1. O que são padrões de projeto?
   1. URL de apoio: <https://sourcemaking.com/design_patterns>
   2. Na engenharia de software, um padrão de design é uma solução repetível geral para um problema comum no design de software. Um padrão de design não é um design final que pode ser transformado diretamente em código. É uma descrição ou modelo de como resolver um problema que pode ser usado em muitas situações diferentes.
2. Para que servem os padrões de projeto?
   1. Para resolver problemas de arquitetura/organização que são recorrentes e comuns no dia a dia do desenvolvedor.
3. Padrões de projeto são diferentes para linguagens diferentes?
   1. Padrões de Projeto devem ser independentes de linguagem, porém sua implementação pode variar um pouco pra atender uma especificidade de uma linguagem.
4. Quem criou o termo?
   1. URL de apoio: <https://en.wikipedia.org/wiki/Software_design_pattern>
   2. Christopher Alexander
5. 5. Quais categorias existem?
   1. Criacional, Comportamental, Estrutural
6. 6. Quantos e quais são os padrões por categoria?
   1. Criacional:
      1. Singleton
      2. Abstract Factory
      3. Builder
      4. Prototype
      5. Factory Mathod
   2. Estruturais:
      1. Decorator
      2. Proxy
      3. FlyWeigth
      4. Facade
      5. Composite
      6. Bridge
      7. Adapter
   3. Comportamental
      1. Template Method
      2. Visitor
      3. Command
      4. Strategy
      5. Chair of Responsability
      6. Iterator
      7. Mediator
      8. Memento
      9. Interpreter
      10. State
      11. Observer

7. Faça uma tabela explicando a utilidade de cada um dos padrões GoF

|  |  |
| --- | --- |
| Nome | Utilidade |
| Singleton | assegura que somente um objeto de uma determinada classe seja criado em todo o projeto; |
| Abstract Factory | permite que um cliente crie famílias de objetos sem especificar suas classes concretas; |
| Builder | encapsular a construção de um produto e permitir que ele seja construído em etapas; |
| Prototype | permite você criar novas instancias simplesmente copiando instancias existentes; |
| Factory Mathod | as subclasses decidem quais classes concretas serão criadas. |
| Decorator | envelopa um objeto para fornecer novos comportamentos; |
| Proxy | envelopa um objeto para controlar o acesso a ele; |
| FlyWeigth | uma instancia de uma classe pode ser usada para fornecer muitas “instancias virtuais”; |
| Facade | simplifica a interface de um conjunto de classes; |
| Composite | Os clientes tratam as coleções de objetos e os objetos individuais de maneira uniforme; |
| Bridge | permite criar uma ponte para variar não apenas a sua implementação, como também as suas abstrações; |
| Adapter | envelopa um objeto e fornece a ele uma interface diferente; |
| Template Method | As subclasses decidem como implementar os passos de um algoritimo; |
| Visitor | permite acrescentar novos recursos a um composto de objetos e o encapsulamento não é importante; |
| Command | encapsula uma solicitação como um objeto; |
| Strategy | encapsula comportamentos intercambiáveis e usa a delegação para decidir qual deles será usado; |
| Chair of Responsability | permite dar a mais de um objeto a oportunidade de processar uma solicitação; |
| Iterator | fornece uma maneira de acessar seqüencialmente uma coleção de objetos sem expor a sua implementação; |
| Mediator | centraliza operações complexas de comunicação e controle entre objetos relacionados; |
| Memento | permite restaurar um objeto a um dos seus estados prévios, por exemplo, quando o usuário seleciona um “desfazer”; |
| Interpreter | permite construir um intérprete para uma linguagem; |
| State | encapsula comportamentos baseados em estados e usa a delegação para alternar comportamentos; |
| Observer | permite notificar outros objetos quando ocorre uma mudança de estado. |

8. Idem para GRASP

|  |  |
| --- | --- |
| Nome | Utilidade |
| Information Expert | Determina quando devemos delegar a responsabilidade para um outro objeto que seja especialista naquele domínio. |
| Creator | Determina qual classe deve ser responsável pela criação certos objetos. |
| Controller | Atribui a responsabilidade de lidar com os eventos do sistema para uma classe que representa a um cenário de caso de uso do sistema global. |
| Low Coupling | Determina que as classes não devem depender de objetos concretos e sim de abstrações, para permitir que haja mudanças sem impacto. |
| High Cohesion | este princípio determina que as classes devem se focar apenas na sua responsabilidade. |
| Polymorphism | As responsabilidades devem ser atribuídas a abstrações e não a objetos concretos, permitindo que eles possam variar conforme a necessidade. |
| Pure Fabrication | é uma classe que não representa nenhum conceito no domínio do problema, ela apenas funciona como uma classe prestadora de serviços, e é projetada para que possamos ter um baixo acoplamento e alta coesão no sistema. |
| Protected Variations | Protege o sistema com a variação de componentes, encapsulando o comportamento que realmente importa.. |

1. 9. Existem outros padrões além destes? Dê exemplos.
   1. Sim, o padrão delegate por exemplo.
2. 10. Você já pensou em criar uma biblioteca de padrões seus? Mostre um exemplo em caso positivo.
   1. Não.