Disciplina:

PYTHON ENGENHARIA DE DADOS

Professor: Nelson Júnior

PUC Minas

MOTIVAÇÕES

- O desenvolvimento de aplicações de software estão cada vez mais complexas;
- Cresceram as demandas por metodologias que pudessem abstrair e modularizar as estruturas básicas de programas; e
- A maioria das linguagens de programação suportam orientação a objetos: Haskell, Java, C++, Python, PHP, Ruby, Pascal, entre outras.

HISTORIA

- Em 1967, Kristen Nygaard e Ole-Johan Dahl, do Centro Norueguês de Computação em Oslo, desenvolveram a linguagem Simula 67 que introduzia os primeiros conceitos de orientação a objetos;
- Em 1970, Alan Kay, Dan Ingalls e Adele Goldberg, do Centro de Pesquisa da Xerox, desenvolveram a linguagem totalmente orientada a objetos;
- Em 1979–1983, Bjarne Stroustrup, no laboratório da AT & T, desenvolveu a linguagem de programação C++, uma evolução da linguagem C; e
- Maior divulgação a partir de 1986 no primeiro workshop "Object-Oriented Programming Languages, Systems and Applications".

CARACTERISTICAS

- Aumento de produtividade;
- Reúso de código;
- Redução das linhas de código programadas;
- Separação de responsabilidades;
- Componentização;
- Maior flexibilidade do sistema; e
- Facilidade na manutenção.

OBJETOS

- É a metáfora para se compreender a tecnologia orientada a objetos;
- Estamos rodeados por objetos: mesa, carro, livro, pessoa, etc; e
- Os objetos do mundo real têm duas características em comum:
  - Estado – representa as propriedades (nome, peso, altura, cor, etc.); e
  - Comportamento – representa ações (andar, falar, calcular, etc.).

Ilustações

OBJETOS

Definição

É um paradigma para o desenvolvimento de software que basea-se na utilização de componentes individuais (objetos) que colaboram para construir sistemas mais complexos.

- A colaboração entre os objetos é feita através do envio de mensagens;
- Descreve uma série de técnicas para estruturar soluções para problemas computacionais; e
- É um paradigma de programação no qual um programa é estruturado em objetos.

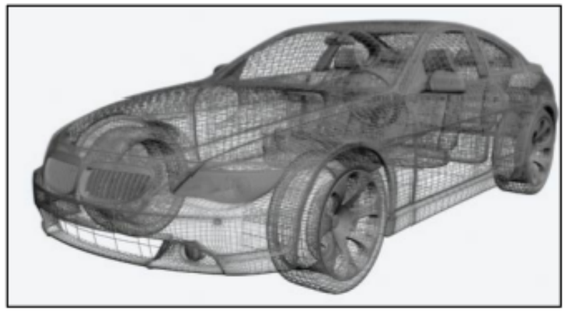
Pilares

1. Abstração;
2. Encapsulamento;
3. Herança; e
4. Polimorfismo.

Classes

- A estrutura fundamental para definir novos objetos é a classe; e
- Uma classe é definida em código-fonte.

Ilustração



Classe

Objeto

Classes Python

Estrutura

class nome_da_classe:

atributos

construtor

métodos

Exemplo

```
class Conta:
    numero = None
    saldo = None
```

Instância

- Uma instância é um objeto criado com base em uma classe definida;
- Classe é apenas uma estrutura, que especifica objetos, mas que não pode ser utilizada diretamente;
- Instância representa o objeto concretizado a partir de uma classe;
- Uma instância possui um ciclo de vida:
  - Criada;
  - Manipulada; e
  - Destruída.

Estrutura

variável = **Classe()**

Instância

Exemplo

```
conta = Conta()
conta.numero = 1
conta.saldo = 10
print(conta.numero)
print(conta.saldo)
```

EXERCÍCIOS

1)Crie uma classe automóvel com os seguintes atributos:

a)Tipo do automóvel; Cor; Marca; Modelo; Ano de fabricação; Numero de portas; Chassi; Placa.

b)Crie cinco instâncias diferentes da classe criada, digitada pela usuário.

2) Crie os seguintes métodos para a classe criada.

a)Liga, Desligar, frear; acelerar; travar portas

b)Insira uma mensagem dentro de cada metodo quando o mesmo for invocado.

c)Crie um método Imprimir recebendo por parametro Automóvel e imprimindo seus atributos.

Construtor

Exemplo

```
class Conta:
    def __init__(self,numero):
        self.numero = numero
        self.saldo = 0.0

conta = Conta(1)
print(conta.numero)
print(conta.saldo)
```

Construtor

- Determina que ações devem ser executadas quando da criação de um objeto; e
- Pode possuir ou não parâmetros.

Estrutura

```
def __init__(self,parâmetros):
```

EXERCÍCIOS

1)Crie uma classe na classe Conta um construtor que receba, agencia e conta.

a) A classe conta devera ter agencia, conta e saldo.

PUC Minas

Funções

- Representam os comportamentos de uma classe;
- Premitem que acessemos os atributos, tanto para recuperar os valores, como para alterá-los caso necessário;
- Podem retornam ou não algum valor; e
- Podem possuir ou não parâmetros.

Estrutura

```
def nome_do_método(self,parâmetros):
```

Importante

O parâmetro **self** é obrigatório.

PUC Minas

Funções

Exemplo

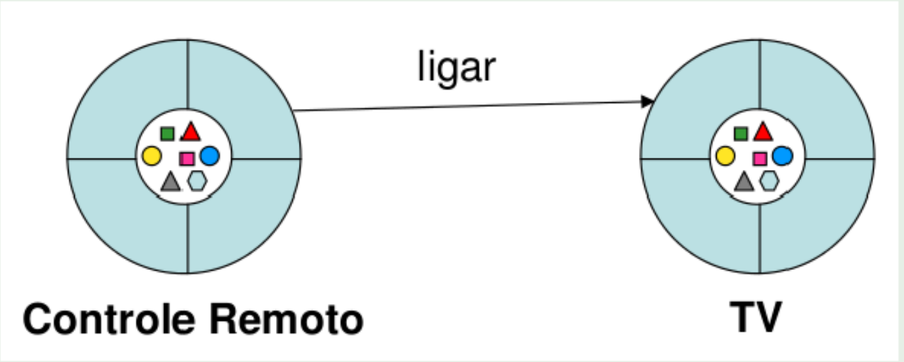
```
class Conta:
    def __init__(self,numero):
        self.numero = numero
        self.saldo = 0.0

    def consultar_saldo(self):
        return self.saldo

    def creditar(self,valor):
        self.saldo += valor

    def debitar(self,valor):
        self.saldo -= valor

    def transferir(self,conta,valor):
        self.saldo -= valor
        conta.saldo += valor

conta1 = Conta(1)
conta1.creditar(10)
conta2 = Conta(2)
conta2.creditar(5)
print(conta1.consultar_saldo())
print(conta2.consultar_saldo())
conta1.transferir(conta2,5)
print(conta1.consultar_saldo())
print(conta2.consultar_saldo())
```

Modificadores

- Em Python, existem dois tipos de modificadores de acesso para atributos e métodos:
  - Público; ou
  - Privado.
- Atributos ou métodos iniciados por dois sublinhados são privados e todas as outras formas são públicas.

- Consiste em separar os aspectos externos de um objeto dos detalhes internos de implementação;
- Evita que dados específicos de uma aplicação possa ser acessado diretamente; e
- Protege os atributos ou métodos de uma classe.

Ilustração



Exemplo

```
class Conta:
    def __init__(self, numero):
        self.__numero = numero
        self.__saldo = 0.0

    def consultar_saldo(self):
        return self.__saldo

    def creditar(self, valor):
        self.__saldo += valor

    def debitar(self, valor):
        self.__saldo -= valor

    def transferir(self, conta, valor):
        self.__saldo -= valor
        conta.__saldo += valor

conta = Conta(1)
conta.creditar(100)
conta.__saldo = 200.0 #Não é possível alterar o saldo da conta

print(conta.consultar_saldo())
```

EXERCÍCIOS

PUC Minas

1) **Crie uma classe avião com os seguintes atributos e encapsule-os.**

a) Altitude, velocidade e preencha-los com informações solicitadas pelo usuário.

b) Crie os métodos: obter velocidade, aumentar velocidade obter altitude

Herança

- É uma forma de abstração utilizada na orientação a objetos;
- Pode ser vista como um nível de abstração acima da encontrada entre classes e objetos;
- Na herança, classes semelhantes são agrupadas em hierarquias;
- Cada nível de uma hierarquia pode ser visto como um nível de abstração;
- Cada classe em um nível da hierarquia herda as características das classes nos níveis acima;
- É uma forma simples de promover reuso através de uma generalização;
- Facilita o compartilhamento de comportamento comum entre um conjunto de classes semelhantes; e
- As diferenças ou variações de uma classe em particular podem ser organizadas de forma mais clara.

Estrutura

class nome_da_classe(classe_pai_1, classe_pai_2, classe_pai_n):
    atributos
    métodos

Exemplo

```
class Poupanca(Conta):
    def __init__(self,numero):
        super().__init__(numero)
        self.__rendimento = 0.0

    def consultar_rendimento(self):
        return self.__rendimento

    def gerar_rendimento(self,taxa):
        self.__rendimento += super().consultar_saldo() * taxa / 100

conta = Poupanca(1)
conta.creditar(200.0)
conta.gerar_rendimento(10)
print(conta.consultar_saldo())
print(conta.consultar_rendimento())
```

Herança

- É originário do grego e significa “muitas formas” (poli = muitas, morphos = formas);
- Indica a capacidade de abstrair várias implementações diferentes em uma única interface;
- É o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos; e
- Quando polimorfismo está sendo utilizado, o comportamento que será adotado por um método só será definido durante a execução.

Exemplo

```
class Poupanca(Conta):
    def __init__(self,numero):
        super().__init__(numero)
        self.__rendimento = 0.0

    def consultar_rendimento(self):
        return self.__rendimento

    def gerar_rendimento(self,taxa):
        self.__rendimento += super().consultar_saldo() * taxa / 100

    def consultar_saldo(self):
        return super().consultar_saldo() + self.__rendimento

conta = Poupanca(1)
conta.creditar(200.0)
conta.gerar_rendimento(5)
print(conta.consultar_saldo())
```

Polimorfismo

- É originário do grego e significa “muitas formas” (poli = muitas, morphos = formas);
- Indica a capacidade de abstrair várias implementações diferentes em uma única interface;
- É o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos; e
- Quando polimorfismo está sendo utilizado, o comportamento que será adotado por um método só será definido durante a execução.

Exemplo

```
class Poupanca(Conta):
    def __init__(self,numero):
        super().__init__(numero)
        self.__rendimento = 0.0

    def consultar_rendimento(self):
        return self.__rendimento

    def gerar_rendimento(self,taxa):
        self.__rendimento += super().consultar_saldo() * taxa / 100

    def consultar_saldo(self):
        return super().consultar_saldo() + self.__rendimento

conta = Poupanca(1)
conta.creditar(200.0)
conta.gerar_rendimento(5)
print(conta.consultar_saldo())
```