Design Patterns in Python for the Untrained Eye

Material for the tutorial:

bit.ly/2V819eq

Ariel Ortiz

Full time faculty member

Tecnológico de Monterrey

Mexico

What is a Pattern?

Patterns capture experiences in software development that have been proven to work again and again, and thus provide a solution to specific problems.

They are not invented, they are discovered.

Categories by Abstraction Level

- Idioms
- Architectural Patterns
- Design Patterns

Relevance of Design Patterns

- Tried and tested solutions
- Common language

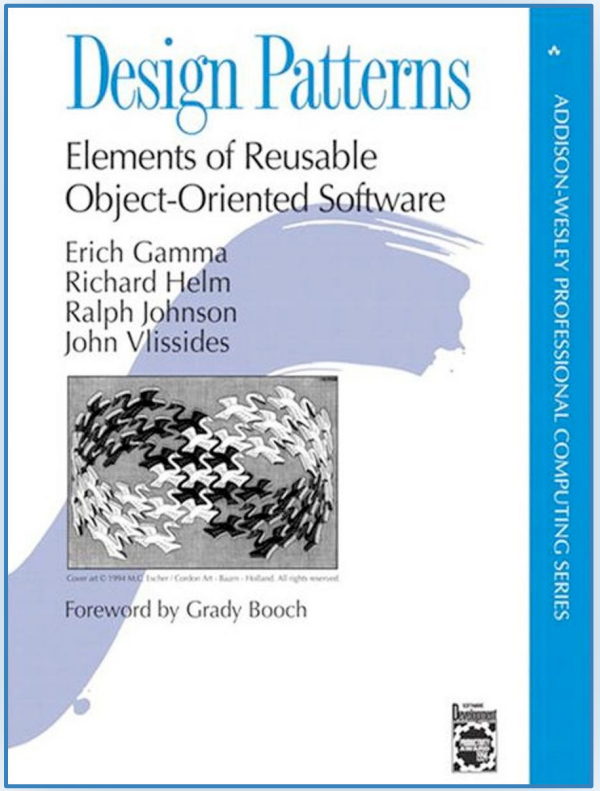
Limitations of Design Patterns

- Unjustified Use
- Kludges for a Weak Programming Language
- Inefficient Solutions

A Bit of History

The Gang of Four

GoF



23 Design Patterns Classified Based on their Use

- Creational Pattern
- Structural Patterns
- Behavioral Patterns

Design Principles

Design Principles

1. **Separate out the things that change from those that stay the same**

2. Program to an interface, not an implementation

Design Principles

3. Prefer composition over inheritance

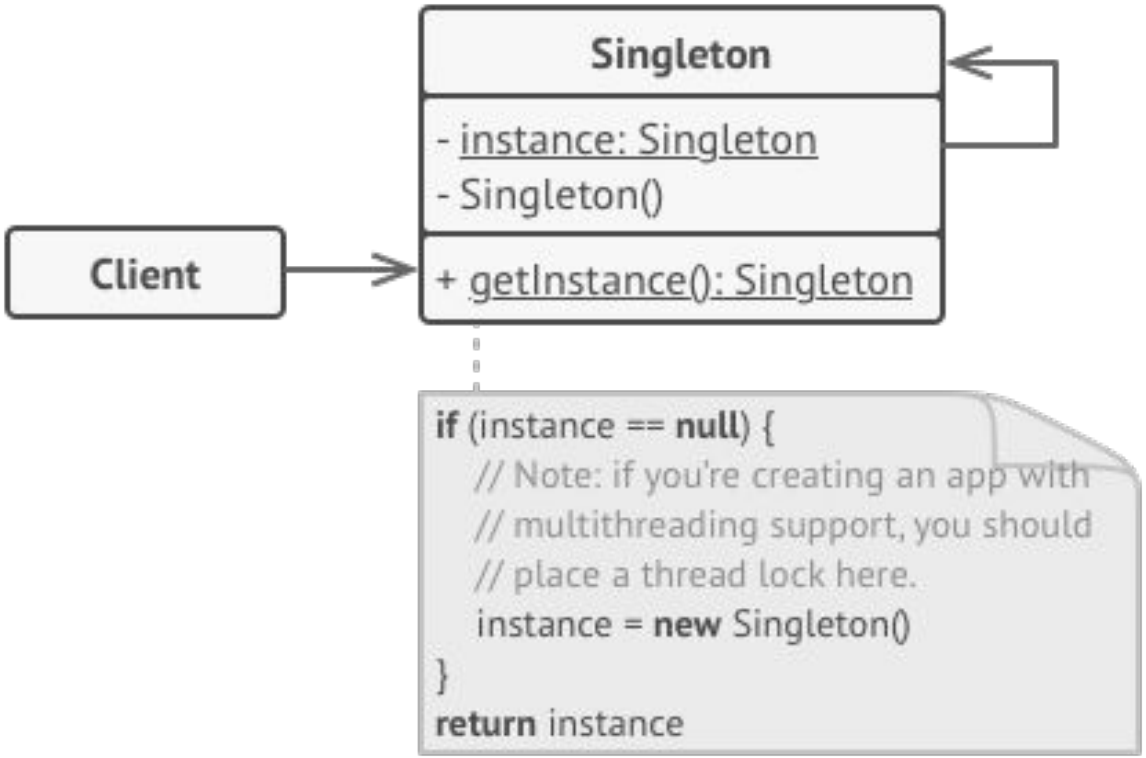
4. Delegation

Anatomy of a Design Pattern

- Intent
- Motivation
- Structure
- Implementation

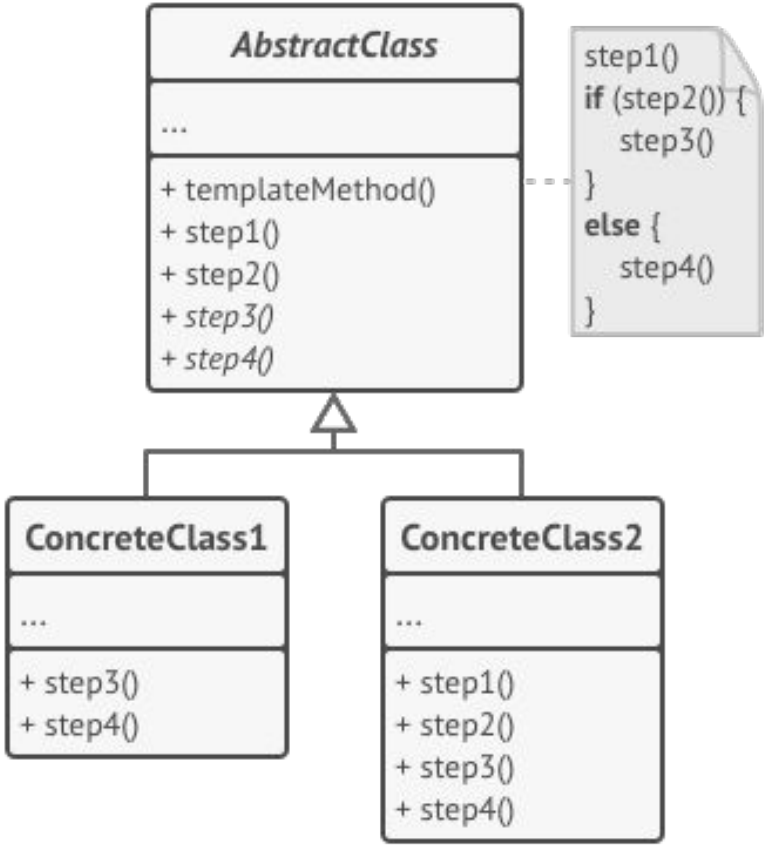
Singleton Pattern

Singleton *is a* ***creational design pattern*** *that lets you ensure that a class has only one instance, while providing a global access point to this instance.*


Singleton
- instance: Singleton
- Singleton()
+ getInstance(): Singleton
Client
if (instance == null) {
// Note: if you're creating an app with
// multithreading support, you should
// place a thread lock here.
instance = new Singleton()
}
return instance

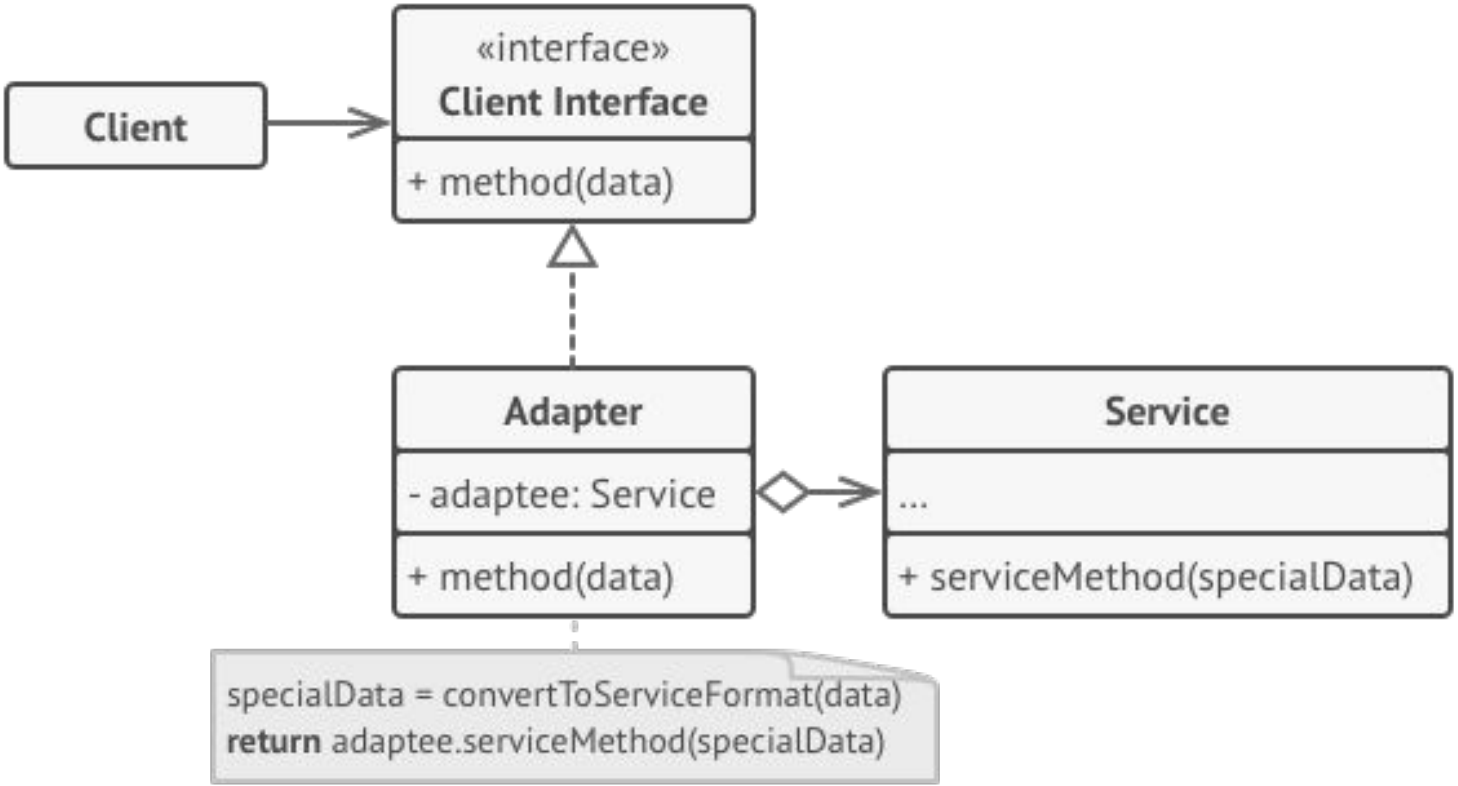
Template Method Pattern

Template Method *is a* ***behavioral design pattern*** *that defines the skeleton of an algorithm in the base class but lets derived classes override specific steps of the algorithm without changing its structure.*


AbstractClass
...
+ templateMethod()
+ step1()
+ step2()
+ step3()
+ step4()
step1()
if (step2()) {
step3()
}
else {
step4()
}
ConcreteClass1
...
+ step3()
+ step4()
ConcreteClass2
...
+ step1()
+ step2()
+ step3()
+ step4()

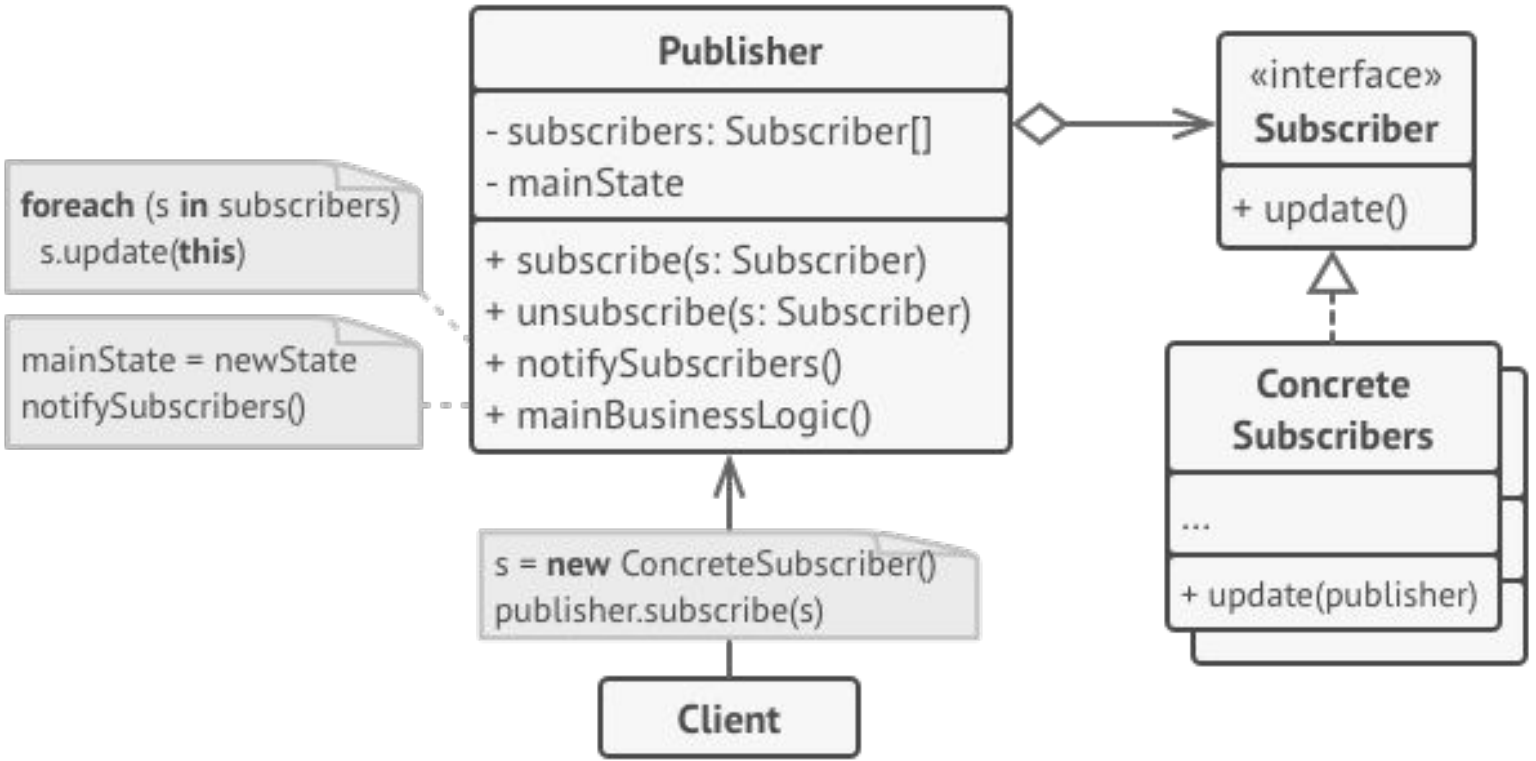
Adapter Pattern

Adapter *is a* ***structural design pattern*** *that converts the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.*


«interface»
Client Interface
+ method(data)
Client
Adapter
- adaptee: Service
+ method(data)
Service
...
+ serviceMethod(specialData)
specialData = convertToServiceFormat(data)
return adaptee.serviceMethod(specialData)

Observer Pattern

Observer *is a* ***behavioral design pattern*** *that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.*


Publisher
- subscribers: Subscriber[]
- mainState
+ subscribe(s: Subscriber)
+ unsubscribe(s: Subscriber)
+ notifySubscribers()
+ mainBusinessLogic()
«interface»
Subscriber
+ update()
foreach (s in subscribers)
s.update(this)
mainState = newState
notifySubscribers()
Concrete
Subscribers
...
+ update(publisher)
s = new ConcreteSubscriber()
publisher.subscribe(s)
Client

CONCLUSIONS

Thanks!

ariel.ortiz.ramirez@gmail.com

Credits

Special thanks to:

- Presentation template by SlidesCarnival
- Icons made by Freepik from Flaticon