DESIGN PATTERNS

Entrega 5.0

Padrões de Criação:

Utilizamos o **Singleton** como **Gerenciador de conexão com o banco de dados.** Queremos garantir que só exista uma única instância de conexão com o banco de dados em todo o sistema. A classe mantém uma instância estática de si mesma e só permite a criação de um único objeto através do método **getInstance()**

```
public class DB_Connection {
    private static DB_Connection instance;

    private DB_Connection() {
        System.out.println("Conexão com o banco de dados estabelecida.");
    }

    public static DB_Connection getInstance() {
        if (instance == null) {
            instance = new DB_Connection();
        }
        return instance;
    }
}
```

Utilizamos o **Factory Method** como **Criador de Flashcards.**

Com ele, conseguimos criar objetos de forma mais flexível e centralizada, sem precisar chamar new FlashCard(...) diretamente. O método criarFlashcard retorna um novo FlashCard, permitindo que futuramente possamos criar outros tipos de Flashcards sem modificar o código principal.

```
package projeto_flashcards;

// Factory Method: Criador de Flashcards
public class FlashCardFactory {
      public static FlashCard criarFlashcard(String pergunta, String
resposta) {
        return new FlashCard(pergunta, resposta);
    }
}
```

Padrões Estruturais:

Utilizamos o **Facade** como **Gerenciador de Flashcards.**

Com ele, conseguimos esconder a complexidade do sistema e fornecer uma interface mais simples para a Main. A FlashcardFacade gerencia Deck e SistemaNotificacao, permitindo que a Main interaja com o sistema sem precisar conhecer seus detalhes internos.

```
public class FlashCardFacade {
        private Deck deck;
           private SistemaNotificacao sistemaNotificacao;

           public FlashCardFacade() {
               this.sistemaNotificacao = new SistemaNotificacao();
           }

           public void criarDeck(String nome) {
               this.deck = new Deck(nome);
               System.out.println("Deck '" + nome + "' criado com sucesso!");
           }

           public void adicionarFlashcard(String pergunta, String resposta) {
               if (deck == null) {
                   System.out.println("Crie um deck antes de adicionar um
flashcard!");
                   return;
               }
               FlashCard card = FlashCardFactory.criarFlashcard(pergunta,
resposta);
               deck.addFlashcard(card);
               sistemaNotificacao.notificar("Nova carta adicionada ao
deck!");
           }

           public void visualizarDeck() {
               if (deck == null) {
                   System.out.println("Deck não criado.");
                   return;
               }
               System.out.println(deck.getFlashCards());
           }

           public void adicionarUsuarioNotificacao(Usuario usuario) {
               sistemaNotificacao.adicionarUsuario(usuario);
           }
}
```

Padrões Comportamentais:

Utilizamos o **Observer**, cujo objetivo é permitir que múltiplos objetos sejam notificados automaticamente sempre que um evento ocorrer. No nosso caso, sempre que um **novo flashcard é adicionado**, os usuários cadastrados no sistema recebem uma notificação.

```java
public interface Observer {
    void atualizar(String mensagem);
}

class Usuario implements Observer {
    private String nome;

    public Usuario(String nome) {
        this.nome = nome;
    }

    @Override
    public void atualizar(String mensagem) {
        System.out.println(nome + " recebeu notificação: " + mensagem);
    }
}

class SistemaNotificacao {
    private List<Observer> usuarios = new ArrayList<>();

    public void adicionarUsuario(Observer usuario) {
        usuarios.add(usuario);
    }

    public void notificar(String mensagem) {
        for (Observer usuario : usuarios) {
            usuario.atualizar(mensagem);
        }
    }
}
```