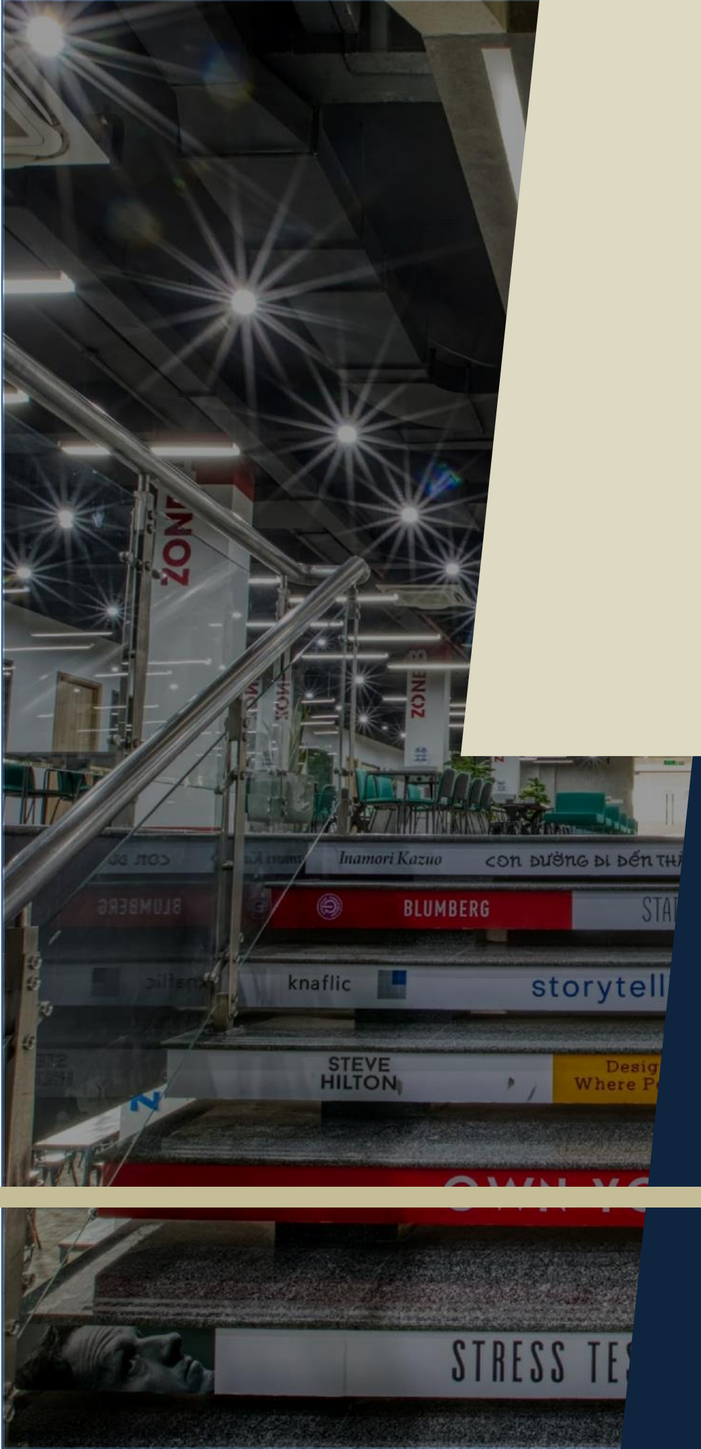


EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Coding Practice

Design patterns

Design patterns

The ratio of time spent reading code to writing code is 10:1 so it is important to write code that is as readable as possible.

~Robert C. Martin~

Even a fool can write code that a computer can understand.
Good programmers write code that humans can understand.

~Martin Fowler~

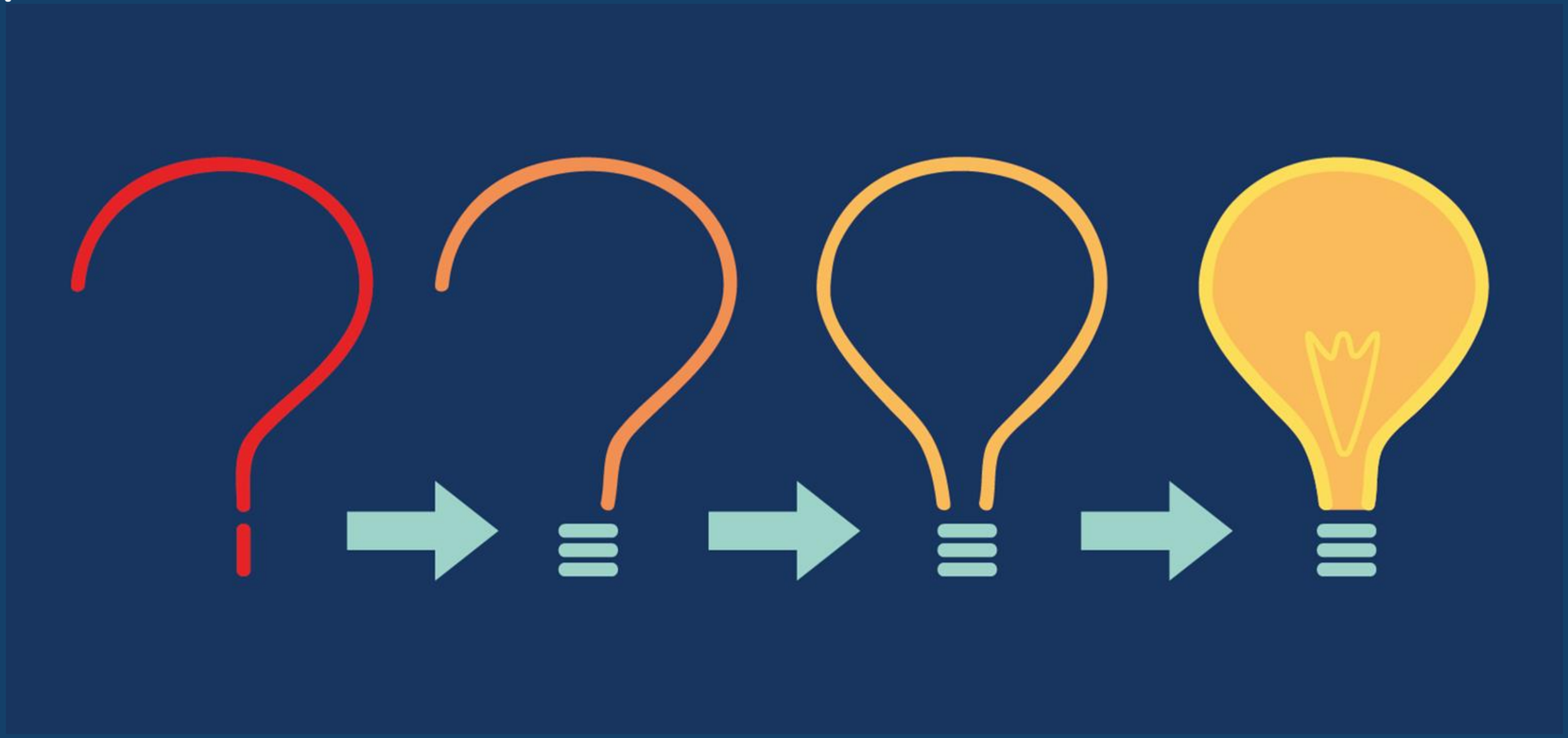
Don't reinvent the wheel !!!

Learn programming languages

Keywords	Syntax	Style
• Variable • Function • Class • Object • Expression • Operator • …	• Command • Condition • Loop • Class declaration • Call method • Identifier • …	• Convention • Principle • Pattern • …

Introduction

What is design ?



Analysis and problem solving

Introduction

What is a good design ?

- Not only solves current problems But also solve the problem in the future.
- Not only does it run well when the system is small But also runs well when the system is larger
- Not only runs well with current version But more versions can be well.

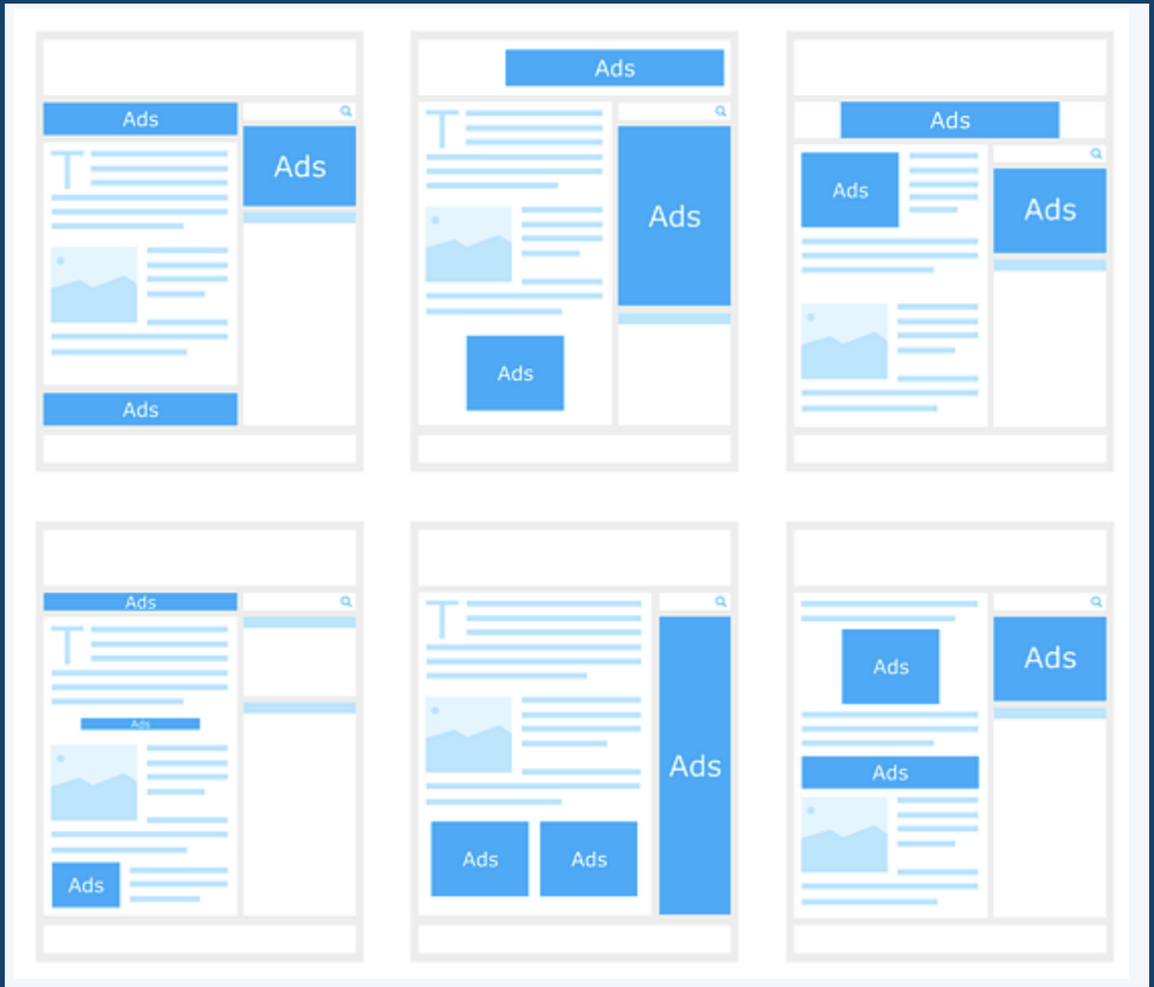
Introduction

Design Principles

- Encapsulate what varies
- Favor composition over inheritance
- Programming to interfaces not to implementations
- Strive for loosely coupled between objects that interact
- Classes should OPEN for extension and CLOSE for modification
- Depends on abstraction, not concrete classes
- Don't call us, we'll call you
- A class should have only one reason to change

Introduction

Template

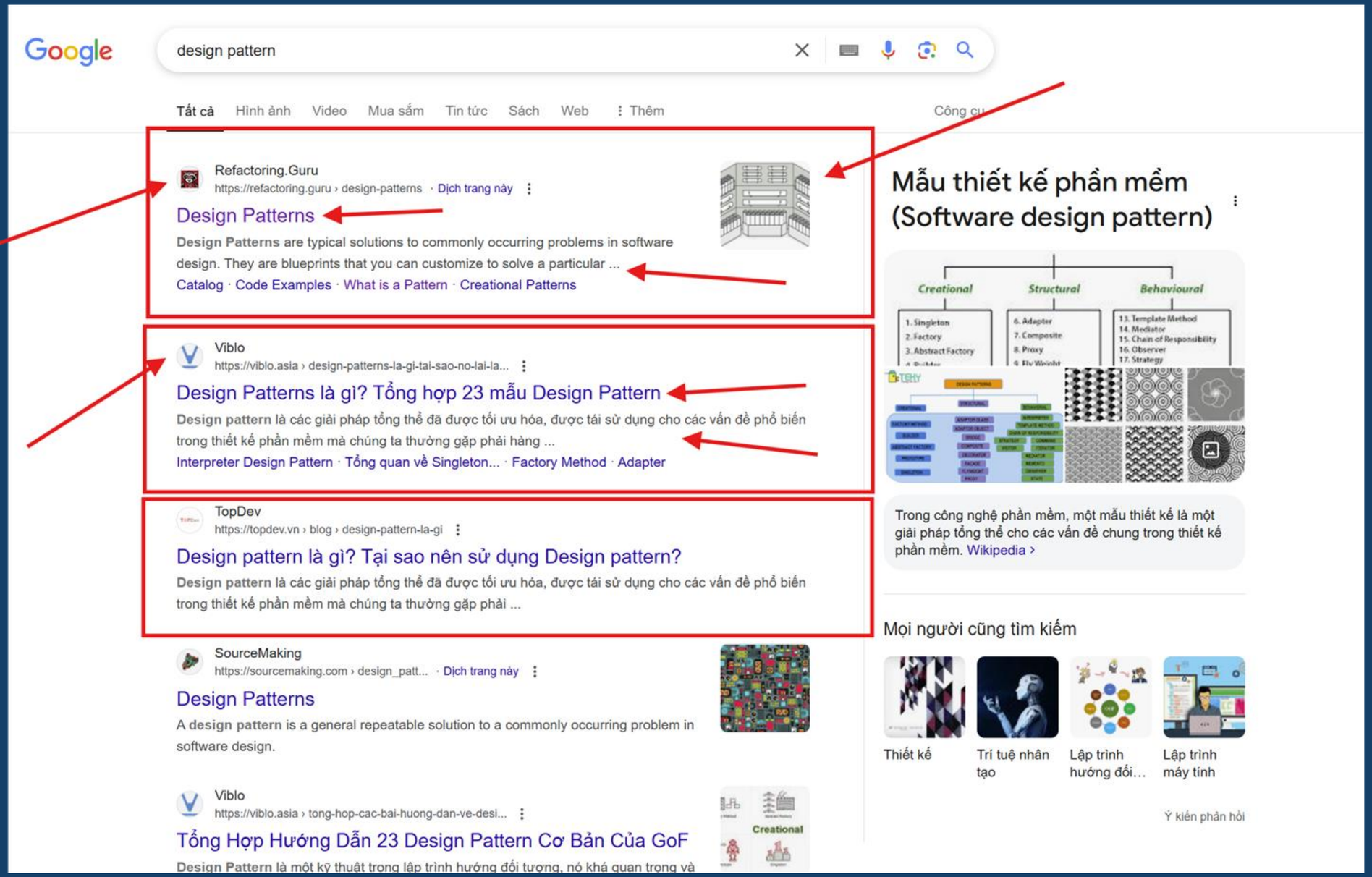


Introduction

Patterns are everywhere

Introduction

Patterns are everywhere



What is Design Pattern ?

- Is a reusable general solution to common problems
- Is a model or description of how to solve a problem
- A proven technique
- Design Pattern is not a complete design that can be directly transformed into source code.
- Design Patterns are templates for solving problems in different situations.

Introduction

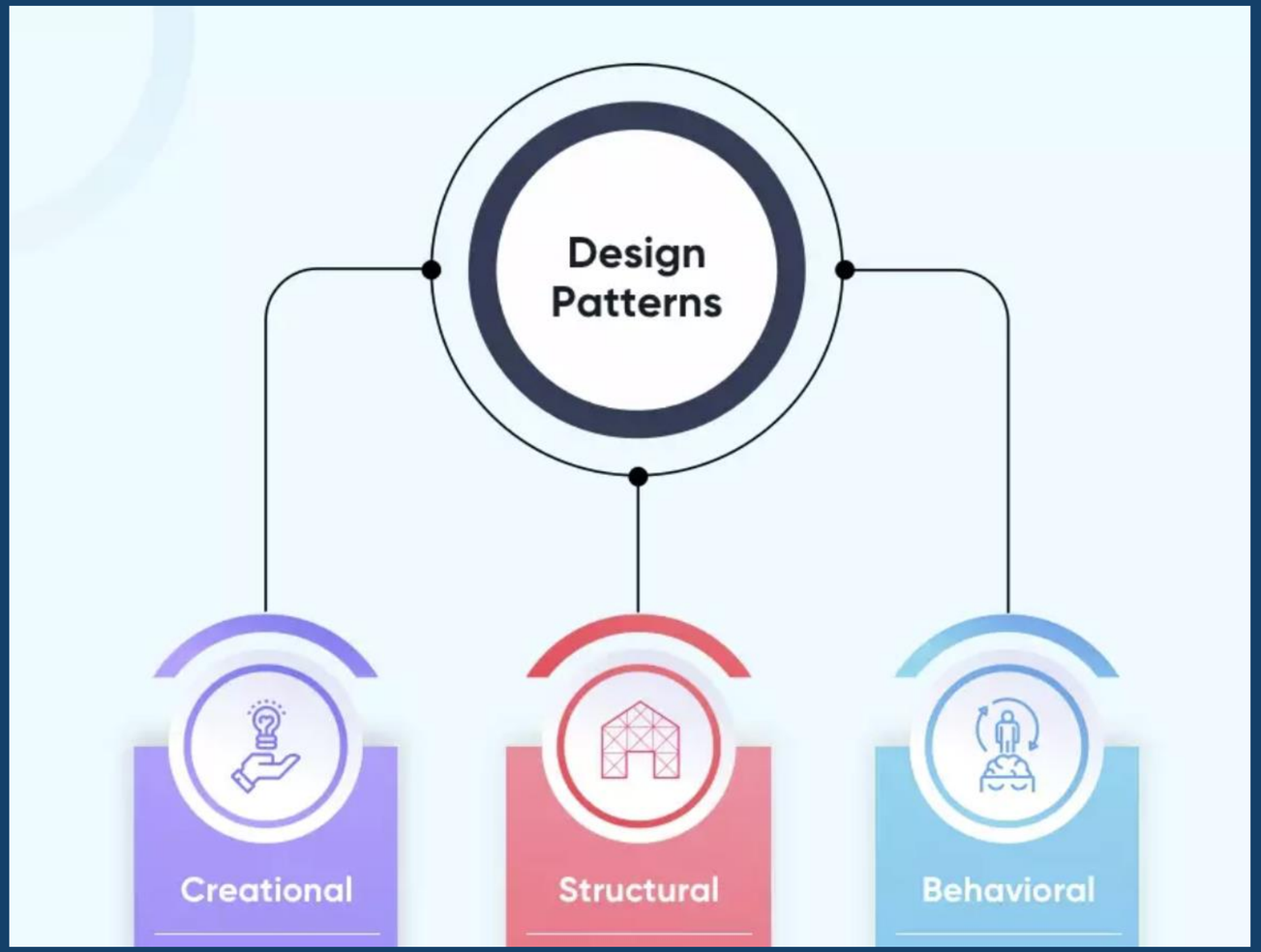
Why do we need Design Pattern?

- Speed up software design and development
- Proven quality of solution
- Prevent problems that arise if the design is not good
- Can be applied to many different situations
- Easily collaborate, share designs and source code between parties

Classification of Design Patterns

Design patterns are classified into three main categories:

- **Creational Patterns**: Deal with object creation mechanisms.
- **Structural Patterns**: Focus on object composition and relationships.
- **Behavioral Patterns**: Concerned with communication between objects.



Overview of Creational Patterns

Creational patterns deal with object creation mechanisms, optimizing how objects are created to be flexible and reusable.

Common Creational Patterns:

1. Singleton Pattern
2. Factory Method Pattern
3. Abstract Factory Pattern
4. Builder Pattern
5. Prototype Pattern

Singleton Pattern

Purpose: Ensures that a class has only one instance and provides a global point of access to it.
Example: Managing a single database connection in an application.

Singleton Pattern

```
public class Singleton
{
    // Static field to hold the single instance of the class
    3 references
    private static Singleton _instance;

    // Object used for locking to ensure thread safety
    1 reference
    private static readonly object _lock = new object();

    // Private constructor to prevent instantiation from outside the class
    1 reference
    private Singleton() { }

    // Static method to provide a global access point to the single instance
    0 references
    public static Singleton GetInstance()
    {
        // Lock ensures only one thread can execute this block at a time
        lock (_lock)
        {
            // Check if the instance has already been created
            if (_instance == null)
                _instance = new Singleton();  // Create the instance if it doesn't exist
            return _instance;  // Return the single instance
        }
    }
}
```

Factory Method Pattern

Purpose: Provides an interface for creating objects, but allows subclasses to alter the type of objects created.

Example: Creating different types of shapes (circle, square) without specifying their exact classes.

```
// Common interface for shape
2 references
public interface IShape
{
    2 references
    void Draw();
}

// Circle
0 references
public class Circle : IShape
{
    1 reference
    public void Draw()
    {
        Console.WriteLine("Drawing a Circle");
    }
}
```

```
// Square
0 references
public class Square : IShape
{
    1 reference
    public void Draw()
    {
        Console.WriteLine("Drawing a Square");
    }
}
```

Structural Patterns

Structural patterns deal with how classes and objects are composed to form larger structures.

- **Common Structural Patterns:**
  - Adapter Pattern
  - Decorator Pattern
  - Facade Pattern
  - Composite Pattern
  - Proxy Pattern

Adapter Pattern

Purpose: Converts the interface of a class into another interface that clients expect.
Example: Adapting legacy code to work with a new system.

```
public interface ITarget
{
    1 reference
    void Request();
}
2 references
public class Adaptee
{
    1 reference
    public void SpecificRequest()
    => Console.WriteLine("Called SpecificRequest");
}
```

```
public class Adapter : ITarget
{
    2 references
    private readonly Adaptee _adaptee;

    0 references
    public Adapter(Adaptee adaptee)
    {
        _adaptee = adaptee;
    }

    1 reference
    public void Request()
    {
        _adaptee.SpecificRequest();
    }
}
```

Structural Patterns

Decorator Pattern

Purpose: Allows behavior to be added to an individual object dynamically without affecting the behavior of other objects from the same class.
Example: Adding new functionalities to a window (scrolling, borders).

Example: Click Here

Behavioral Patterns

Overview of Behavioral Patterns

Behavioral patterns focus on communication between objects and how responsibilities are distributed.

- **Common Behavioral Patterns:**
  - Observer Pattern
  - Strategy Pattern
  - Command Pattern
  - State Pattern
  - Mediator Pattern

Behavioral Patterns

Observer Pattern

Purpose: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified.

Example: Event handling in a GUI application.

Behavioral Patterns

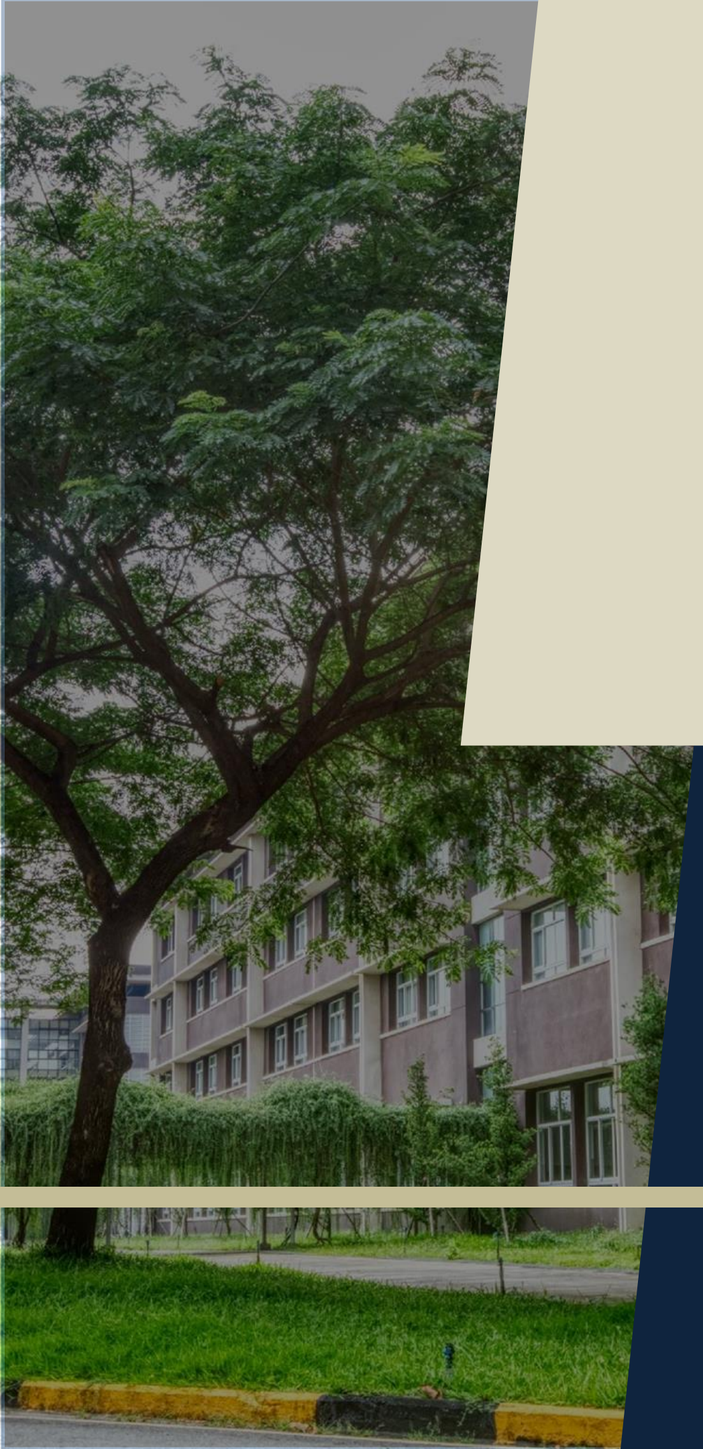
Command Pattern

Purpose: Encapsulates a request as an object, thereby allowing users to parameterize clients with different requests, queue or log requests, and support undoable operations.

Example: Click here

Good material to learn about Design Patterns

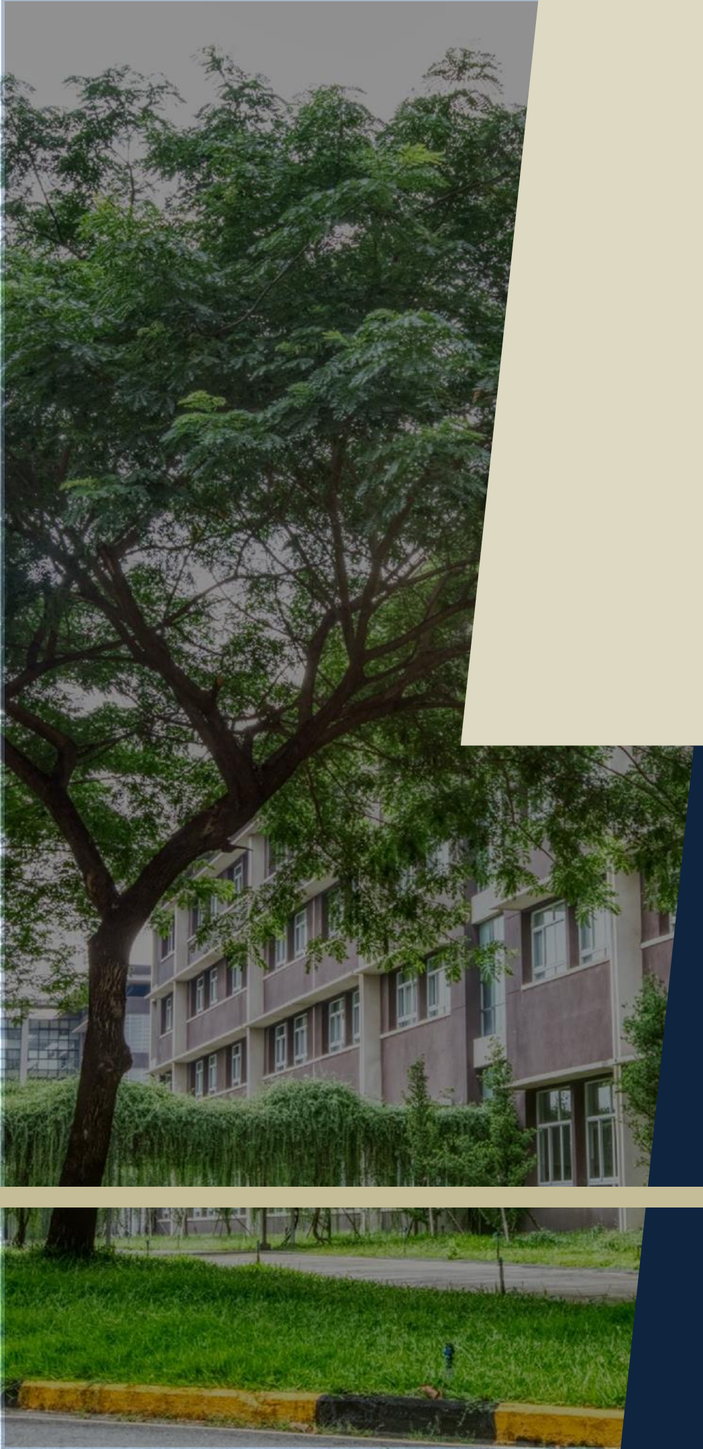
1. Design Patterns: Elements of Reusable Object-Oriented Software
Author: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Nhóm "Gang of Four")
2. Head First Design Patterns
Author: Eric Freeman, Elisabeth Robson
3. Patterns of Enterprise Application Architecture
Author: Martin Fowler
4. Refactoring to Patterns
Author: Joshua Kerievsky
5. Clean Architecture: A Craftsman's Guide to Software Structure and Design
Author: Robert C. Martin (Uncle Bob)
6. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design
Author: Craig Larman
7. **Pro Design Patterns in C#**
Author: Chaur Wu



EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Start your future at EIU

Q&A



EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

Start your future at EIU

THANK YOU