- Classes são abstrações de elementos do mundo real. Os dados de uma classe não podem e não devem ser manipulados diretamente por uma funcionalidade implementada em outra classe. Na implementação de uma classe, qualquer alteração nos dados de uma classe deve acontecer pela invocação de um método da própria classe. Esta proteção é conhecida como -> Encapsulamento
- É correto afirmar que "um objeto da subclasse é, também, um objeto da superclasse, ou seja, os objetos da subclasse podem ser tratados como objetos da superclasse" -> Herança
- Em POO (Programação Orientada a Objetos), dizer que a classe A estende a classe B é o mesmo que dizer que -> A classe A é derivada de B;
- Os acessos e alterações dos dados de um objeto acontecem por meio de métodos implementados nesse objeto, para evitar que ocorram acessos diretos aos dados e assim evitando erros de alterações. Os dados ficam escondidos para dentro do objeto -> Encapsulamento
- Os objetos de uma classe podem herdar atributos e métodos de mais de uma classe base. Pode introduzir complexidade, como o problema do diamante de herança e ambiguidades -> Herança Múltipla
- É a característica única que permite que diferentes objetos respondam a mesma mensagem cada um a sua maneira. Em termos de programação, representa a capacidade de uma única referência invocar métodos diferentes, dependendo do seu conteúdo -> Polimorfismo
- Sejam A e B duas classes em um programa orientado a objetos. Se A é SUBCLASSE de B, então objetos da classe A PODEM POSSUIR MAIS atributos que objetos da classe B
- Neste exemplo, a classe Camisa está fazendo uma sobrescrita de método (são métodos diferentes) I. a classe Camisa implementa um outro método cor, diferente daquele da classe Blusa. IV. a classe Camisa poderá fazer uso de métodos pela herança direta da classeBlusa.
- De acordo com o código analisado, considere as seguintes asserções:
  - II. na linha 18 a classe B está herdando as características da classe base A;
  - III. a linha 26 contém polimorfismo (Sobrecarga).
- Verifique o código a seguir e selecione quais conceitos de OO estão sendo utilizados:
  - I. polimorfismo e herança.

Herança Simples (classe derivada)

- Processo de criação
  - de novas classes (classe derivada)
  - baseado em classes existentes (classe base)
- A classe derivada herda todas as características da classe base, além de incluir características próprias adicionais
- Própria classe -> Tudo; - Classes derivadas -> Público e Protegido!

Ser X Ter

- Herança (SER)
  - Um Cliente é uma Pessoa; Um Funcionário é uma Pessoa; Uma Conta Corrente é uma Conta
- Composição (TER)
  - Uma Pessoa tem uma Data de Aniversário; Um Cliente tem uma senha; Uma Conta tem um saldo

Classe Pessoa e Classe Funcionário

Sobrescrita

- As subclasses podem ter métodos com o mesmo NOME que a classe pai (sobrecarga)
- As subclasses podem ter métodos com a mesma ASSINATURA que a classe pai (sobrescrita)
- Finalidades: Reescrita e Extensão

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def getNome(self):
        return self.nome

    def __str__(self):
        return f'{self.nome} : {self.idade} anos'

class Funcionario(Pessoa):
    def __init__(self, nome, idade, salario):
        super().__init__(nome, idade)
        self.salario = salario

    def getSalario(self):
        return self.salario

    def getSalarioAnual(self):
        return self.salario * 12

    def __str__(self):
        return super().__str__() + '\nSalário=' + str(self.salario)

    def getBonificação(self):
        return self.salario * 0.10

class Professor(Funcionario):
    def getBonificação(self):
        return self.salario * 0.15

class Aluno(Pessoa):
    def __init__(self, nome, idade):
        super().__init__(nome, idade)

    def __str__(self):
        return super().__str__()

#main
f1 = Funcionario('Josias', 26, 3000)
p1 = Professor('Malaquias', 56, 5000)
a1 = Aluno('Bruno', 26)

print( f1.getBonificação() )
print( p1.getBonificação() )
print( a1.__str__())
```

Operadores

- Operadores podem ser sobrecarregados em classes através de métodos especiais conhecidos como métodos mágicos ou métodos especiais - iniciam e terminam por __(2 underlines)
- **len**(self)
- **add**(self, other) - **sub**(self, other) - **mul**(self, other) - **truediv**(self, other) - Div Inteira **floordiv**(self, other) - Resto da Div **mod**(self, other) - **pow**(self, other) - **and**(self, other) - **or**(self, other)
- Adição: **pos**(self) - precedido por um sinal de (+).
- Subtração: **neg**(self)precedido por um sinal de (-).
- Absoluto: **abs**(self) -> Negação lógica: **not**(self)
- Igualdade: **eq**(self, other) -> Diferença: **ne**(self, other)
- Menor que: **lt**(self, other) -> Menor ou igual a: **le**(self, other)
- Maior que: **gt**(self, other) -> Maior ou igual a: **ge**(self, other)
- Conversão para String: **str**(self) -> Conversão para Inteiro: **int**(self)
- Conversão para Float: **float**(self) -> Conversão para Booleano: **bool**(self)
- Acesso por Índice: **getitem**(self, key) -> - Def por Índice: **setitem**(self,key,value)
- Deleção por Índice: **delitem**(self, key) -> - Comprimento (len): **len**(self)
- Iteração (iter): **iter**(self)

```
class Numero:
    def __init__(self, valor):
        self.valor = valor
    def __pos__(self):
        return +self.valor
    def __neg__(self):
        return -self.valor
    def __abs__(self):
        return abs(self.valor)
num1 = Numero(-5)
print("Pos=", +num1)
print("Pos=", -num1)
print("Abs=", abs(num1))

class Playlist:
    def __init__(self):
        self.musics = []
    def __getitem__(self,index):
        return self.musics[index]
    def __setitem__(self, index, music):
        self.musics[index] = music
    def __len__(self):
        return len(self.musics)
    def adicionar_musica(self, music):
        self.musics.append(music)
playlist = Playlist()
playlist.adicionar_music("Thriller")
print("Músicas na playlist:")
for musica in playlist:
    print('-', musica)

print("Músicas na playlist:")
for i in range(len(playlist)):
    print('-', playlist[i])
```

```
class Bebida:
    nome = "Bebida"   # atributo de classe
    @classmethod
    def imprimir_nome(cls):
        printn(f"Esta bebida é do tipo {cls.nome})

class Café(Bebida):
    nome = "Café"
class Chá(Bebida):
    nome = "Chá"
#main
Bebida.imprimir_nome()
Café.imprimir_nome()
Chá.imprimir_nome()
```

- Método estático é um método que se comporta como uma função regular, mas pertence a uma classe.
  - o método não tem acesso nem à classe (cls)
  - o método não tem acesso à instância (self) na qual é chamado.
  - não pode modificar o estado da instância do objeto nem o estado da classe.

- Atributo de classe é aquele que é compartilhado por todos os objetos pertencentes àquela classe (Variável ÚNICA dentro da classe)
- Método de classe é um método que se comporta como uma função regular, mas pertence a uma classe.
  - recebe a classe como o primeiro argumento.
  - consegue manipular atributos da classe
  - o método não tem acesso à instância (self) na qual é chamado.

```
class Classe1:
    def metodo_objeto(self):
        return 'met instancia chamado'
    @classmethod
    def metodo_classe(cls):
        return 'met classe chamado'
    @staticmethod
    def metodo_estatico():
        return 'met estatico chamado'
obj1 = Classe1()
print( obj1.metodo_objeto())
print( Classe1.metodo_classe())
print( Classe1.metodo_estatico())
```