All Design Pattern(low level design)

Object-Based Modeling ek technique hai jisme real-world entities ko **objects** ke form mein represent kiya jata hai. Ye objects ke andar **attributes** (data) aur **methods** (functions) hote hain.

Main Points:

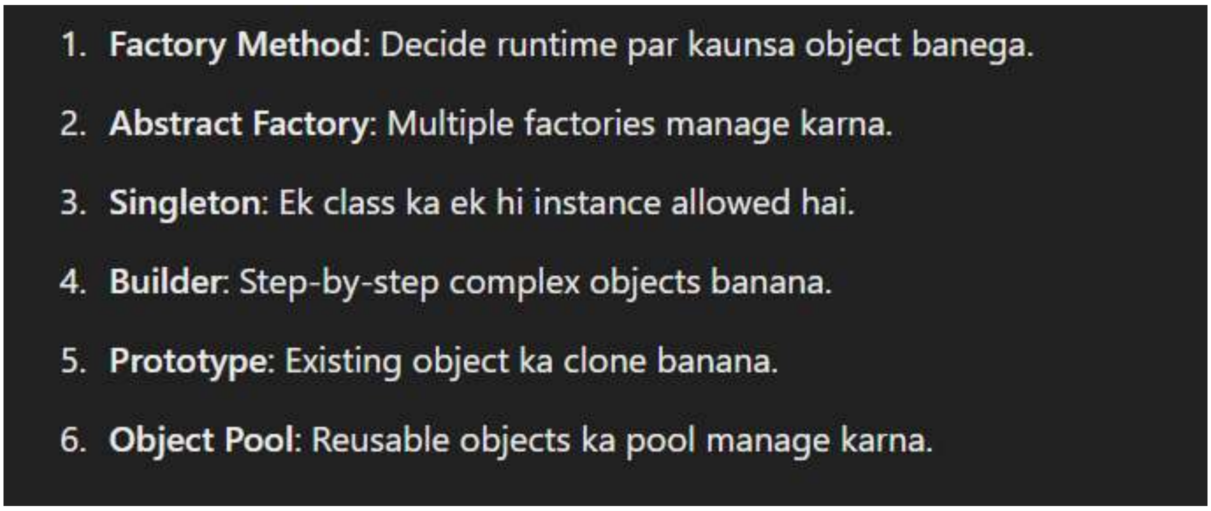
1. **Objects**: Real-world entities ka representation. Example: Car, Book.
2. **Classes**: Objects banane ki blueprint. Example: "Car" class se Honda aur Maruti objects bante hain.
3. **Encapsulation**: Data aur methods ko ek object me bundle karna.
4. **Relationships**: Objects ke beech interaction. Example: Member borrows Book.
5. **Behavior**: Objects kaise interact karte hain.

Example:

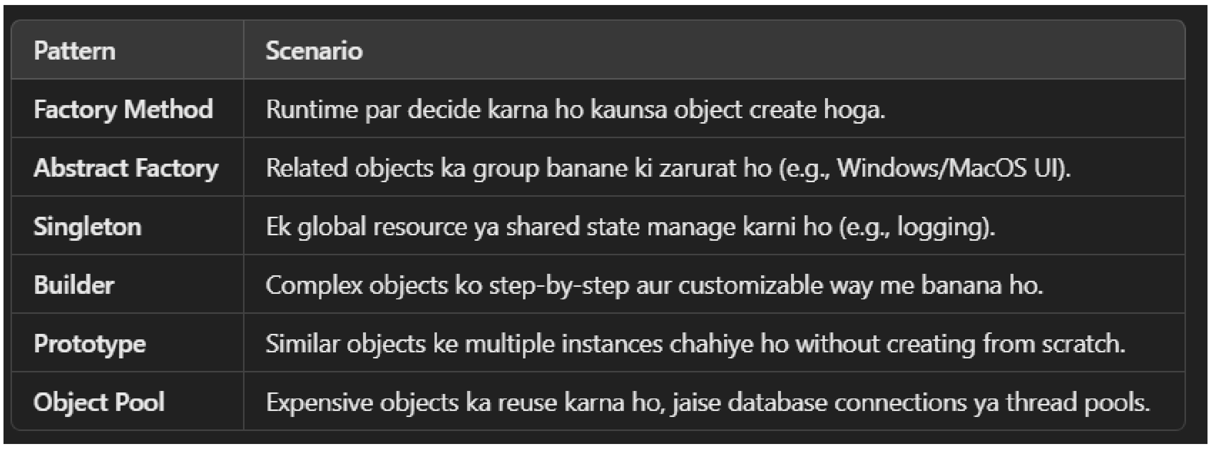
- **Library System**:
    - Objects: Book, Member, Librarian.
    - Attributes: Book ke liye title, author; Member ke liye name, ID.
    - Methods: issueBook(), returnBook().

Creational Pattern

Creational Design Patterns software design ke wo templates hote hain jo object creation ko simplify aur organize karte hain. Ye patterns ensure karte hain ki objects efficiently create ho aur system ke flexibility aur reusability maintain ho.



1. **Factory Method**: Decide runtime par kaunsa object banega.
2. **Abstract Factory**: Multiple factories manage karna.
3. **Singleton**: Ek class ka ek hi instance allowed hai.
4. **Builder**: Step-by-step complex objects banana.
5. **Prototype**: Existing object ka clone banana.
6. **Object Pool**: Reusable objects ka pool manage karna.



Pattern	Scenario
Factory Method	Runtime par decide karna ho kaunsa object create hoga.
Abstract Factory	Related objects ka group banane ki zarurat ho (e.g., Windows/MacOS UI).
Singleton	Ek global resource ya shared state manage karni ho (e.g., logging).
Builder	Complex objects ko step-by-step aur customizable way me banana ho.
Prototype	Similar objects ke multiple instances chahiye ho without creating from scratch.
Object Pool	Expensive objects ka reuse karna ho, jaise database connections ya thread pools.

1. Factory Method

- **Explanation**:
  Factory Method ek interface provide karta hai jo subclasses ko allow karta hai ki wo runtime par decide karein kaunsa object create hoga. and all business logic and object keep at one place(agr kuchh changes bhi krna hua to ek bar sirf factory class me krenge) .
- **Scenario**:
  - Jab object creation runtime par depend kare.
  - Jab har object ka exact type pehle se fix na ho.
- **Example**: notification system

2. Abstract Factory

- **Explanation**:
  Factory ka ek group (factory of factory) jo related objects banata hai bina exact classes ko specify kiye.
- **Scenario**:

    - Jab multiple families of related objects create karne ho.
    - Jab platform-specific objects banane ho (e.g., Windows vs MacOS).
- **Example**: UI Component Factory

3. Singleton

- **Explanation**:
  Singleton ek design pattern hai jo ensure karta hai ki kisi class ka ek hi instance ho aur us instance ko globally accessible banata hai..
- **Scenario**:
    - Jab ek shared resource ho (e.g., logging system, configuration settings).
- **Example**: Logger Class

4. Builder

- **Explanation**:
  Complex objects ko step-by-step construct karta hai, aur customize karne ki flexibility deta hai.
- **Scenario**:
    - Jab ek object ke multiple configurations ho.
- **Example**: Pizza Builder

5. Prototype

- **Explanation**:
  Ek existing object ka shallow ya deep copy banata hai.
- **Scenario**:
    - Jab object creation expensive ho, ya ek object ke multiple similar copies chahiye ho.
- **Example**: Shape Cloning

Structural Design Patterns

Structural design patterns object aur class ke relationships ko simplify karte hain, taki code modular aur flexible rahe. Ye patterns focus karte hain structure aur composition par.

Combination of different objects and classes.

1. Adapter Pattern

- **Explanation**:
  Do incompatible interfaces ko connect karne ke liye ek adapter class banata hai.
- **Scenario**:
  Jab ek class ki functionality dusri class ke interface se match nahi karti ho.
- **Example**:
  Phone charger adapter jo Type-C ko USB mein convert karta hai.

2. Bridge Pattern

- **Explanation**:
  Abstraction (concept) aur **implementation** (code) ko alag-alag rakhta hai, taki dono ko independently change kiya ja sake.
- **Scenario**:
  Jab ek cheez ke multiple features aur implementations ho (e.g., shapes with colors).
- **Example**:
  Shapes (Circle, Rectangle) ko different colors ke saath draw karna.

3. Composite Pattern

- **Explanation**:
  Ek **tree structure** banata hai jisme objects (single ya group) ko ek jaisa treat kiya jaye.
- **Scenario**:
  Jab single item aur group of items ko ek hi logic se handle karna ho (e.g., files and folders).

- **Example**:
  Folder ke andar files aur sub-folders ka system.

4. Decorator Pattern

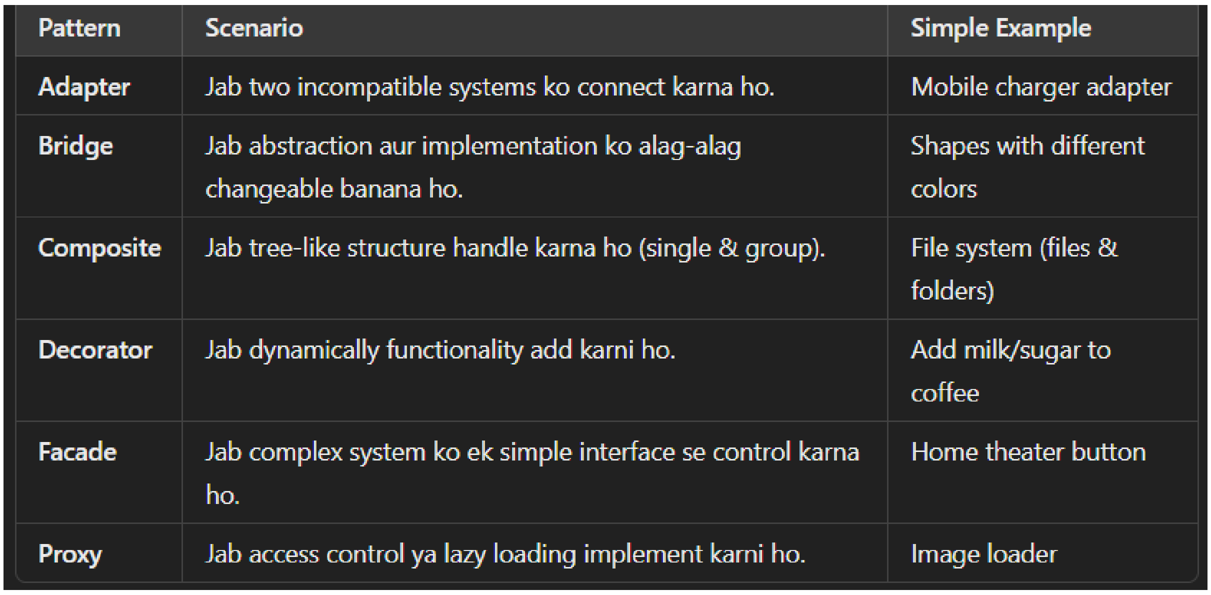
- **Explanation**:
  Ek object ki existing functionality ko dynamically badhata hai bina uski original class ko modify kiye.
- **Scenario**:
  Jab ek object ka behavior customize karna ho (e.g., coffee customization).
- **Example**:
  Coffee ke andar milk ya sugar add karna.

5. Facade Pattern (mukhauta) like light system

- **Explanation**:
  Multiple complex systems ko access karna easy banata hai by providing a single simplified interface (hide complexity like threater system).
- **Scenario**:
  Jab user ko ek hi interface ke through pura system control karwana ho (e.g., home theater).
- **Example**:
  Ek button dabao aur pura home theater system ready ho jaye (kaise hua user ko nhi pta complexity).

6. Proxy Pattern

- **Explanation**:
  Ek **proxy** class real object ka representative hoti hai jo access control ya lazy loading provide karti hai.
- **Scenario**:
  Jab ek heavy object ko lazily initialize karna ho ya access ko restrict karna ho.
- **Example**:
  Image ka proxy loader, jo tabhi load kare jab zarurat ho.



Pattern	Scenario	Simple Example
Adapter	Jab two incompatible systems ko connect karna ho.	Mobile charger adapter
Bridge	Jab abstraction aur implementation ko alag-alag changeable banana ho.	Shapes with different colors
Composite	Jab tree-like structure handle karna ho (single & group).	File system (files & folders)
Decorator	Jab dynamically functionality add karni ho.	Add milk/sugar to coffee
Facade	Jab complex system ko ek simple interface se control karna ho.	Home theater button
Proxy	Jab access control ya lazy loading implement karni ho.	Image loader

Behavioral Design Patterns

Behavioral patterns ka focus **objects ke interaction aur communication** par hota hai. Ye patterns ensure karte hain ki objects ke beech relationships flexible aur maintainable rahen.

1. Chain of Responsibility Pattern

- **Explanation**:
  Ek request ko sequentially multiple handlers ke through pass karne ka system provide karta hai, jab tak ek handler request ko process na kare.
- **Scenario**:
  Jab multiple objects ko ek request handle karne ka chance dena ho.
- **Example**:
  Complaint handling system.

2. Command Pattern

- **Explanation**:
  Actions ko objects ke roop mein wrap karta hai, taki unhe dynamically execute, undo ya redo kiya ja sake.
- **Scenario**:
  Jab action ko runtime par execute ya queue karna ho.
- **Example**:
  TV remote button press.

3. Iterator Pattern

- **Explanation**:
  Collections ke elements ko sequentially access karne ka system provide karta hai bina internal details reveal kiye.
- **Scenario**:
  Jab collections (e.g., list, array) ko iterate karna ho.
- **Example**:
  Songs playlist iterator.

4. Mediator Pattern

- **Explanation**:
  Multiple objects ke interaction ko ek **mediator** object ke through manage karta hai, taki loose coupling ho.
- **Scenario**:
  Jab objects directly communicate karne ke bajay ek mediator se baat karein.
- **Example**:
  Chat room system.

5. Observer Pattern

- **Explanation**:
  Ek object (subject) ke state change hone par saare dependent objects (observers) ko notify karta hai.
- **Scenario**:
  Jab ek object ke update hone par multiple objects ko notify karna ho.
- **Example**:
  News subscription system.

6. Strategy Pattern

- **Explanation**:
  Ek family of algorithms ko define karta hai aur runtime par dynamically select karne ki facility deta hai.
- **Scenario**:
  Jab multiple algorithms mein se dynamically choose karna ho.
- **Example**:
  Payment system (Credit card, PayPal, etc.).

7. State Pattern

- **Explanation**:
  Ek object apne behavior ko state ke basis par change karta hai.
- **Scenario**:
  Jab ek object ke behavior ko dynamically change karna ho (e.g., document lifecycle).
- **Example**:
  Traffic light system.

1. Template Pattern

- **Explanation**:
  Ye pattern ek **framework define karta hai** jisme steps ka structure fix hota hai, lekin kuch steps ko subclasses implement karte hain.
- **Scenario**:
  Jab ek algorithm ka structure fix ho, lekin kuch steps customize karne ki zarurat ho.
- **Example**:
  Food preparation system - Cooking steps fix hain (prepare, cook, serve), lekin recipe (spices, ingredients) vary karti hai.

2. Interpreter Pattern

- **Explanation**:
  Ye pattern ek **language ya grammar ke rules define karta hai aur expressions ko evaluate karta hai**.

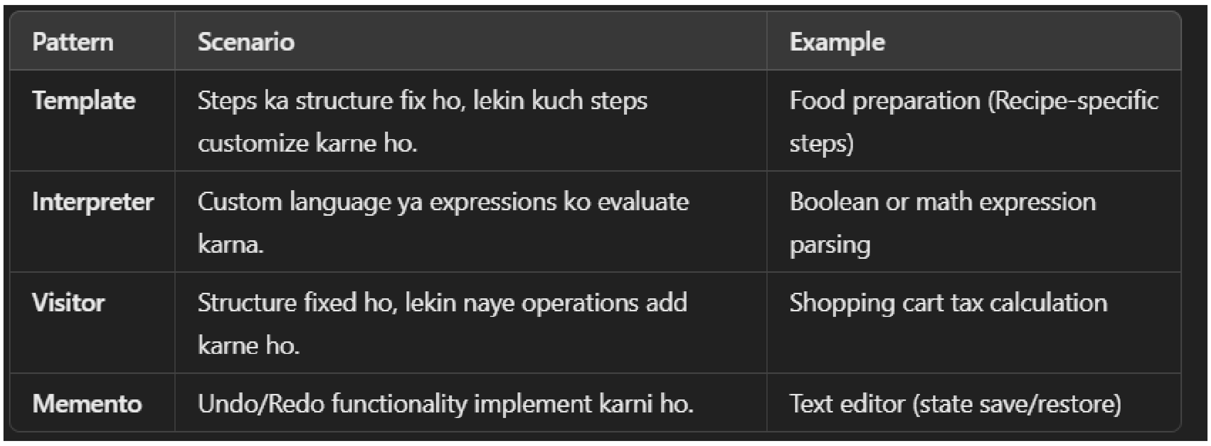
- **Scenario**:
  Jab kisi **custom language** ko parse karna ho ya mathematical expressions ko evaluate karna ho.
- **Example**:
  Boolean expressions (A AND B).

3. Visitor Pattern

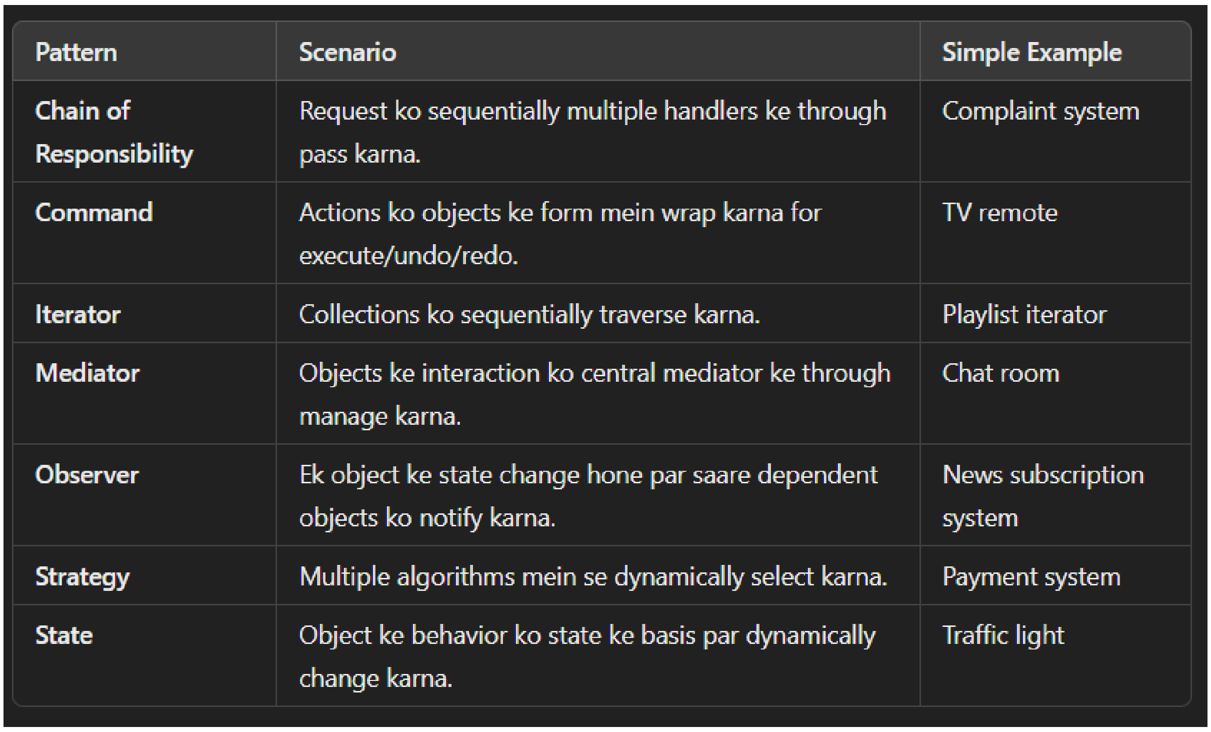
- **Explanation**:
  Visitor pattern allow karta hai **ek object structure ke elements par operations perform karna bina structure ko modify kiye**.
- **Scenario**:
  Jab naye operations add karne ho bina objects ko modify kiye.
- **Example**:
  Shopping cart mein alag-alag items ke liye tax calculation.

4. Memento Pattern

- **Explanation**:
  Ye pattern ek object ki state ko save aur restore karne ka system provide karta hai.
- **Scenario**:
  Jab kisi system ko **undo/redo** functionality chahiye.
- **Example**:
  Text editor mein document ka state save karna.



Pattern	Scenario	Example
Template	Steps ka structure fix ho, lekin kuch steps customize karne ho.	Food preparation (Recipe-specific steps)
Interpreter	Custom language ya expressions ko evaluate karna.	Boolean or math expression parsing
Visitor	Structure fixed ho, lekin naye operations add karne ho.	Shopping cart tax calculation
Memento	Undo/Redo functionality implement karni ho.	Text editor (state save/restore)



Pattern	Scenario	Simple Example
Chain of Responsibility	Request ko sequentially multiple handlers ke through pass karna.	Complaint system
Command	Actions ko objects ke form mein wrap karna for execute/undo/redo.	TV remote
Iterator	Collections ko sequentially traverse karna.	Playlist iterator
Mediator	Objects ke interaction ko central mediator ke through manage karna.	Chat room
Observer	Ek object ke state change hone par saare dependent objects ko notify karna.	News subscription system
Strategy	Multiple algorithms mein se dynamically select karna.	Payment system
State	Object ke behavior ko state ke basis par dynamically change karna.	Traffic light