**Fundação Universidade Regional de Blumenau**

**Disciplina:** Análise de Algoritmo

**Professor:** Jose Carlos Altholf

**Aluno:** Daniel Krüger, Lucas Fritzke e Luiz Henrique Martendal

Perguntas referente padrões de projeto

1. **Quais as categorias dos padrões de projeto GoF?**

*Resposta:* Os três principais padrões de projeto definidos pelo livro “Design Patterns: Elements of Reusable Object-Oriented” são os padrões criacionais, estruturais e comportamentais. Esses padrões foram divididos e agrupados de acordo com a natureza do problema que eles solucionam.

1. **Quais os tipos de padrões de projeto GoF?**

*Resposta:*

**Padrões criacionais**: estes padrões oferecem diversas alternativas de criação de objetos, o que aumenta a flexibilidade e a reutilização de código.

**Padrões estruturais:** Nos mostram como montar objetos e classes em estruturas maiores, sem perder a eficiência e flexibilidade.

**Padrões comportamentais:** Nos ajudam a trabalhar melhor com os algoritmos e com a delegação de responsabilidades entre os objetos.

1. **Descreva em um parágrafo a funcionalidade de cada padrão de projeto GoF**

**Abstract Factory:** Fornecer uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

**Adapter:** Converte a interface de uma classe em outra interface que os clientes esperam. Adapter permite que as classes trabalhem juntas que, de outra forma, não poderiam por causa de interfaces incompatíveis.

**Bridge**: Desacople uma abstração de sua implementação para que as duas possam variar independentemente.

**Builder:** Separe a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar diferentes representações.

**Chain of Responsibility:** Evite acoplar o remetente de uma solicitação ao seu receptor, dando a mais de um objeto a chance de lidar com a solicitação. Encadeie os objetos receptores e passe a solicitação ao longo da cadeia até que um objeto a trate.

**Command:** Encapsule uma solicitação como um objeto, permitindo assim que você parametrize clientes com diferentes solicitações, enfileire ou registre solicitações e suporte operações reversíveis.

**Composite:** Componha objetos em estruturas de árvore para representar hierarquias de parte-todo. Composite permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.

**Decorator:** Anexe responsabilidades adicionais a um objeto dinamicamente. Os Decorators fornecem uma alternativa flexível à herança para estender a funcionalidade.

**Facade:** Fornece uma interface unificada para um conjunto de interfaces em um subsistema. Facade define uma interface de nível superior que torna o subsistema mais fácil de usar.

**Factory Method:** Define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe instanciar. Factory Method permite que uma classe adie a instanciação para subclasses.

**Flyweight:** Use o compartilhamento para suportar um grande número de objetos granulares de maneira eficiente.

**Interpreter:** Dado um idioma, defina uma representação para sua gramática junto com um interpretador que usa a representação para interpretar sentenças no idioma.

**Iterator:** Forneça uma maneira de acessar os elementos de um objeto agregado sequencialmente sem expor sua representação subjacente.

**Mediator:** Defina um objeto que encapsula como um conjunto de objetos interage. O mediador promove o acoplamento solto, evitando que os objetos se refiram uns aos outros explicitamente, e permite que você varie sua interação independentemente.

**Memento:** Sem violar o encapsulamento, capture e externalize o estado interno de um objeto para que o objeto possa ser restaurado para este estado posteriormente.

**Observer:** Defina uma dependência de um para muitos entre objetos para que, quando um objeto muda de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.

**Prototype:** Especifique os tipos de objetos a serem criados usando uma instância prototípica e crie novos objetos copiando este protótipo.

**Proxy:** Forneça um substituto ou espaço reservado para outro objeto para controlar o acesso a ele.

**Singleton:** Garanta que uma classe tenha apenas uma instância e forneça um ponto global de acesso a ela.

**State:** Permita que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá mudar sua classe.

**Strategy:** Defina uma família de algoritmos, encapsule cada um e torne-os intercambiáveis. A estratégia permite que o algoritmo varie independentemente dos clientes que o utilizam.

**Template Method:** Defina o esqueleto de um algoritmo em uma operação, adiando algumas etapas para subclasses. O método de template permite que as subclasses redefinam certas etapas de um algoritmo sem alterar a estrutura do algoritmo.

**Visitor:** Represente uma operação a ser realizada nos elementos de uma estrutura de objeto. O visitante permite que você defina uma nova operação sem alterar as classes dos elementos nos quais opera.

1. **Qual padrão de projeto tem o propósito de assegurar o controle da quantidade de instâncias da classe?**

a) Singleton b) Façade c) Proxy d) MVC e) Command

1. **Padrões de projeto são técnicas adotadas com o objetivo de estruturar a implementação de um sistema seguindo princípios preestabelecidos. Na**

**implementação de um sistema fictício, foi detectado que o custo de criação de quaisquer objetos é muito alto. Dessa forma, cada objeto somente deve ser**

**instanciado no momento em que ele é realmente necessário. Nesse caso, qual deve ser o padrão de projeto utilizado?**

a) Façade b) Mock Object c) Singleton d) Lazy Loading e) Method Chaining

1. **Sobre os padrões de projeto, é correto afirmar:**

a) O uso do padrão de projeto Command em uma aplicação possibilita que os objetos envolvidos possam ser acessados sequencialmente de uma forma padronizada.

b) O padrão de projeto Facade busca fornecer uma interface de acesso simplificada para um subestima complexo, de forma a tornar o uso de tal subsistema mais simples.

c) O padrão de projeto Template Method é considerado um padrão de criação, onde é disponibilizada uma interface para a criação de uma família de objetos relacionados.

d) O padrão de projeto Abstract Factory é considerado comportamental, pois busca oferecer recursos para dinamicamente definir o comportamento dos objetos envolvidos.

e) O padrão de projeto Observer define a forma como a aplicação pode garantir a

existência de apenas uma instância de um objeto, sendo que tals instância será

compartilhada entre os demais objetos.

1. **De acordo com Gamma, padrões de projeto são soluções reutilizáveis de software orientado a objetos. Considere as três afirmativas a seguir e depois informe a alternativa correta.**

**I - Padrões de projeto de criação são aqueles que abstraem o processo de**

**instanciação de objetos.**

**II - Padrões de projeto estruturais se preocupam com a forma como classes e**

**objetos são compostos para formar estruturas maiores.**

**III - Padrões de projeto comportamentais se preocupam com algoritmos e a**

**atribuição de responsabilidades entre objetos.**

a) Apenas a afirmativa I está correta.

b) Apenas a afirmativa I e II estão corretas.

c) Apenas as afirmativa II e III estão corretas.

d) Apenas as afirmativa I e III estão corretas.

e) Todas as três afirmativas estão corretas.

1. **Um bacharel de TI está desenvolvendo uma aplicação onde existe um objeto X**

**que contém informações importantes para outros objetos, de forma de que**

**quando o valor do objeto X mudar, os outros objetos deverão ser notificados**

**dessa alteração. Além disso, ele precisa especificar uma sequência de etapas**

**em uma classe, onde algumas dessas etapas deverão ser delegadas para as**

**subclasses.**

**Quais os dois padrões de projetos que podem ser empregados nesse cenário?**

a) Adapter e Iterator

b) Command e Visitor

c) Builder e Composite

d) Observer e Template Method

e) Factory Method e Abstract Factory