Padrões de Projeto no Sistema de Gerenciamento de Biblioteca (SGB)

1. **Singleton/Facade (Verde Claro)**

Classes envolvidas:
Fachada

Objetivo:

O padrão **Singleton** garantirá que exista apenas uma única instância da classe **Fachada** em todo o sistema, funcionando como um ponto de acesso global. Já o padrão **Facade** (ou Fachada) tem como objetivo simplificar a interação com o sistema, escondendo a complexidade dos seus componentes internos.

A classe **Fachada** atua como a porta de entrada para todas as operações da biblioteca, repassando as solicitações para os gerenciadores responsáveis por Usuários, Livros e Empréstimos. Com isso, os clientes do sistema não precisam se preocupar com os detalhes de cada subsistema.

2. **Command(Laranja)**

Classes envolvidas:
Comando (interface abstrata)
CadastrarUsuarioComando;
AtualizarUsuarioComando;
DeletarUsuarioComando;
CadastrarLivroComando;
AtualizarLivroComando;
DeletarLivroComando;
RealizarEmprestimoComando;
DevolverLivroComando;
Invoker.

Objetivo:

O padrão Command permite encapsular uma solicitação como um objeto, o que traz várias vantagens: é possível parametrizar clientes com diferentes comandos, enfileirar operações e até permitir que ações sejam desfeitas.

No sistema de biblioteca, cada comando representa uma operação específica — como cadastrar um usuário ou realizar um empréstimo. A execução dessas ações fica por conta do **Invoker**, que também mantém um histórico das operações realizadas. Com isso, é possível implementar funcionalidades como "desfazer a última ação".
Esse padrão torna o sistema mais flexível e facilita a adição de novas funcionalidades no futuro, sem a necessidade de alterar o código que já está funcionando.

3. **Memento (Rosa)**

Classes envolvidas

Memento;
GerenciadorUsuarios (guarda mementos);
GerenciadorLivros (guarda mementos).

Objetivo:

O padrão Memento permite capturar e armazenar o estado interno de um objeto sem violar seu encapsulamento, possibilitando que ele seja restaurado para esse estado no futuro.

No sistema de biblioteca, esse padrão é usado para guardar versões anteriores de entidades como usuários e livros. Assim, se uma atualização for feita de forma errada ou indesejada, é possível reverter facilmente para o estado anterior.

Isso torna o sistema mais robusto e melhora a experiência do usuário, já que oferece uma maneira prática de desfazer mudanças e recuperar informações anteriores quando necessário.

4. **Factory Methond(Azul Claro)**

Classes envolvidas:
DAOFactory (interface);
MemoriaDAOFactory;
ArquivoDAOFactor.

Objetivo:

O padrão **Factory Method** define uma interface para a criação de objetos, mas delega às subclasses a responsabilidade de decidir qual classe será instanciada.

No sistema de biblioteca, esse padrão é usado nas fábricas de DAOs, que criam os objetos correspondentes — como UsuarioDAO, LivroDAO ou EmprestimoDAO — de acordo com o tipo de armazenamento escolhido, seja em memória ou em arquivo.

Isso torna o sistema mais flexível, permitindo trocar a estratégia de persistência dos dados sem precisar alterar o código que usa os DAOs. Além disso, centraliza e organiza a lógica de criação dos objetos, deixando o código mais limpo e fácil de manter.

5. **Template Method (Roxo)**

Classes envolvidas:

RelatorioTemplate (classe abstrata);
RelatorioAcessos;
RelatorioEmprestimos.

Objetivo:

O padrão Template Method define a estrutura básica de um algoritmo, deixando que algumas etapas sejam implementadas pelas subclasses.

No sistema de biblioteca, a classe abstrata RelatorioTemplate estabelece a estrutura geral de um relatório — como o cabeçalho, o processamento dos dados e o rodapé. As subclasses, como RelatorioAcessos e RelatorioEmprestimos, implementam as partes específicas, definindo como os dados são processados e como os cabeçalhos devem ser montados para cada tipo de relatório.

Esse padrão favorece o reuso de código ao manter a estrutura comum em um só lugar, ao mesmo tempo que permite personalizações nas subclasses. Isso torna o sistema mais organizado, fácil de manter e flexível para a criação de novos tipos de relatórios no futuro.

6. **DAO (Azu Turquesa)**

Classes envolvidas:

AbstractDAO (interface abstrata);
UsuarioDAO , LivroDAO , EmprestimoDAO (interfaces);
Implementações concretas:
UsuarioDAOMemoria , UsuarioDAOArquivo;
LivroDAOMemoria , LivroDAOArquivo;
EmprestimoDAOMemoria , EmprestimoDAOArquivo.

Objetivo:

O padrão DAO (Data Access Object) tem como objetivo separar a lógica de acesso aos dados da lógica de negócios da aplicação.

No sistema de biblioteca, os DAOs oferecem uma interface padronizada para acessar diferentes tipos de armazenamento, como memória ou arquivos, escondendo os detalhes técnicos de como esses dados são manipulados. Isso permite que o restante do sistema funcione da mesma forma, independentemente de onde ou como os dados estão sendo armazenados.

Além disso, as operações básicas — criar, ler, atualizar e deletar (CRUD) — seguem um padrão comum para todas as entidades, como usuários, livros e empréstimos. Isso facilita a manutenção, promove a reutilização de código e torna o sistema mais organizado e preparado para futuras mudanças na forma de persistência.

7.Adapter

Embora não haja uma classe "Adapter" explícita no sistema, o uso das fábricas de DAOs (DAOFactory, MemoriaDAOFactory, ArquivoDAOFactory) atua como uma ponte entre as camadas de alto nível — como a Fachada e os Gerenciadores — e os diferentes mecanismos de persistência (memória ou arquivo). Essa estrutura desempenha um papel semelhante ao do padrão Adapter, ao permitir que o sistema utilize diferentes implementações de armazenamento através de uma interface padronizada (UsuarioDAO, LivroDAO, etc.), sem precisar conhecer os detalhes de cada uma.

8. Abstract Factory (Azul Claro)

Classes envolvidas:

DAOFatory;
MemoriaDAOFactory;
ArquivoDAOFactory.

Objetivo:

O padrão **Abstract Factory** ajuda a tornar o sistema mais flexível e organizado quando se trata de lidar com diferentes formas de armazenamento de dados. Em vez de o código precisar saber exatamente se está lidando com dados em memória ou em arquivos, ele simplesmente pede para a "fábrica" (DAOFactory) fornecer os objetos certos para cada caso. Assim, tudo continua funcionando da mesma forma, independentemente de onde os dados realmente estão sendo guardados. Isso facilita muito caso seja necessário trocar o tipo de armazenamento no futuro, além de manter o código mais limpo e fácil de manter.