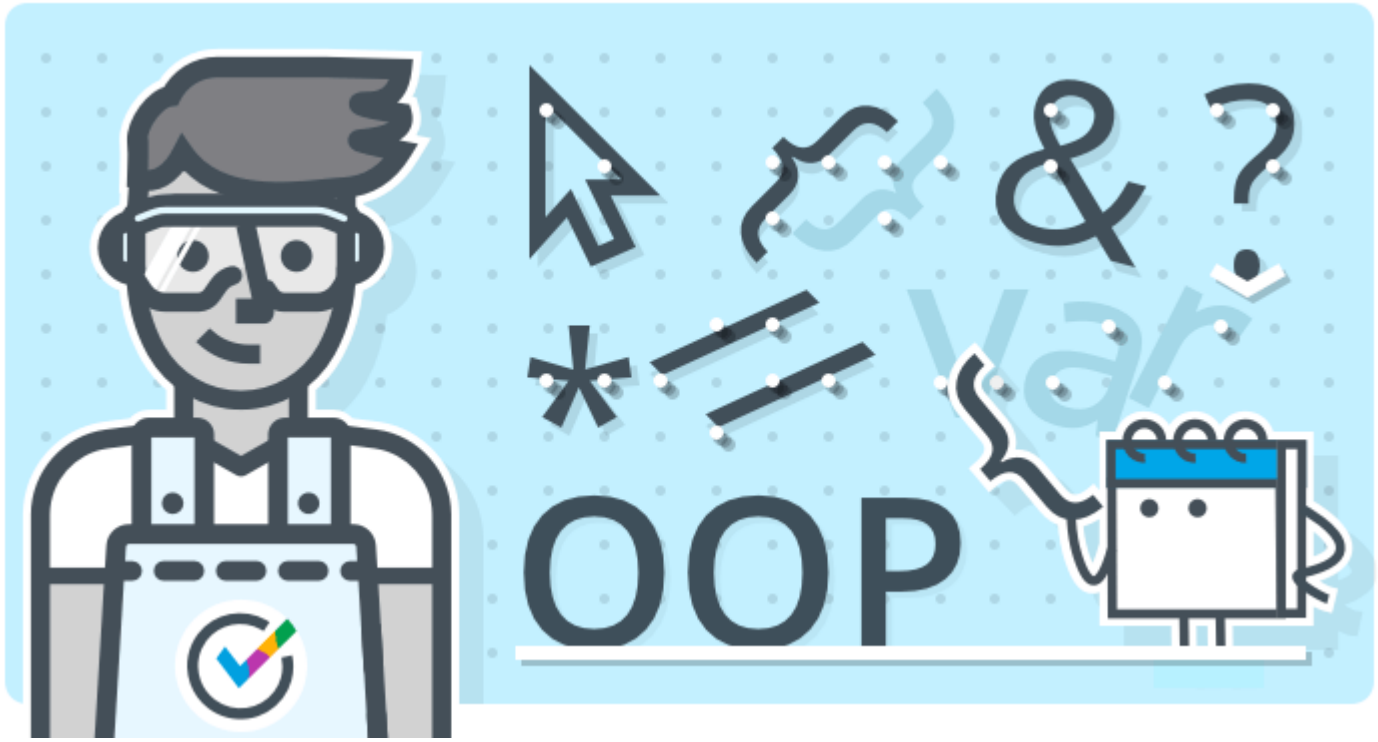
# Geek University

## Evolua seu lado geek!

www.geekuniversity.com.br

# Princípios do Design de Software Orientado a Objetos



# Princípios do Design de Software Orientado a Objetos

Quando entrarmos realmente na parte de Padrões de Projeto iremos utilizar os princípios do design de software orientado a objetos como uma caixa de ferramentas para nos auxiliar na criação de softwares bons e confiáveis.

# Princípios do Design de Software Orientado a Objetos

**Princípio do aberto/fechado**

# Princípios do Design de Software Orientado a Objetos

**Princípio do aberto/fechado**

Este princípio determina que classes ou métodos devem estar abertos para extensão mas fechados para modificações.

Isso significa que ao desenvolver nossa aplicação deveremos escrever nossas classes e métodos de forma genérica, de modo que sempre que sentirmos necessidade de estender o comportamento não precisaremos alterar a classe propriamente dita.

Ao invés disso uma extensão simples da classe deverá ajudar a implementar o novo comportamento.

```python
from abc import ABC, abstractmethod
from uuid import uuid4


class Pessoa(ABC):

    def __init__(self, nome):
        self.__nome = nome

    @abstractmethod
    def ganhar_dinheiro(self):
        pass
```

```python
class Aluno(Pessoa):

    def __init__(self, nome):
        super().__init__(nome)
        self.__matricula = str(uuid4()).split('-')[0].upper()

    def ganhar_dinheiro(self):
        print('Como ganhar dinheiro? ')



aluno1 = Aluno('Angelina')
print(aluno1.__dict__)
```

# Princípios do Design de Software Orientado a Objetos

**Princípio da inversão de controle**

# Princípios do Design de Software Orientado a Objetos

**Princípio da inversão de controle**

Este princípio determina que módulos de alto nível não devem ser dependentes de módulos de baixo nível pois ambos devem ser dependentes de abstrações. Os detalhes devem depender das abstrações e não o inverso.

Ou seja, quaisquer dois módulos não devem ser altamente dependentes um do outro. Eles devem estar desacoplados com uma camanda de abstração entre eles.

De forma simples, devemos evitar a dependência entre módulos, ou seja, eles devem ser independentes.

Desta forma a complexidade/rigidez do sistema diminui e fica mais fácil lidar com dependências entre os módulos.

# Princípios do Design de Software Orientado a Objetos

**Princípio da segregação de interfaces**

# Princípios do Design de Software Orientado a Objetos

**Princípio da segregação de interfaces**

Este princípio determina que clientes não devem ser forçados a depender de interfaces que não utilizam.

Desta forma, nós desenvolvedores, estaremos escrevendo boas interfaces escrevendo apenas métodos relacionados à funcionalidade para que não exista nenhum método não relacionado à interface, de forma que a classe dependente da interface tenha que implementá-lo desnessariamente.

Por exemplo, uma interface Pizza não deveria ter um método chamado adiciona_frango() pois a classe PizzaVegana não deveria ser forçada a implementar este método…

As vantagens deste princício são:

- Força os desenvolvedores a escrever interfaces enxutas e a ter métodos que sejam específicos da interface;
- Ajuda você a não encher as interfaces com métodos indesejados;

# Princípios do Design de Software Orientado a Objetos

**Princípio da responsabilidade única**

# Princípios do Design de Software Orientado a Objetos

**Princípio da responsabilidade única**

Este princípio determina que uma classe deve ter apenas um motivo para mudar.

Segundo este princípio, quando desenvolvemos uma classe, ela deve cuidar bem de sua funcionalidade em particular.

Se uma classe estiver tratando de duas funcionalidades, será melhor dividí-la.

Este princípio refere-se à funcionalidade como um motivo para mudança.

A vantagem deste princício é fazer com que nossas classes sejam objetivas e enxutas.

# Princípios do Design de Software Orientado a Objetos

**Princípio da substituição**

# Princípios do Design de Software Orientado a Objetos

**Princípio da substituição**

Este princípio determina que classes derivadas devem ser capazes de substituir totalmente as classes-pai.

De forma simples, este princípio sugere que a classe derivada deve estar o mais próximo possível da classe-pai de modo que a classe derivada possa substituir a classe-pai sem qualquer modificação no código.

```python
class Pessoa:

    def __init__(self, nome):
        self.__nome = nome

    def andar(self):
        print('Estou andando...')


class Aluno(Pessoa):

    def __init__(self, nome, matricula):
        super().__init__(nome)
        self.__matricula = matricula
```

# Geek University

## Evolua seu lado geek!

www.geekuniversity.com.br