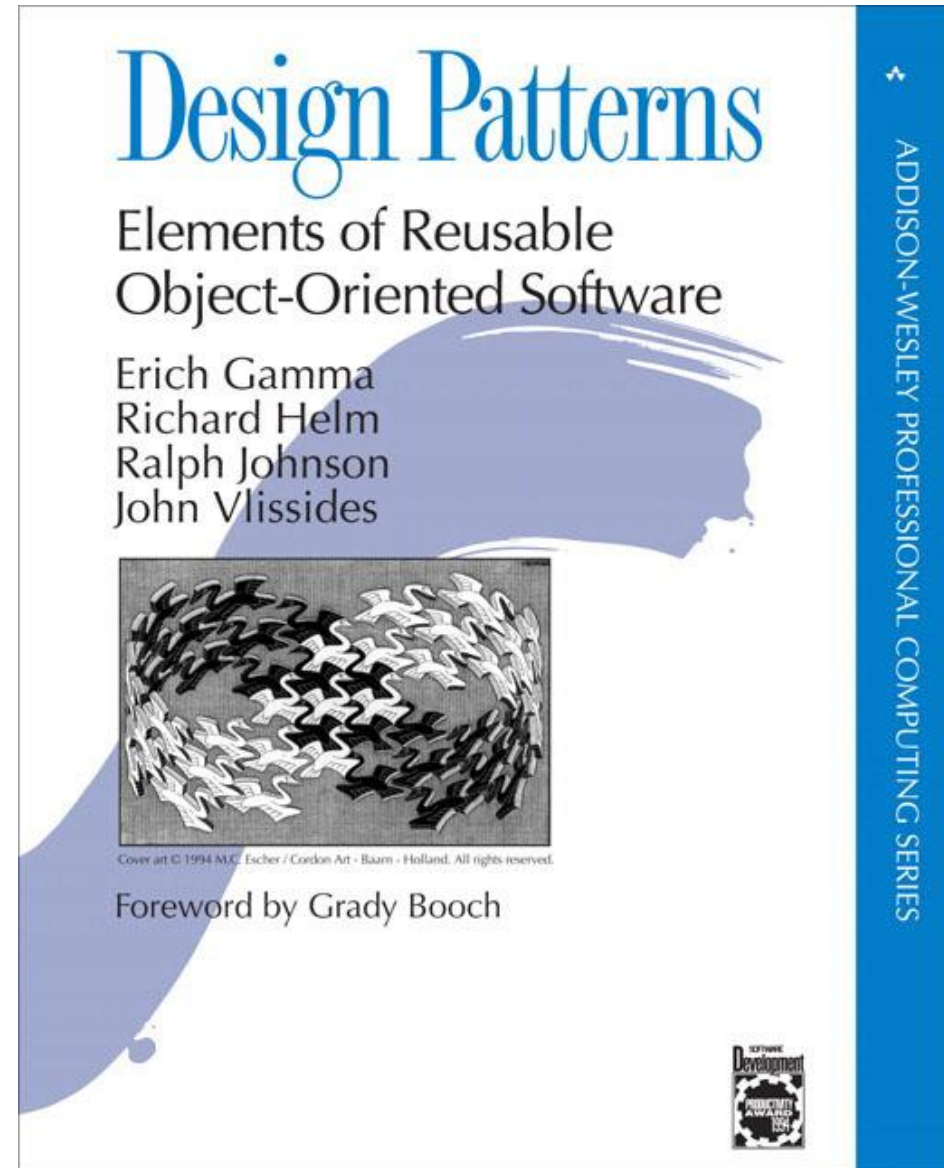# DESIGN PATTERNS IN C#
PART 2: STRUCTURAL PATTERNS

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)
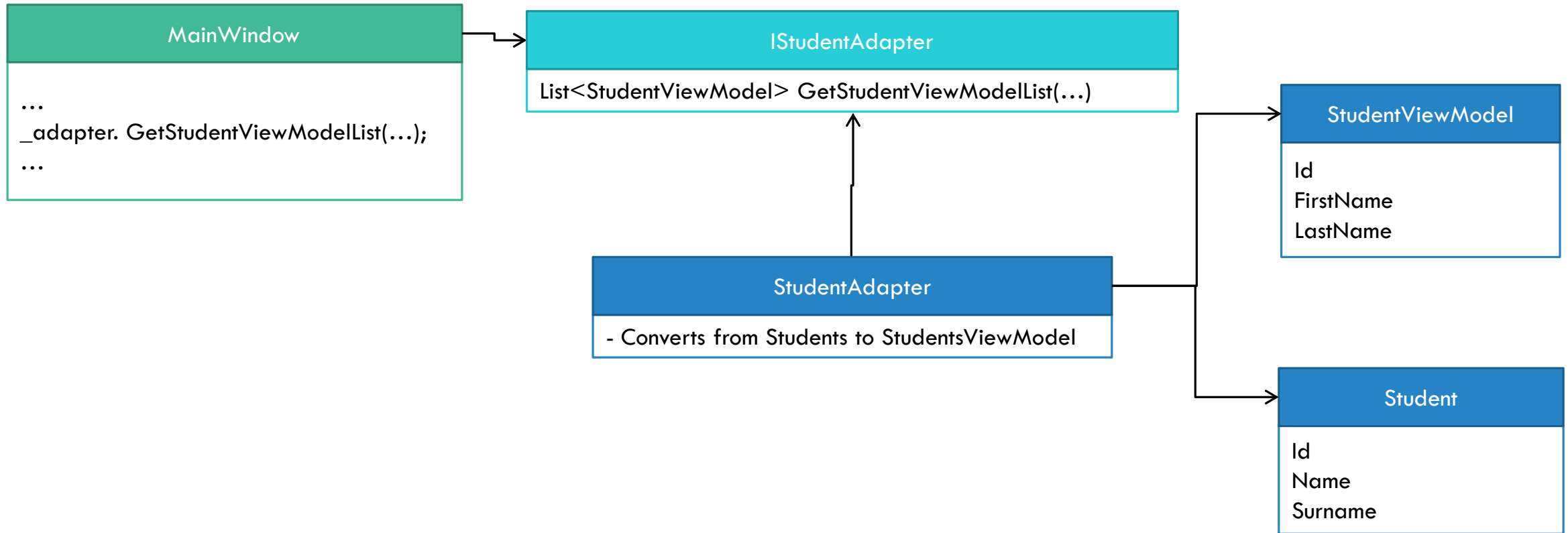
ADAPTER – WHEN TO USE

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**
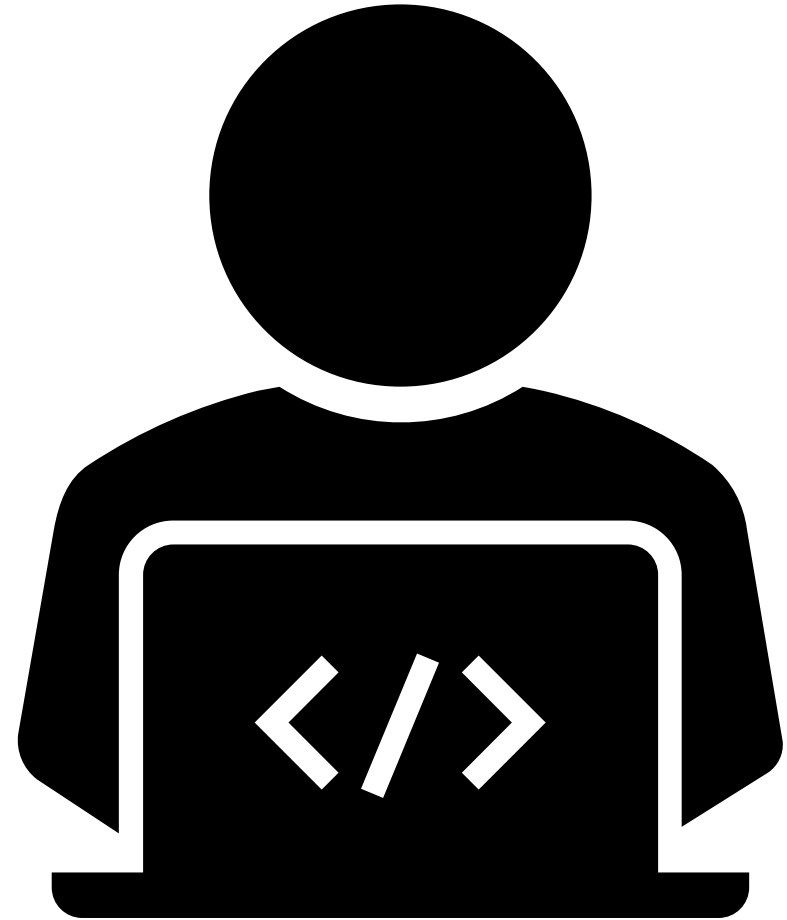- Interface = public data (properties, fields, methods)

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
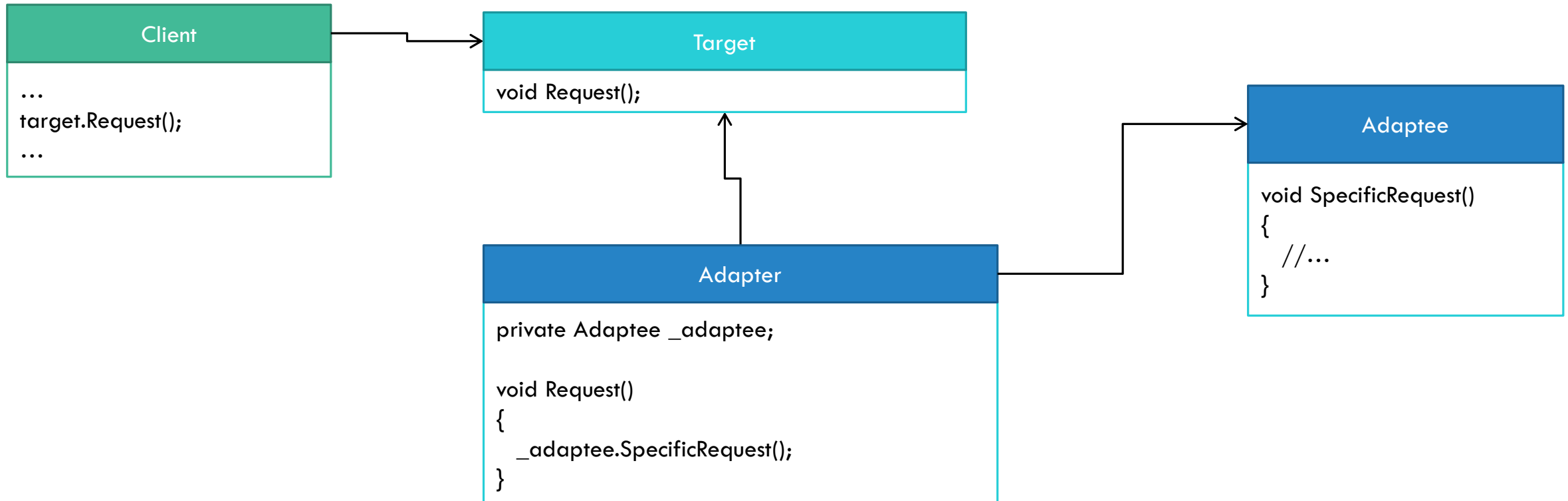
ADAPTER – DIAGRAM – STUDENT

MainWindow

...
_adapter. GetStudentViewModelList(...);
...

IStudentAdapter

List<StudentViewModel> GetStudentViewModelList(...)

StudentAdapter

- Converts from Students to StudentsViewModel

StudentViewModel

Id
FirstName
LastName

Student

Id
Name
Surname

DEMO

Adapter - Student

ADAPTER – DIAGRAM

Client
… target.Request(); …

Target
void Request();

Adapter

private Adaptee _adaptee;
 	
void Request()
{
    _adaptee.SpecificRequest();
}

Adaptee

void SpecificRequest()
{
    //…
}

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
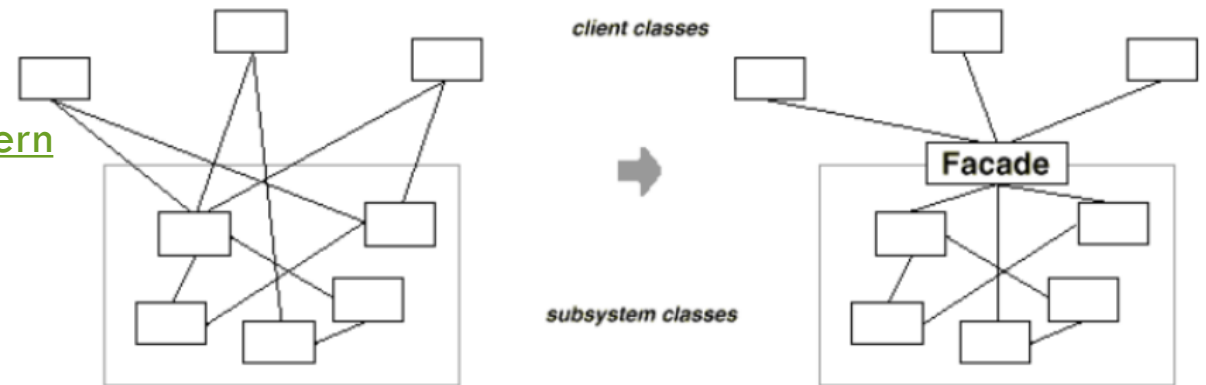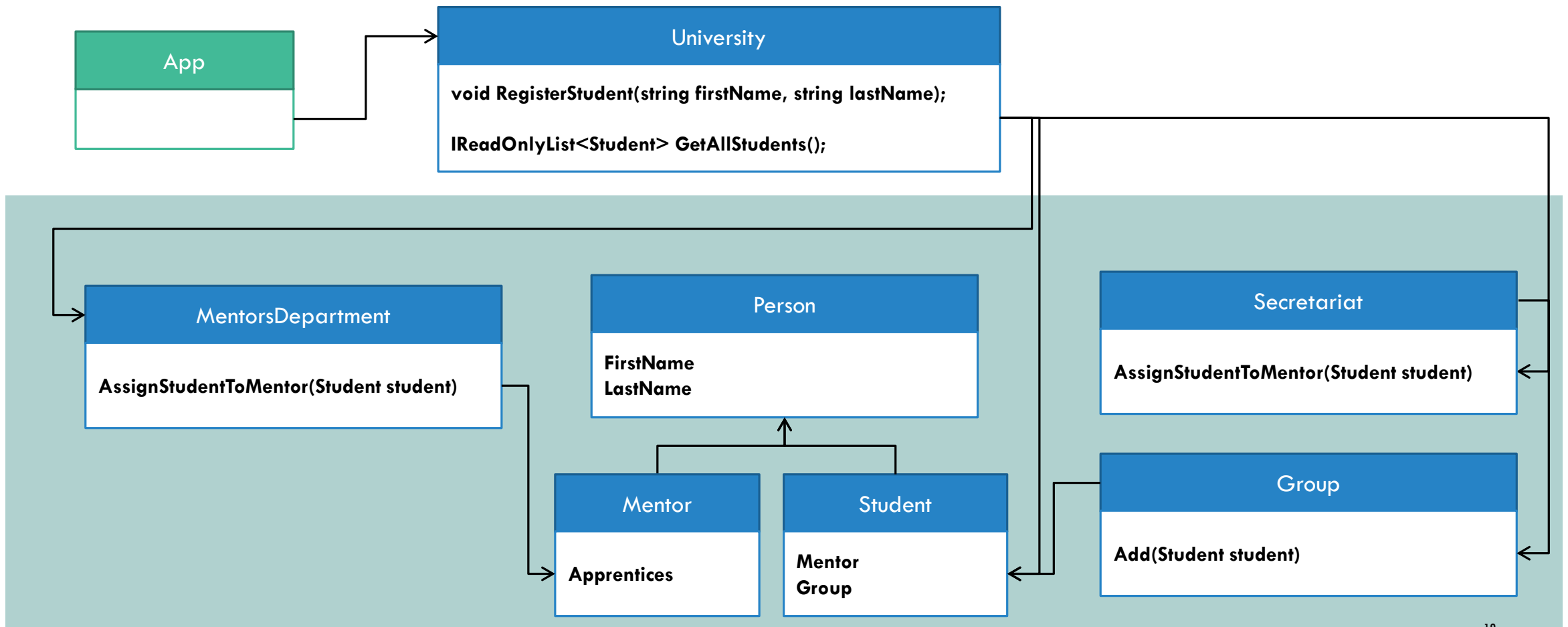
Q&A
ADAPTER

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

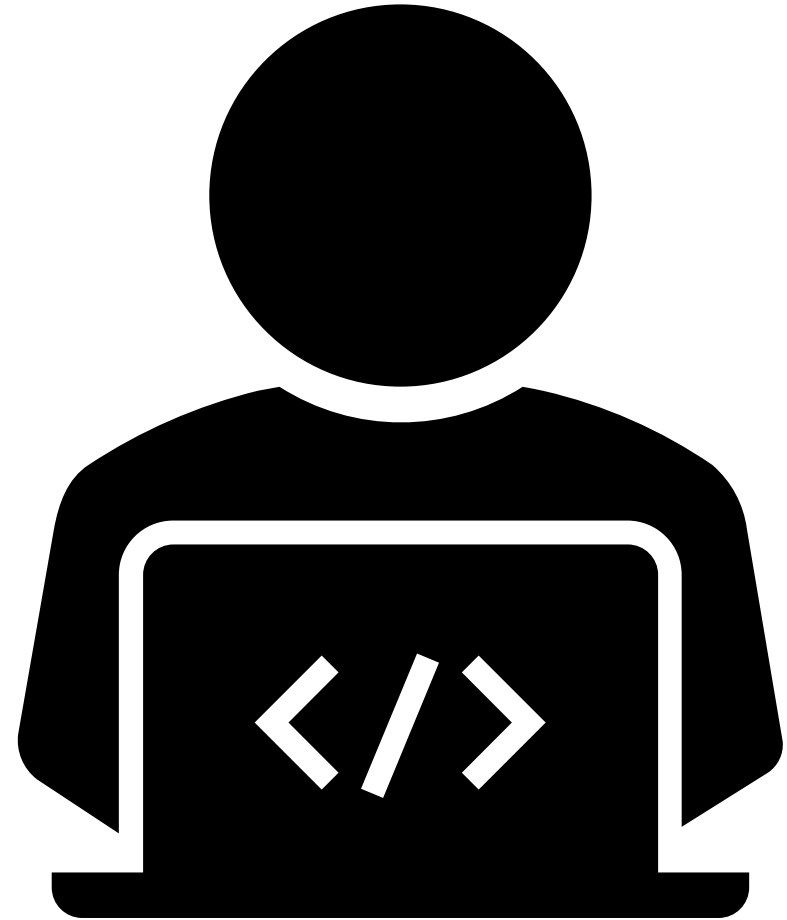- Wrap poorly designed systems in a better designed one

- More:
  - https://refactoring.guru/design-patterns/facade
  - https://www.dofactory.com/net/facade-design-pattern

FACADE – DIAGRAM – UNIVERSITY

App

University

void RegisterStudent(string firstName, string lastName);

IReadOnlyList<Student> GetAllStudents();

MentorsDepartment

AssignStudentToMentor(Student student)

Person

FirstName
LastName

Mentor

Apprentices

Student

Mentor
Group

Secretariat

AssignStudentToMentor(Student student)
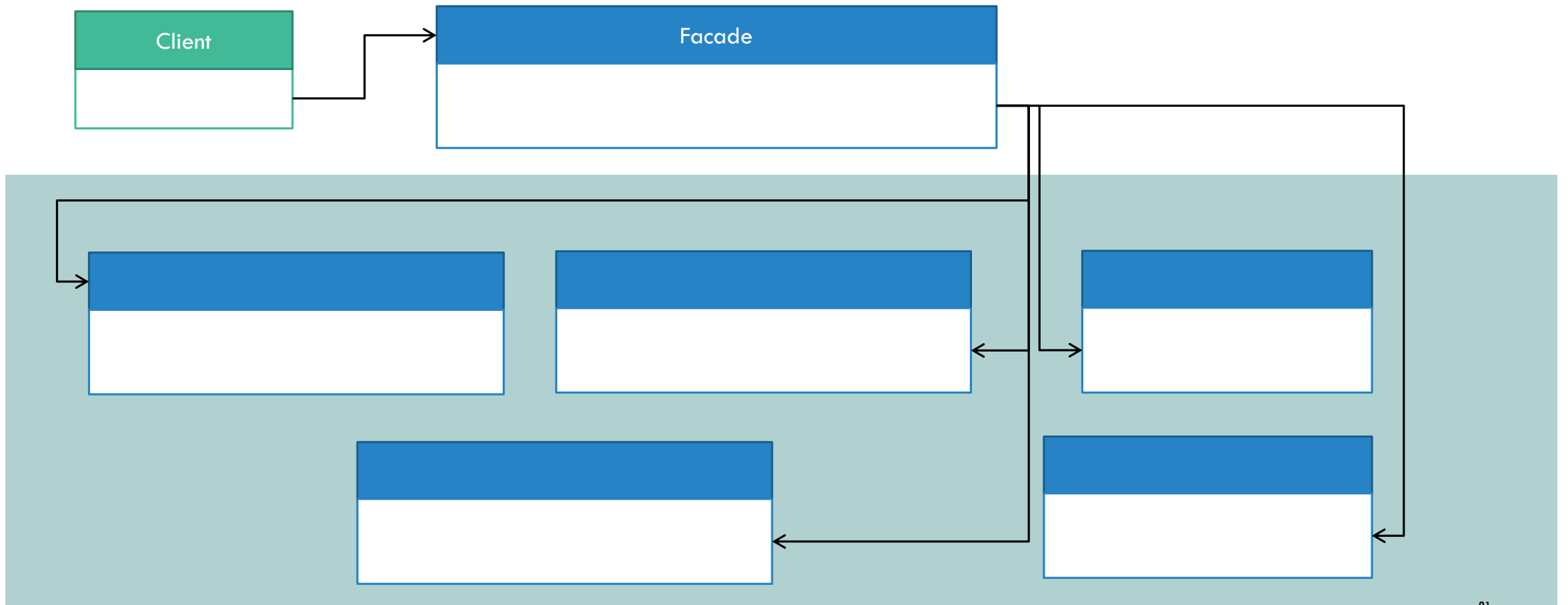
Group

Add(Student student)

DEMO

Facade - University

FACADE – DIAGRAM

Client

Facade

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