Semana 23 - Aula 1

Tópico Principal da Aula: Pilares da Programação Orientada a Objetos: Implementação de Herança em um Projeto Prático

Subtítulo/Tema Específico: Introdução à Herança e Criação da Classe Base ProdutoEletronico

Código da aula: [SIS]ANO1C3B3S23A1

Objetivos da Aula:

- Entender como ocorre a implementação de herança em um projeto prático utilizando um exemplo de sistema de gestão de uma loja de eletrônicos.
- Compreender o papel da classe base na reutilização de código e na definição de uma hierarquia de classes.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 05 - Objetivos da Aula

- **Definição:** **O objetivo desta aula é capacitar o estudante a implementar o conceito de herança em um projeto prático, utilizando como base um sistema de gestão para uma loja de eletrônicos.**
- **Aprofundamento/Complemento (se necessário):** A herança é um dos pilares da Programação Orientada a Objetos (POO), permitindo que novas classes (subclasses) herdem atributos e métodos de classes existentes (superclasses). Isso promove a reutilização de código, reduzindo a duplicação e facilitando a manutenção e a extensão do software.
- **Exemplo Prático:** Imagine um sistema onde você precisa gerenciar diversos tipos de produtos eletrônicos, como smartphones, notebooks e televisões. Embora sejam produtos distintos, todos compartilham características comuns, como nome, marca e preço. A herança nos permite modelar essa relação de forma eficiente.

Referência do Slide: Slides 06-08 - Cenário e Ponto de Partida

- **Definição:** O cenário proposto é o desenvolvimento de um sistema para uma loja de eletrônicos. A partir desse contexto, surgem questionamentos sobre como estruturar as classes, promover a reutilização de código e estabelecer relações hierárquicas entre elas.
- **Aprofundamento/Complemento (se necessário):** No desenvolvimento de sistemas complexos, é comum identificar entidades que possuem características e comportamentos semelhantes, mas também particularidades. A POO, através da

herança, oferece uma solução elegante para modelar essas relações, evitando a criação de código redundante para cada tipo de entidade.

- **Exemplo Prático:** Para a loja de eletrônicos, questiona-se: Deveríamos criar uma classe separada para cada tipo de produto (Smartphone, Notebook, Televisao)? Ou existe uma forma de organizar o código para que as características comuns não sejam repetidas em cada nova classe de produto? A herança é a resposta para essa questão, permitindo a criação de uma classe mais genérica que sirva de base para as mais específicas.

Referência do Slide: Slides 09-12 - Construindo o Conceito: A Classe Base `ProdutoEletronico`

- **Definição:** A primeira etapa na implementação da herança é a definição de uma classe base, `ProdutoEletronico`, que encapsula os atributos e métodos comuns a todos os produtos eletrônicos. Esta classe servirá como um molde para as classes mais específicas.
- **Aprofundamento/Complemento (se necessário):** A classe base, também conhecida como superclasse ou classe pai, define o comportamento e os dados que serão compartilhados por todas as suas subclasses. Ela atua como um contrato, garantindo que as subclasses possuam uma estrutura fundamental. O método `__init__` é o construtor da classe em Python, responsável por inicializar os atributos do objeto. O método `exibir_informacoes()` é um exemplo de método comum que pode ser implementado na classe base para exibir detalhes gerais do produto.
- **Exemplo Prático:**
- Python

```python
class ProdutoEletronico:
    def __init__(self, nome, marca, preco):
        self.nome = nome
        self.marca = marca
        self.preco = preco

    def exibir_informacoes(self):
        print(f"Nome: {self.nome}, Marca: {self.marca}, Preço: R${self.preco:.2f}")

# Exemplo de uso da classe base (apenas para demonstração)
# produto_generico = ProdutoEletronico("Eletrônico Genérico", "Marca XYZ", 500.00)
# produto_generico.exibir_informacoes()
```

Referência do Slide: Slides 13-16 - Construindo o Conceito: A Subclasse `Smartphone`

- **Definição:** A subclasse `Smartphone` herda da classe base `ProdutoEletronico`, estendendo suas funcionalidades com atributos específicos (como capacidade de armazenamento) e adaptando métodos, se necessário, para incluir informações particulares de um smartphone.
- **Aprofundamento/Complemento (se necessário):** A palavra-chave `super().__init__(...)` é fundamental ao implementar a herança. Ela garante que o construtor da classe pai (`ProdutoEletronico`) seja chamado, inicializando os atributos herdados antes que a subclasse adicione seus próprios atributos. A sobrescrita do

método exibir_informacoes() permite que a subclasse forneça uma implementação mais específica, adicionando detalhes relevantes apenas para smartphones.

- **Exemplo Prático:**
- Python

```
class Smartphone(ProdutoEletronico):
    def __init__(self, nome, marca, preco, capacidade_armazenamento):
        super().__init__(nome, marca, preco) # Chama o construtor da classe pai
        self.capacidade_armazenamento = capacidade_armazenamento

    def exibir_informacoes(self):
        super().exibir_informacoes() # Chama o método da classe pai
        print(f"Capacidade de Armazenamento: {self.capacidade_armazenamento}GB")

# Exemplo de uso da subclasse Smartphone
# meu_smartphone = Smartphone("Galaxy S23", "Samsung", 4500.00, 256)
# meu_smartphone.exibir_informacoes()
```

Semana 23 - Aula 2

Tópico Principal da Aula: Pilares da Programação Orientada a Objetos: Implementação de Herança em um Projeto Prático

Subtítulo/Tema Específico: Continuação da Implementação da Herança com Novas Subclasses

Código da aula: [SIS]ANO1C3B3S23A2

Objetivos da Aula:

- Aprofundar a compreensão da herança através da criação de subclasses adicionais no projeto prático de gestão de loja de eletrônicos.
- Perceber a flexibilidade e o poder da reutilização de código proporcionados pela herança.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slides 05-08 - Construindo o Conceito: A Subclasse Notebook

- **Definição:** Dando continuidade ao projeto, criamos a subclasse Notebook, que herda de ProdutoEletronico e adiciona características específicas de laptops, como tipo de processador e quantidade de memória RAM.
- **Aprofundamento/Complemento (se necessário):** Assim como o Smartphone, a classe Notebook demonstra como a herança permite especializar um produto

genérico em um tipo mais específico, mantendo a base comum. A adição de atributos como `processador` e `memoria_ram` diferencia um notebook de outros produtos eletrônicos.

- **Exemplo Prático:**
- Python

```
class Notebook(ProdutoEletronico):
    def __init__(self, nome, marca, preco, processador, memoria_ram):
        super().__init__(nome, marca, preco)
        self.processador = processador
        self.memoria_ram = memoria_ram

    def exibir_informacoes(self):
        super().exibir_informacoes()
        print(f"Processador: {self.processador}, Memória RAM: {self.memoria_ram}GB")

# Exemplo de uso da subclasse Notebook
# meu_notebook = Notebook("IdeaPad", "Lenovo", 3800.00, "Intel Core i5", 8)
# meu_notebook.exibir_informacoes()
```

-
-

Referência do Slide: Slides 09-12 - Construindo o Conceito: A Subclasse `Televisao`

- **Definição:** A subclasse `Televisao` é mais um exemplo da aplicação da herança, estendendo `ProdutoEletronico` com atributos como tamanho da tela e tipo de resolução, que são relevantes para televisões.
- **Aprofundamento/Complemento (se necessário):** A criação da classe `Televisao` reforça o conceito de hierarquia e especialização. Cada subclasse adiciona detalhes que a tornam única, enquanto a funcionalidade base é herdada, evitando a reescrita de código para atributos e métodos comuns a todos os produtos eletrônicos.
- **Exemplo Prático:**
- Python

```
class Televisao(ProdutoEletronico):
    def __init__(self, nome, marca, preco, tamanho_tela, resolucao):
        super().__init__(nome, marca, preco)
        self.tamanho_tela = tamanho_tela
        self.resolucao = resolucao

    def exibir_informacoes(self):
        super().exibir_informacoes()
        print(f"Tamanho da Tela: {self.tamanho_tela} polegadas, Resolução: {self.resolucao}")

# Exemplo de uso da subclasse Televisao
# minha_tv = Televisao("Smart TV QLED", "Samsung", 2999.00, 55, "4K UHD")
# minha_tv.exibir_informacoes()
```

-
-

Referência do Slide: Slides 13-14 - Reflexão sobre a Herança

- **Definição:** A atividade de criar as classes e suas subclasses nos leva a pensar de forma mais orientada a objetos, identificando objetos do mundo real e seus relacionamentos de herança.
- **Aprofundamento/Complemento (se necessário):** A herança não é apenas uma técnica de codificação, mas uma forma de modelar o mundo real em um sistema. Ela nos permite criar uma estrutura de classes que reflete as relações "é um tipo de" (e.g., um Smartphone *é um tipo de* ProdutoEletronico), facilitando a compreensão e a escalabilidade do código.
- **Exemplo Prático:** No projeto da loja de eletrônicos, podemos ver claramente que um `Smartphone` é um `ProdutoEletronico`, um `Notebook` é um `ProdutoEletronico`, e uma `Televisao` é um `ProdutoEletronico`. Essa modelagem hierárquica simplifica a adição de novos tipos de produtos no futuro.

Semana 23 - Aula 3

Tópico Principal da Aula: Pilares da Programação Orientada a Objetos: Implementação de Herança em um Projeto Prático

Subtítulo/Tema Específico: Aplicação Prática e Polimorfismo

Código da aula: [SIS]ANO1C3B3S23A3

Objetivos da Aula:

- Praticar a instanciação e utilização de objetos das subclasses criadas.
- Compreender o conceito de polimorfismo através da sobrescrita de métodos.
- Entender brevemente o conceito de composição como alternativa ou complemento à herança.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slides 05-06 - Aplicação Prática: Instanciando e Utilizando Objetos

- **Definição:** Após a definição das classes e subclasses, o próximo passo é a criação de instâncias (objetos) dessas classes para simular o uso no sistema da loja de eletrônicos.
- **Aprofundamento/Complemento (se necessário):** Instanciar um objeto significa criar uma representação concreta de uma classe. Cada objeto possui seus próprios

valores para os atributos definidos na classe e pode invocar os métodos herdados ou específicos.

- **Exemplo Prático:**
- Python

```
# Criando instâncias dos produtos
smartphone = Smartphone("iPhone 15 Pro", "Apple", 8000.00, 512)
notebook = Notebook("MacBook Air", "Apple", 7500.00, "M2", 16)
televisao = Televisao("OLED C3", "LG", 6000.00, 65, "4K UHD")

# Exibindo informações dos produtos
print("--- Informações do Smartphone ---")
smartphone.exibir_informacoes()
print("\n--- Informações do Notebook ---")
notebook.exibir_informacoes()
print("\n--- Informações da Televisão ---")
televisao.exibir_informacoes()
```

-
-

Referência do Slide: Slides 07-08 - O Polimorfismo em Ação

- **Definição:** O polimorfismo, outro pilar da POO, é demonstrado quando o método `exibir_informacoes()` se comporta de maneira diferente para cada subclasse, mesmo tendo o mesmo nome, exibindo informações específicas para cada tipo de produto.
- **Aprofundamento/Complemento (se necessário):** Polimorfismo significa "muitas formas". Na POO, refere-se à capacidade de um objeto assumir diferentes formas ou comportamentos, geralmente através da sobrescrita de métodos. Embora o Python não tenha sobrecarga de métodos por assinatura (overload) como algumas outras linguagens, ele suporta polimorfismo por sobrescrita (override) e por coerção/duck typing. Isso permite que diferentes classes com uma interface comum (mesmo nome de método) sejam tratadas de forma uniforme.
- **Exemplo Prático:** Ao chamar `produto.exibir_informacoes()` para um `Smartphone`, um `Notebook` ou uma `Televisao`, o Python executa a versão específica do método `exibir_informacoes()` definida em cada subclasse, mostrando os detalhes relevantes para aquele tipo de produto. Isso acontece automaticamente, sem a necessidade de verificar o tipo do objeto explicitamente.
- **Vídeos Adicionais:**

Referência do Slide: Slide 09 - Composição (Saiba Mais)

- **Definição:** Além da herança, a composição é uma técnica de reutilização de código onde uma classe contém objetos de outras classes, formando um relacionamento "tem um" (has-a).
- **Aprofundamento/Complemento (se necessário):** Enquanto a herança modela um relacionamento "é um tipo de" (e.g., um carro *é um tipo de* veículo), a composição modela um relacionamento "tem um" (e.g., um carro *tem um* motor). Ambos são mecanismos poderosos para reutilização de código e organização, e a escolha entre

um e outro depende da natureza do relacionamento entre as entidades. Frequentemente, a composição é preferida quando a relação não é estritamente hierárquica.

- **Exemplo Prático:** Um sistema de gestão escolar pode ter uma classe `Escola` que *tem* objetos da classe `Funcionario` e `Aluno`. A `Escola` não *é um tipo de* `Funcionario` nem `Aluno`, mas ela *contém* ou *tem* funcionários e alunos.