Disciplina: IPOO - Introdução à Programação Orientada a Objetos

Prof. Luiz Cláudio Dalmolin - luiz.dalmolin@udesc.br

Orientação a Objetos com Python:

Encapsulamento (+ get e set)

In [10]:
```
# Slides - parte 10
# Exemplo 1
class ClasseFracamentePrivada():
    _atributoFracamentePrivado = None
    def __init__(self):
        self._atributoFracamentePrivado = 100
    def _MetodoFracamentePrivado(self):
        print("método fracamente privado")
##################################################
p1 = ClasseFracamentePrivada()
p1._MetodoFracamentePrivado()
print(p1._atributoFracamentePrivado)
```

```
método fracamente privado
100
```

In [7]:
```
# Exemplo 2
class ClasseFortementePrivada():
    __atributoFortementePrivado = None
    def __init__(self):
        self.__atributoFortementePrivado = 100
    def __MetodoFortementePrivado(self):
        print("método fortemente privado")
##################################################
p1 = ClasseFortementePrivada()
#p1.__MetodoFortementePrivado()
#p1._ClasseFortementePrivada__MetodoFortementePrivado()
#p1.__MetodoFortementePrivado()
print(p1.__atributoFortementePrivado)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[7], line 13
      9 p1 = ClasseFortementePrivada()
     10 #p1.__MetodoFortementePrivado()
     11 #p1._ClasseFortementePrivada__MetodoFortementePrivado()
     12 #p1.__MetodoFortementePrivado()
---> 13 print(p1.__atributoFortementePrivado)

AttributeError: 'ClasseFortementePrivada' object has no attribute '__atributoFortementePrivado'
```

In [11]:
```
# Acessores e Modificadores - Exemplo
class Cliente:
    __nome = None
    __telefone = None

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

    def getnome (self):
        return self.__nome
```

```
    def setnome (self, nome):
        self.__nome = nome

    def gettelefone (self):
        return self.__telefone

    def settelefone (self, telefone):
        self.__telefone = telefone
###
CadastroClientes = []
while True:
    nome = input("Digite o nome (fim para encerrar): ")
    if nome == "fim":
        break
    telefone = input("Telefone: ")
#    c1 = Cliente(nome,telefone)
#    CadastroClientes.append(c1)
    CadastroClientes.append(Cliente(nome,telefone))
print("Telefone Nome")
print("-------- ---------")
# for x, e in enumerate(CadastroClientes):
for e in CadastroClientes:
    print(e.gettelefone(), e.getnome())
```

```
Digite o nome (fim para encerrar): fim
Telefone Nome
-------- ---------
```

In [12]:

```
class Cliente:
    def __init__ (self, nome, telefone):
        self.__nome = nome
        self.__telefone = telefone
# se gravar a classe acima em um arquivo chamado 'clientes.py', usar a importação abaixo
#from clientes import Cliente
joao = Cliente("Joao da Silva", "777-1234")
maria = Cliente("Maria da Silva", "555-4321")
```

In [13]:

```
# Encapsulamento - Exemplo (slide 11)
# Para resolver o problema do Banco X, precisa-se de outra classe,
# Conta (salvar como contas.py, incluindo getters e setters), para
# representar uma conta do banco com seus clientes e seu saldo. Ficando:
class Conta:
    # incluir os atributos do construtor
    def __init__ (self, clientes, numero, saldo=0):
        self._saldo = saldo
        self.__clientes = clientes
        self.__numero = numero

    def resumo(self):
        print ("CC Número: ", self.__numero, "Saldo:", self._saldo)

    def saque(self, valor):
        if self._saldo >= valor:
            self._saldo -= valor

    def deposito(self, valor):
        self._saldo += valor
```

In [15]:

```
# Encapsulamento - Exemplo (slide 12)
# Alterando a classe Conta de forma a adicionar um atributo que é a lista
# de operações realizadas. Ficando:
class Conta:
    # incluir os atributos do construtor, os getters e os setters
    def __init__ (self, clientes, numero, saldo=0):
```

```
        self._saldo = 0
        self.__clientes = clientes
        self.__numero = numero
        self._operacoes = []
        self.deposito(saldo)
    def resumo(self):
        print ("CC Numero: ", self.__numero, "Saldo:", self._saldo)
    def saque(self, valor):
        if self._saldo >= valor:
            self._saldo -= valor
            self._operacoes.append(["Saque", valor])
    def deposito(self, valor):
        self._saldo += valor
        self._operacoes.append(["Deposito",valor])
    def extrato(self):
        print("Extrato CC Nr: %3d" % self.__numero)
        for e in self._operacoes:
            print("%10s %10.2f" % (e[0],e[1]))
        print ("\nSaldo: %10.2f\n" % self._saldo)
```

In [16]:

```
# Encapsulamento - Exemplo (slide 13)
# Modifique também o programa testes para imprimir o extrato de cada conta.
# Ficando conforme código abaixo
# Obs:
# faça as importações abaixo se as classes foram gravadas em arquivos separados.
# from exemplo_clientes import Cliente
# from exemplo_contas import Conta

joao = Cliente("Joao da Silva", "777-1234")
maria = Cliente("Maria da Silva", "555-4321")

conta1 = Conta([joao], 1, 1000)
conta2 = Conta([maria, joao], 2, 500)
conta1.saque(50)
conta2.deposito(300)
conta1.saque(190)
conta2.deposito(95.15)
conta2.saque(250)
conta1.extrato()
conta2.extrato()
```

```
Extrato CC Nr:   1
  Deposito    1000.00
     Saque      50.00
     Saque     190.00

Saldo:     760.00

Extrato CC Nr:   2
  Deposito     500.00
  Deposito     300.00
  Deposito      95.15
     Saque     250.00

Saldo:     645.15
```

In [1]:

```
# Resumo sobre Encapsulamento
# Banco - programa completo
# classe Cliente
class Cliente:
    def __init__ (self, nome, telefone):
        self.__nome = nome
        self.__telefone = telefone
    def getNome(self):
        return self.__nome
```

```
# classe Conta
class Conta:
    # incluir os atributos do construtor, os getters e os setters
    def __init__ (self, clientes, numero, saldo=0):
        self._saldo = 0
        self.__clientes = clientes
        self.__numero = numero
        self._operacoes = []
        self.deposito(saldo)
    def resumo(self):
        print ("CC Numero: ", self.__numero, "Saldo:", self._saldo)
    def saque(self, valor):
        if self._saldo >= valor:
            self._saldo -= valor
            self._operacoes.append(["Saque", valor])
    def deposito(self, valor):
        self._saldo += valor
        self._operacoes.append(["Deposito",valor])
    def extrato(self):
        print("Extrato CC Nr: %3d - Titular: %s" % (self.__numero,self.__clientes[0].getNome())
        for o in self._operacoes:
            print("%10s %10.2f" % (o[0],o[1]))
        print ("Saldo R$ %12.2f\n" % self._saldo)

## criando objetos clientes e contas
joao = Cliente("Joao da Silva", "777-1234")
maria = Cliente("Maria da Silva", "555-4321")

conta1 = Conta([joao], 1, 1000)
conta2 = Conta([maria, joao], 2, 500)
conta1.saque(50)
conta2.deposito(300)
conta1.saque(190)
conta2.deposito(95.15)
conta2.saque(250)
conta1.extrato()
conta2.extrato()
```

```
Extrato CC Nr:   1 - Titular: Joao da Silva
  Deposito    1000.00
     Saque      50.00
     Saque     190.00
Saldo R$       760.00

Extrato CC Nr:   2 - Titular: Maria da Silva
  Deposito     500.00
  Deposito     300.00
  Deposito      95.15
     Saque     250.00
Saldo R$       645.15
```

In [21]:

```
# teste com atributos públicos e privados
class Pessoa:
    __cpf = None
    __nome = None

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

    def getNome(self):
        return self.__nome

    def setNome(self,nome):
        self.__nome = nome
```

```
    def getCPF(self):
        return self.__cpf

class Aluno(Pessoa):
    __matricula = None

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

    def getMatricula(self):
        return self.__matricula

    def setMatricula(self,matricula):
        self.__matricula = matricula

##
p1 = Pessoa("38000011100","Luiz Antunes")
a1 = Aluno("11111111111","João José","20220101")
a2 = Aluno(p1.getCPF(),p1.getNome(),"20220101")

print(p1.getNome())
print(a1.getNome())
print(a2.getNome())

x = p1
if type(x) is Pessoa:
    print("A pessoa %s tem o CPF %s" % (x.getNome(),x.getCPF()))
if type(x) is Aluno:
    print("O aluno %s tem a matrícula %s" % (x.getNome(),x.getMatricula()))
x = a1
if type(x) is Pessoa:
    print("A pessoa %s tem o CPF %s" % (x.getNome(),x.getCPF()))
if type(x) is Aluno:
    print("O aluno %s tem a matrícula %s" % (x.getNome(),x.getMatricula()))

#print(p1.__nome) # erro
#print(p1._cpf) # erro: o _ na frente do atributo "esconde" o mesmo (privado).
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 36
     34 a1 = Aluno("11111111111","João José","20220101")
     35 a2 = Aluno(p1.getCPF(),p1.getNome(),"20220101")
---> 36 a3 = Aluno(p1,"20220101")
     38 print(p1.getNome())
     39 print(a1.getNome())

TypeError: Aluno.__init__() missing 1 required positional argument: 'matricula'
```

In [ ]: