**Design Patterns**

**🔰 What Are Design Patterns?**

Design Patterns are **proven solutions** to **common problems** in software design. They help you **structure your code** in a standard, reusable way.

**✅ 1. Creational Patterns – how objects are created.**

* Singleton
* Factory
* Abstract Factory
* Builder
* Prototype

**✅ 2. Structural Patterns – how classes/objects are composed.**

* Adapter
* Decorator
* Proxy
* Composite
* Facade
* Bridge
* Flyweight

**✅ 3. Behavioral Patterns – how objects communicate.**

* Strategy
* Observer
* Template Method
* Command
* Iterator
* State
* Chain of Responsibility

**🧠1. What is Singleton Pattern?**

The **Singleton Pattern** ensures that a **class has only one instance** and provides a global point of access to it.

**✅ When to Use Singleton?**

* Logger classes
* Configuration classes
* Database connections
* Caches
* Shared resources

// Singleton Logger Class

public class Logger {

private static Logger instance;

// private constructor to prevent external instantiation

private Logger() {}

// public method to get the single instance

public static Logger getInstance() {

if (instance == null) {

instance = new Logger(); // create the instance once

}

return instance;

}

public void log(String message) {

System.out.println("Log: " + message);

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("User logged in.");

logger2.log("User clicked on dashboard.");

// Check if both are same instances

System.out.println(logger1 == logger2); // true ✅

}

}

**Singleton** = "Only one object allowed. Everyone uses the same one."

**🧠 2. What is the Factory Pattern?**

The **Factory Pattern** is like a **real-world factory** 🏭. You tell the factory **what you want**, and it **builds the object** for you.

You don’t create objects directly — you ask the **factory to make them**.

**🎂 Real-life Analogy (Child-friendly):**

Imagine you go to an **ice cream factory** 🍦. You say:  
“I want **chocolate** ice cream” or “I want **vanilla**”.

You don’t make it yourself — the factory makes the right flavor and gives it to you.

**✅ Code Example: Ice Cream Factory**

**🧱 Step 1: Create an Interface**

public interface IceCream {

void eat();

}

**🍫 Step 2: Create Different Types of Ice Cream**

public class ChocolateIceCream implements IceCream {

public void eat() {

System.out.println("Eating Chocolate Ice Cream! 🍫");

}

}

public class VanillaIceCream implements IceCream {

public void eat() {

System.out.println("Eating Vanilla Ice Cream! 🍦");

}

}

**🏭 Step 3: Create the Factory**

public class IceCreamFactory {

public static IceCream getIceCream(String flavor) {

if (flavor.equalsIgnoreCase("chocolate")) {

return new ChocolateIceCream();

} else if (flavor.equalsIgnoreCase("vanilla")) {

return new VanillaIceCream();

} else {

throw new IllegalArgumentException("Unknown flavor: " + flavor);

}

}

}

**🎮 Step 4: Use the Factory**

public class Main {

public static void main(String[] args) {

IceCream ice1 = IceCreamFactory.getIceCream("chocolate");

IceCream ice2 = IceCreamFactory.getIceCream("vanilla");

ice1.eat();

ice2.eat();

}

}

**✅ Output:**

Eating Chocolate Ice Cream! 🍫

Eating Vanilla Ice Cream! 🍦

**🔥 Summary**

* You ask the **Factory** for an object (by giving a type or name).
* The factory **decides** which class to create.
* Useful when object creation logic is **complex or dynamic**.

**🧠 3. What is the Abstract Factory Pattern?**

The **Abstract Factory Pattern** provides an interface to **create families of related objects** without specifying their concrete classes.

Think of it as a **super factory** that creates other factories.

**🎯 Key Points:**

* Use **Factory Pattern** when you need to create **one object**.
* Use **Abstract Factory Pattern** when you need to create **families of related objects**.

**🧱 4. Builder Pattern — For building complex objects step-by-step**

**🎯 Real-life Example:**

Imagine you’re building a **burger 🍔** at a burger shop.

You don’t create the burger in one go — instead, you add:

* Bun
* Patty
* Cheese
* Sauce

And finally, say **“Done!”**.

That’s what the **Builder Pattern** does — **step-by-step construction** of an object.

**✅ Use When:**

* You have a **complex object** with many parts.
* You want to avoid a **constructor with too many parameters**.
* You want to make the object **immutable** once built.

**🧬 5. Prototype Pattern — For cloning objects**

**🎯 Real-life Example:**

Imagine you are a **cookie maker 🍪**. Instead of baking from scratch every time, you just **copy the first perfect cookie** (clone it).

That’s what **Prototype** is about — **copying an existing object** to create a new one.