Design patterns

Design patterns are **proven, reusable solutions** to common problems that software developers face while designing applications.

They are **not code**, but **general templates** that guide how to structure your classes and objects.

Think of them as **best practices** used by experienced programmers.

Definition

A design pattern is a standard way to solve a common software design problem.

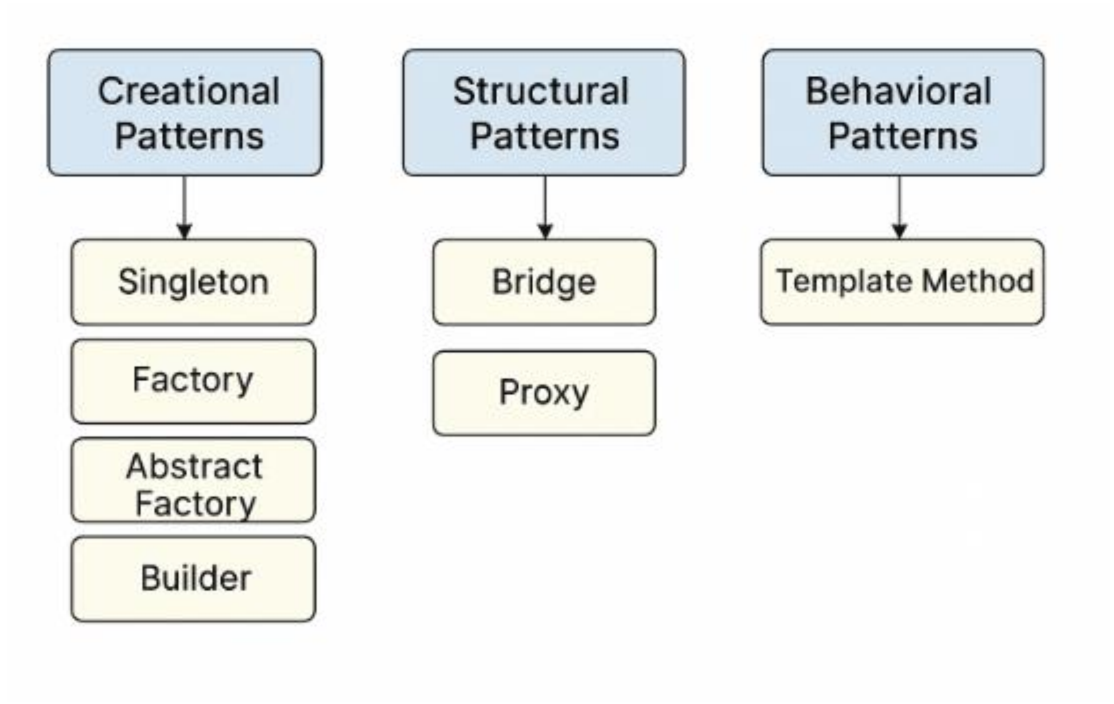
Why Do We Use Design Patterns?

- Improve **code reusability**
- Make code **easy to understand**
- Provide **well-tested solutions**
- Reduce **development time**
- Improve **communication** between developers (“**Use a Factory here”)**

Types of Design Patterns

There are 3 major categories:

1. **Creational Patterns** → deal with object creation
2. **Structural Patterns** → deal with class/object composition
3. **Behavioral Patterns** → deal with object interaction/algorithms



Real-Time Examples

Below are simple real-life analogies + actual software examples.

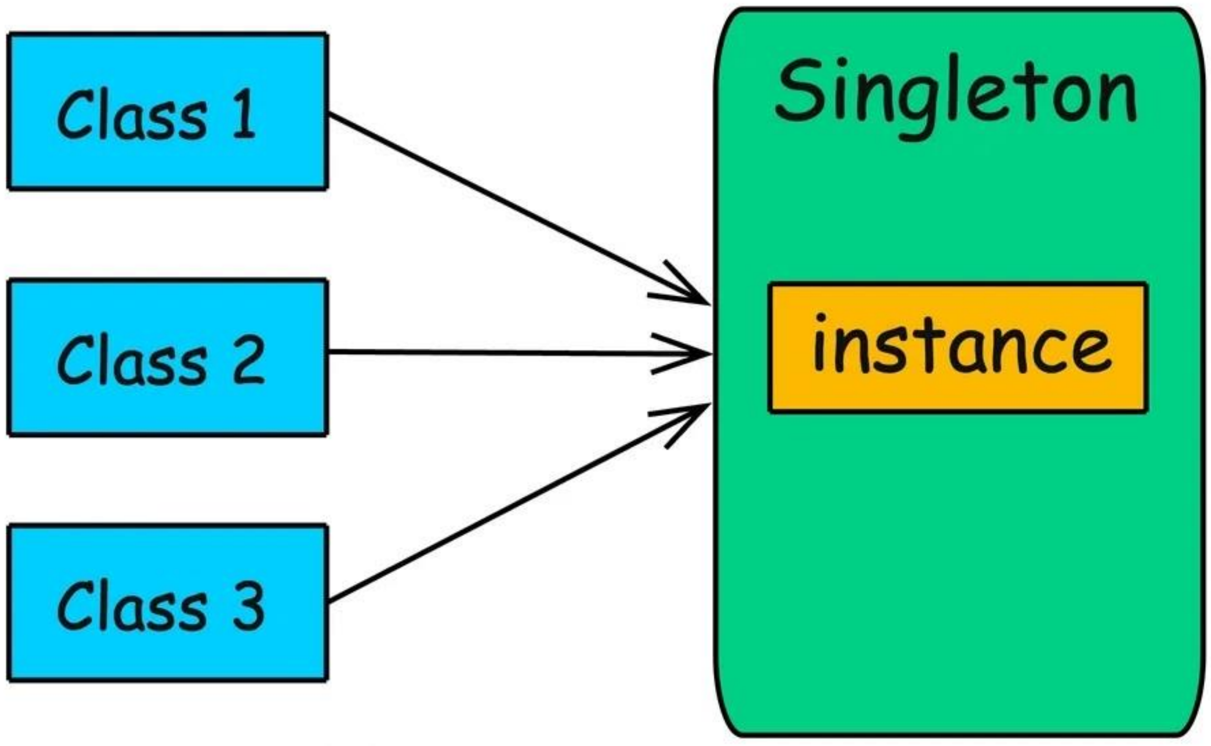
---

1 Singleton Pattern (Creational)

Real-Time Example: "Only one CEO in a company"

- A company can have multiple employees
- But only **one CEO**
- Everyone refers to the same CEO

Software Example:
Database connection → Only one instance is shared across the app.



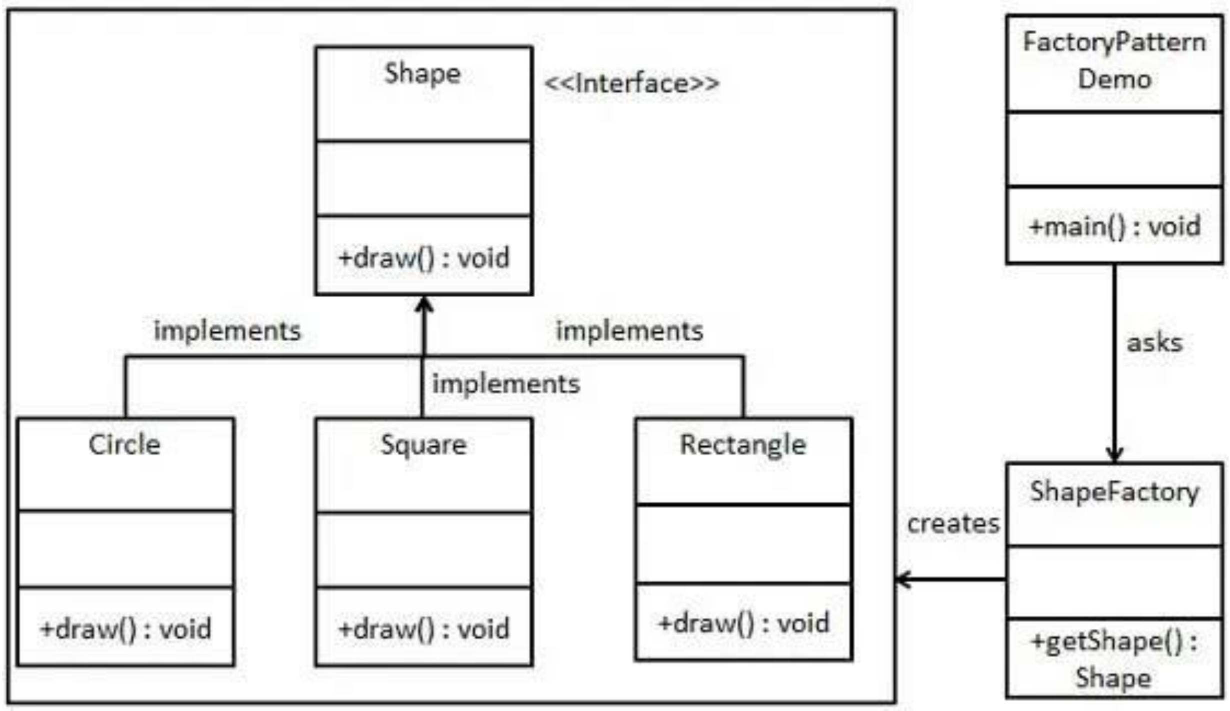
---

2 Factory Pattern (Creational)

Real-Time Example: "Car Factory"

- You tell the factory: *I want a Sedan / SUV / Truck*
- Factory produces the correct car
- You don't build the car yourself

Software Example:
A ShapeFactory creates Circle, Square, Rectangle objects based on input.



---

3. Abstract Factory Pattern (Creational)

Real-Time Example: "Furniture Family Factory"

You buy:

- Victorian Chair
- Victorian Table
- Victorian Sofa
  (All belong to **same family**)

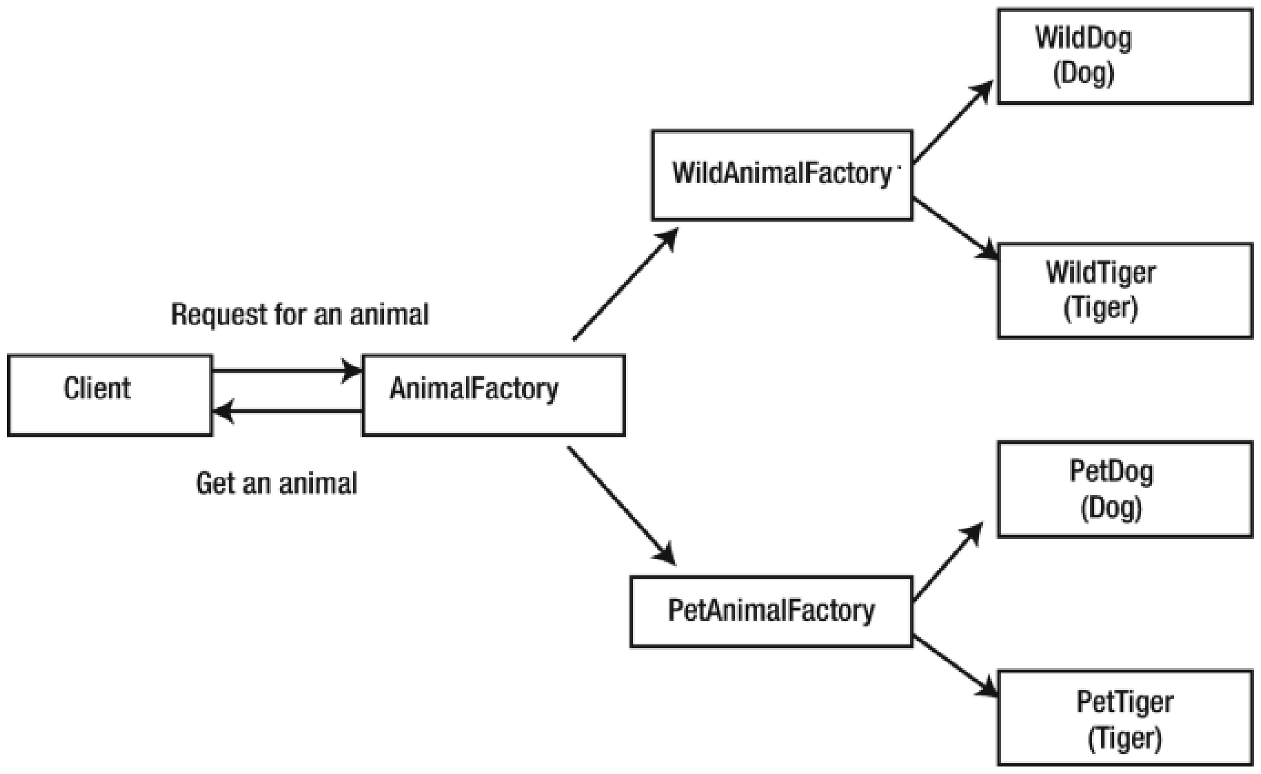
Or you choose:

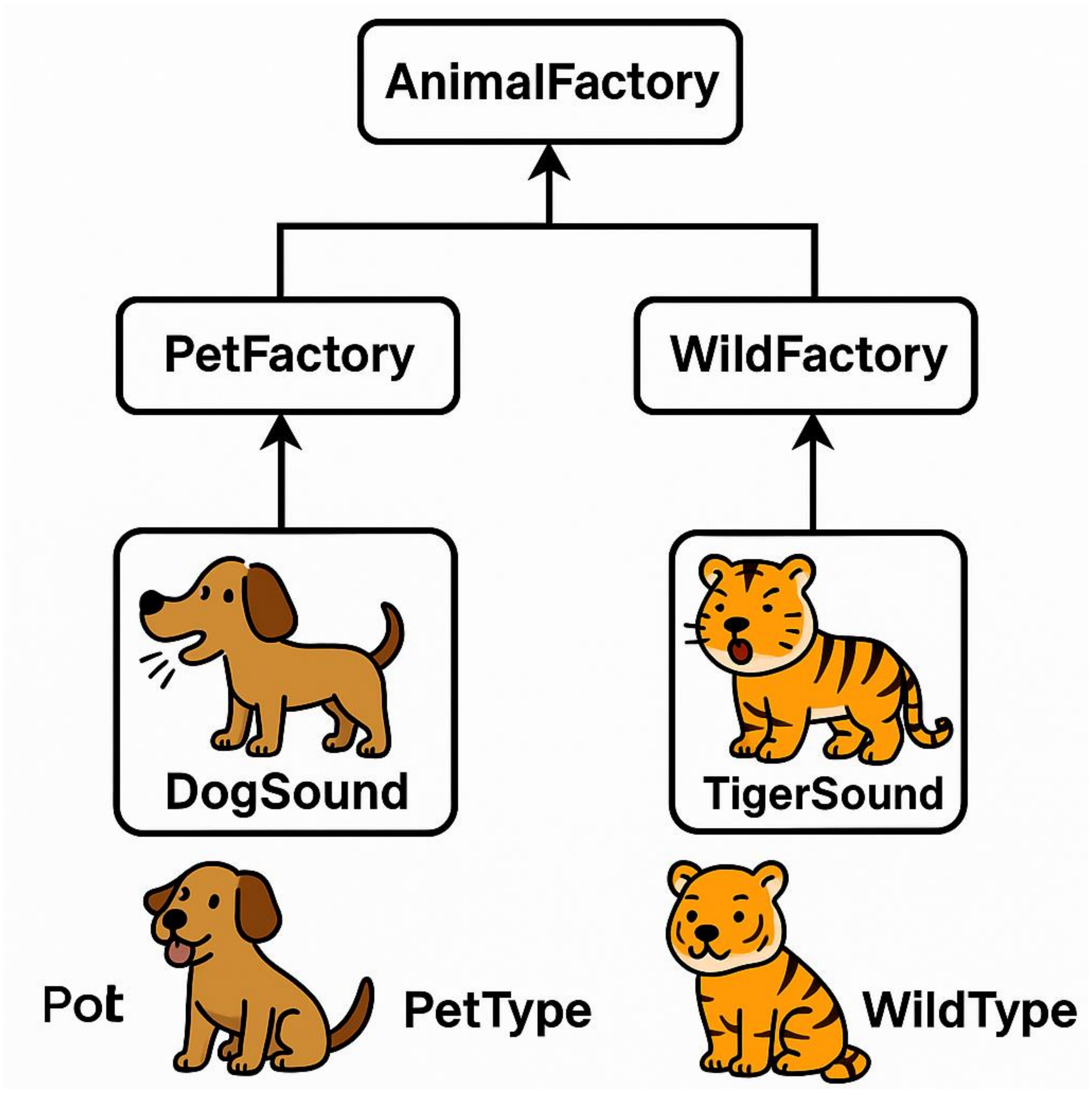
- Modern Chair
- Modern Table
- Modern Sofa

Abstract Factory ensures matching families.

Software Example:
Creating **UI themes** (Light Theme, Dark Theme)

- **Light Button + Light TextField**
- **Dark Button + Dark TextField**





---

4 Builder Pattern (Creational)

Real-Time Example: "Pizza Builder"

You choose:

- Size

- Cheese
- Toppings
- Crust

Pizza is built **step-by-step**.

Software Example:
Building complex objects like:

- HTTP Requests
- User Objects
- Car objects

---

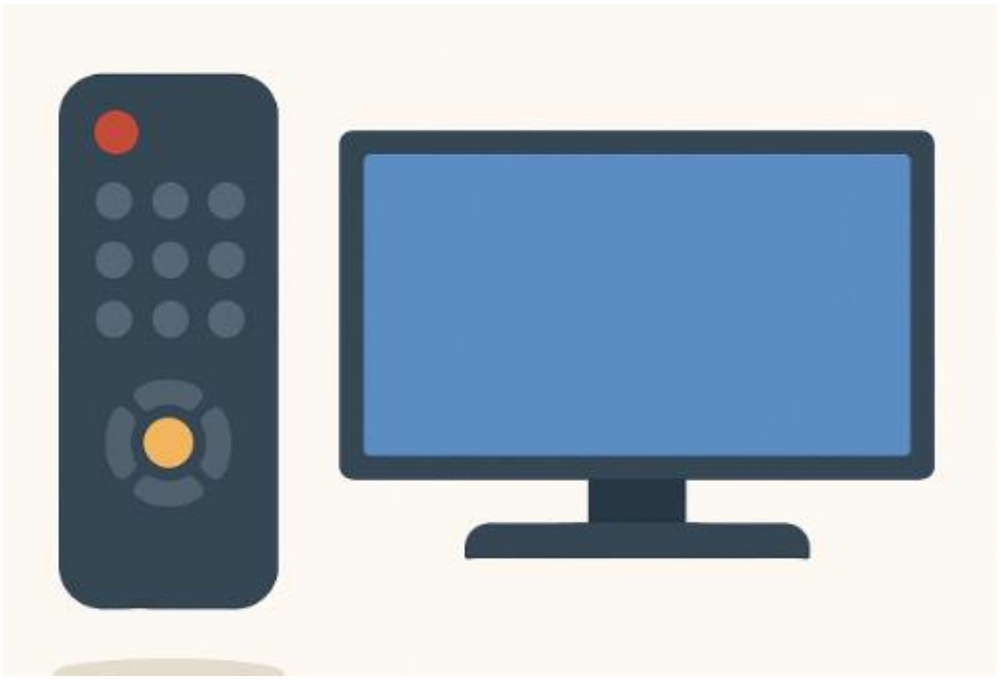
5 Bridge Pattern (Structural)

Real-Time Example: "Remote → TV"

One remote works with:

- Sony TV
- Samsung TV
- LG TV

Remote = Abstraction
TV = Implementation
Both can change independently.



---

6 Proxy Pattern (Structural)

Real-Time Example: "Credit Card is a proxy for Cash"

- You don’t carry cash
- Credit card acts as a **proxy** to your bank account

Software Example:

- Proxy: controls access to heavy objects
- Lazy-loading images
- Internet access control

---

7 Template Method Pattern (Behavioral)

Real-Time Example: "Cooking recipe"

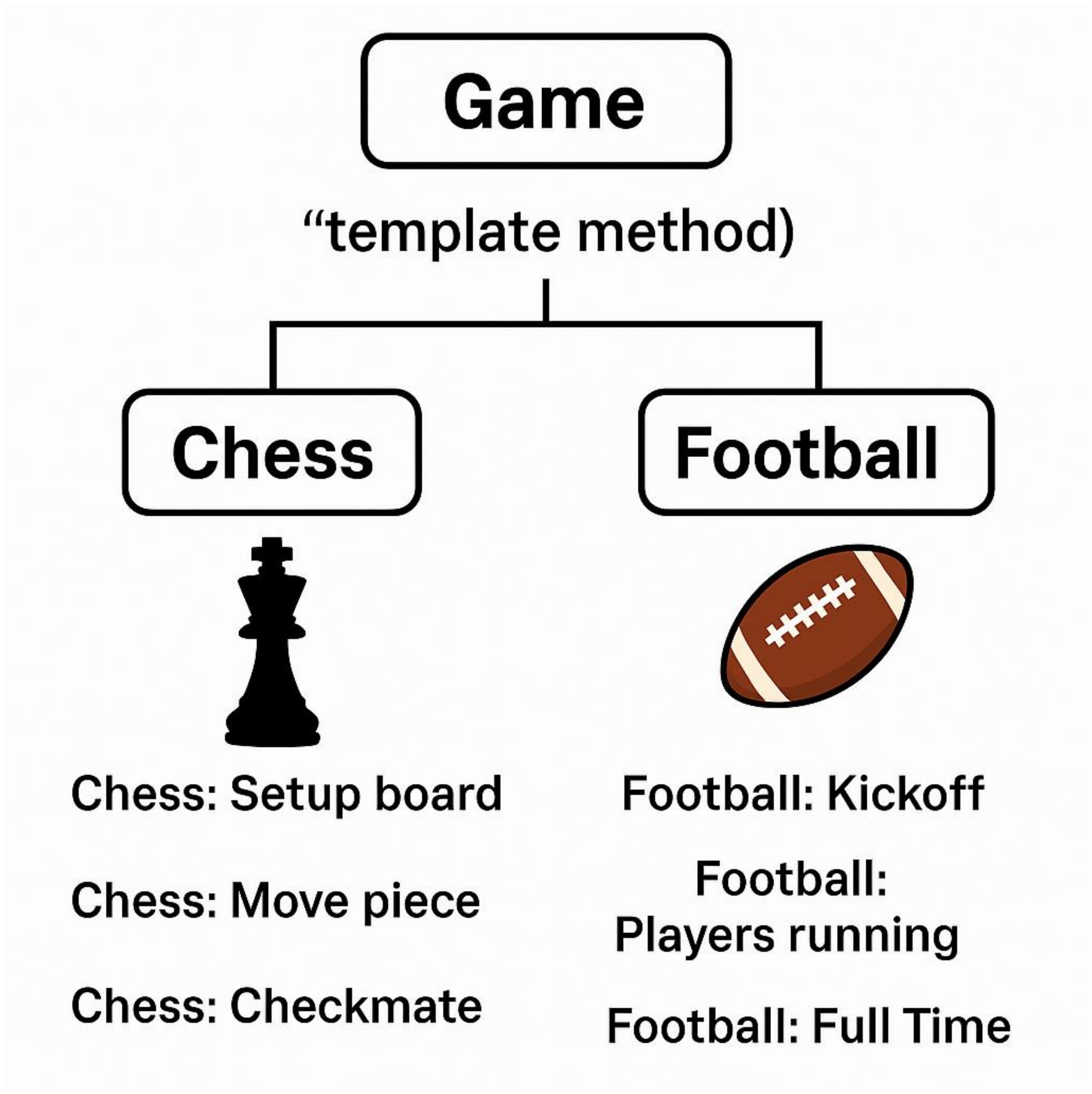
Steps are same:

1. Wash
2. Cook
3. Serve

But different dishes implement the steps differently.

Software Example:
Games:

- Cricket game steps
- Chess game steps
- Football game steps



---

8 Immutable Class

Real-Time Example: “Your Birth Certificate”

Once created:

- Name fixed

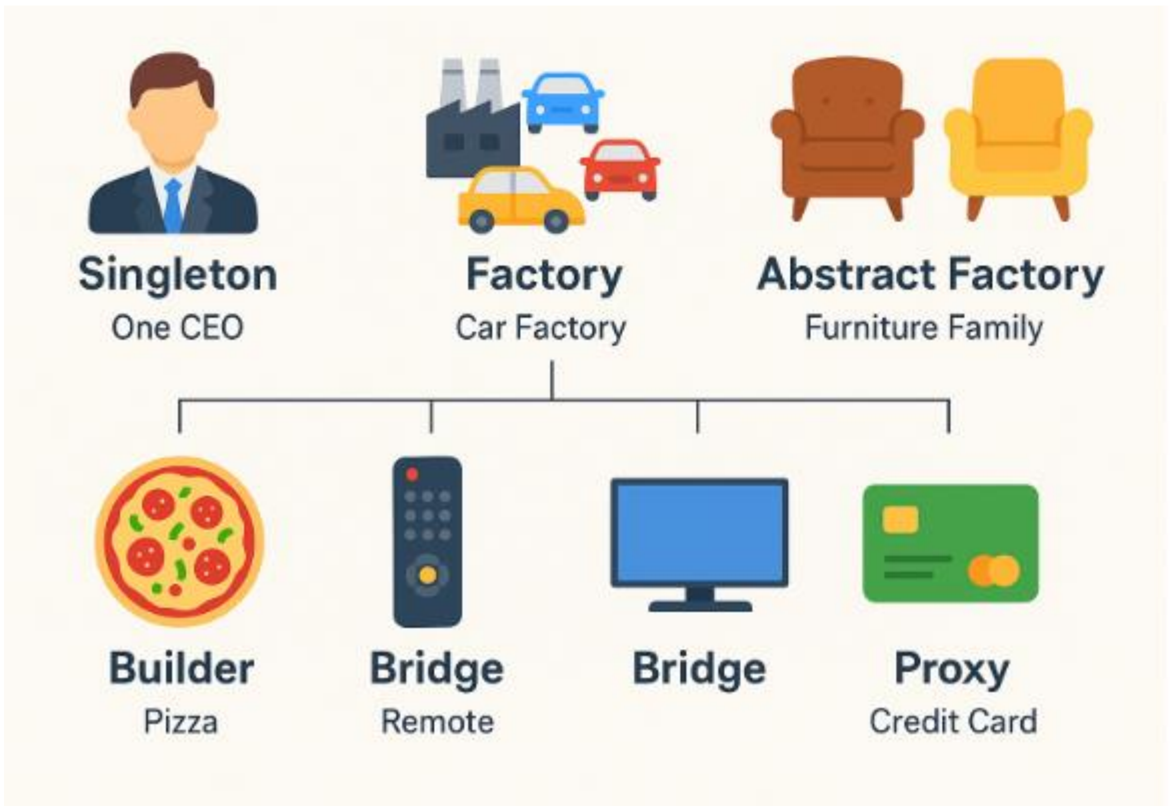
- Date of birth fixed
- Parents fixed

Software Example:

- Java String class
- Data objects used in multithreading

□ In Short

Pattern	Solves What Problem?
Singleton	Only one object needed
Factory	Choose correct object
Abstract Factory	Create related object families
Builder	Build complex objects easily
Bridge	Decouple abstraction from implementation
Proxy	Protect / control / enhance real object
Template Method	Define algorithm in steps
Immutable Class	Create unchangeable objects


Singleton
One CEO
Factory
Car Factory
Abstract Factory
Furniture Family
Builder
Pizza
Bridge
Remote
Bridge
Proxy
Credit Card