# Aula 09 – Padrões de Projeto (Design Patterns)

**Padrão de Projeto:** descreve uma abordagem de sucesso, amplamente testada, para um determinado problema. Possibilitando o reúso desse padrão em outros projetos. Afinal, na maioria dos casos, não é necessário “reinventar a roda”.

Os padrões mais conhecidos foram descritos por volta da década de 90, pela chamada “Gangue dos Quatro”, ou “GoF”. Segundo registro, há 23 padrões de projeto, que são divididos em três grupos: padrões de comportamento, de criação e estruturais. Por fim, para usar esses padrões é necessário ter um nível razoável de experiência.

***Observação***: Padrão de Projeto, ou Design Patterns, é algo mais próximo do nível de código, da implementação de fato (mão na massa), diferenciado-se consideravelmente do conceito de Padrão de Arquitetura; que por ser mais abstrato, foco mais no “como” será a construção da estrutura: organização/definição de componentes, boas práticas e etc.

Correção do trecho acima: Padrões de arquitetura descrevem questões maiores como a estrutura e a organização geral do sistema, por exemplo, quais camadas de código o sistema terá, ou como o código será organizado nos servidores. Design patterns propõem soluções específicas para questões recorrentes de implementação (código).

**Elementos Básicos de um Padrão de Projeto**: São importantes para ajudar os desenvolvedores a entender o contexto em que o padrão de projeto é aplicável, bem como suas vantagens e desvantagens. Além disso, os exemplos concretos ajudam a tornar o padrão mais aplicável na prática. Embora não seja a mesma coisa, lembra-me muito a parte de análise sobre qual(quais) Padrão(es) de Arquitetura, levando em consideração os requisitos funcionais X não funcionais, se adequaria melhor a um determinado software.

Os 4 elementos básicos são: Contexto, problema, solução, consequência e padrões associados.

**Objetivos de design**: são diretrizes que ajudam os desenvolvedores a

criar sistemas de software de alta qualidade, que sejam fáceis de manter, extensíveis, escaláveis e eficiente. Com ênfase na parte de requisitos não funcionais: escalabilidade, confiabilidade, adaptatividade, performance e etc.

**Design orientado a objetos:** é um paradigma de design de software que se baseia nos conceitos de orientação a objetos: encapsulamento, herança, polimorfismo, composição e etc. Nessa abordagem é muito comum o uso de interfaces, composição em detrimento da herança, aberto-fechado (aberto para extensão e fechado para modificações) e etc.

**Padrão de Projeto Observer**: é um padrão, comportamental, que representa uma relação de 1 para N objetos, de modo que quando ocorre uma mudança de estado, todos os objetos dependentes são notificados dessa alteração.

Dentro do Observer temos o objeto observado, também conhecido como “subject”, e os objetos observadores, “observers”. O objeto observado é responsável por notificar os observadores sobre a mudança ocorrida no objeto.

Suposição: No Youtube temos o produtor de conteúdo e seus seguidores. Quando o produtor publica um vídeo, todos os seus seguidores recebem uma notificação sobre o novo conteúdo. Nessa lógica, o canal/produtor de conteúdo seria o objeto observável? Enquanto seus seguidores, o objeto observador? Faz sentido essa analogia no contexto do padrão observer?

**Padrão de Projeto Singleton:** é um padrão, do tipo criacional, que permite a criação de apenas uma única instância de uma classe. Isso garante que o sistema não uma duplicação.

Suposição: uma aplicação que gerencia uma instância de conexão de um banco de dados usaria esse padrão? Onde é permitido apenas uma instância desse processo no sistema inteiro?

Correção do trecho acima: Isso faria sentido “por usuário”, por exemplo, o sistema permitiria apenas uma instância de conexão por usuário. Caso contrário, os outros usuários teriam que esperar o usuário conectado se desconectar.

**Padrão de Projeto Factory:** é um padrão, do tipo criacional, que fornece uma interface para criar objetos em uma superclasse. Assim, as subclasses podem facilmente realizar modificações nos objetos criados.

Correção do trecho acima: Na verdade, esse padrão especifica uma classe dedicada a criar objetos. Para isso, faz uso de interfaces e classes que implementam estas interfaces.