Semana 21 - Aula 1

Tópico Principal da Aula: Pilares da Programação Orientada a Objetos

Subtítulo/Tema Específico: Herança: Reutilização de Código e Hierarquia de Classes

Código da aula: [SIS]ANO1C3B3S21A1

Objetivos da Aula:

- Compreender a herança de classes e sua hierarquia.
- Analisar a reutilização de código através do conceito de herança na Programação Orientada a Objetos (POO).
- Estruturar sistemas utilizando herança para flexibilidade e escalabilidade.

Recursos Adicionais (Sugestão, pode ser adaptado):

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

Exposição do Conteúdo:

Referência do Slide: Slide 02 - Herança: reutilização de código e hierarquia de classes

- **Definição:** **A herança é um dos pilares fundamentais da Programação Orientada a Objetos (POO) que permite que uma classe herde atributos (características) e métodos (comportamentos) de outra classe. Esse conceito promove a reutilização de código e o estabelecimento de uma hierarquia de classes.**
- **Aprofundamento/Complemento (se necessário):** A herança é uma maneira de criar uma nova classe (subclasse ou classe derivada) com base em uma classe existente (superclasse ou classe base). Isso significa que a subclasse automaticamente possui todas as características e funcionalidades da superclasse, podendo, no entanto, adicionar suas próprias características específicas, **modificar o comportamento dos métodos existentes (sobrescrita) ou fornecer sua própria implementação de métodos específicos. É um mecanismo essencial para a modularidade e extensibilidade do código.**
- **Exemplo Prático:** Imagine um sistema de gerenciamento de veículos. Em vez de criar classes `Carro`, `Moto` e `Caminhão` do zero, cada uma com seus próprios atributos de marca, modelo, ano e métodos como `ligar()` e `desligar()`, podemos criar uma classe base `Veiculo`.
- `Veiculo` conteria esses atributos e métodos comuns. As classes `Carro`, `Moto` e `Caminhão` então herdariam de `Veiculo` e adicionariam suas características únicas, como número de portas para `Carro`, cilindrada para `Moto` ou capacidade de carga para `Caminhão`.

Referência do Slide: Slide 07 - Primeiras ideias: Sistema de gerenciamento bancário

- **Definição:** Introdução a um cenário prático para aplicar o conceito de herança: um sistema de gerenciamento bancário que lida com diferentes tipos de contas (corrente, poupança e investimento). O desafio é identificar as características específicas de cada tipo de conta e como a herança pode ser utilizada para compartilhar funcionalidades comuns.
- **Aprofundamento/Complemento (se necessário):** Antes de codificar, é crucial analisar os requisitos. Quais são os atributos comuns a todas as contas (ex: número da conta, saldo)? Quais são as operações básicas que todas as contas devem realizar (ex: depositar, sacar, consultar saldo)? E quais são as particularidades de cada tipo de conta (ex: limite de cheque especial para conta corrente, rendimento de juros para poupança, produtos de investimento para conta investimento)? A herança permite modelar essa relação "é um" (ex: "Conta Corrente É UMA Conta Bancária").
- **Exemplo Prático:** Para o sistema bancário, a classe `ContaBancaria` seria a superclasse, com atributos como `numero_conta` e `saldo`, e métodos como `depositar()`, `sacar()` e `consultar_saldo()`. As subclasses
- `ContaCorrente`, `ContaPoupanca` e `ContaInvestimento` herdariam essas características básicas.

Referência do Slide: Slides 09 e 10 - Construindo Conceito: Herança - Classe Base e Classe Derivada

- **Definição:** **No contexto da herança, a classe de onde as características são herdadas é chamada de classe base ou superclasse. A classe que recebe ou herda esses atributos e métodos é denominada classe derivada ou subclasse. A subclasse pode estender a superclasse adicionando novos atributos e métodos, e também pode modificar o comportamento de métodos existentes através de sobrescrita (method overriding).**
- **Aprofundamento/Complemento (se necessário):** **A relação entre superclasse e subclasse é de generalização para especialização. A superclasse define o comportamento geral, enquanto as subclasses fornecem implementações mais específicas ou adicionam novas funcionalidades. O uso da palavra-chave `super()` em muitas linguagens de programação (como Python) permite que a subclasse chame o construtor ou métodos da superclasse para inicializar atributos herdados ou reutilizar lógicas já implementadas.**
- **Exemplo Prático:** Voltando ao sistema bancário, `ContaBancaria` é a **superclasse**.
- `ContaCorrente` é uma **subclasse** que herda de `ContaBancaria`.
- `ContaCorrente` pode ter um atributo `limite_cheque_especial` e um método `emitir_cheque()` que são específicos dela, mas ainda utiliza os métodos `depositar()` e `sacar()` da `ContaBancaria`.

Referência do Slide: Slides 11 e 12 - Construindo o Conceito: Exemplo de Herança com `Veiculo` e `ContaBancaria`

- **Definição:** Os slides reforçam o conceito de herança através de dois exemplos claros: a hierarquia `Veiculo` -> `Carro`, `Moto`, `Caminhão` e `ContaBancaria` -> `ContaCorrente`, `ContaPoupanca`, `ContaInvestimento`. Eles ilustram como a classe base contém atributos e métodos comuns, enquanto as classes derivadas adicionam suas especificidades.

- **Aprofundamento/Complemento (se necessário): A herança não é apenas sobre reutilização de código, mas também sobre a capacidade de polimorfismo, onde objetos de diferentes classes derivadas podem ser tratados como objetos de sua classe base.** Isso permite um design de software mais flexível e manutenção simplificada, **pois as alterações nos métodos da classe base se propagam automaticamente para as subclasses (a menos que sejam sobrescritas).**
- **Exemplo Prático:** No exemplo do sistema bancário, a classe `ContaBancaria` implementa os métodos `depositar`, `sacar` e `consultar_saldo`. As subclasses como
- `ContaPoupanca` (que pode ter `calcular_juros_mensal`) e `ContaInvestimento` (com `realizar_investimento`) herdam esses métodos comuns e adicionam suas próprias funcionalidades especializadas. Isso evita que o código de depósito, saque e consulta seja reescrito para cada tipo de conta.
- **Video: O que é Herança em POO:**
- **https://youtu.be/QY0Kdg83orY?si=QTKi5CwjLaVECuy3**

O Pix Sob a Ótica da POO: Estrutura, História e Reconhecimento Internacional

De forma resumida, o Pix, sistema de pagamentos instantâneos brasileiro, pode ser analisado sob a ótica dos pilares da Programação Orientada a Objetos (POO), revelando uma arquitetura robusta e flexível. Sua história é marcada pela inovação e o seu sucesso culminou em um prestigioso prêmio internacional para o Banco Central do Brasil.

A Estrutura do Pix e a Analogia com os Pilares da POO

Embora o Pix não seja um software que "entregamos" ao cliente final, mas sim um ecossistema complexo, podemos traçar paralelos de sua estrutura com os pilares da POO:

- **Abstração:** Para o usuário final, o Pix é extremamente simples: uma chave (CPF, e-mail, telefone ou aleatória) e um valor. Toda a complexidade da liquidação financeira, comunicação entre diferentes instituições, segurança e validação de dados é completamente abstraída. O usuário interage com uma interface simples que esconde os detalhes complexos, focando no que é essencial: a transferência.
- **Encapsulamento:** Cada instituição financeira (banco, fintech) participante do Pix funciona como um "objeto" encapsulado. Ela detém seus próprios dados de clientes e implementações de segurança internas. O Banco Central, através do Diretório de Identificadores de Contas Transacionais (DICT), não precisa conhecer os detalhes internos da conta de um usuário, apenas como se comunicar de forma segura com a instituição para ordenar a transação. Os dados e as operações estão protegidos dentro de cada participante.
- **Herança:** Podemos pensar nas diferentes modalidades do Pix como uma forma de "herança". Existe uma transação "base" (a transferência P2P simples). A partir dela, "herdam-se" novas funcionalidades com características específicas, como o **Pix Saque** e o **Pix Troco**, que adicionam o comportamento de retirada de dinheiro em espécie. O **Pix Cobrança** herda da transação base e adiciona funcionalidades de vencimento e juros, similar a um boleto. Todos compartilham a mesma essência (transferência instantânea), mas possuem especializações.

- **Polimorfismo:** Este é um dos conceitos mais evidentes. Uma mesma ação, "iniciar um pagamento Pix", pode ser executada de múltiplas formas, com o sistema se comportando adequadamente para cada uma. Você pode iniciar um Pix:
    - Digitando uma chave.
    - Lendo um QR Code estático (um mesmo código para várias transações).
    - Lendo um QR Code dinâmico (um código único por transação, que pode incluir valor e dados do recebedor).
    - Usando a tecnologia NFC (Pix por aproximação).

A "mensagem" é a mesma (pagar), mas a forma como o sistema a processa (o "método" invocado) é diferente dependendo do "objeto" (chave, QR Code, NFC), demonstrando o polimorfismo.

História de Sucesso: A Trajetória do Pix

A ideia de um sistema de pagamentos instantâneos no Brasil começou a ser gestada pelo Banco Central (BC) por volta de 2016, com estudos sobre a necessidade de modernizar e democratizar o sistema financeiro nacional, reduzindo custos e a dependência de dinheiro em espécie, TED e DOC.

O projeto foi oficialmente anunciado em 2018 e desenvolvido pelo Banco Central em parceria com diversos agentes do mercado financeiro. A marca "Pix" foi lançada em fevereiro de 2020. Após um período de cadastro de chaves em outubro, o sistema entrou em operação plena para todos os brasileiros em **16 de novembro de 2020**.

A adesão foi meteórica. Em poucos meses, o Pix superou as transações de TED, DOC e boletos, tornando-se o meio de pagamento mais utilizado no Brasil. Sua simplicidade, gratuidade para pessoas físicas e disponibilidade 24/7 foram fatores cruciais para essa rápida popularização.

Prêmio Internacional: O Reconhecimento do Banco Central

O sucesso estrondoso e o design inovador do Pix renderam ao Banco Central do Brasil um reconhecimento global de grande prestígio. Em maio de 2022, o BC foi o vencedor do prêmio **"BIS Central Bank Award for Inclusion"** (Prêmio de Inclusão do Banco Central, do BIS), concedido pelo Banco de Compensações Internacionais (Bank for International Settlements - BIS), considerado o "banco central dos bancos centrais".

O prêmio destacou o papel do Pix na promoção da **inclusão financeira** no Brasil. O comitê de premiação reconheceu que o sistema permitiu a milhões de brasileiros, muitos dos quais eram desbancarizados ou utilizavam primariamente dinheiro em espécie, acesso rápido, barato e seguro a serviços financeiros digitais. O Pix foi celebrado não apenas como uma maravilha tecnológica, mas como uma poderosa ferramenta de transformação social e econômica.

Semana 21 - Aula 2

Tópico Principal da Aula: Herança: Reutilização de Código e Hierarquia de Classes

Subtítulo/Tema Específico: Reutilização de Código com Classes Utilitárias e Métodos Estáticos

Código da aula: [SIS]ANO1C3B3S21A2

Objetivos da Aula:

- Compreender a herança de classes e a reutilização do código.
- Analisar a aplicação de classes utilitárias e métodos estáticos para promover a reutilização de código.
- Desenvolver soluções para problemas de duplicação de lógica através de design modular.

Recursos Adicionais (Sugestão, pode ser adaptado):

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

Exposição do Conteúdo:

Referência do Slide: Slide 07 e 08 - Construindo Conceito: Novo requisito - Cálculo de Tarifas

- **Definição:** Apresenta um novo requisito para o sistema bancário: a necessidade de implementar um cálculo de tarifas que varia conforme o tipo de conta e critérios específicos (número de transações, saldo médio), mas que a lógica de cálculo deve ser compartilhada para evitar duplicação de código. A solução proposta é a criação de uma classe utilitária, como
- `CalculadoraTarifas`.
- **Aprofundamento/Complemento (se necessário):** A reutilização de código não se limita apenas à herança. Em casos onde a lógica não se encaixa em uma relação "é um" (como uma conta "é uma" calculadora de tarifas), mas sim em uma relação "tem um" ou "usa um", a composição ou o uso de classes utilitárias com métodos estáticos são abordagens eficazes. Métodos estáticos pertencem à classe, não a uma instância específica, tornando-os ideais para funcionalidades que não dependem do estado de um objeto, como cálculos utilitários.
- **Exemplo Prático:** Para o cálculo de tarifas, a classe `CalculadoraTarifas` pode ter métodos estáticos como `calcular_tarifa_base()`, `calcular_tarifa_transacao(numero_transacoes)`, e `calcular_tarifa_saldo(saldo)`. Essas funções podem ser chamadas diretamente pela classe
- `CalculadoraTarifas` sem precisar criar um objeto dela, e são reutilizadas por todas as subclasses de `ContaBancaria` para determinar suas tarifas específicas.

Referência do Slide: Slides 09, 10 e 11 - Construindo Conceito: Nova Programação - Implementação da `CalculadoraTarifas`

- **Definição:** Apresenta o código da classe `CalculadoraTarifas` com métodos estáticos para o cálculo de diferentes tipos de tarifas. Demonstra como essa classe utilitária é integrada à classe `ContaBancaria` para que as subclasses possam utilizá-la para calcular suas tarifas.
- **Aprofundamento/Complemento (se necessário):** **A utilização de métodos estáticos em classes utilitárias é uma prática comum para encapsular lógicas que são independentes do estado de um objeto e que precisam ser acessíveis de forma global em um determinado contexto. Isso evita a proliferação de código duplicado e torna a manutenção mais centralizada. Ao modificar a lógica de cálculo de tarifa, basta alterar a classe `CalculadoraTarifas`, e todas as outras classes que a utilizam se beneficiarão da mudança automaticamente.**
- **Exemplo Prático:** A classe `ContaBancaria` poderia ter um método `calcular_tarifa()` que chamaria os métodos estáticos da `CalculadoraTarifas` para compor o valor total da tarifa. Por exemplo, `tarifa = CalculadoraTarifas.calcular_tarifa_base() + CalculadoraTarifas.calcular_tarifa_transacao(self.numero_transacoes)`. Isso mostra como a lógica de cálculo é centralizada e reutilizada pelas subclasses como
- `ContaCorrente` e `ContaPoupanca`.

Semana 21 - Aula 3

Tópico Principal da Aula: Herança: Reutilização de Código e Hierarquia de Classes

Subtítulo/Tema Específico: Prática e Extensão do Sistema de Contas Bancárias com Transferência e Ética

Código da aula: [SIS]ANO1C3B3S21A3

Objetivos da Aula:

- Praticar os conhecimentos em herança de classes, a reutilização do código e a hierarquia de classes.
- Implementar funcionalidades adicionais em sistemas orientados a objetos, como a transferência de valores entre contas.
- Analisar e discutir desafios éticos e de segurança em sistemas bancários.

Recursos Adicionais (Sugestão, pode ser adaptado):

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

Exposição do Conteúdo:

Referência do Slide: Slide 06 - Colocando em prática: Recurso de transferência entre contas

- **Definição:** Desafio prático de estender o sistema de contas bancárias, implementando um método `transferir()` na classe `ContaBancaria`. Este método deve

permitir a movimentação de dinheiro de uma conta para outra, incluindo a verificação de saldo disponível na conta de origem.

- **Aprofundamento/Complemento (se necessário):** A implementação de uma funcionalidade de transferência em um sistema bancário envolve mais do que apenas a movimentação de valores. É crucial considerar:
    - **Atomiciade:** A operação de transferência (saque de uma conta e depósito em outra) deve ser atômica; ou seja, ou ambas as operações são bem-sucedidas ou nenhuma delas é. Isso evita inconsistências no sistema em caso de falhas.
    - **Validação:** Além do saldo, outras validações podem ser necessárias, como a existência da conta de destino.
    - Tratamento de Erros: Como o sistema reagirá se não houver saldo suficiente ou se a conta de destino for inválida?
      A prática permite consolidar os conceitos de herança e interação entre objetos.
- **Exemplo Prático:** O método `transferir` na classe `ContaBancaria` receberia a `conta_destino` e o `valor`.
- Python

```
class ContaBancaria:
    # ... (atributos e outros métodos) ...

    def transferir(self, conta_destino, valor):
        if self.saldo >= valor:
            self.sacar(valor)  # Reutiliza o método sacar da própria classe
            conta_destino.depositar(valor) # Reutiliza o método depositar da conta de destino
                                print(f"Transferência de R${valor} de {self.numero_conta} para
{conta_destino.numero_conta} realizada com sucesso.")
        else:
            print("Saldo insuficiente para realizar a transferência.")
```

- Isso demonstra a reutilização dos métodos `sacar` e `depositar` já existentes.

Referência do Slide: Slide 07 - Situação: Problemas Éticos em um Sistema Bancário

- **Definição:** Apresenta uma discussão sobre os problemas éticos que sistemas bancários podem enfrentar, focando em privacidade de dados, segurança das transações financeiras e transparência nas práticas de empréstimo e investimento.
- **Aprofundamento/Complemento (se necessário):** A segurança da informação e a ética no desenvolvimento de software são cruciais, especialmente em sistemas que lidam com dados sensíveis e dinheiro.
    - **Privacidade:** Implementar criptografia, anonimização de dados, controles de acesso rigorosos e estar em conformidade com regulamentações como a LGPD (Lei Geral de Proteção de Dados) no Brasil.

    - **Segurança:** Utilizar práticas de codificação segura, auditorias de segurança, testes de penetração, e monitoramento contínuo para prevenir fraudes e ataques cibernéticos.
    - **Transparência:** Garantir que todas as informações sobre produtos e serviços sejam claras, acessíveis e compreensíveis para os clientes, evitando termos ambíguos ou "letras miúdas". A rastreabilidade de todas as operações também é fundamental.
- **Exemplo Prático:** Ao projetar o sistema bancário, deve-se pensar em:
    - **Criptografia:** Garantir que senhas e dados sensíveis sejam armazenados de forma criptografada.
    - **Logs de Auditoria:** Registrar todas as operações financeiras e acessos para rastreabilidade e detecção de atividades suspeitas.
    - **Autorização:** Implementar sistemas de permissão para que apenas usuários autorizados possam acessar e modificar informações bancárias específicas.