**Estudo de Padrões de Projeto**

**GoF**

**- Categorias**

Os padrões de projeto são divididos em três categorias:

\*Creational

Visam abstrair o processo de como os objetos são criados na aplicação.

\*Structural

Lidam com a composição de classes e objetos.

\* behavioural

Caracterizam como classes e objetos interagem e distribuem responsabilidade na aplicação.

**- SOLID – princípios do design orientados a objetos**

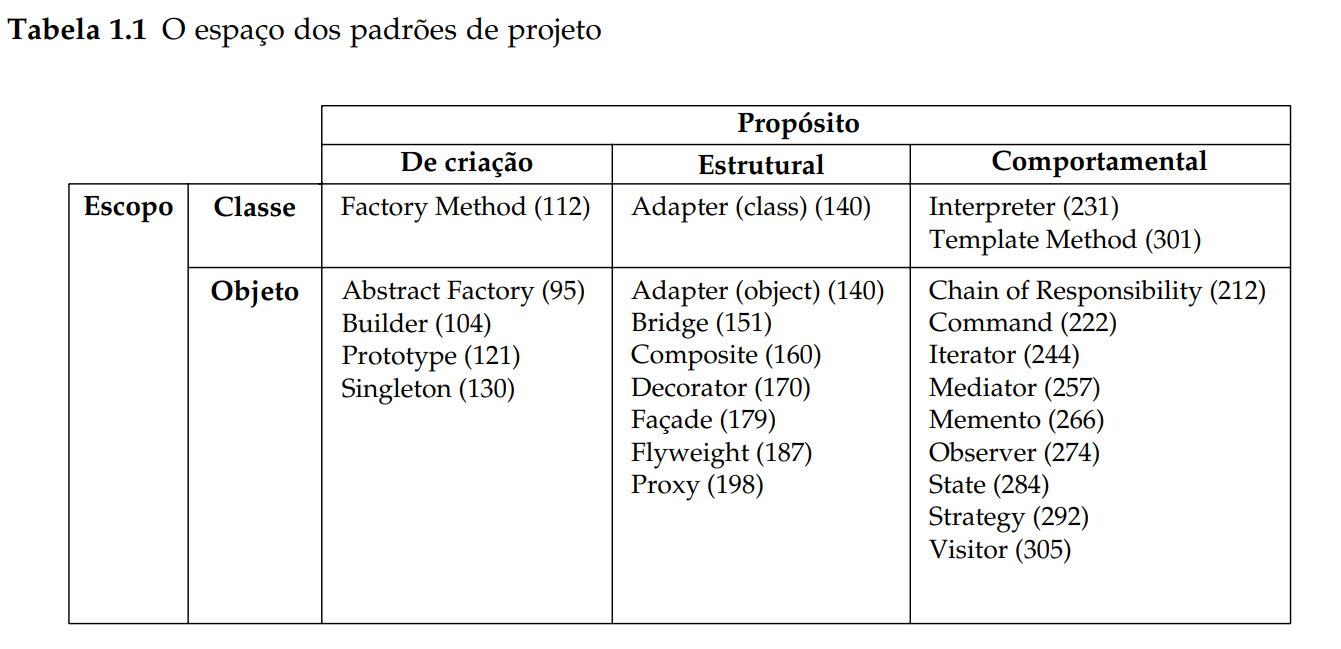
**S – single responsability princíple** (princípio da responsabilidade única) – uma classe deve ter apenas um motivo para mudar.

**O – open/close principle** (princípio do aberto/fechado) – classes, objetos e métodos devem estar abertos para extensão, mas fechado para modificações.

**L – linkov substitution principle** (princípio de substituição de Linkov) – classes derivadas devem ser capazes de substituir totalmente classes-bases.

**I – interface segregation principle** (princípio da segregação de interface) – os clientes não devem ser forçados a depender de interfaces que não utilizam.

**D – dependecy inversion principle** (princípio da inversão de dependência) – módulo de alto nível não devem ser dependentes de módulos de baixo nível, ambos devem depender de abstrações. Detalhes devem depender de abstrações e não o inverso.



**- Padrão Singleton – padrão de criação**

Possui apenas uma classe.

Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela.

Geralmente usado para acessar recursos compartilhados, como acesso a base de dados, classes de login, interface gráficas, servidores de impressão e mais.

Também usados para substituir variáveis globais, como o caso de uso de objetos de configuração de sistemas.

**- Padrão Builder – padrão de construção**

Separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.

O padrão sugere a separação do código de cria e o código que usa o objeto.

O código que usa é chamado de código cliente. Visa separar a construção de um código complexo de um código cliente.

Permite criar um objeto complexo por etapas.

**- Padrão Prototype - padrão de construção**

Especifica os tipos de objetos a serem criados usando uma instância prototípica e criar novos objetos copiando esse protótipo.

Protótipos são objetos diretamente ligados a outros objetos.

Você faz um objeto herdar de outro.

O tipo do objeto criada é determinada pelo protótipo.

Geralmente é criado por apenas um método.

Evita que o cliente conheça as classes que criam o objeto.

É usado para evitar recriar objetos caros computacionalmente.

**- Factory Method – método de criação**

Define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. O Factory Method permite a uma classe postergar (defer) a instanciação às subclasses.

Factory são geralmente ações que criam objetos.

Oculta a lógica de instanciação do código cliente. O método fábrica será responsável por instanciar as classes desejadas.

É obtido através de herança.

**- Abstract Factory – método de criação**

Fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

É composto por vários Factory Method.

**- Composite - método estrutural**

Compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. O Composite permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.

Usado em estruturas que possam ter um tratamento hierárquico, objeto dentro um objeto, como árvores.

Pode-se tratar a estrutura inteira de maneira uniforme.

Prioriza a composição ao invés de herança.