**1.Khái niệm:**

Design Pattens là các giải pháp chung được thiết kế sẵn để giải quyết những vấn đề phổ biến trong OOP.Trong Java,các mẫu thiết kế giúp cải thiện khả năng tái sử dụng mã,tính linh hoạt và dễ bảo trì của các ứng dụng.

Có ba loại chính của các mẫu thiết kế trong Java:

**1.Creational Pattents(Mẫu khỏi tạo)**

Mẫu khởi tạo giúp quản lý quá trình tạo đối tượng,thường che giấu sự phức tạp của việc khởi tạo và làm cho quá trình này trở nên linh hoạt hơn.

**Một số mẫu phổ biến:**

* **Singleton Pattern:** Đảm bảo rằng chỉ có duy nhất một instance của một class và cung cấp một điểm toàn cục cho nó
* **Factory Method Pattern**: Cung cấp một giao diện để tạo đối tượng,nhưng để các lớp con quyết định lớp nào sẽ được khởi tạo
* **Builder Pattern:**Tách việc xây dựng một đối tượng phức tạp ra khỏi biểu diễn của nó,cho phép xây dựng đối tượng đó theo nhiều bước khác nhau

**2.Structural Patterns(Mẫu cấu trúc)**

Mẫu cấu trúc giải quyết vấn đề về cách tổ chức các đối tượng và lớp để tạo thành các cấu trúc lớn hơn mà vẫn đảm bảo tính linh hoạt và hiệu quả.

**Một số mẫu phổ biến:**

* **Adapter Pattern:** Cho phép các lớp có giao diện không tương thích làm việc với nhau bằng cách cung cấp một lớp Adapter
* **Decorator Pattern:**Cho phép thêm chức năng vào một đối tượng mà không làm thay đổi cấu trúc của lớp

**3.Behavioral Patterns(Mẫu hành vi)**

Mẫu hành vi tập trung vào cách các đối tượng giao tiếp với nhau và cách chúng phân chia trách nhiệm để thực hiện các hành vi phức tạp

**Một số mấu phổ biến:**

* **Observer Pattern:** Định nghĩa mối quan hệ phụ thuộc một-nhiều, sao cho khi một đối tượng thay đổi trạng thái, các đối tượng phụ thuộc khác sẽ tự động được thông báo và cập nhật.
* **Strategy Pattern:** Cho phép lựa chọn các thuật toán khác nhau tại runtime bằng cách đóng gói từng thuật toán vào một lớp riêng biệt.

**2.Ưu điểm và nhược điểm:**

**1.Ưu điểm:**

* **Tái sử dụng mã:**Các mẫu thiết kế cung cấp các giải pháp chuẩn mà ta có thể tái sử dụng nhiều lần
* **Dễ bảo trì:**Bằng cách sử dụng các mẫu thiết kế đã được kiểm chứng,ta có thể giảm các lỗi phát sinh và cải thiện tính bảo trì của mã.
* **Tăng tính linh hoạt:**Các mẫu thiết kế thường tập trung vào việc làm cho hệ thống dễ dàng mở rộng mà không thay đổi quá nhiều mã hiện tại
* **Giảm sự phụ thuộc:**Nhiều mẫu giúp tách biệt các thành phần trong hệ thống,làm cho chúng ít phụ thuộc vào nhau hơn

**2.Nhược điểm:**

* **Phức tạp hơn:**Đôi khi việc áp dụng mẫu thiết kế có thể làm tang sự phức tạp của mã nếu không đc sử dụng đúng cách
* **Quá mức cần thiết:**Sử dụng mẫu thiết kế ở những nơi đơn giản có thể làm tang khối lượng công việc và gây khó hiểu cho người mới

**3.Singleton và Factory**

**1.Singleton:**

**Khái niệm:**Mẫu thiết kế Singleton đảm bảo rằng một lớp chỉ có một instance(thể hiện duy nhất) trong suốt vòng đời của ứng dụng và cung cấp một điểm truy cập toàn cục để lấy instance đó.

**Các cách triển khai Singleton trong Java:**

Có nhiều cách để triển khai Singleton,bao gồm:

**1.Eager Initialization(Khởi tạo trước):**

-Trong cách này,instance được tạo ngay khi lớp được tải lên bộ nhớ,không quan tâm đến việc có sử dụng instance đó hay không,cách này đơn giản,dễ triển khai nhưng gây lãng phí bộ nhớ nếu không bao giờ được sử dụng

**2.Lazy Initialization(Khởi tạo khi cần):**

-Instance chỉ được tạo khi cần thiết,tức là khi getInstance được gọi lần đầu tiên

|  |  |  |
| --- | --- | --- |
|  | **Singleton** | **Factory** |
| **Mục đích** | Đảm bảo chỉ có một instance duy nhất của lớp | Cung cấp phương thức để tạo đối tượng mà không cần chỉ rõ lớp cụ thể |
| **Trường hợp sử dụng** | Khi cần quản lý một tài nguyên duy nhất(logger,database,connection). | Khi cần thay đổi loại đối tượng được tạo ra mà không làm thay đổi mã nguồn |
| **Phương thức** | Trả về một instance duy nhất của class | Trả về một đối tượng của lớp con,được tạo ra thông qua phương thức factory |
| **Ưu điểm** | Dễ dàng truy cập và chỉ có một instance duy nhất | Dễ dàng thay đổi lớp đối tượng mà không thay đổi mã sử dụng. |
| **Nhược điểm** | Không thể tạo nhiều instance | Cần thêm các lớp creator, tăng sự phức tạp. |