Information System Design

Lecture 2: Object Oriented Analysis and Design

Dr. Moustafa Alzantot

Programming vs Chess

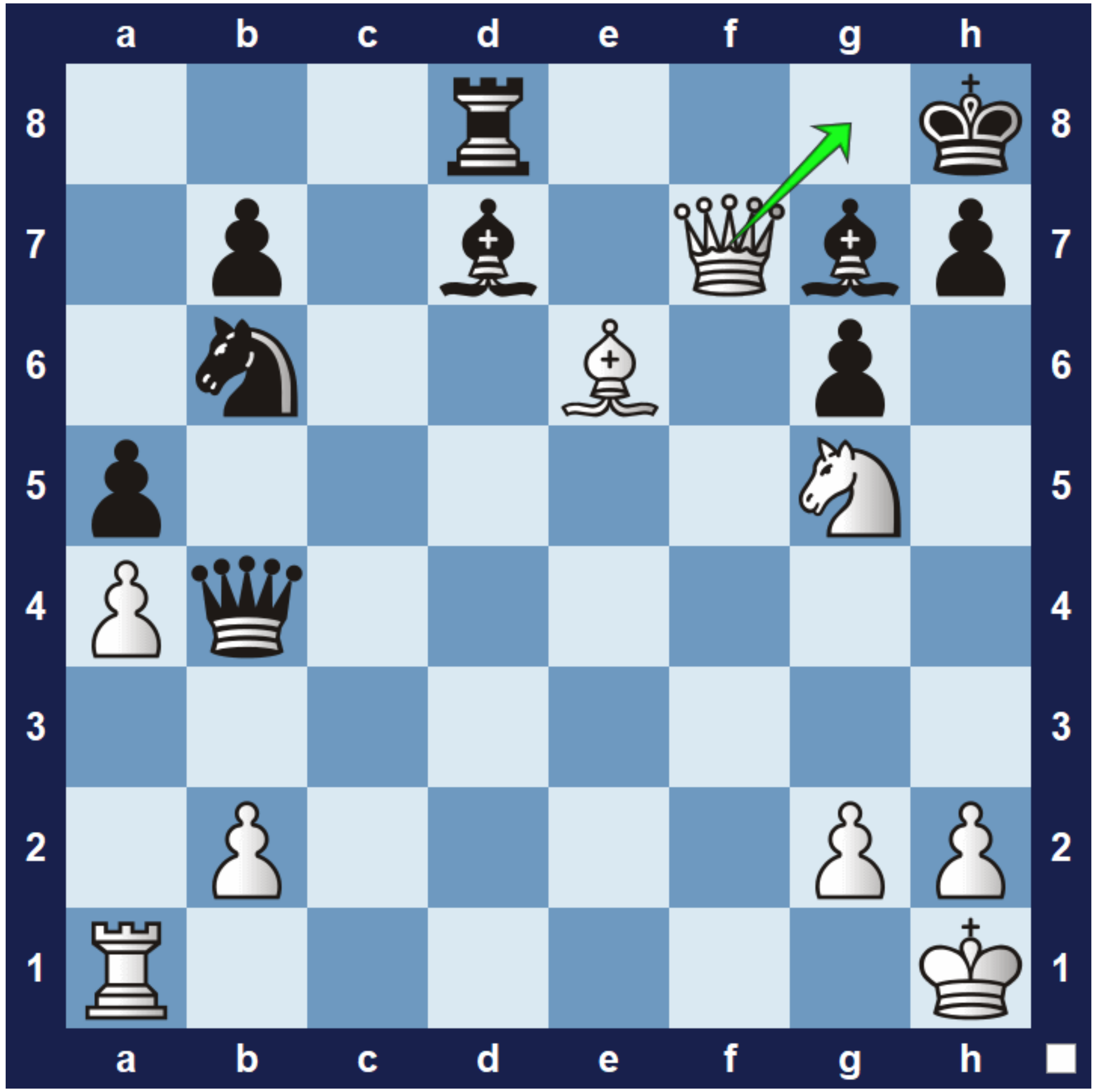
- Learning how to program is quite similar to learning how to play chess.
- You first learn how to move pieces (i.e. writing pieces of working code)
- But to become a professional player, you need to do a lot of handwork.

Programming vs Chess

- Chess benefits from imagination and visualization chess.
  - *So does programming.*

Programming vs Chess

- ***And both involve a lot of pattern recognition.***
  - *Identifying if a given position (or problem) is similar to a previously seen and you know the best move (or how to solve)*



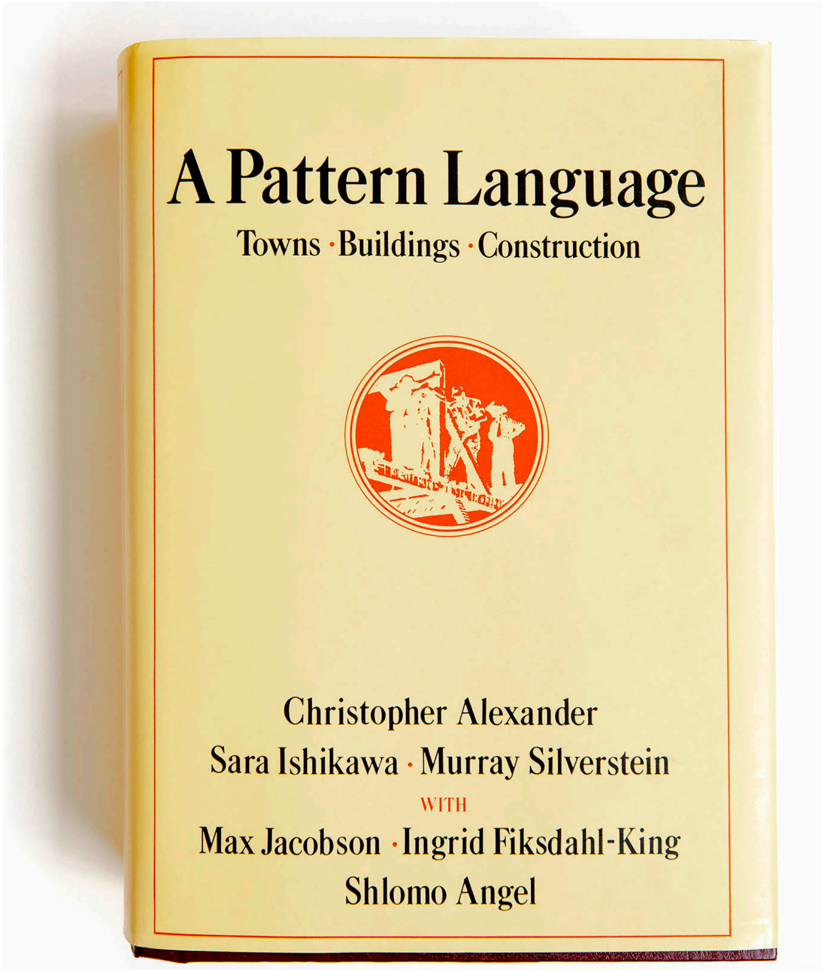
Software Design Patterns

- ***Design Pattern***: a practical proven solution to a recurring design problem.
- Design patterns are ***not*** pieces of code in any programming language, but they act as templates that you can follow to solve a problem.
- They are highly optimized solutions designed and revised by experts.

History of Design Patterns

- The concept of “***Patterns***” originated in architecture. Often attributed to Christopher Alexander, an American architect who wrote a book “**A Pattern Language”** in 1977.

  “ … each pattern represents our current best guess as to what arrangement of the physical environment will work to solve the problem presented.”


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

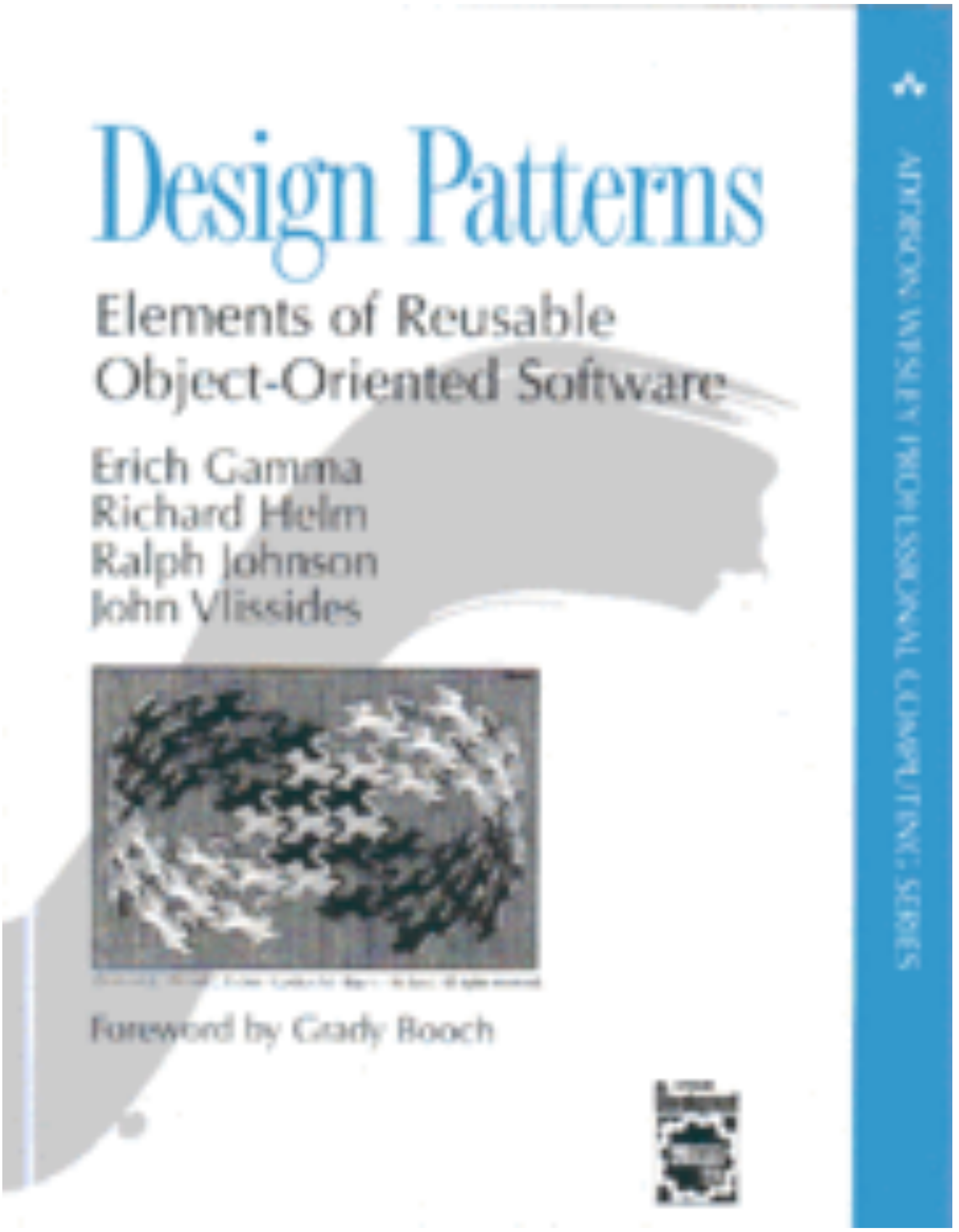
History of Design Patterns

Software Design Patterns

- In a 1995, four authors wrote a book about design patterns for software.

Design Patterns: Elements of Reusable Object-Oriented Software

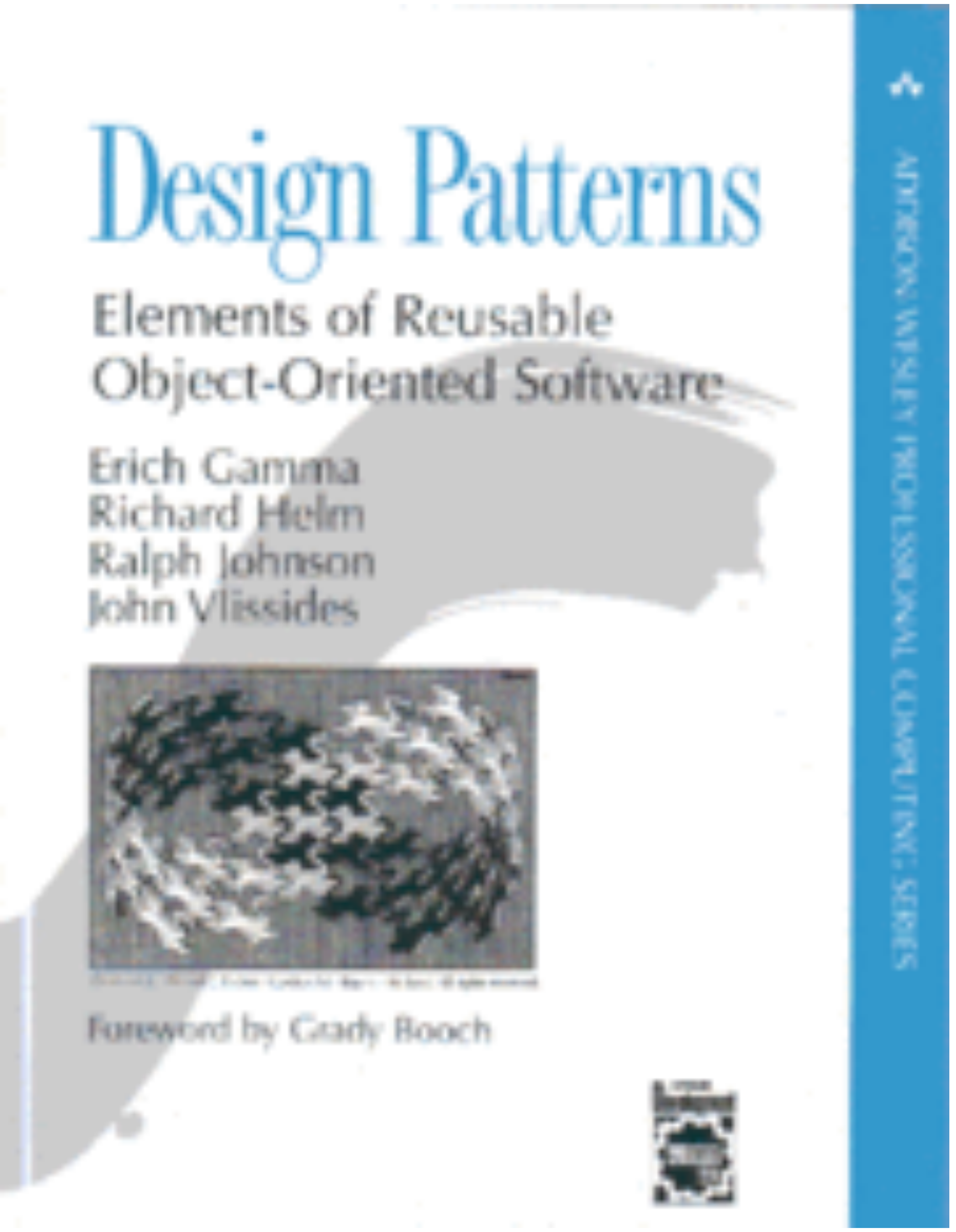
Widely known as the “***Gang of Four (GoF)***” book.


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

History of Design Patterns

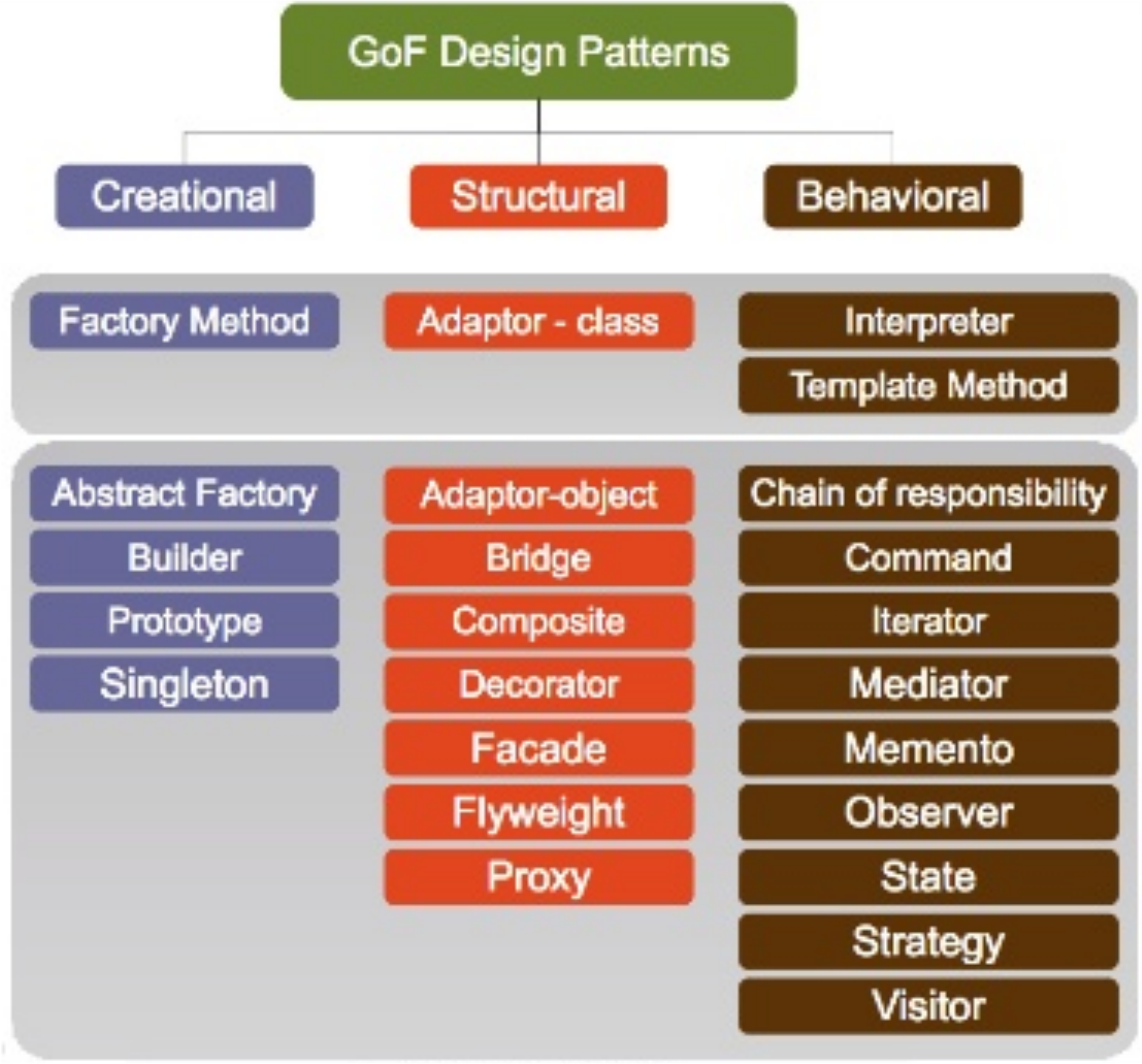
Software Design Patterns

- The GoF book defined 23 useful design patterns for recurring situations.
- These design patterns are categorized by their purpose into 3 categories:
    - **Creational**
    - **Structural**
    - **Behavioral**


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

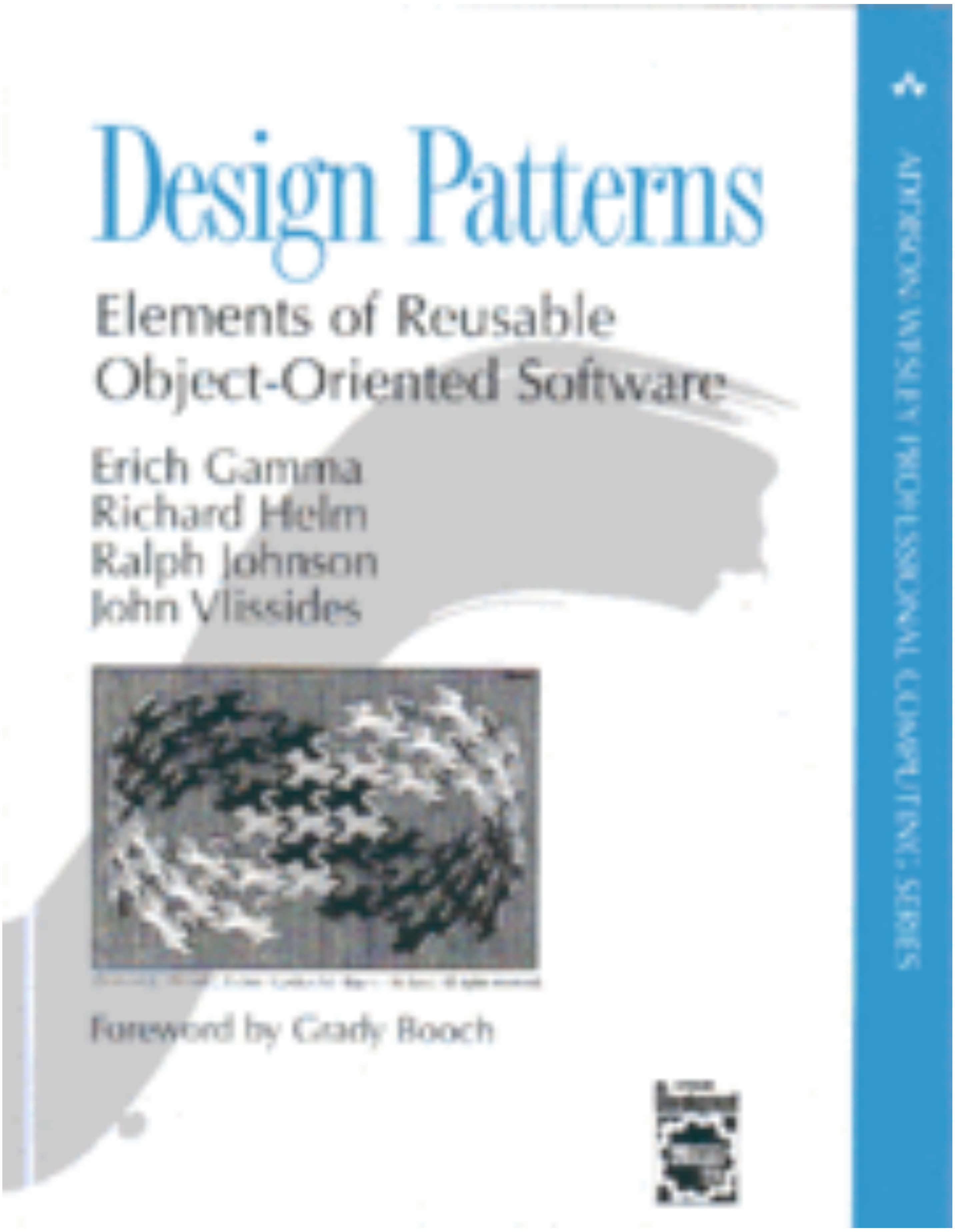
History of Design Patterns

Software Design Patterns



GoF Design Patterns

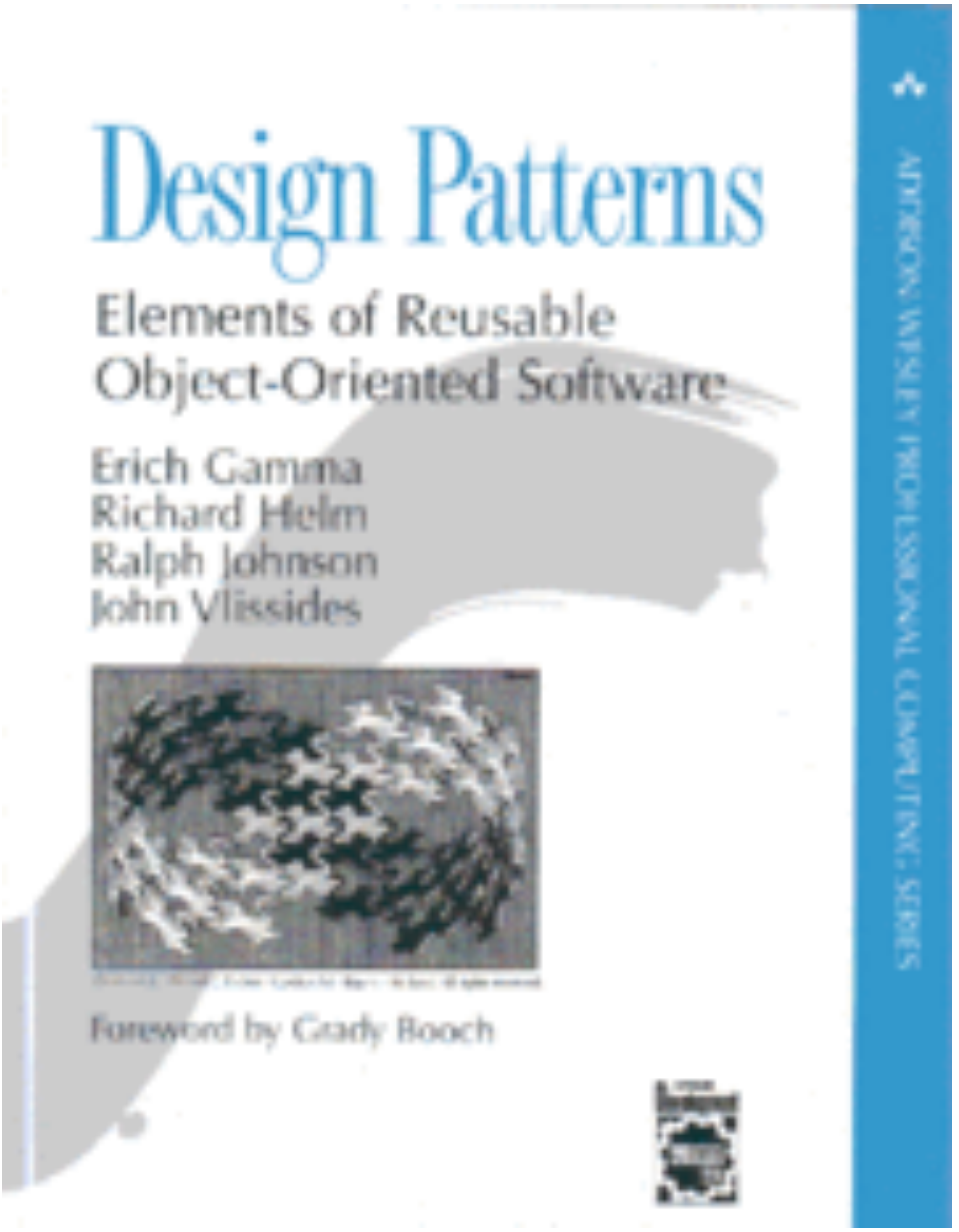
Creational	Structural	Behavioral
Factory Method	Adaptor - class	Interpreter
		Template Method
Abstract Factory	Adaptor-object	Chain of responsibility
Builder	Bridge	Command
Prototype	Composite	Iterator
Singleton	Decorator	Mediator
	Facade	Memento
	Flyweight	Observer
	Proxy	State
		Strategy
		Visitor


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

History of Design Patterns

Software Design Patterns

- More design patterns emerged later.


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Reasons for using Design Patterns

Software Design Patterns

Why to use design patterns ?

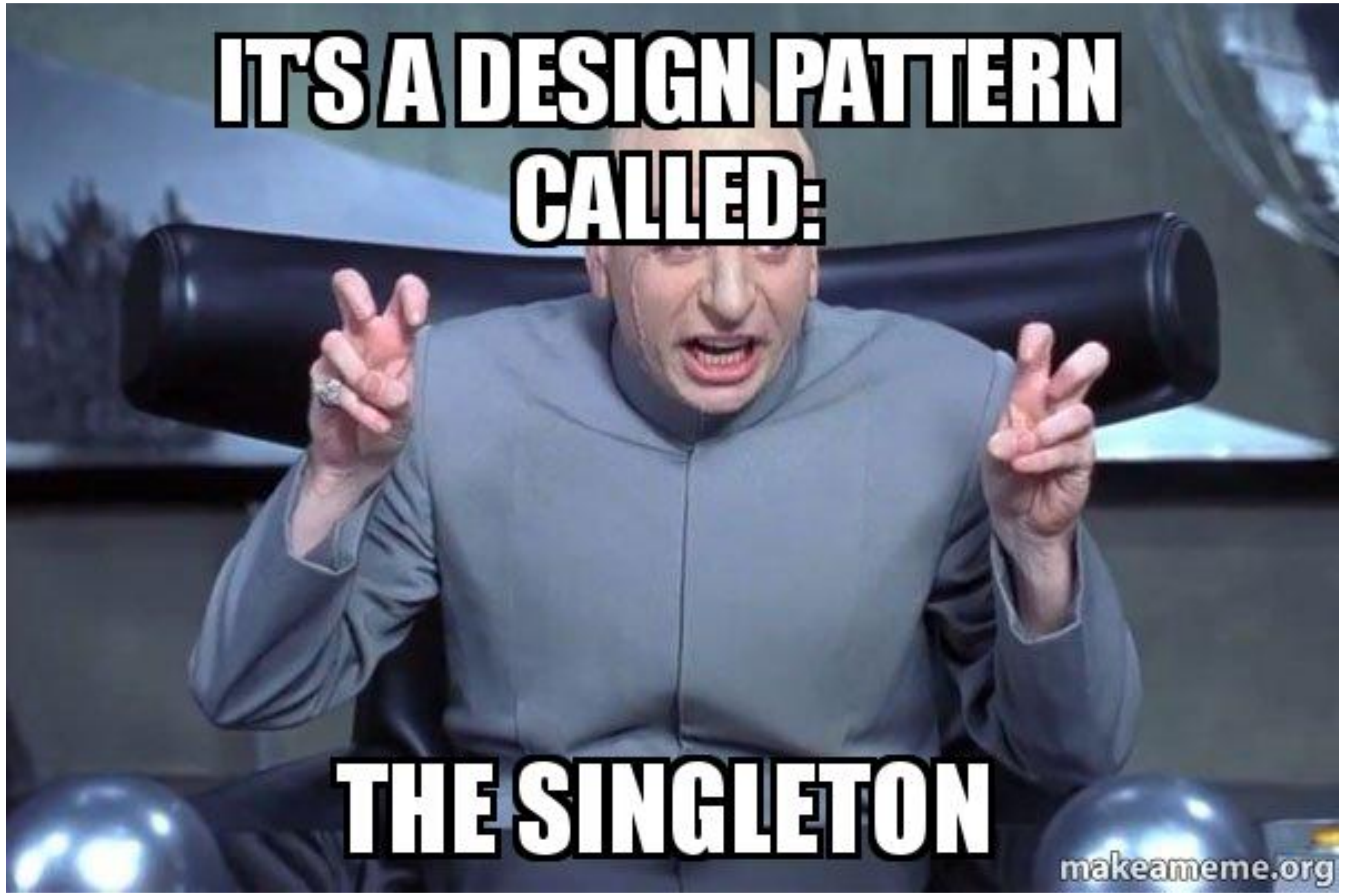
- Saving time/effort to design a solution for an already solved problem.
- They are proven to be highly optimal. Since they have been designed and revised by experts.
- Makes it easier to document and explain your design.

Real world example:

Design a class that can have only one instance in the program ?

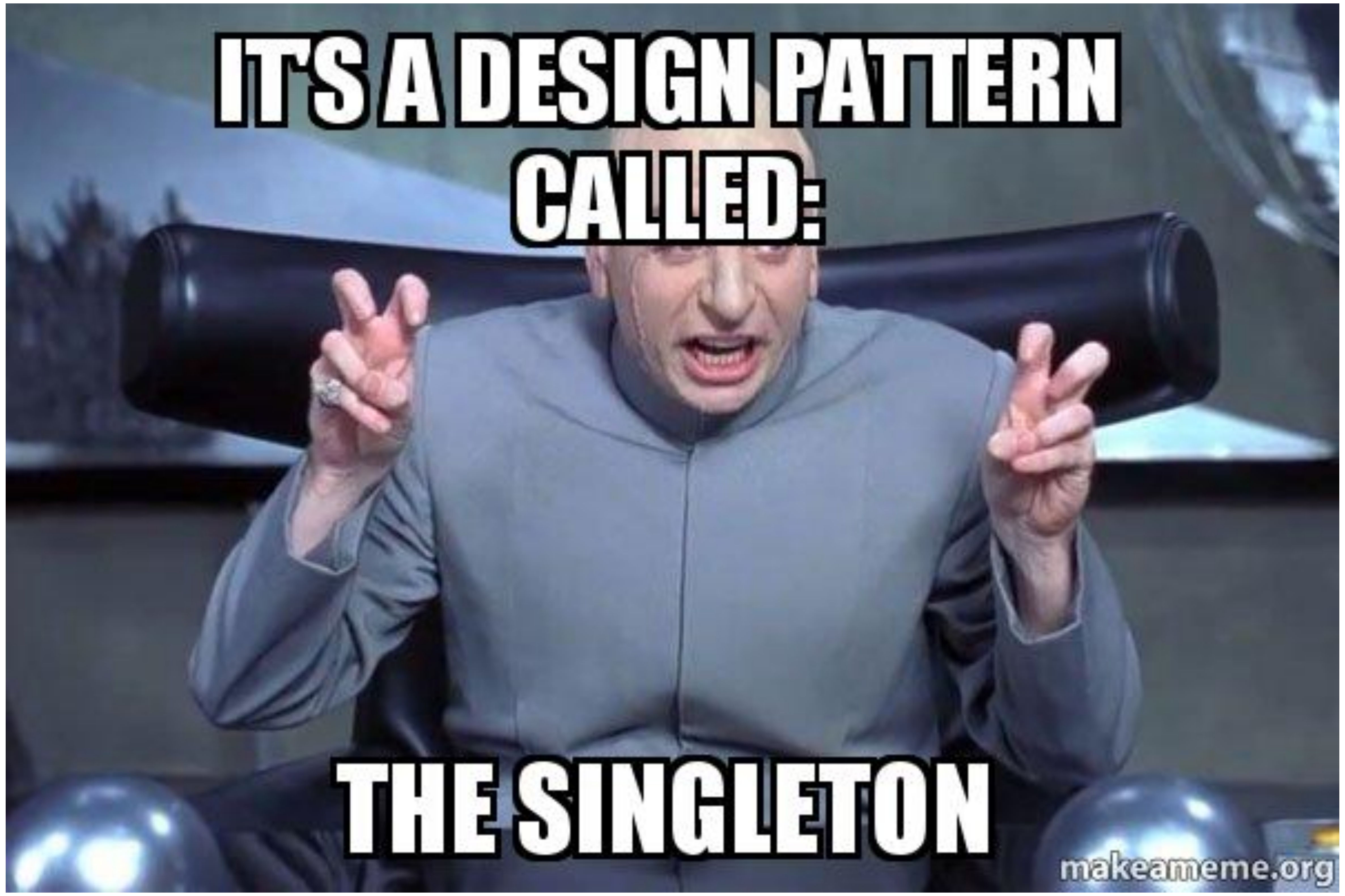
Real world example:

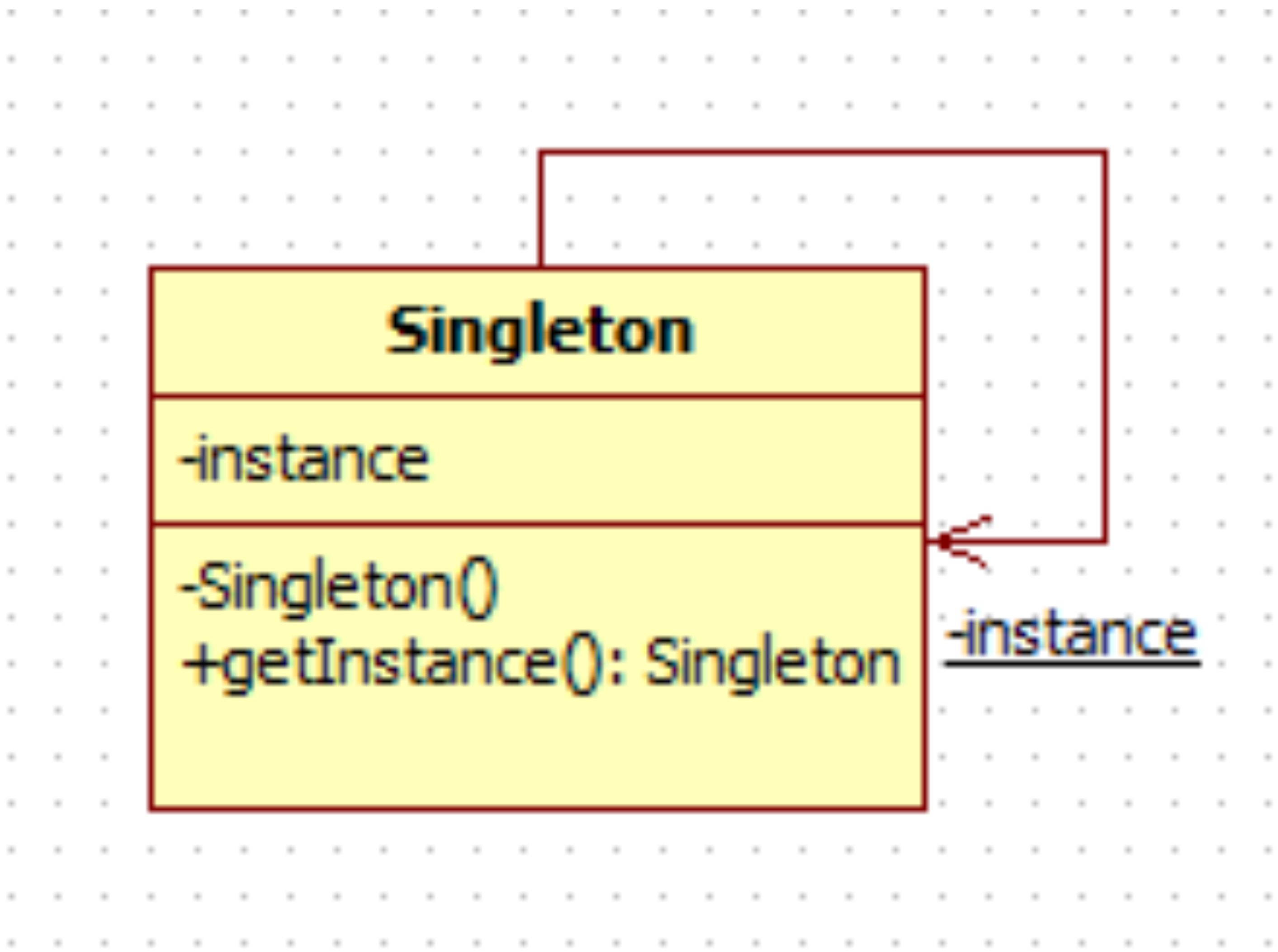
Design a class that can have only one instance in the program ?



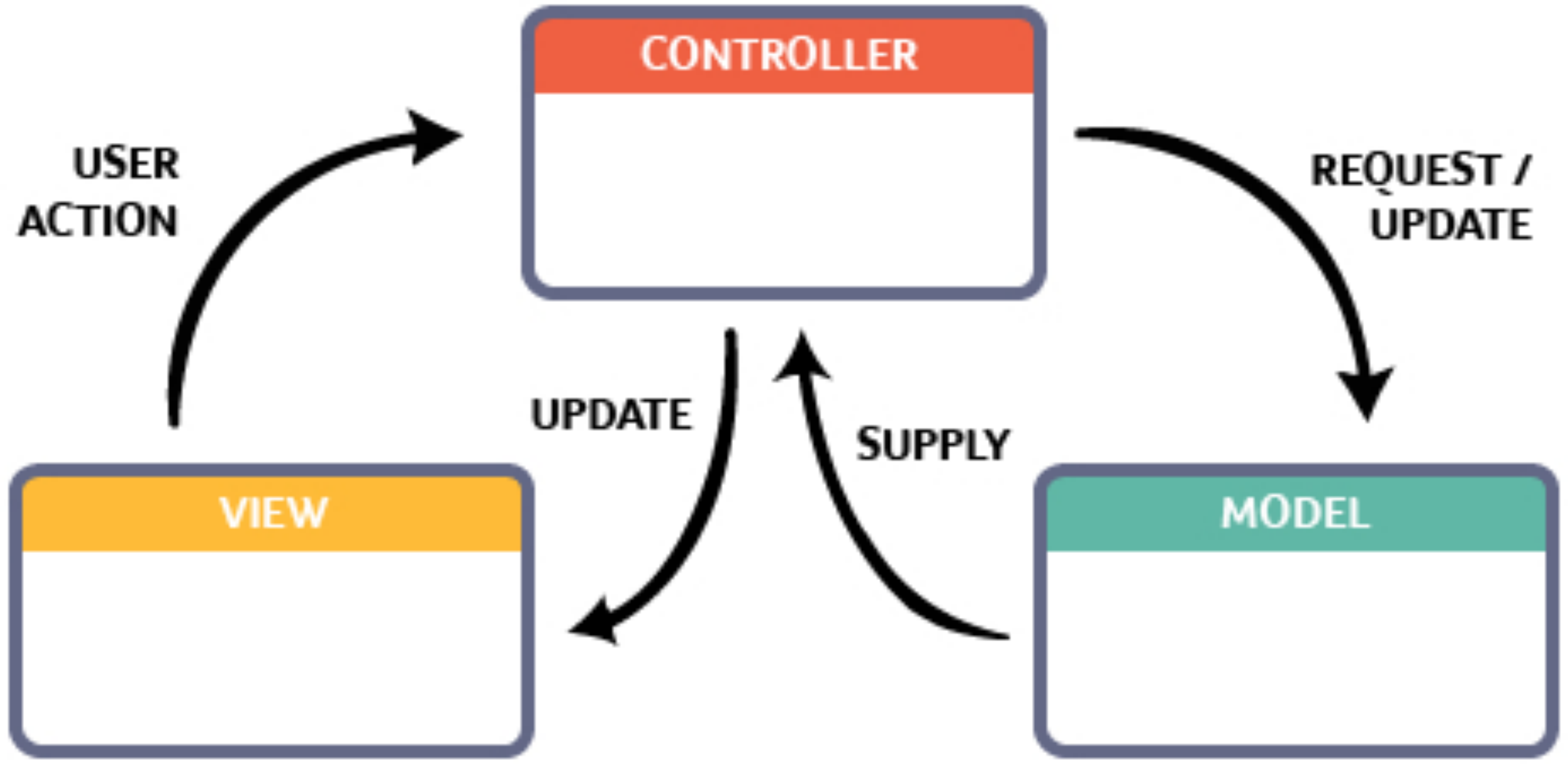
Real world example:

Design a class that can have only one instance in the program ?



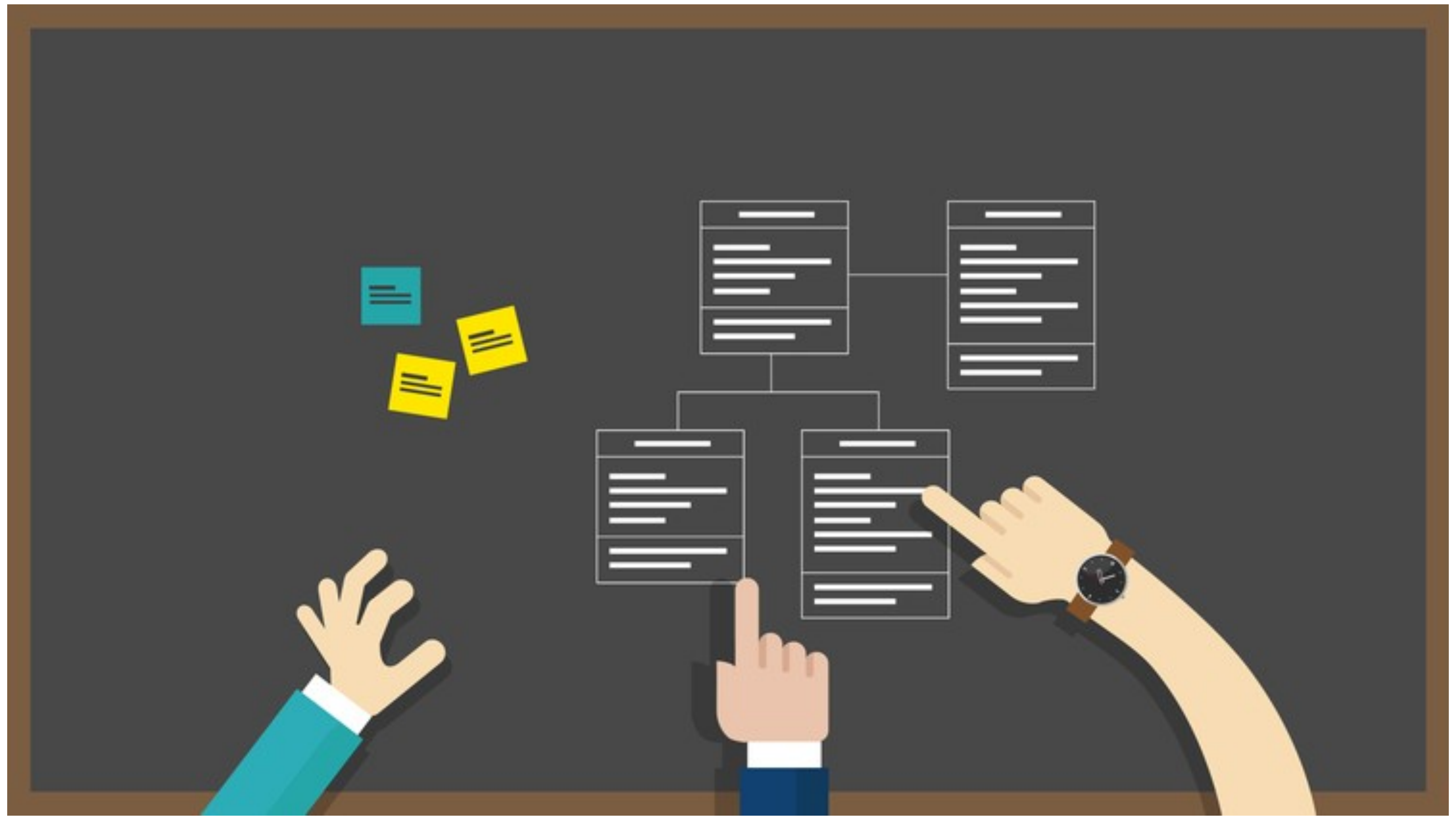


Real world example:



Model-View-Controller (MVC) pattern is widely used in web and mobile app development frameworks

Design Patterns are used Everywhere



Object Oriented Analysis and Design Crash Course

Object Oriented Design

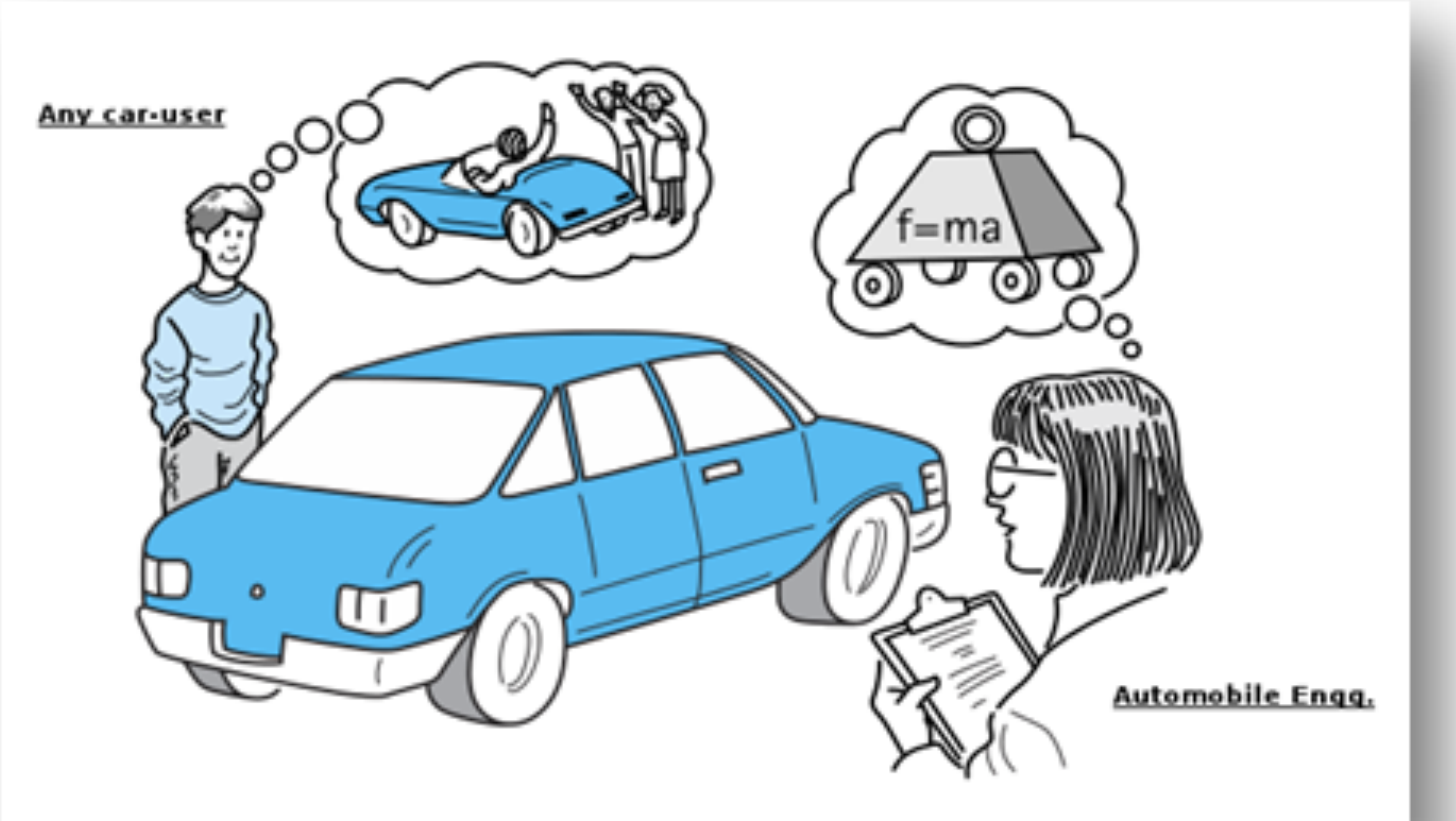
- Object Oriented Programming languages allows programmer to create models of how objects are represented in the world.
- **Major design principles to create OO program:**
    - Abstraction
    - Encapsulation
    - Decomposition
    - Generalization

Object Oriented Design

- **Abstraction**:
  - One way humans use to deal with complexity.
  - Simplifying a concept in the problem domain.
  - Abstraction breaks a concept down to a simplified description that ignores unimportant details and emphasize the essentials needed for the concept within the problem context.

Object Oriented Design

- **Abstraction:**



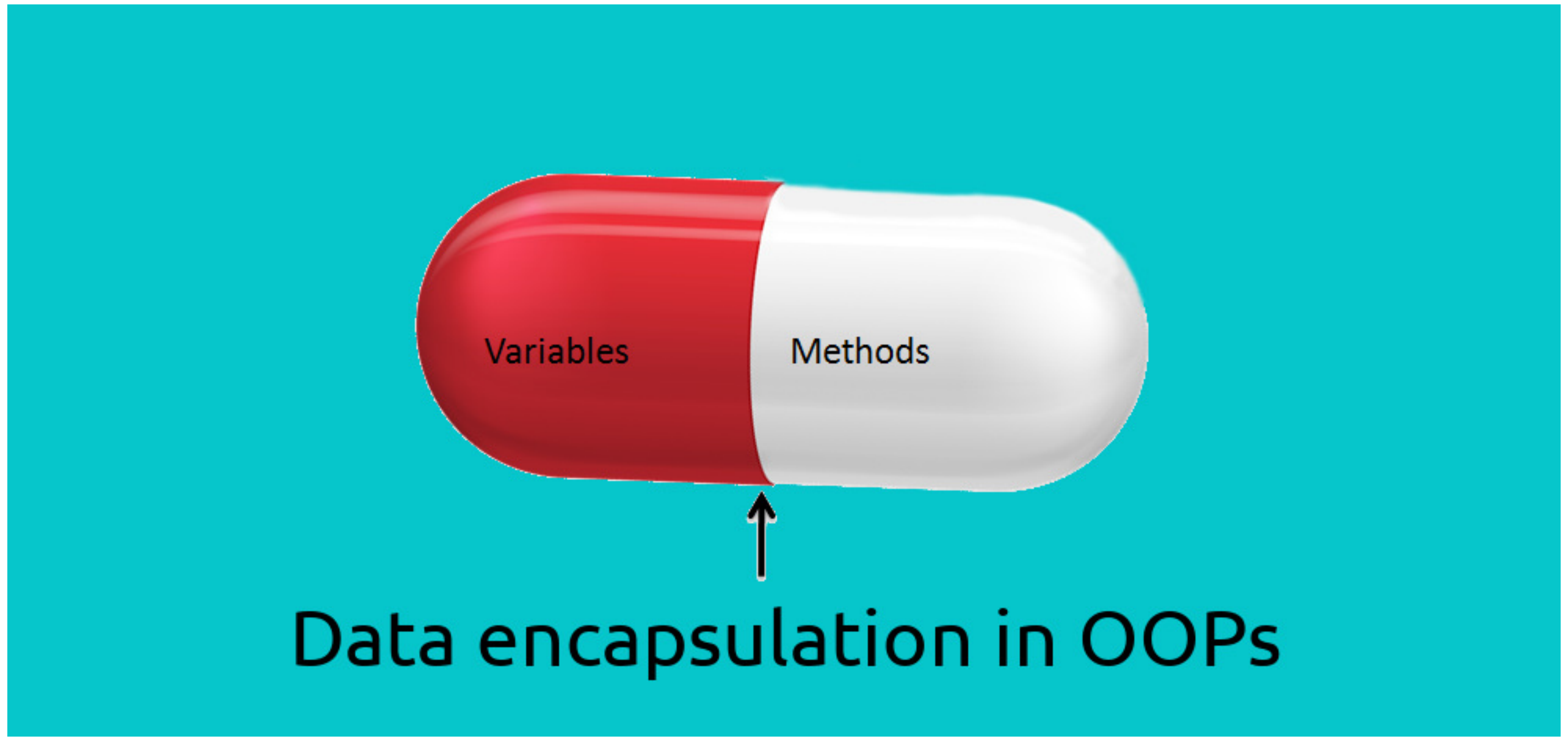
An abstraction includes the essential details relative to the perspective of the viewer

Object Oriented Design

- **Abstraction**:
    - Each object has **attributes** and **behaviors (functions)**.
    - The problem context determines what are the relevant attributes and behaviors.

Object Oriented Design

- **Encapsulation**:
  - Bundling together the object attributes and behaviors.
  - Certain attributes and behaviors are accessible by other objects while others are not.



Object Oriented Design

- **Decomposition**:
    - Decomposition allows taking separate parts with different functionalities and combining them together to create a whole.
    - It makes it possible to break larger problems into smaller parts that are easier to understand and solve.

Object Oriented Design

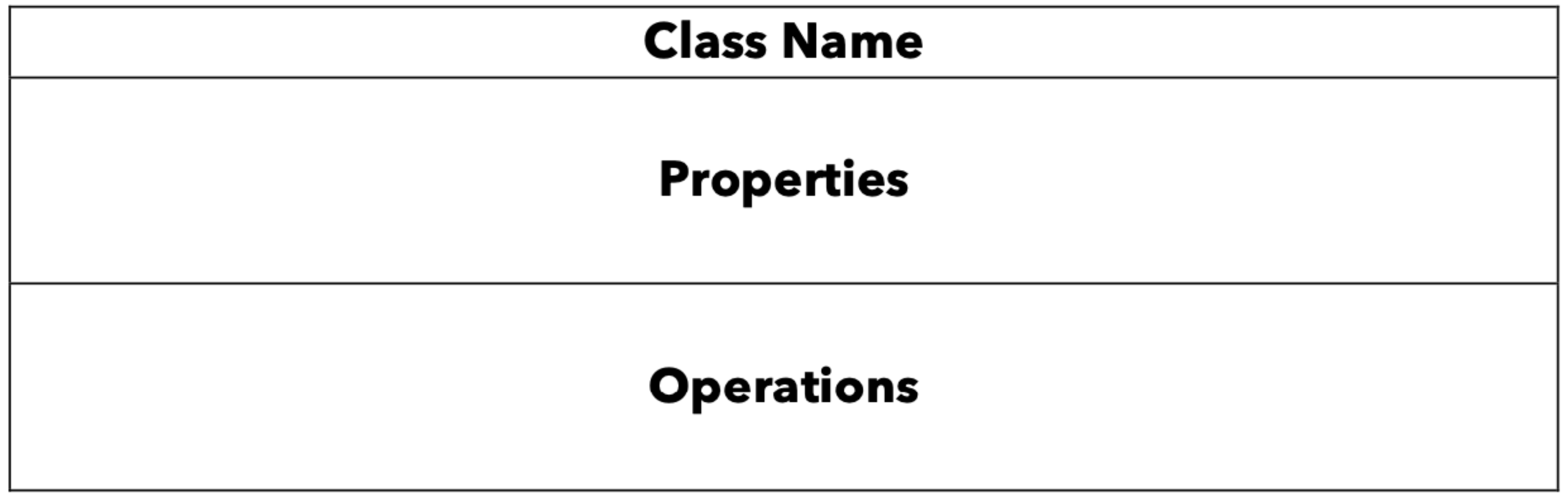
- **Generalization**:
  - Generalization helps reduce redundancy when solving problems.
  - In OO modeling, generalization is achieved by **inheritance**.
  - **Inheritance**: ***child*** class inherits attributes and behaviors from ***parent*** class.
  - Common characteristics and behaviors between two or more classes are defined in a common parent class.
  - Other terminology: (Parent : ***Superclass***) and (Child : ***Subclass***)

UML

- UML: Unified Modeling Language.
- A standard visual modeling language for documenting software architecture.

UML class diagrams

- Each class is represented as a box with three sections
- **Class name**: the name of the class.
- **Properties**: attributes or member variables.
- **Operations**: methods or behaviors.



Class Name

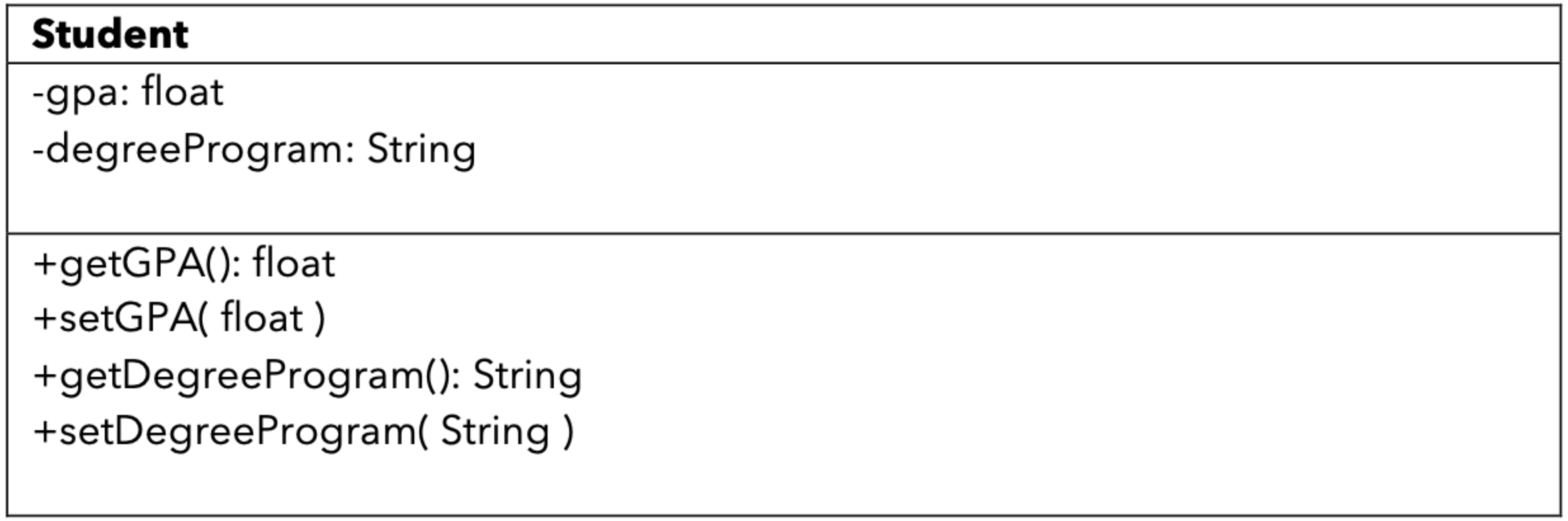
Properties

Operations

UML class diagrams

- Example:

- The (+) or (-) symbols define which attributes/behaviors are **public** *(accessible within and outside the class)* and which
  ones are **private** *(accessible only within the class)*



Student

-gpa: float
-degreeProgram: String

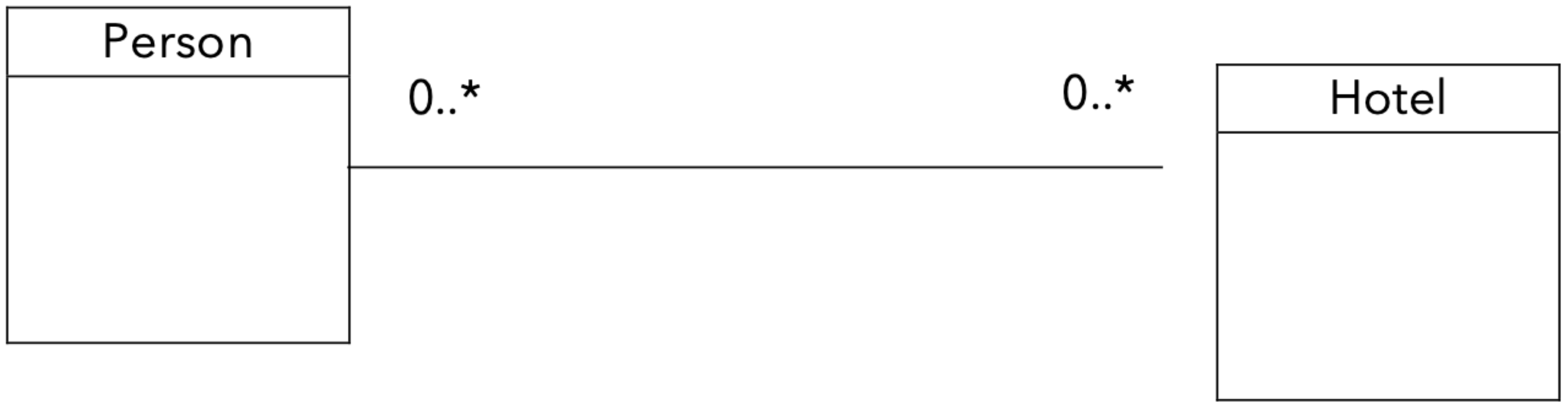
+getGPA(): float
+setGPA(float)
+getDegreeProgram(): String
+setDegreeProgram(String)

UML class diagrams

- Decomposition takes separate parts and combines them together to form a whole.
- Three different types of decomposition according the relationship between the whole and the parts:
  - Association
  - Aggregation
  - Composition

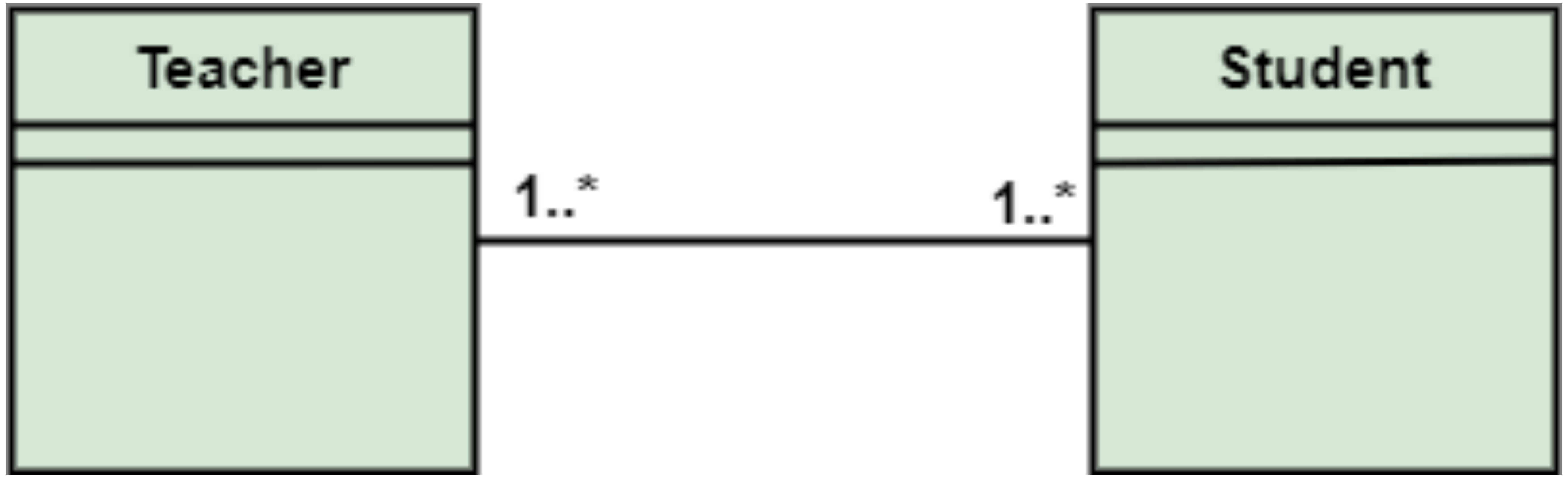
UML class diagrams

- Association: a loose relationship between two objects. They interact with each other for some time, but they are not dependent on each other.
- Association relationship is represented using an arrow.
- The (0..*) means that each Person is associated with zero or more Hotel objects.



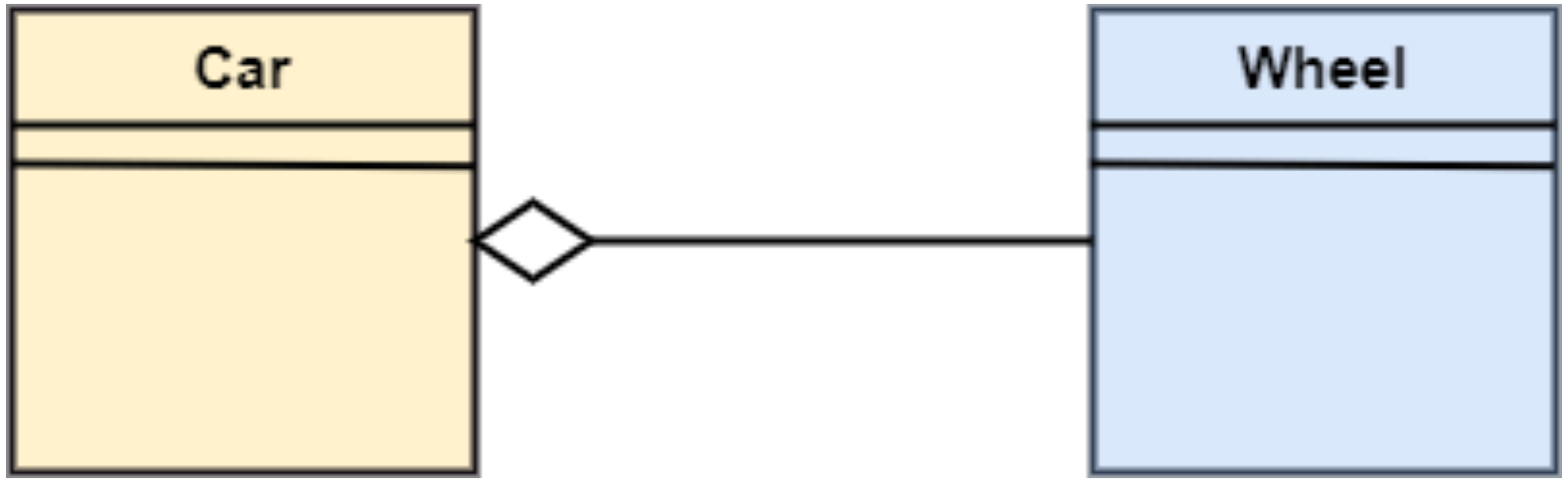
UML class diagrams

- **Association**: a loose relationship between two objects. They interact with each other for some time, but they are not dependent on each other.
- Association relationship is represented using an arrow.
- The (1..*) means that each Student is associated with 1 or more Teacher objects.



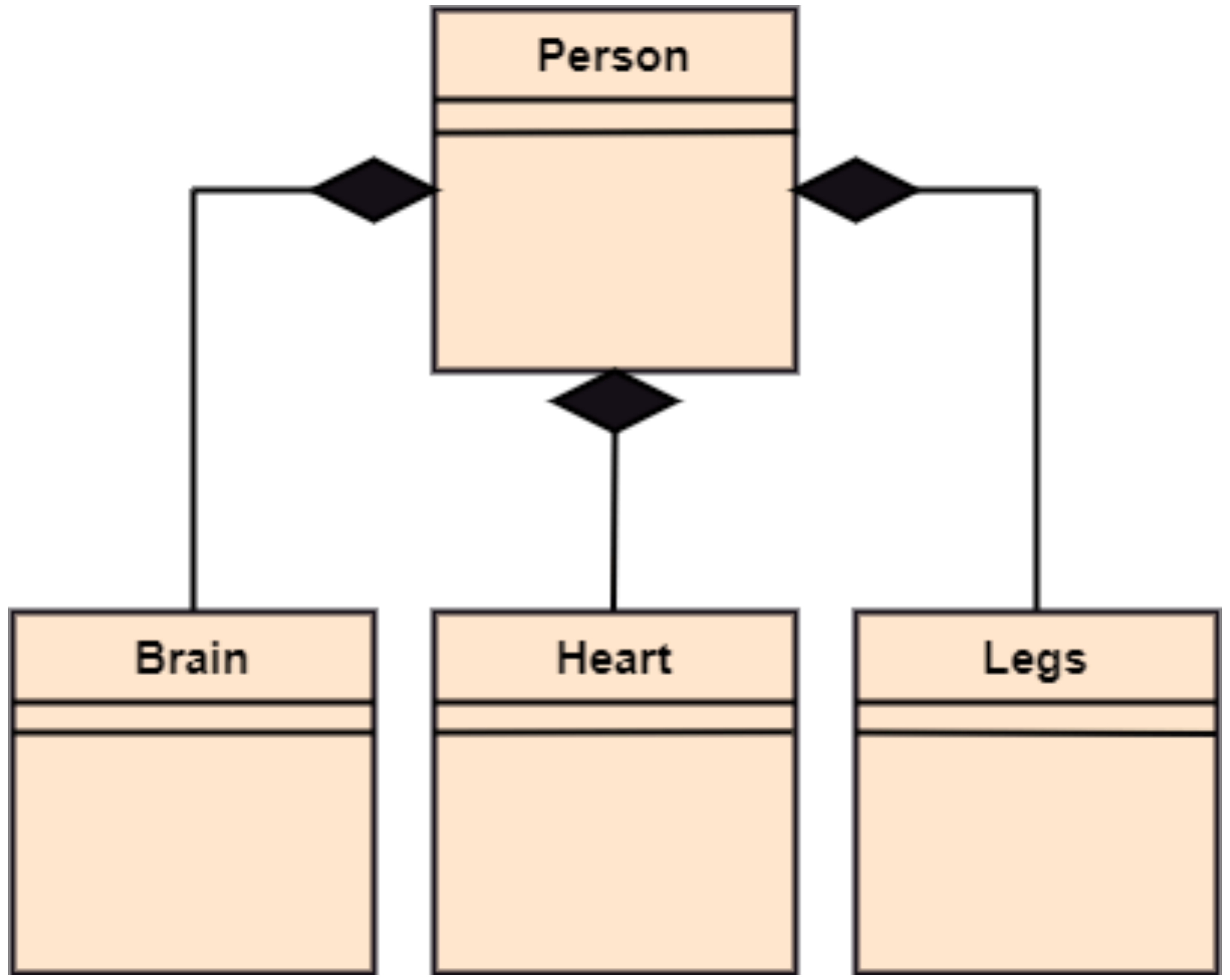
UML class diagrams

- **Aggregation**: is “has-a” relationship. Aggregation is “part-of” relationship.
- Aggregation relationship is represented by a straight line with an empty diamond at one end (the whole).



UML class diagrams

- **Composition**: is strong “has-a” relationship. The whole cannot exist without the part, if the whole is destroyed the parts are destroyed too.
- The composition relationship is represented by a straight line with a black diamond at one end (the whole).



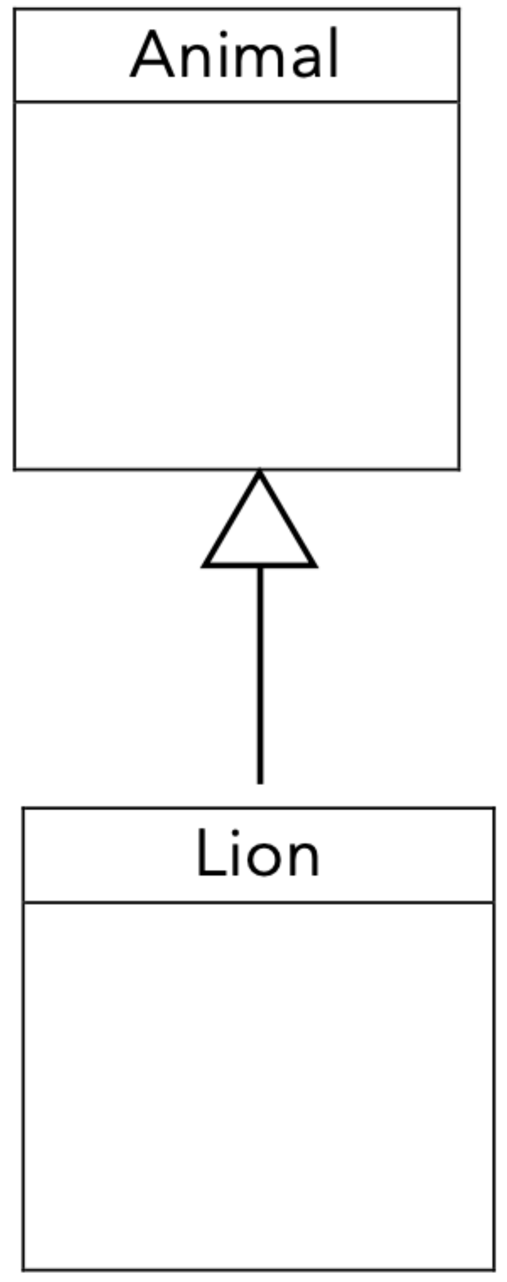
Association vs Composition vs Aggregation

- Read more: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

UML class diagrams

Inheritance

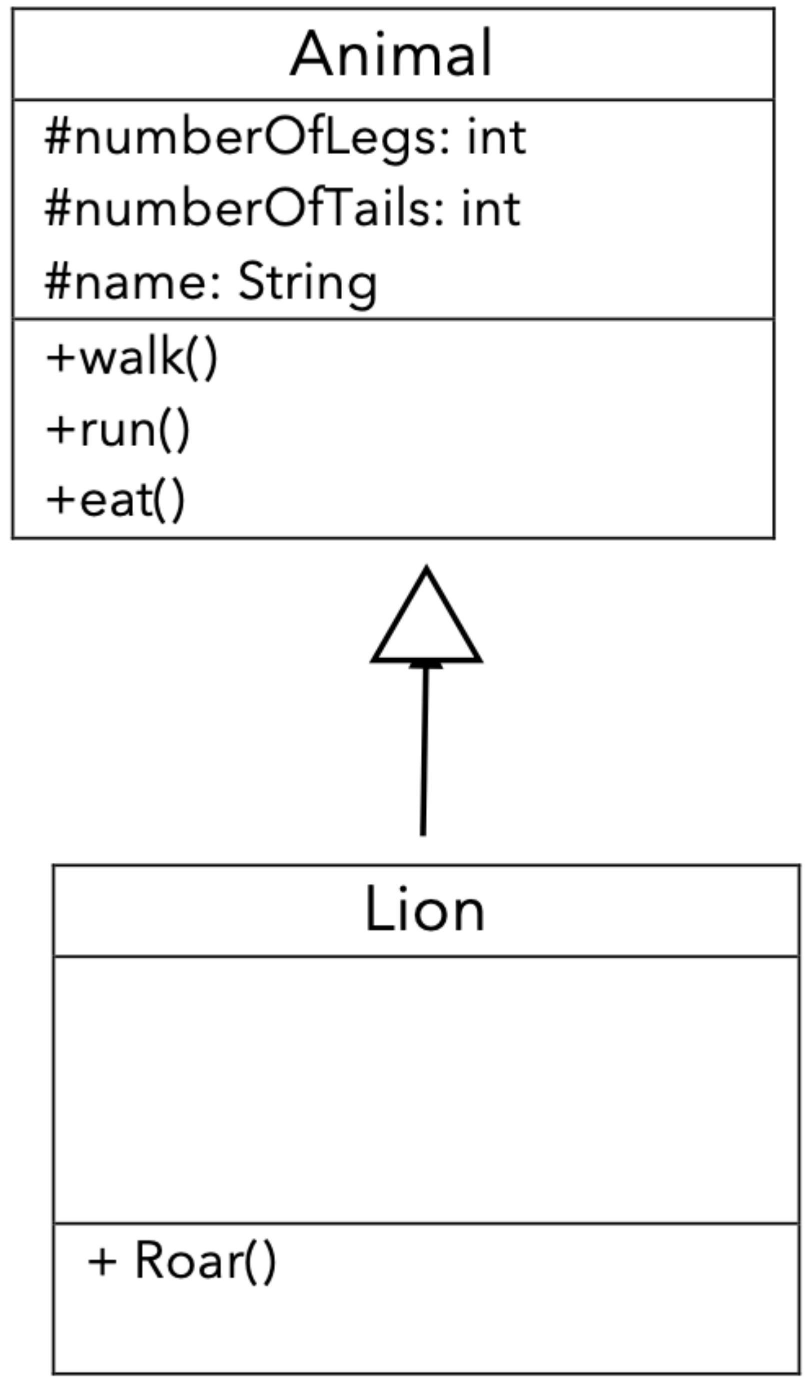
- **Inheritance** can be represented as solid-lined arrow.
- The parent (superclass) is at the head of the arrow.



UML class diagrams

Inheritance

- **Inheritance** can be represented as solid-lined arrow.
- Inherited attributes and behaviors do not need to be rewritten in subclasses.



Animal
#numberOfLegs: int
#numberOfTails: int
#name: String
+walk()
+run()
+eat()
Lion
+ Roar()