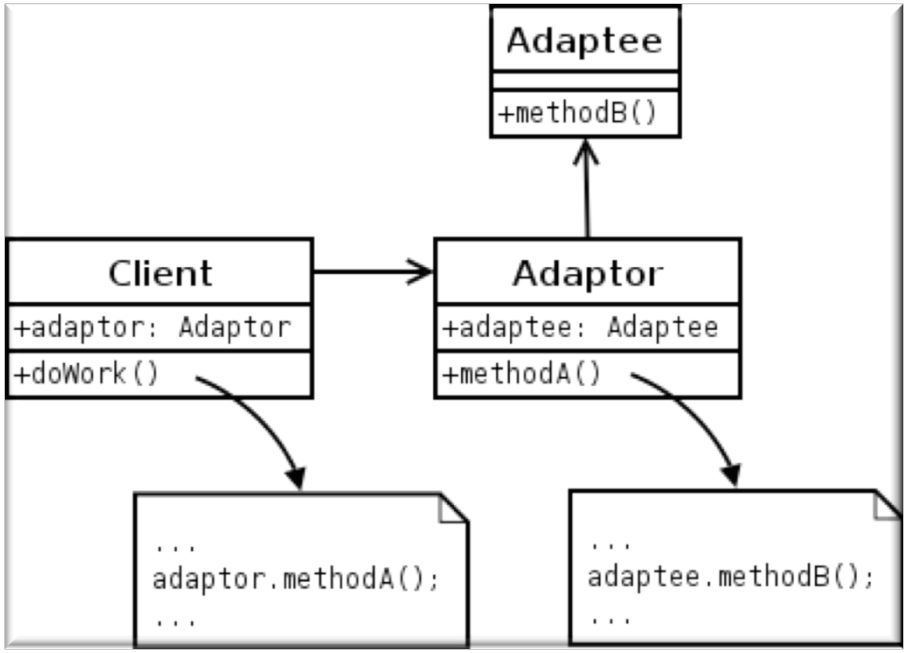
Structural

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

Adapter: Structural Pattern

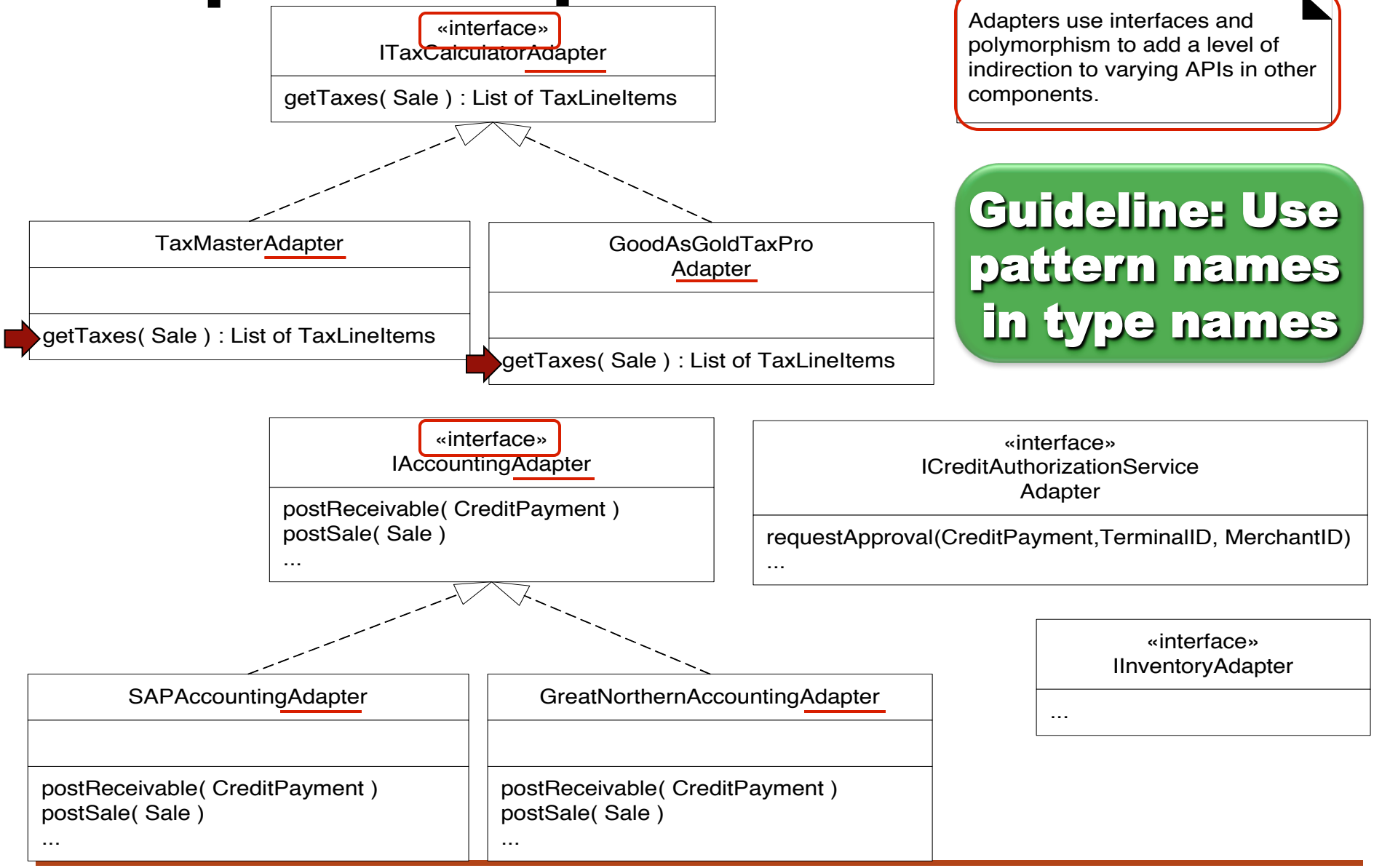
Problem: How do we provide a single, stable interface to similar components with different interfaces?

- How do we resolve incompatible interfaces?



Solution: Use an intermediate *adapter* object to convert calls to the appropriate interface for each component

Adapter Examples



Adapters use interfaces and polymorphism to add a level of indirection to varying APIs in other components.

Guideline: Use pattern names in type names

Which GRASP Principles in Adapter?

- Low coupling
- High cohesion
- Information Expert
- Creator
- Controller
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

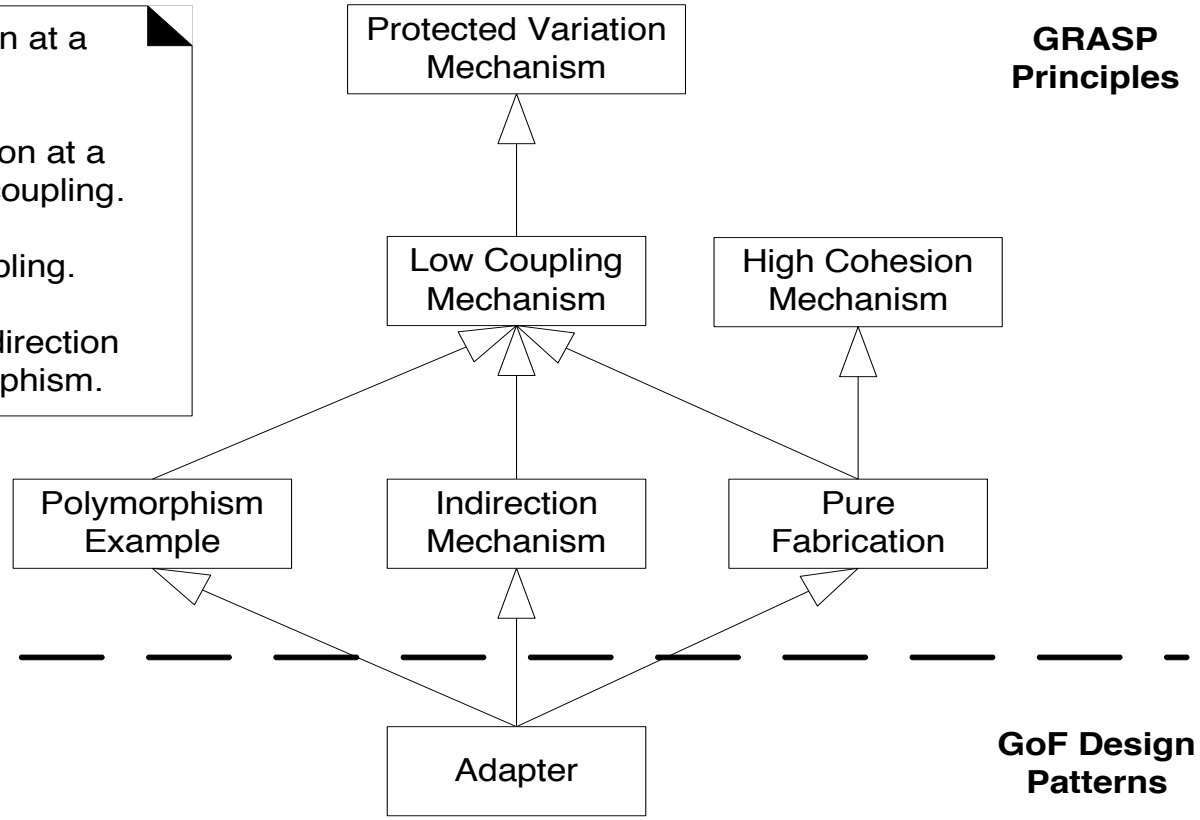
GoF Adapter mapped to GRASP

Low coupling is a way to achieve protection at a variation point.

Polymorphism is a way to achieve protection at a variation point, and a way to achieve low coupling.

An indirection is a way to achieve low coupling.

The Adapter design pattern is a kind of Indirection and a Pure Fabrication, that uses Polymorphism.

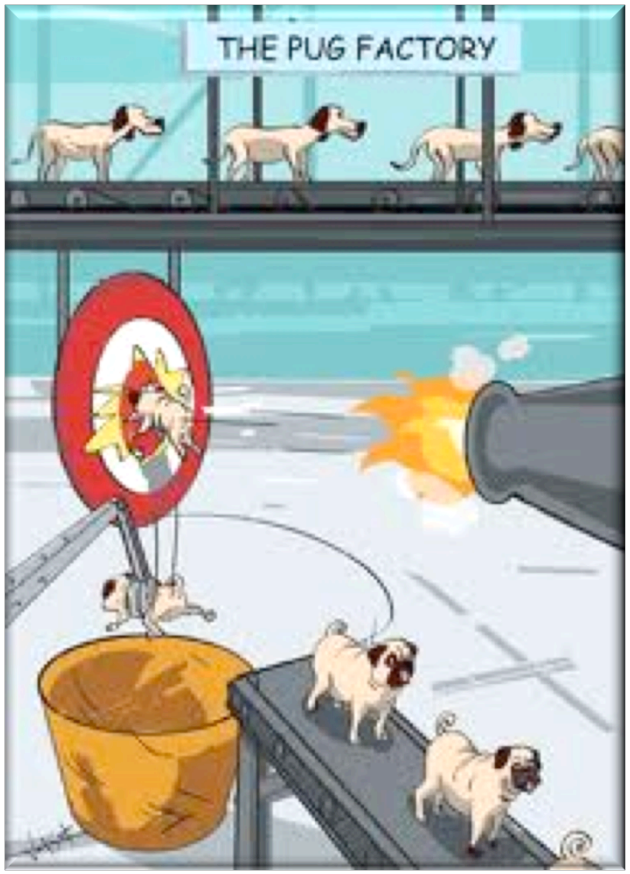


Factory: Creational Pattern

Problem: Who should be responsible for creating objects when there are special considerations like:

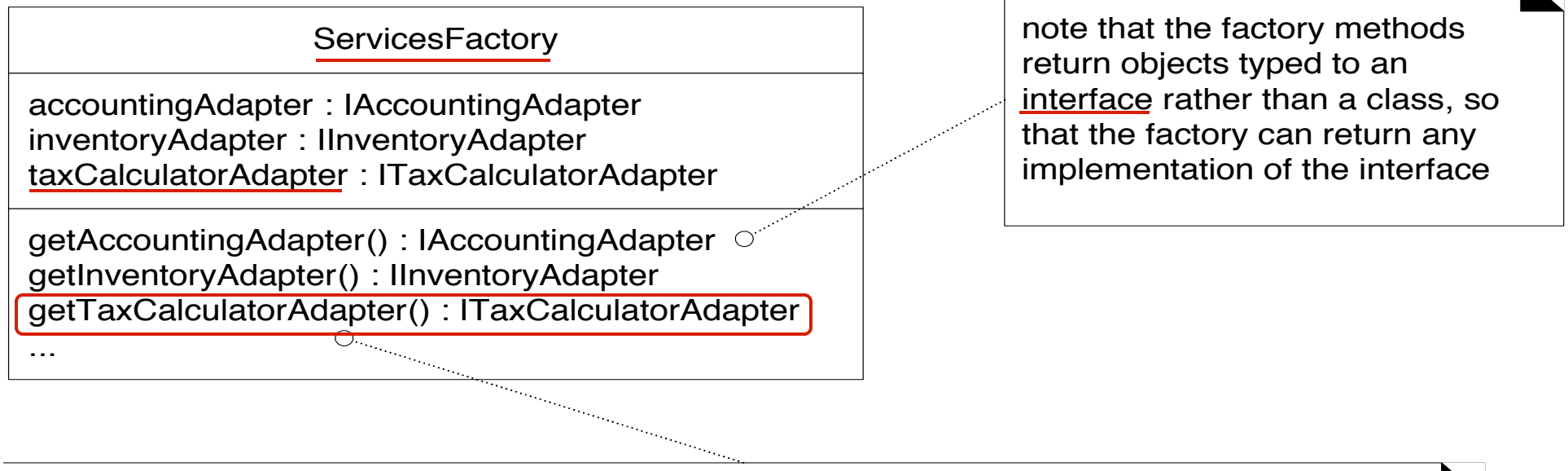
- Complex creation logic
- Separating creation to improve cohesion
- A need for caching

Solution: Create a "Pure Fabrication" called a "Factory" to handle the creation



Also known as Simple Factory or Concrete Factory

Factory Example

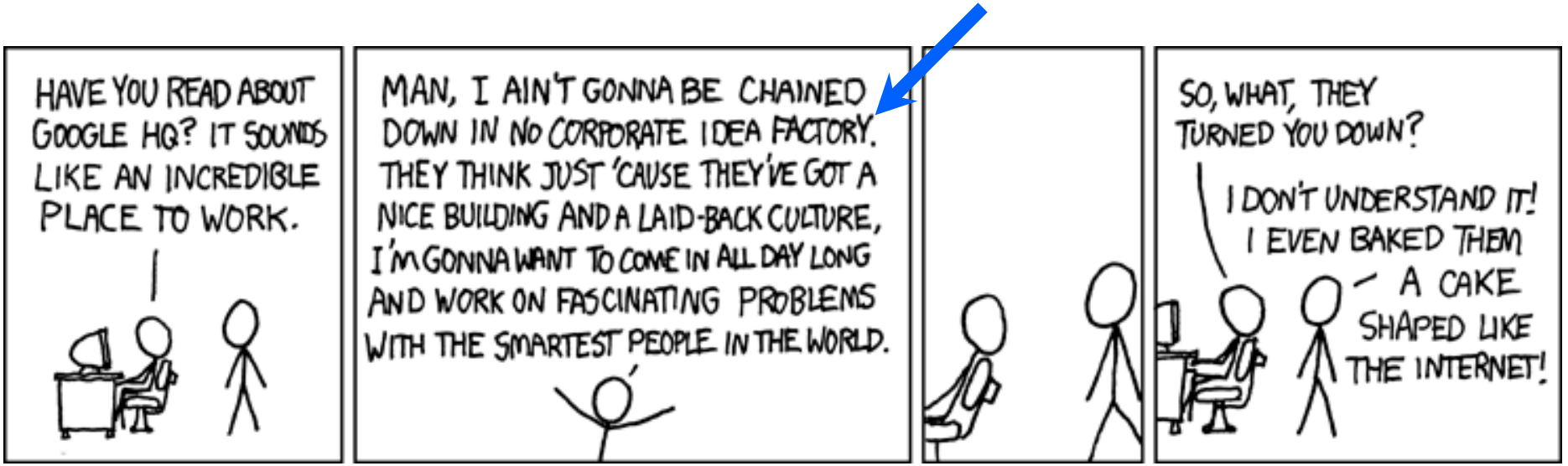



```
if ( taxCalculatorAdapter == null )
{
  // a reflective or data-driven approach to finding the right class: read it from an
  // external property

  String className = System.getProperty( "taxcalculator.class.name" );
  taxCalculatorAdapter = (ITaxCalculatorAdapter) Class.forName( className ).newInstance();

}
return taxCalculatorAdapter;
```

Advantages of Factory

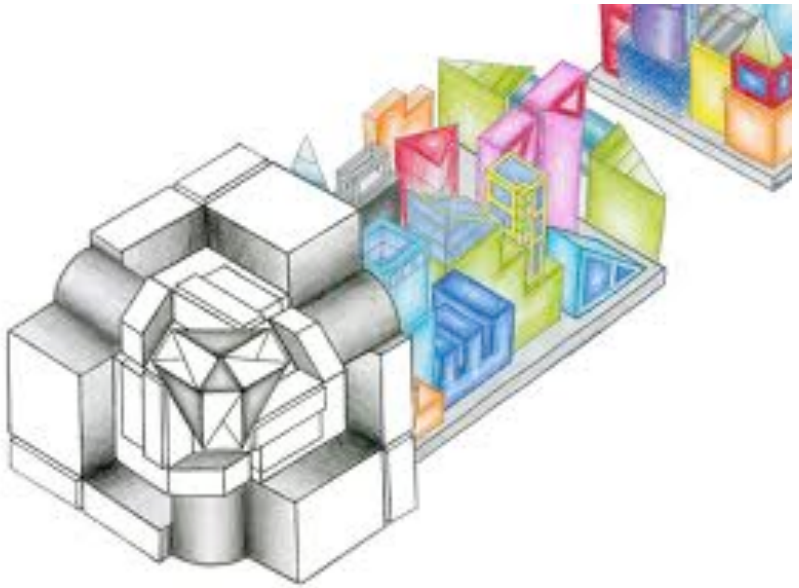
- Puts responsibility of creation logic into a separate, cohesive class —*separation of concerns*
- Hides complex creation logic
- Allows performance enhancements:
  - Object caching
  - Recycling

Working for Google



I hear once you've worked there for 256 days they teach you the secret of levitation.

Who creates the Factory?



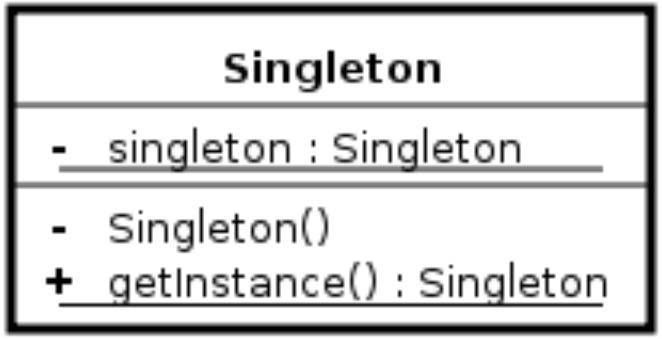
- Several classes need to access Factory methods
- Options:
  - Pass instance of Factory to classes that need it
  - Provide global visibility to a Factory instance

Dependency Injection

Singleton

Singleton: Creational Pattern

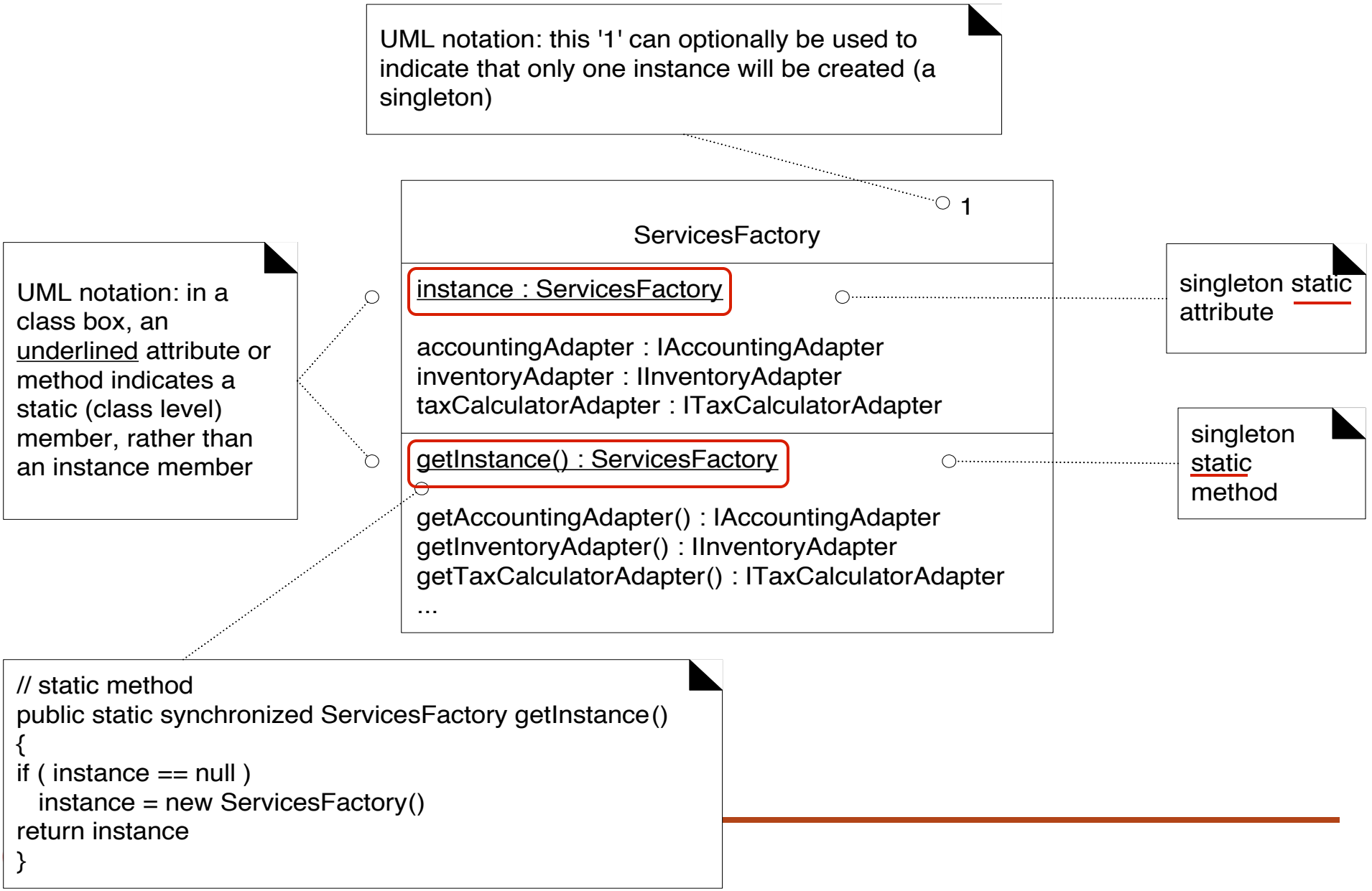
Problem: How do we ensure that exactly one instance of a class is created and is globally accessible?



Solution: Define a static method in the class that returns the *singleton* instance

- Created only once for the life of the program (a non-creational pattern?)
- Provides single global point of access to instance
  - Similar to a static or global variable variable

Singleton Example



Lazy vs. Eager Initialization

- **Lazy**:

```
private static ServicesFactory instance;
public static synchronized Services Factory
getInstance() {
  if (instance == null)
        instance = new ServicesFactory();
  return instance;
}
```

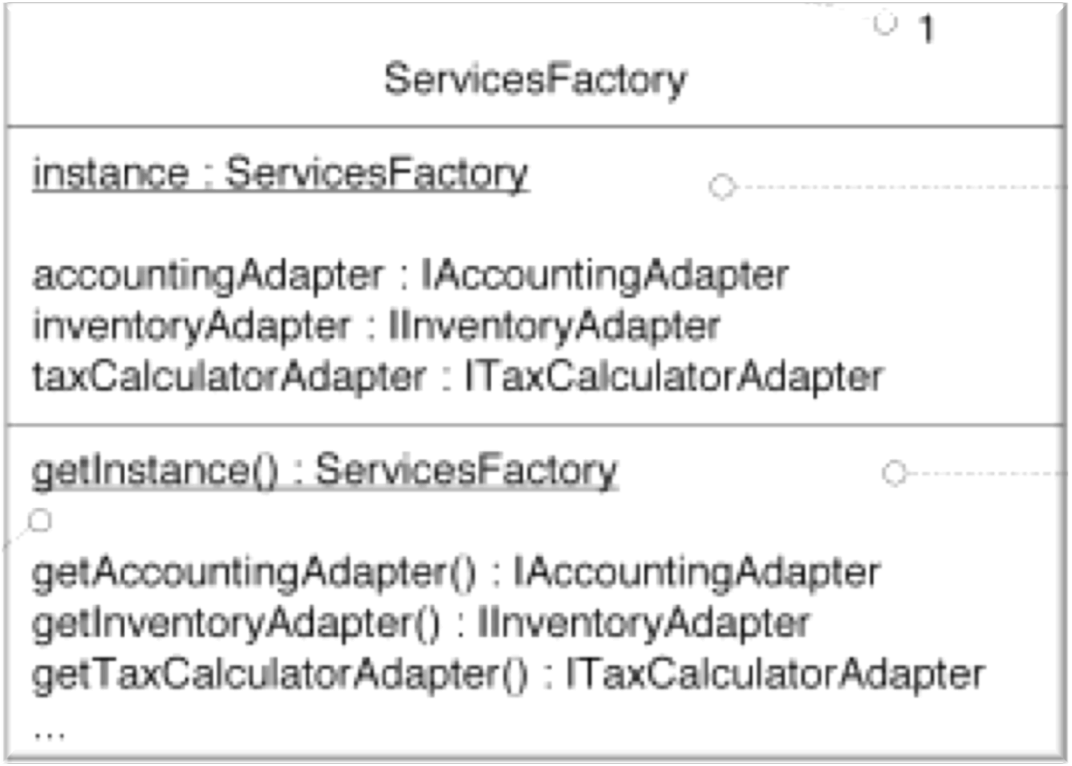
- **Eager**:

```
private static ServicesFactory instance = new
ServicesFactory();
public static Services Factory getInstance()
{
  return instance;
}
```

Pros and cons?

Why don't we just make all the methods static?

- Instance methods permit subclassing
- Instance method allow easier migration to "multi-ton" status



Singleton Considered Harmful?

Favor Dependency Injection

- Hides dependencies by introducing global visibility
- Hard to test since it introduces global state (also leaks resources)
- A singleton today is a multi-ton tomorrow
- Low cohesion — class is responsible for domain duties *and* for limiting number of instances

Instead, use Factory to control instance creation

http://blogs.msdn.com/scottdensmore/archive/2004/05/25/140827.aspx

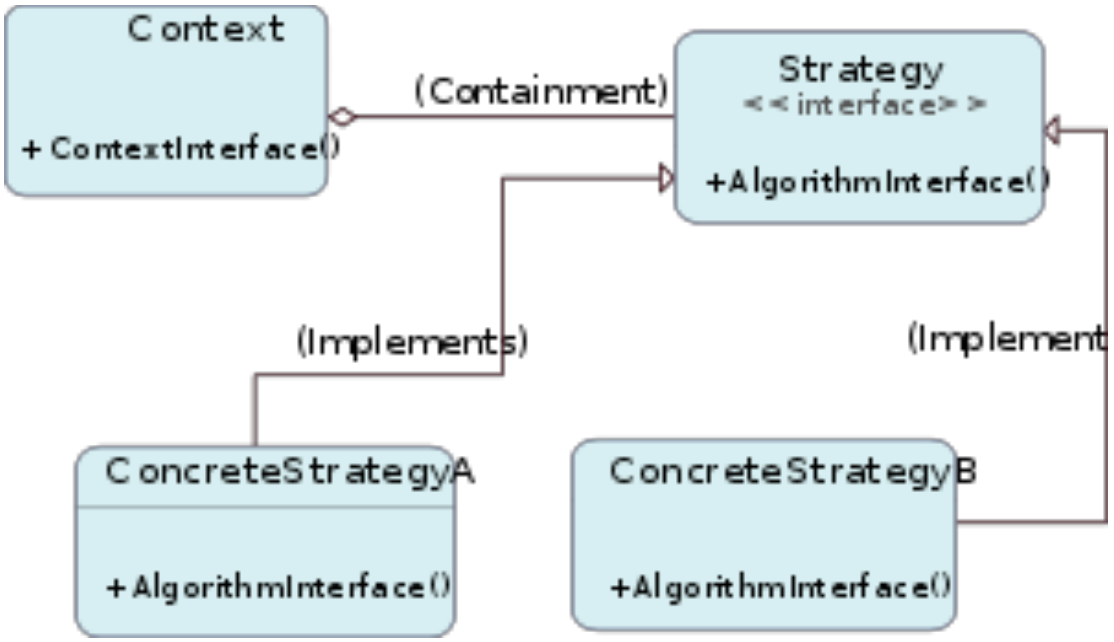
http://tech.puredanger.com/2007/07/03/pattern-hate-singleton/

Strategy: Behavior Pattern

Problem:

How do we design for varying, but related, algorithms or policies?

How do we design for the ability to change these?

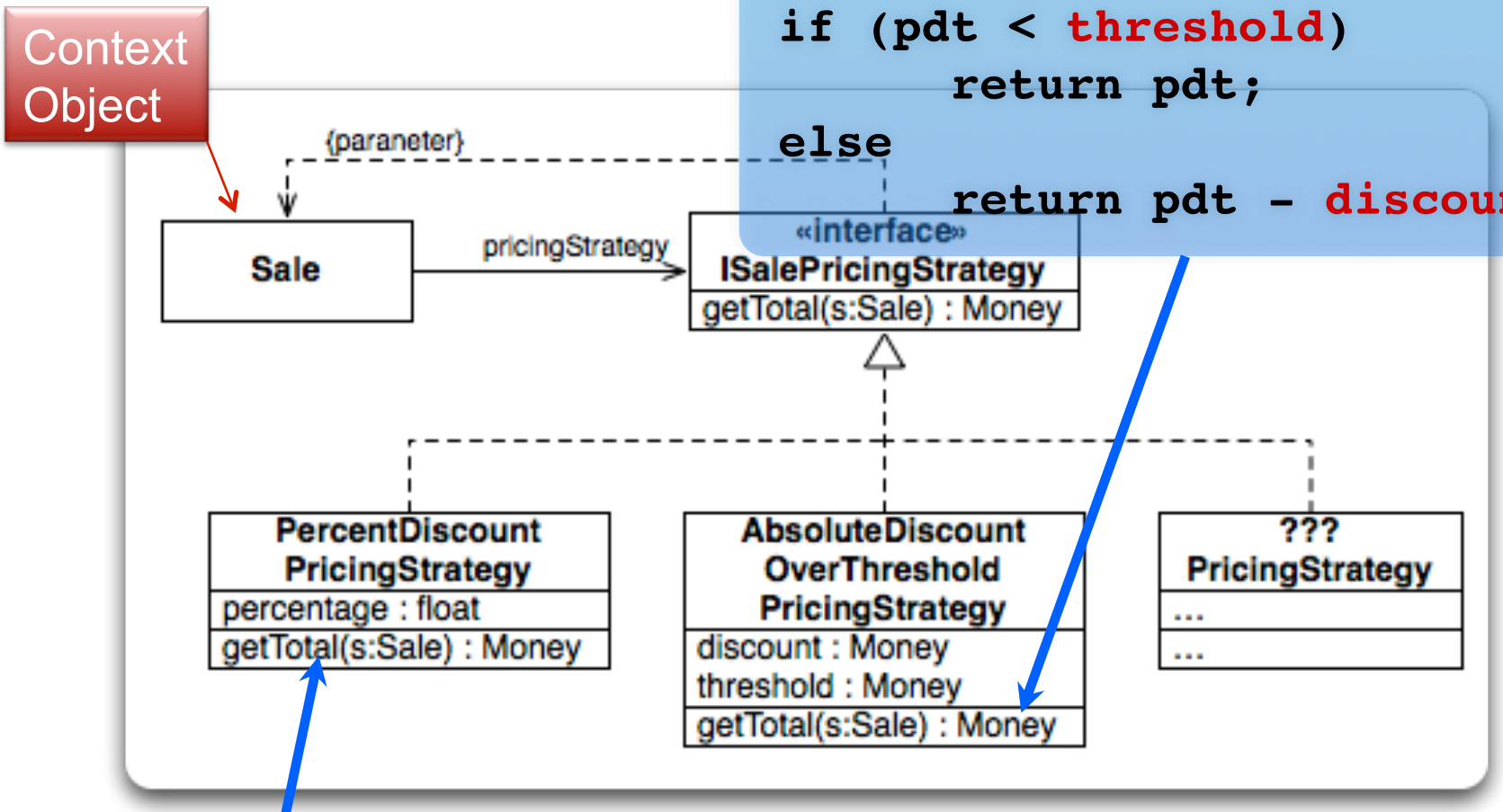


Solution:

Define each algorithm or policy in a separate class with a common interface.

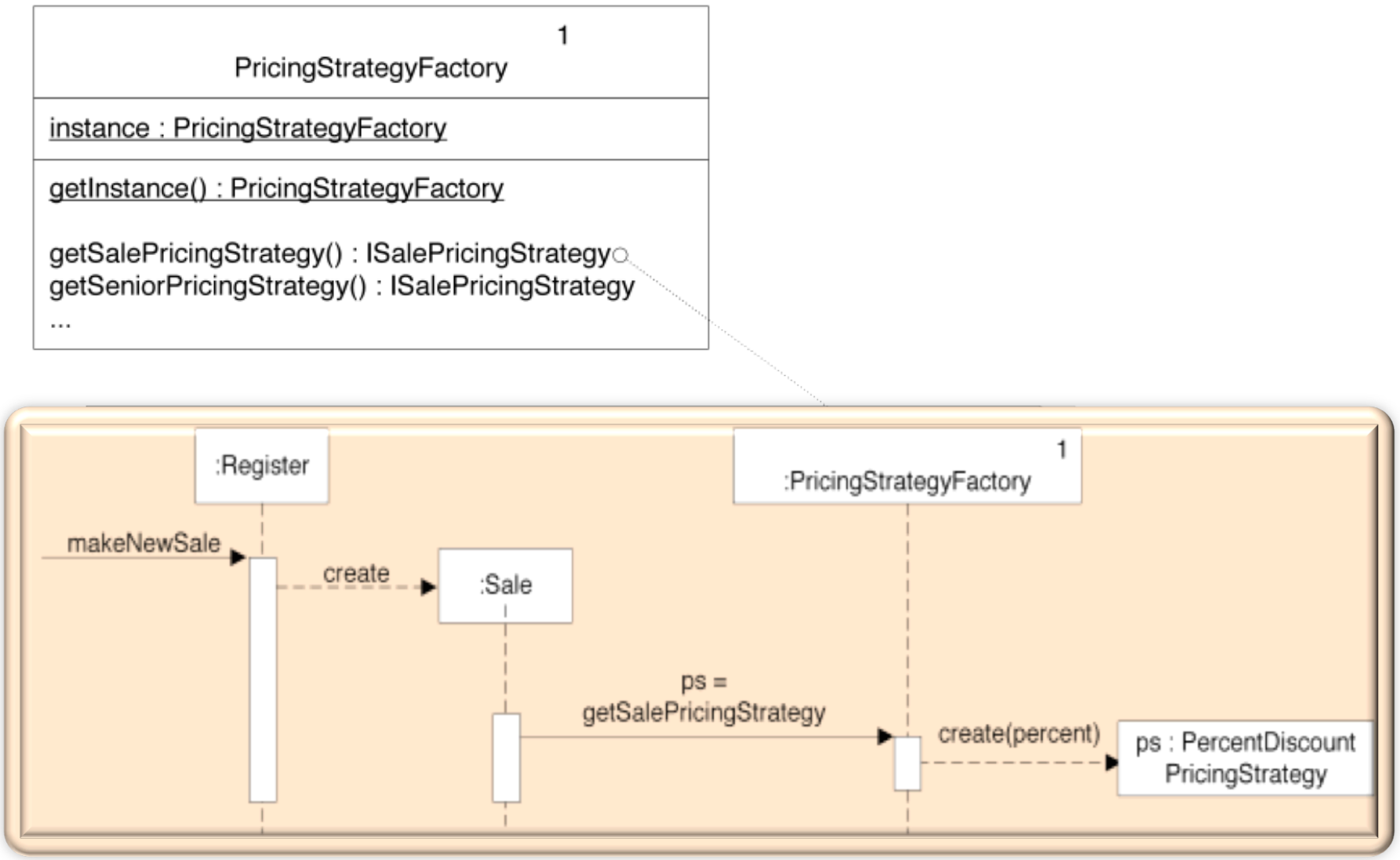
Strategy Example

```
pdt = s.getPreDiscountTotal();
if (pdt < threshold)
        return pdt;
else
        return pdt - discount;
```




```
return s.getPreDiscountTotal() * percentage;
```

Where does the *PricingStrategy* come from?



Examples of Change and Patterns

What Varies	Design Pattern
Algorithms	Strategy, Visitor
Actions	Command
Implementations	Bridge
Response to change	Observer
Interactions between objects	Mediator
Object being created	Factory Method, Abstract Factory, Prototype
Structure being created	Builder
Traversal Algorithm	Iterator
Object interfaces	Adapter
Object behavior	Decorator, State

Design Studios

Objective is to share your design with others to communicate the approach or to leverage more eyes on a problem.

- **Minute or so to set up…**
- **5-6 minute discussion**
- **1-2 minute answering questions**

1. **Team 2.2 - Rovio**

Homework and Milestone Reminders

- Continue Reading Chapter 26 on Gang of Four (GoF) Design Patterns
- Milestone 4 – Junior Project Design with More GRASP'ing
  - Due by 11:59pm on Friday, January 28th, 2011
- Homework 5 – BBVS Design using more GRASP Principles
  - Due by 11:59pm Tuesday, January 25th, 2011