Fatec Araras

Tópicos Especiais em Sistemas para Internet III

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Agenda

Fatec Araras

- POO
  - Fundamentos
  - Princípio SOLID
- Padrões de projetos
  - Tipos / classificação
  - Padrão Criacional
    - Singleton

POO

Programação Orientada a Objetos

Fatec Araras

Fatec Araras

Abstração

Na maioria das vezes quando você está criando um programa com a POO, você molda os objetos do programa baseado em objetos do mundo real.

Encapsulamento

Para começar um motor de carro, você precisa apenas girar a chave ou apertar um botão. Você não precisa conectar os fios debaixo do capô, rotacionar o eixo das manivelas e cilindros, e iniciar o ciclo de força do motor.

Fatec Araras

Herança

A Herança é a habilidade de construir novas classes em cima de classes já existentes. O maior benefício da herança é a reutilização de código. Se você quer criar uma classe que é apenas um pouco diferente de uma já existente, não há necessidade de duplicar o código.

Polimorfismo

O Polimorfismo é a habilidade de um programa detectar a classe real de um objeto e chamar sua implementação mesmo quando seu tipo real é desconhecido no contexto atual.

Princípio SOLID

Princípio SOLID

Fatec Araras

Robert Martin os introduziu no livro *Agile Software Development, Principles, Patterns, and Practices*.

O SOLID é uma sigla mnemônica em inglês para cinco princípios de projeto destinados a fazer dos projetos de software algo mais compreensivo, flexível, e sustentável.

Como tudo na vida, usar estes princípios sem cuidado pode causar mais males que bem. O custo de aplicar estes princípios na arquitetura de um programa pode ser torná-lo mais complicado que deveria ser.

Single Responsibility Principle

Fatec Araras

Tente fazer com que cada classe seja responsável por uma única parte da funcionalidade fornecida pelo software, e faça aquela responsabilidade ser inteiramente encapsulada pela classe.

O objetivo principal deste princípio é reduzir a complexidade. Você não precisa inventar um projeto sofisticado para um programa que tem apenas 200 linhas de código. Faça uma dúzia de métodos bonitos, e você ficará bem.

Open / Close Principle

Fatec Araras

As classes devem ser abertas para extensão mas fechadas para modificação.

Uma classe é aberta se você pode estendê-la, produzir uma sub-classe e fazer o que quiser com ela (adicionar novos métodos ou atributos, sobrescrever o comportamento base, etc.)

Algumas linguagens de programação permitem que você restrinja a futura extensão de uma classe com palavras chave como *final*.

Liskov Substitution Principle

Fatec Araras

Quando estendendo uma classe, lembre-se que você deve ser capaz de passar objetos da subclasse em lugar de objetos da classe mãe sem quebrar o código cliente.

Isso significa que a subclasse deve permanecer compatível com o comportamento da superclasse. Quando sobrescrevendo um método, estenda o comportamento base ao invés de substituí-lo com algo completamente diferente.

Interface Segregation Principle

Fatec Araras

Clientes não devem ser forçados a depender de métodos que não usam.

Tente fazer com que suas interfaces sejam reduzidas o suficiente para que as classes cliente não tenham que implementar comportamentos que não precisam.

Dependency Inversion Principle

Fatec Araras

Classes de alto nível não deveriam depender de classes de baixo nível. Ambas devem depender de abstrações. As abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.

O princípio de inversão de dependência quase sempre anda junto com o princípio aberto/fechado: você pode estender classes de baixo nível para usar diferentes classes de lógica do negócio sem quebrar classes existentes.

Dependency Inversion Principle

Fatec Araras

Geralmente quando fazendo projeto de software, você pode fazer uma distinção entre dois níveis de classes.

- Classes de baixo nível implementam operações básicas tais como trabalhar com um disco, transferindo dados pela rede, conectar-se a uma base de dados, etc.

- Classes de alto nível contém lógica de negócio complexa que direcionam classes de baixo nível para fazerem algo.

Padrões de Projeto

Padrões de Projeto

Fatec Araras

Padrões de projeto são soluções típicas para problemas comuns em projeto de software. Eles são como plantas de obra pré fabricadas que você pode customizar para resolver um problema de projeto recorrente em seu código.

O padrão não é um pedaço de código específico, mas um conceito geral para resolver um problema em particular. Você pode seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do seu próprio programa.

Padrões de Projeto

Fatec Araras

Padrões de projeto são soluções típicas para problemas comuns em projeto de software. Eles são como plantas de obra pré fabricadas que você pode customizar para resolver um problema de projeto recorrente em seu código.

O padrão não é um pedaço de código específico, mas um **conceito geral para resolver um problema em particular**. Você pode seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do seu próprio programa.

Como é um padrão ?

Fatec Araras

- O **Propósito** do padrão descreve brevemente o problema e a solução.

- A **Motivação** explica a fundo o problema e a solução que o padrão torna possível.

- As **Estruturas** de classes mostram cada parte do padrão e como se relacionam.

Classificação

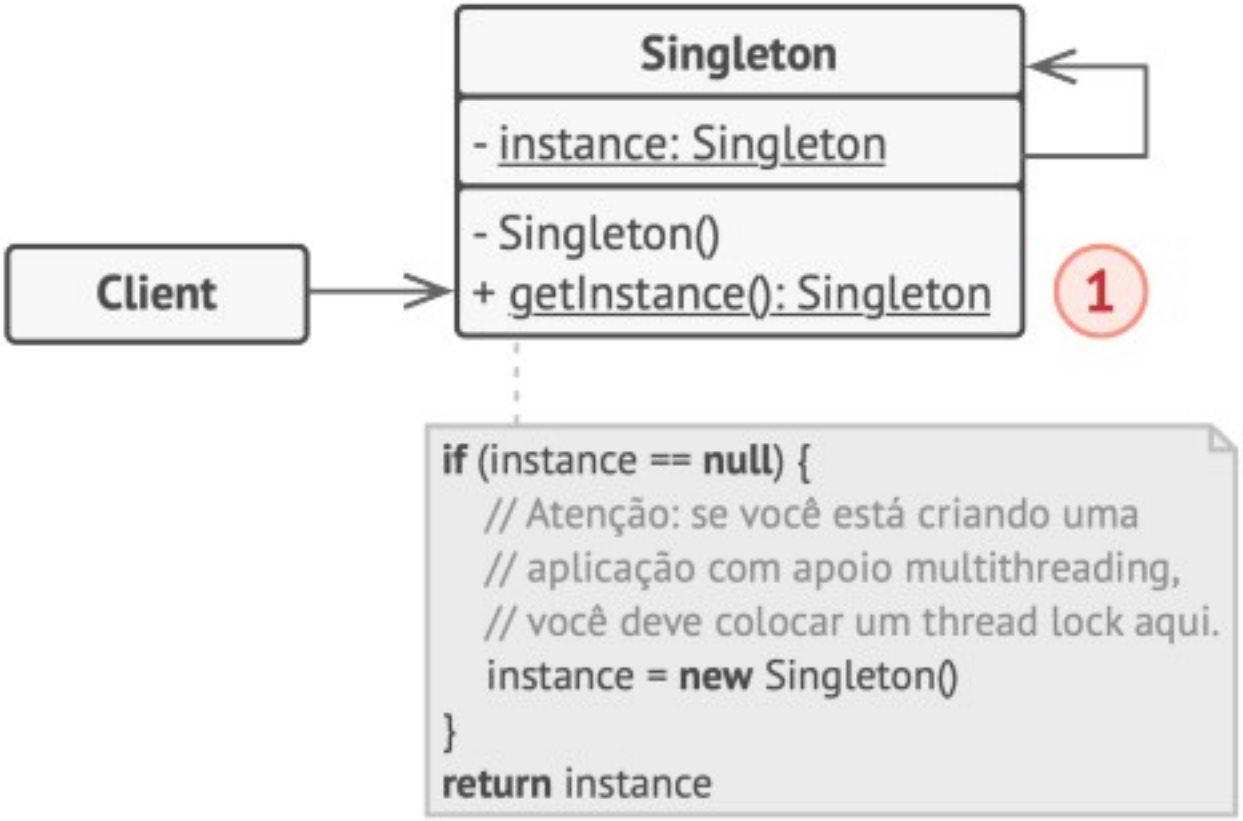
- Os **padrões criacionais** fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização de código.
- Os **padrões estruturais** explicam como montar objetos e classes em estruturas maiores, enquanto ainda mantém as estruturas flexíveis e eficientes.
- Os **padrões comportamentais** cuidam da comunicação eficiente e da assinalação de responsabilidades entre objetos.

Singleton

Singleton

O *Singleton* é um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.

Singleton


Singleton
- instance: Singleton
- Singleton()
+ getInstance(): Singleton
Client
1
if (instance == null) {
// Atenção: se você está criando uma
// aplicação com apoio multithreading,
// você deve colocar um thread lock aqui.
instance = new Singleton()
}
return instance

Singleton

Fatec Araras

O padrão Singleton resolve dois problemas de uma só vez, violando o princípio de responsabilidade única:

- **Garantir que uma classe tenha apenas uma única instância.** Por que alguém iria querer controlar quantas instâncias uma classe tem? A razão mais comum para isso é para controlar o acesso a algum recurso compartilhado—por exemplo, uma base de dados ou um arquivo.

Singleton

Fatec Araras

Fornece um ponto de acesso global para aquela instância. Se lembra daquelas variáveis globais que você (tá bom, eu) usou para guardar alguns objetos essenciais? Embora sejam muito úteis, elas também são muito inseguras uma vez que qualquer código pode potencialmente sobrescrever os conteúdos daquelas variáveis e quebrar a aplicação.

Aplicabilidade

- Utilize o padrão Singleton quando uma classe em seu programa deve ter apenas uma instância disponível para todos seus clientes; por exemplo, um objeto de base de dados único compartilhado por diferentes partes do programa.
- Utilize o padrão Singleton quando você precisa de um controle mais estrito sobre as variáveis globais.

Prós / Contra

Fatec Araras

- Você pode ter certeza que uma classe só terá uma única instância.
- Você ganha um ponto de acesso global para aquela instância.
- O objeto singleton é inicializado somente quando for pedido pela primeira vez.
- Viola o princípio de responsabilidade única. O padrão resolve dois problemas de uma só vez.

- O padrão Singleton pode mascarar um design ruim, por exemplo, quando os componentes do programa sabem muito sobre cada um.

Prós / Contra

Fatec Araras

- O padrão requer tratamento especial em um ambiente multithreaded para que múltiplas threads não possam criar um objeto singleton várias vezes.
- Pode ser difícil realizar testes unitários do código cliente doSingleton porque muitos frameworks de teste dependem de herança quando produzem objetos simulados. Já que o construtor da classe singleton é privado e sobrescrever métodos estáticos é impossível na maioria das linguagem, você terá que pensar em uma maneira criativa de simular o singleton.

Fatec
Araras

Dúvidas

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Desafio

Pesquisar na internet sobre o clássico livro sobre padrões de projetos GoF.