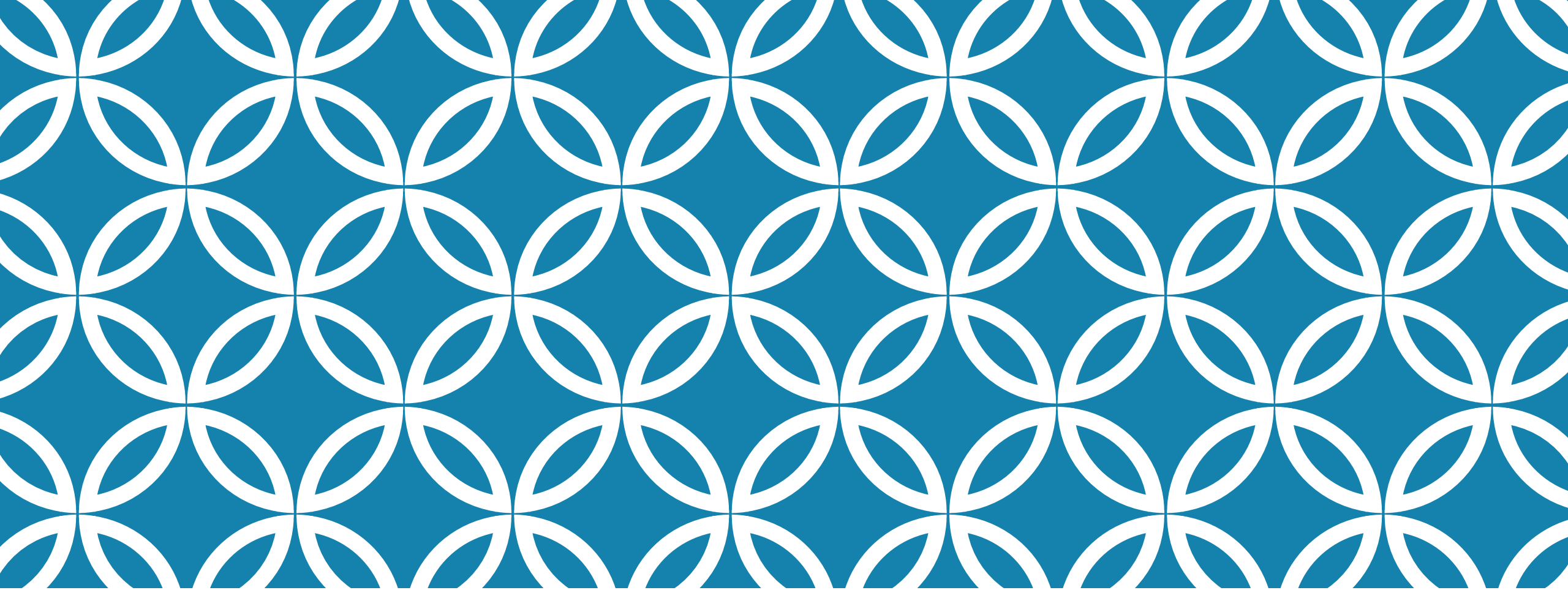
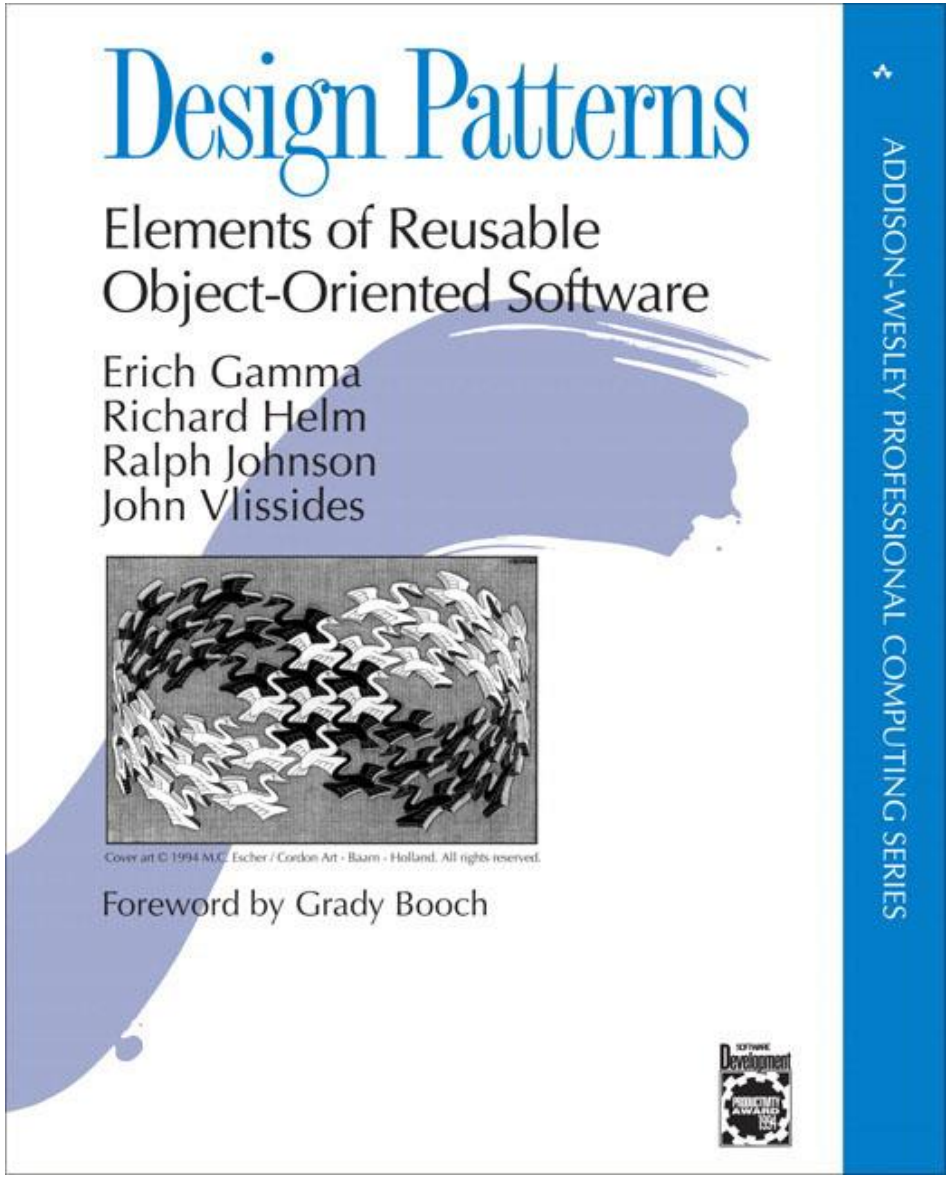


DESIGN PATTERNS IN C#
PART 2: STRUCTURAL PATTERNS

Trainer: Nadia Comanici

DESIGN PATTERNS – THE BOOK

- Published in 1994
- Gang of Four (GoF) = the authors
- You might need to read it twice ☺



WHAT ARE DESIGN PATTERNS?

- A design pattern is a recommended “recipe” to use in case of a certain problem
- Design patterns are:
  - independent of the programming language
  - simple, elegant & object-oriented solutions to a problem
  - not the first solution you would try (intuitively), because they were developed and evolved in time, to offer more flexibility and reusability
  - generally accepted by developers and used in programming

WHY USE THEM?

- Proven solutions, that work
- No need to reinvent the wheel, just use the well-known solution for your problem
- Common vocabulary for developers, easier to communicate and understand the needed solution
- Offer flexibility and reusability of code
- Make future changes more easier
- Object-oriented solutions

SO WHICH ARE THEY?

Scope	Creational	Structural	Behavioral
Class - relationships between classes (static + compile time)	Factory Method	Adapter	Interpreter
			Template Method
Object - relationship between objects (dynamic + runtime)	Abstract Factory	Bridge	Chain of Responsibility
	Builder	Composite	Command
	Prototype	Decorator	Iterator
	Singleton	Façade	Mediator
		Flyweight	Memento
		Proxy	Observer
			State
			Strategy
			Visitor

STRUCTURAL DESIGN PATTERNS

STRUCTURAL DESIGN PATTERNS

- TODO

1. ADAPTER

ADAPTER – WHAT DOES IT DO?

"Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces." (GoF)

ADAPTER – WHEN TO USE

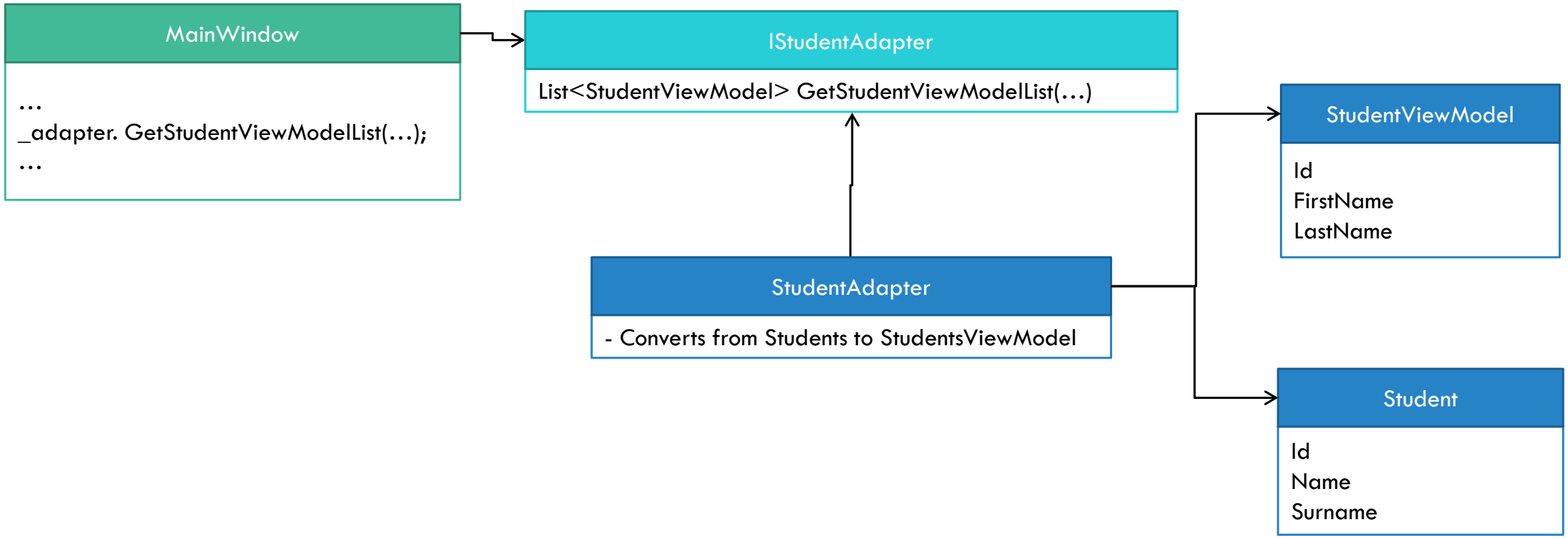
When you need to use a class T, but the interface of T is not the expected one

- **And you don't have control/rights over the T class, to change its interface**
- Interface = public data (properties, fields, methods)

Examples:

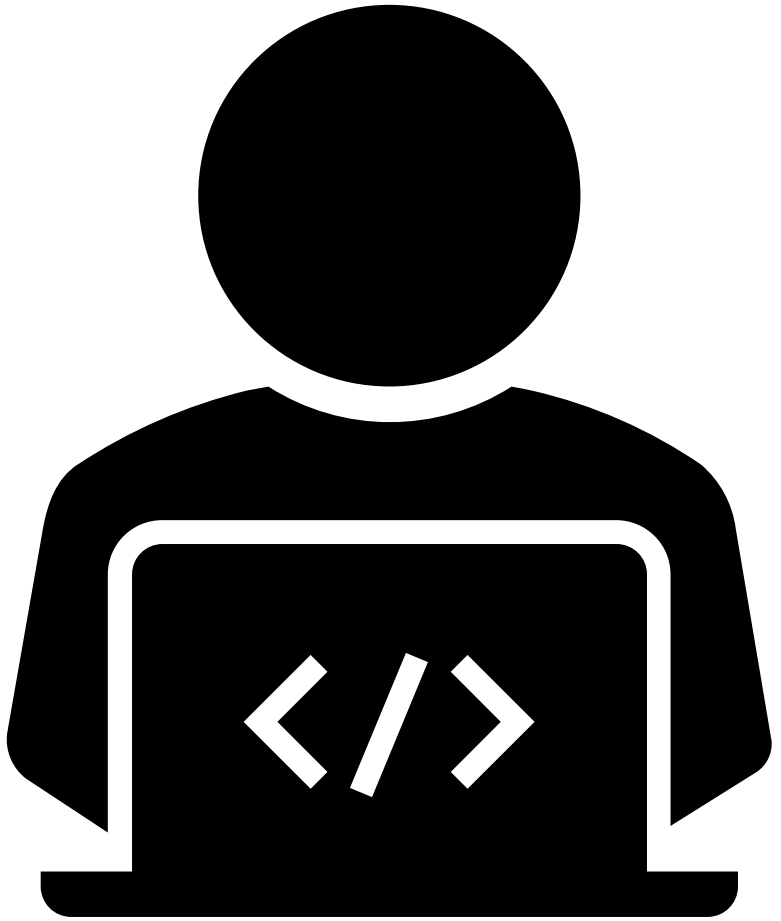
- Model mapped over database table has different structure than the model used in UI
- To create wrappers for a framework class that doesn't implement the interface expected by the domain.
- Create a reusable class, that wraps over existing or future classes, that might not have compatible interfaces

ADAPTER – DIAGRAM – STUDENT

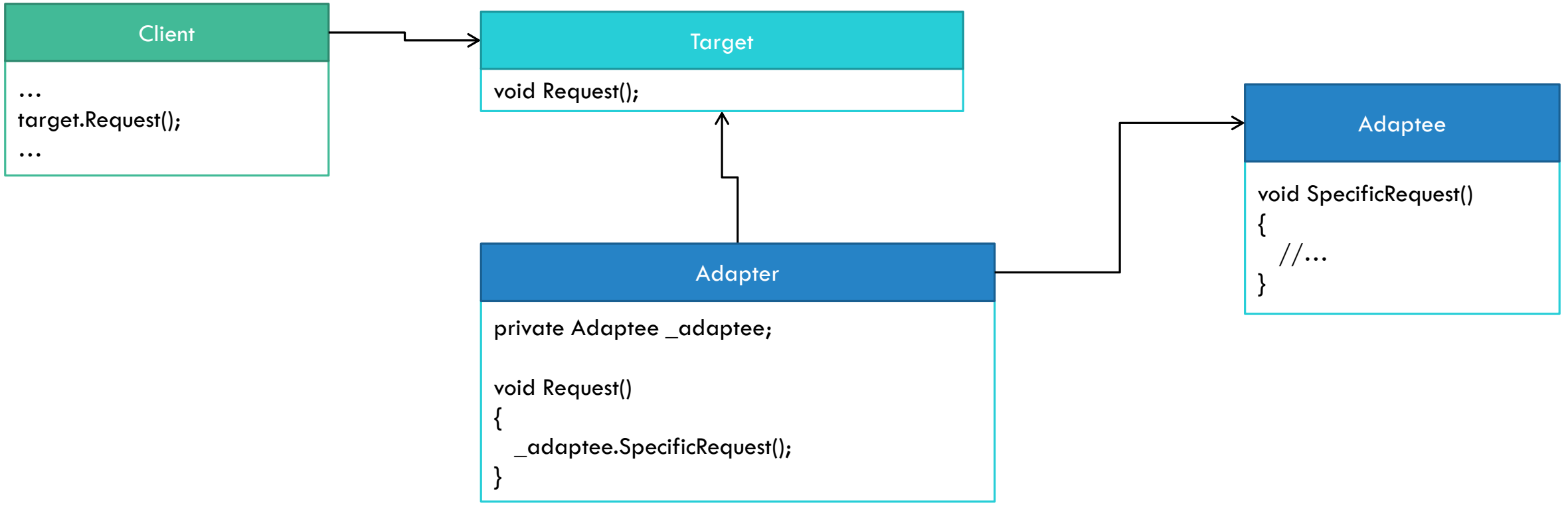


DEMO

Adapter - Student



ADAPTER – DIAGRAM



ADAPTER – VARIANTS YOU MIGHT FIND

1. An adapter class for each combination of 2
   - Methods: ConvertStudentToStudentViewModel + ConvertStudentViewModelToStudent
   - It would be better to have a class for each combination (Single Responsibility Principle)
   - Useful if we need additional methods, too, for this combination of 2

2. An adapter class for multiple combinations
   - Methods: ConvertStudentToStudentViewModel + ConvertStudentViewModelToStudent + ConvertTeacherToTeacherViewModel + ConvertTeacherViewModelToTeacher

3. Class with static methods vs class with non-static methods

4. Extension Methods

5. AutoMapper
   - Useful just for mapping, cannot add additional methods/functionality to the adapter

Q&A
ADAPTER

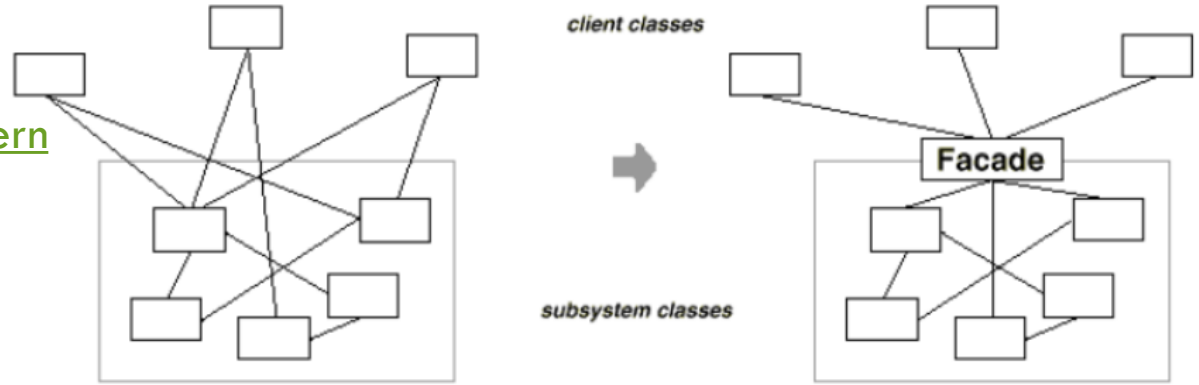
2. FACADE

FACADE – WHAT DOES IT DO?

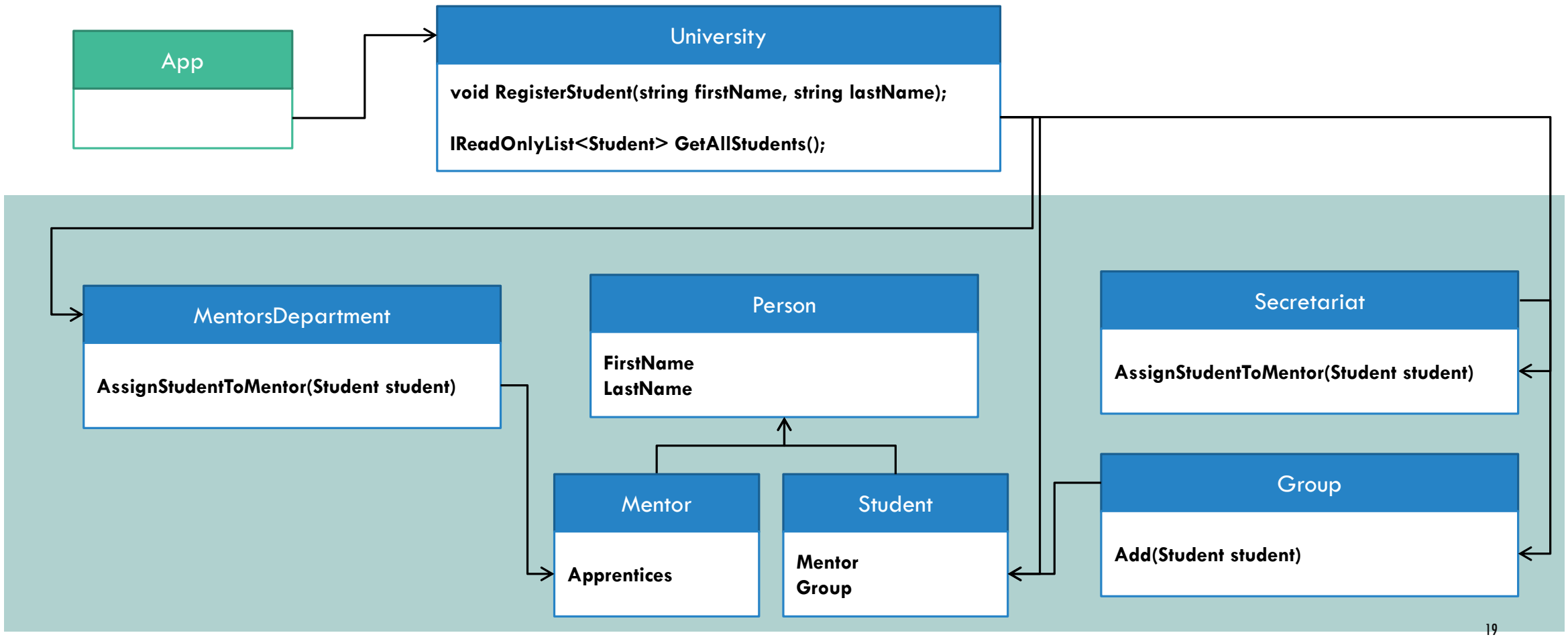
- “Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.” (GoF)

FACADE – WHEN TO USE

- Provide a simplified interface for a complex system, from which you need only part of it, for a certain purpose
- Expose multiple systems under a single interface
- Wrap poorly designed systems in a better designed one
- More:
  - https://refactoring.guru/design-patterns/facade
  - https://www.dofactory.com/net/facade-design-pattern

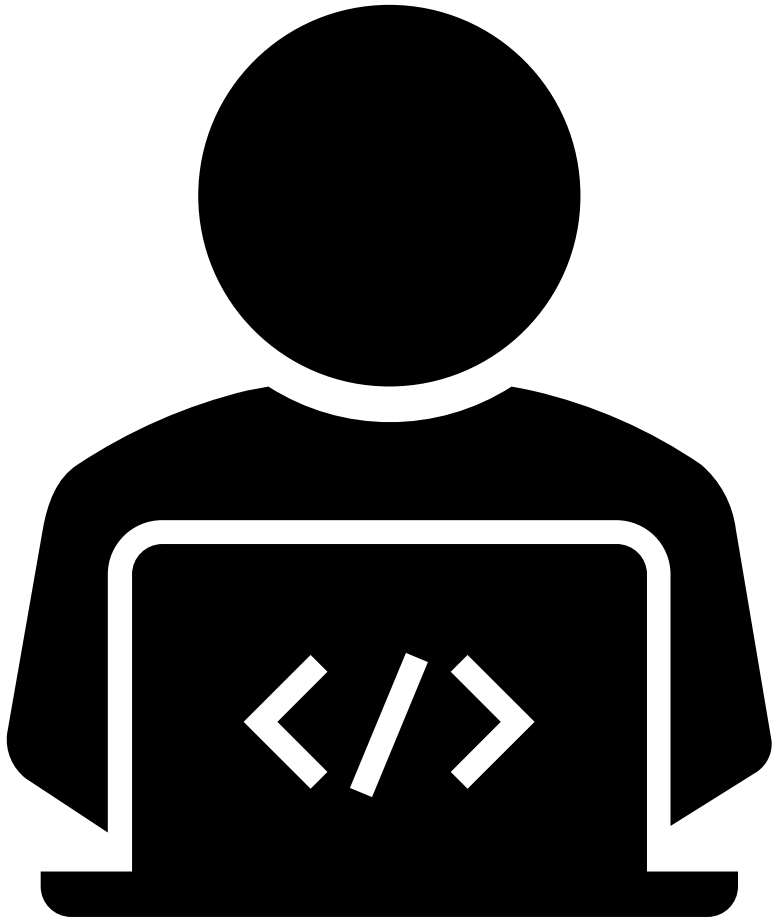


FACADE – DIAGRAM – UNIVERSITY

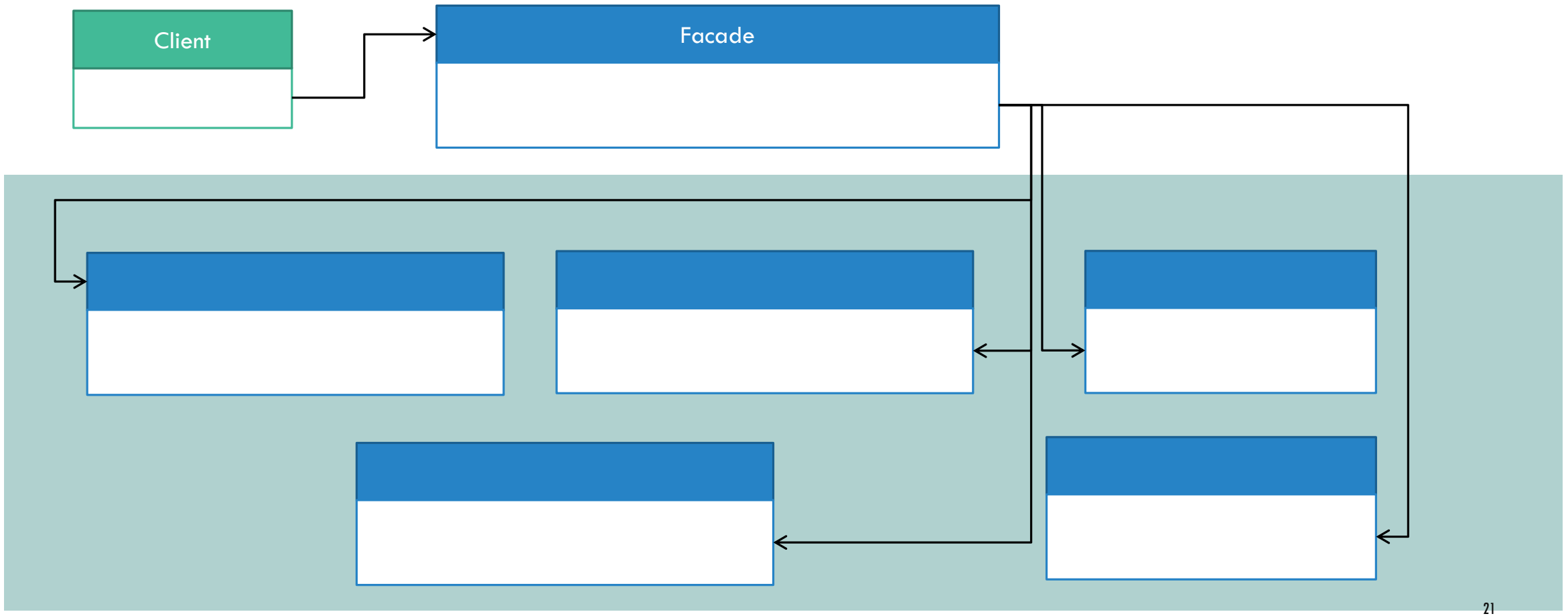


DEMO

Facade - University



FACADE – DIAGRAM



FACADE – ADVANTAGES

- Simplified interface, hides implementation details and connections between elements inside subsystem
  - Anti Corruption Layer
- You might already used it, but not know it has a name
- “Hides” legacy implementation / naming

FACADE – DISADVANTAGES

- Can have “God” classes (see Single Responsibility Principle)

Q&A
FACADE

3. PROXY

PROXY – WHAT DOES IT DO?

"Provide a surrogate or placeholder for another object to control access to it." (GoF)

Examples:

- https://www.dofactory.com/net/proxy-design-pattern
- https://refactoring.guru/design-patterns/proxy/csharp/example
- https://exceptionnotfound.net/proxy-pattern-in-csharp/

PROXY – DESCRIPTION

- A proxy is an object that can be used as a replacement for the real object used by a client.
- The proxy hides the actual real object and whenever receives a call, it does some specific action and then forwards calls to the real object
- The proxy must have the same interface as the real object, and thus it is interchangeable with the real one
- The proxy can use lazy loading for creating the real object

PROXY – WHEN TO USE

- You need a placeholder for an actual object that is expesive to create
  - Display an image – while the actual image is being fetched, you can use a proxy and display a "please wait" message
- You need to provide a local object that stands in place for a remote object and acts in the same way
  - If you access a service over the network, but want to hide the actual networking details
- When you want to add some additional behaviors to an object of some existing class, without modifying the client code
- The proxy might use lazy loading, in order to postpone expensive calls until they are first time actually needed

LAZY LOADING

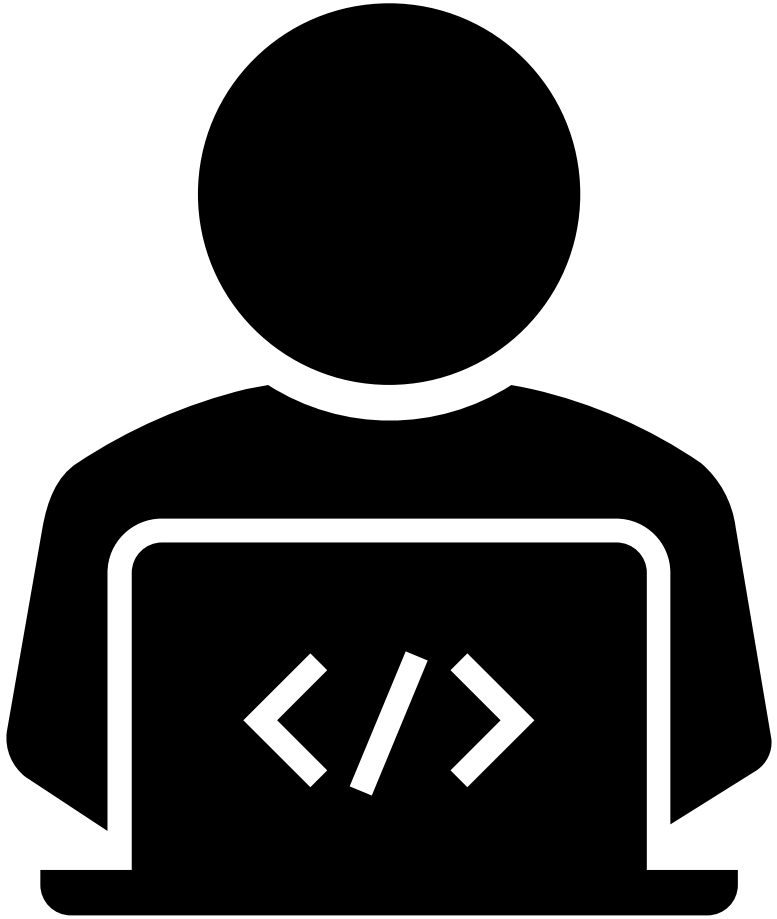
- Code optimization – fetching objects state from persistence only when it is requested by the client code
- Instead of loading everything from the beginning, it returns the information only when it is first time actually needed
- ORM usually have a way of defining which properties to load lazy or not, in order to optimize the application load at startup (or calls, in general)

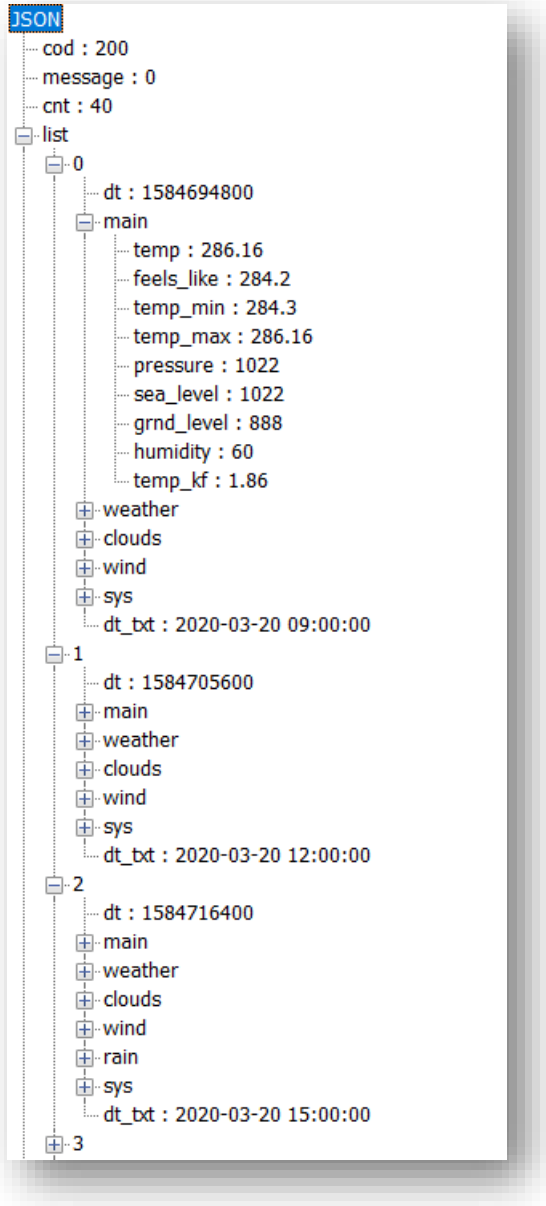
PROXY - TYPES

1. *Remote proxies*
   - A local replacement of a remote object, which hides the details of communicating with the remote object
   - Are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
2. *Virtual proxies*
   - Used to create expensive objects on demand
   - May cache additional information about the real subject so that they can postpone accessing it.
3. *Protection proxies*
   - Checks that the caller has the access permissions required to perform a request from the real object.

DEMO

Proxy – Weather

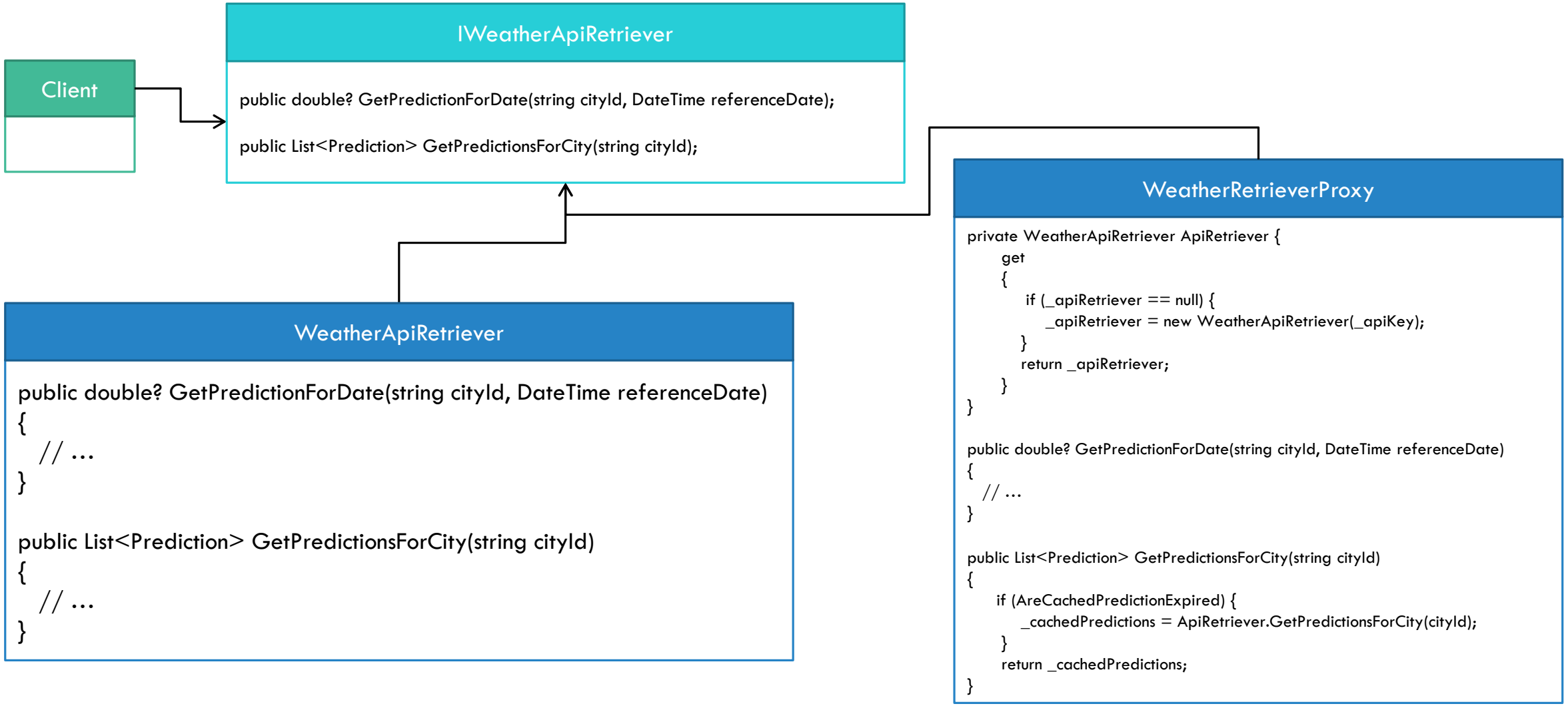




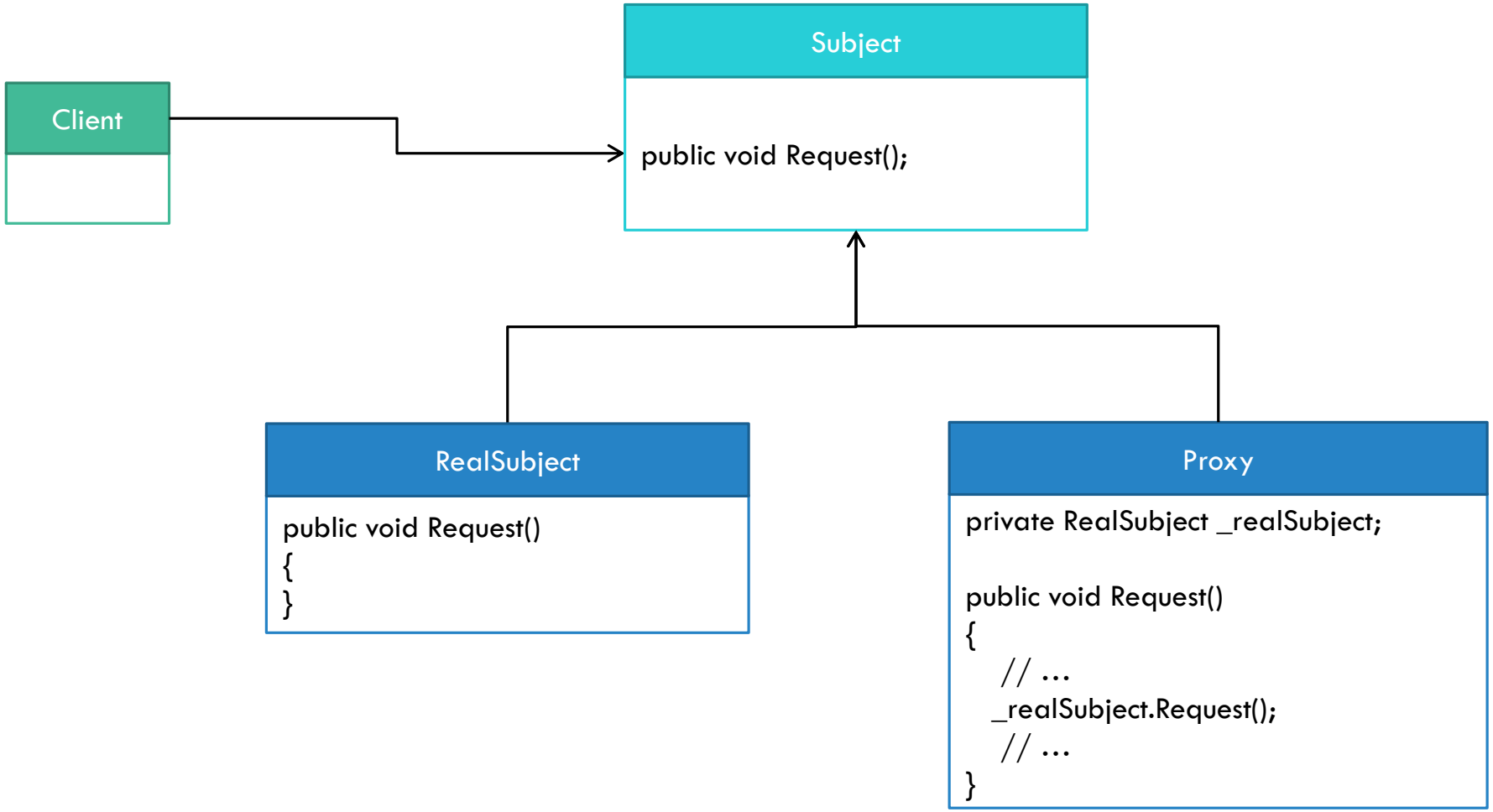
Useful links:

- https://openweathermap.org/appid

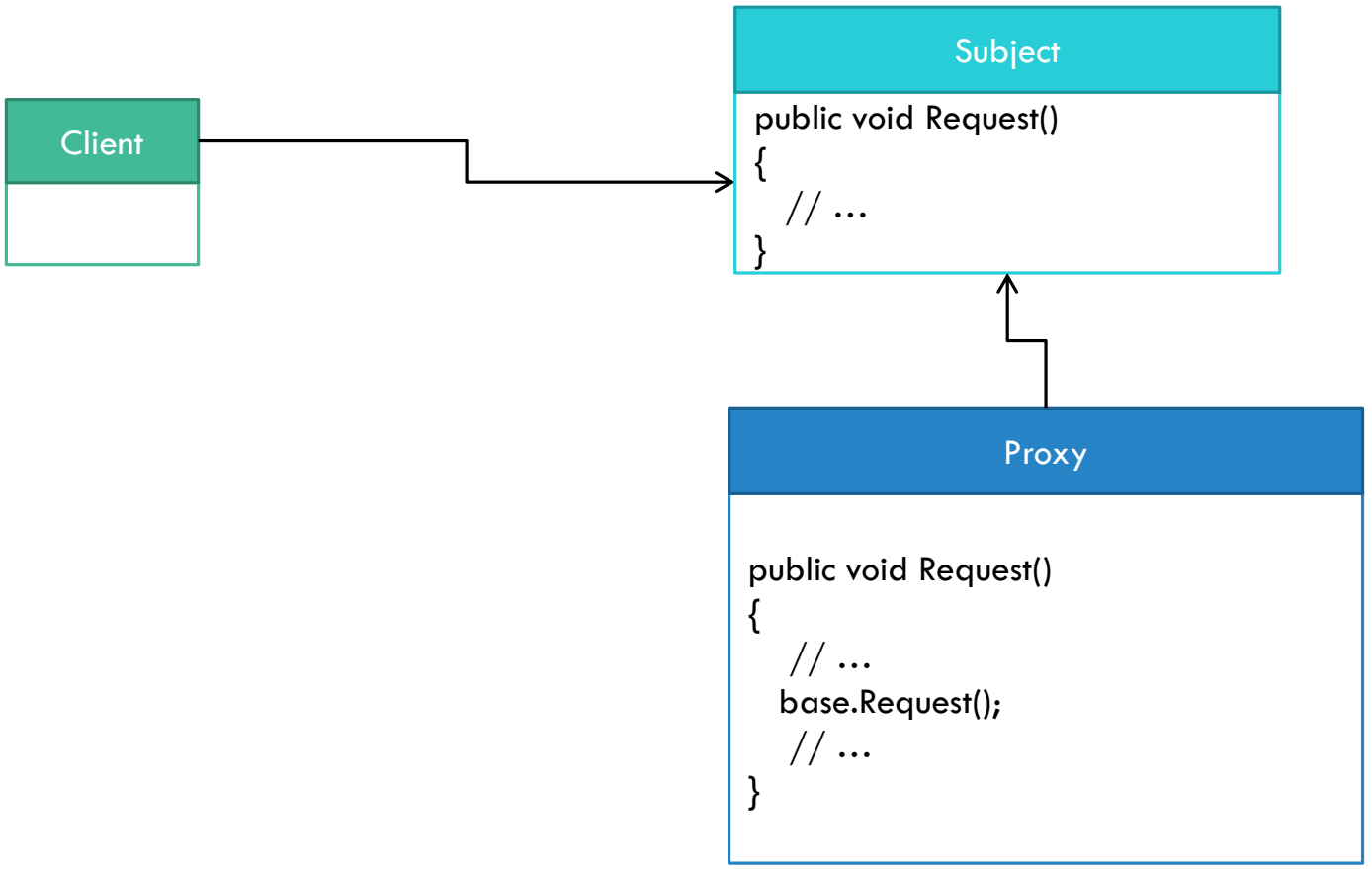
PROXY – DIAGRAM – WEATHER



PROXY – DIAGRAM



PROXY – DIAGRAM (ALTERNATE)



PROXY – ADVANTAGES

- Control access to an object in order to delay expensive operations and thus improve application performance
- Encapsulate access to a remote object

Q&A
PROXY