

UNIVERSITY OF SANTO TOMAS

IT-ELEC2C

## 01 – OVERVIEW AND HISTORY

JOHN DEL PILAR

4IT-H





# 01 – OVERVIEW AND HISTORY

## Design Patterns

- best practices used by **experienced** object-oriented software developers.
- **solutions to general problems** that software developers faced during software development
- obtained by **trial and error** by numerous software developers over quite a substantial period

## History of Design Patterns

- Elements of Reusable Object-Oriented Software
  - 1994
  - initiated the **concept of Design Pattern** in Software Development
  - Gang of Four (**GOF**)
    - Erich Gamma
    - Richard Helm
    - Ralph Johnson
    - John Vlissides
  - primarily based on the following principles of object-oriented design:
    - program to an **interface** not an implementation
    - favor **object composition (has-a)** over inheritance (is-a)

## Usage of Design Pattern

- Common Platform for Developers
  - provide a **standard terminology** and are specific to a scenario.
  - a singleton design pattern signifies **use of single object** so all developers familiar with single design pattern **will make use of single object** and they can tell others that program is following a singleton pattern
- Best Practices
  - **evolved over a long period** and they **provide best solutions** to certain problems faced during software development
  - learning these patterns **help inexperienced developers to learn software design** in an easy and faster way

## Advantages of Design Patterns

- provides the developer with a **selection of tried and tested solutions** to work with
- language neutral so it **can be applied to any language** that supports object-orientation
- aid communication for they are **well-documented** and can be researched if that is not the case
- have a proven track record as they are **widely used** and thus **reduce the technical risk** to the project
- **highly flexible** and can be used in practically any type of application or domain

## Disadvantage of Design Patterns

- can only be applied **using object-oriented programming**
- cannot be used in imperative programming, i.e., procedural and structured programming

## Types of Design Patterns

- Creational Patterns
  - **object creation**
  - provide a way to **create objects while hiding the creation logic**, rather than instantiating objects directly using new operator
  - give program more **flexibility** in deciding which objects need to be created for a given use case
- Structural Patterns
  - **relationship between entities**
  - concern **class and object composition**
  - concept of **inheritance is used to compose interfaces** and define ways to compose objects to obtain new functionalities
- Behavioral Patterns
  - specifically concerned with **communication between objects**



# 01 – OVERVIEW AND HISTORY

## Gang-of-Four Design Patterns

Creational (6)	Structural (7)	Behavioral (10)
Abstract Factory Builder Factory Method Object Pool Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Interpreter Iterator Mediator Memento Null Object Observer Strategy Template Method

- concept of **extending the behavior** of an object
  - a concrete class and an abstract base class or interface inheritance
- an abstract class implements **no functionality** in the base class
- a concrete class will **have some functionality** in the base class

## Loosely Coupled Objects

- **more independent of each other** rather than dependent on one another
- loose coupling makes it **easier to modify an object** with little understanding of the larger system in which the object resides

## Decomposition

- process of **dividing a problem into smaller pieces**
- divide and conquer
- breaking your program into objects, each with **specific responsibilities**
- **reusability**
  - the **result** of good decomposition

## Encapsulation

- **hides** the internal construction of an object
- objects are **accessed through their interface**
- allows an object to change its internal behavior **without changing** the way other objects interact with it

## Polymorphism

- allows objects with a common base class or interface to be **treated in a similar fashion**
- an object can interact with another object **without being completely aware** of the other object's true type

## OOD

- object-oriented analysis and design
- technical approach for analyzing and designing an application, system, or business **by applying object-oriented programming**

## SOLID

- five-dependency principles
- intended to make software designs more:
  - understandable
  - flexible; and
  - maintainable
- 1. Single Responsibility Principle
  - a class should have only **a single responsibility**
- 2. Open-Closed Principle
  - entities should be **open for extension**, but **closed for modification**
- 3. Liskov Substitution Principle
  - **objects in a program should be replaceable** with instances of their subtypes without altering the correctness of that program
- 4. Interface Segregation Principle
  - **many client-specific interfaces** are better than one general-purpose interface
- 5. Dependency Inversion Principle
  - should **depend upon abstractions**, not concretions

## Inheritance