# **Day 23 - 2 August 2025**

**Document Name:**Day 23 - hmuvvala@ - Hari Gopal Muvvala

### What are Design Patterns?

* Design Patterns are **standard, reusable solutions** to commonly occurring problems in software design.
* They act as **blueprints** or **templates** that you can apply to solve particular design issues in object-oriented programming.
* They are **not code**, but rather **conceptual solutions** which can be implemented in any language.

Example: When creating multiple objects of the same type, instead of writing new logic each time, you can use a **Factory Pattern**.

### Why Use Design Patterns?

| **Benefit** | **Description** |
| --- | --- |
| **Reusability** | Encourages use of proven solutions, saving time and effort. |
| **Maintainability** | Makes the codebase more modular and easier to maintain or extend. |
| **Scalability** | Prepares the code for future changes without breaking existing logic. |
| **Improved Communication** | Developers can discuss design clearly using pattern names as vocabulary. |
| **Avoids Re-inventing the Wheel** | Solves problems that others have already solved efficiently. |

### Types of Design Patterns

Design Patterns are broadly classified into **three categories**:

| **Category** | **Purpose** |
| --- | --- |
| **Creational** | Deals with object **creation mechanisms**. Focuses on **how** objects are created. |
| **Structural** | Deals with object **composition**. Helps organize relationships between classes and objects. |
| **Behavioral** | Focuses on **communication** between objects and assigning responsibilities. |

### Creational Design Patterns (Covered Today)

Creational patterns abstract the instantiation process. The main goal is to make a system **independent of how its objects are created, composed, and represented.**

#### a) Factory Pattern

* **Definition:** Provides an interface for creating objects, but **lets subclasses decide** which class to instantiate.
* **Used When:**
  + You don’t know in advance what class objects you need.
  + Object creation logic is **complex or repetitive**.
* **How It Works:**
  + You create a **Factory class** with a method that returns different types of objects **based on input**.
  + The client class **calls the factory method** and gets the object.

**Real-Life Analogy:**

Think of a **pizza shop** where you just place an order by type (Veg, Cheese, Chicken) — and the factory (kitchen) decides **which subclass of Pizza** to make.

**Advantages of Factory Pattern:**

* Encapsulates object creation logic
* Promotes loose coupling between client and object
* Makes code easier to manage and scale

#### b) Singleton Pattern

* **Definition:** Ensures that a class has **only one instance**, and provides a **global point of access** to it.
* **Used When:**
  + You want exactly **one object** across the whole application (e.g., Logger, Configuration Manager).
* **Key Concepts:**
  + **Private constructor** prevents object creation from outside.
  + **Static variable** holds the single instance.
  + **Static method (getInstance)** gives access to the single object.

**Types of Singleton Implementations:**

| **Type** | **Description** |
| --- | --- |
| **Eager Initialization** | Instance is created at class loading. Simple but may waste memory. |
| **Lazy Initialization** | Instance is created only when needed (on-demand). More efficient. |

**Real-Life Analogy:**

A **printer spooler** – you only need **one** spooler object to manage print jobs across the system.

### Best Practices for Design Patterns

* **Understand the problem first** — don’t force-fit a pattern.
* **Use meaningful class names** so others can easily recognize the pattern.
* **Follow SOLID principles** to know **when and why** to use a pattern.
* **Combine patterns** if needed (e.g., Factory + Singleton).

### Quick Summary Table

| **Pattern** | **Type** | **Purpose** |
| --- | --- | --- |
| **Factory** | Creational | Create objects based on input without exposing logic |
| **Singleton** | Creational | Ensure only one instance of a class |