UNIVERSIDADE VEIGA DE ALMEIDA

Conceitos de Linguagem de Programação

JULIA CLARA SIQUEIRA LOPEZ

TRABALHO DA DISCIPLINA [AVA 2]

Rio de Janeiro

2021

Sumário

Introdução

A linguagem escolhida será Python e o Paradigma Orientado a Objeto. A programação orientada a objetos tem em vista a descrição de objetos do mundo real em um computador, para isso, faz uso de classes em torno de objetos com características e ações específicas. Nesse estilo de programação, o objetivo inicial do programador é identificar todos os objetos que ele pretende manipular e a relação entre os mesmos. Com os objetos identificados, são feitas generalizações dos mesmos em forma de classes de objetos que compreendem sequências de lógica para manipular um objeto, chamadas de métodos.

Desenvolvimento

Conceitos da Orientação a Objeto:

1. **Objetos ou Instâncias de Classe:** cada objeto possui propriedades próprias, elas são chamadas de atributos do objeto. Esses atributos são guardados na memória numa região relativa ao objeto específico, permitindo que cada instância tenha valores diferentes.
2. **Classes:** tipo definido pelo usuário, define uma estrutura que possibilita a geração de objetos distintos. Uma classe possui atributos e métodos, que são as bases para a caracterização dos objetos.
3. **Atributos:** são responsáveis por descrever as características compartilhadas por um conjunto de objetos semelhantes. Eles descrevem o objeto de forma geral.
    a. **de Classe:** são atributos universais da própria classe, ou seja, podem ser independentes das instâncias e cada instância pode alterar esse mesmo atributo (Ex: contador de Retângulos criados).
    b. **de Objeto:** são atributos locais, individuais de cada objeto e só existem para aquele objeto específico (Ex: largura do Retângulo).

4. **Métodos:** são funções que definem o comportamento dos objetos. Podem servir para manipular/alterar os atributos do próprio objeto ou para troca de informações entre objetos.
    a. **Getter/Setter:** são métodos especiais para atribuir valores aos atributos dos objetos e recuperar os mesmos, sem permitir que as variáveis sejam acessadas diretamente. Servem como uma forma de intermediador, uma camada de segurança extra.
    b. **Construtores:** são os métodos responsáveis pela definição de valores iniciais, durante a instanciação do objeto e também podem realizar outras ações nesse período.
5. **Herança:** permite que classes mais genéricas passem suas propriedades para classes mais específicas, como no exemplo do código, a interface Poligono é uma superclasse para Retangulo e Quadrado, pois ambos são polígonos e compartilham as mesmas características de um polígono, mas um objeto de Retangulo não necessariamente faz parte de Quadrado, pois pode ter altura diferente da largura.
    a. **Simples:** quando uma subclasse herda as propriedades de uma única superclasse por vez, como pode ser observado no código fonte pelo exemplo da classe Retangulo com a interface Poligono.
    b. **Múltipla:** quando uma subclasse herda as propriedades de mais de uma superclasse ao mesmo tempo, como é evidenciado no código fonte, quando Quadrado está herdando tanto de Retangulo, quanto de Poligono ao mesmo tempo.
6. **Encapsulamento:** forma de “esconder” membros de uma classe, para segurança, impedir que certos atributos sejam alterados após a inicialização ou apenas por serem irrelevantes ao usuário.
7. **Classes Abstratas:** São usadas como superclasses, são incompletas, são caracterizadas por usar pelo menos um método abstrato. Só devem ser utilizadas em situações de heranças, visto que elas não devem ser instanciadas.
    a. **Interface:** é um tipo específico de classe abstrata, onde todos os métodos são abstratos e seus atributos são estáticos, finais e podem ser acessados sem que o objeto seja instanciado. Uma classe pode implementar mais de uma interface. (Ex: Classe Poligono no código).

Código Fonte Comentado:

```
''' PARADIGMA ORIENTADO A OBJETOS '''
## Interface
class Poligono:
    ## Atributos da Classe
    elementos = ['Lados', 'Vértices',
                 'Ângulos Internos',
                 'Ângulos Externos',
                 'Diagonais', 'Convexidade']
    ## Métodos Abstratos
    def numeroLados(self)->int:
        pass
    def numeroVertices(self)->int:
        pass
    def angulosInternos(self)->int:
        pass
    def angulosExternos(self)->int:
        pass
    def numeroDiagonais(self)->int:
        pass
    def ehConvexo(self)->bool:
        pass

## Classe
class Retangulo(Poligono):
    ## Atributos da Classe
    contador = 0
    __lados = 4             ##
    __vertices = 4          ## Atributos
    __angulos = 360         ## Encapsulados
    __diagonais = 2         ##
    __convexidade = True    ##

    ## Método Construtor
    def __init__(self, altura:int, largura:int):
        if (altura and largura > 0):
            ## Atributos do Objeto
            '''
            setAltura(self, altura)
            setLargura(largura)
            '''
            self.__altura = altura
```

```
        self.__largura = largura
        Retangulo.contador += 1

'''
## Métodos Setter
def __setAltura(self, altura:int):
    ## Encapsulamento
    self.__altura = altura

def setLargura(self, largura:int):
    ## Encapsulamento
    self.__largura = largura
'''
## Métodos Getter
def getAltura(self)->int:
    return self.__altura

def getLargura(self)->int:
    return self.__largura

## Métodos da Interface
def numeroLados(self)->int:
    return self.__lados

def numeroVertices(self)->int:
    return self.__vertices

def angulosInternos(self)->int:
    return self.__angulos

def angulosExternos(self)->int:
    return self.__angulos

def numeroDiagonais(self)->int:
    return self.__diagonais

def ehConvexo(self)->bool:
    return self.__convexidade

## Métodos básicos
def area(self)->int:
    return self.__altura * self.__largura
```

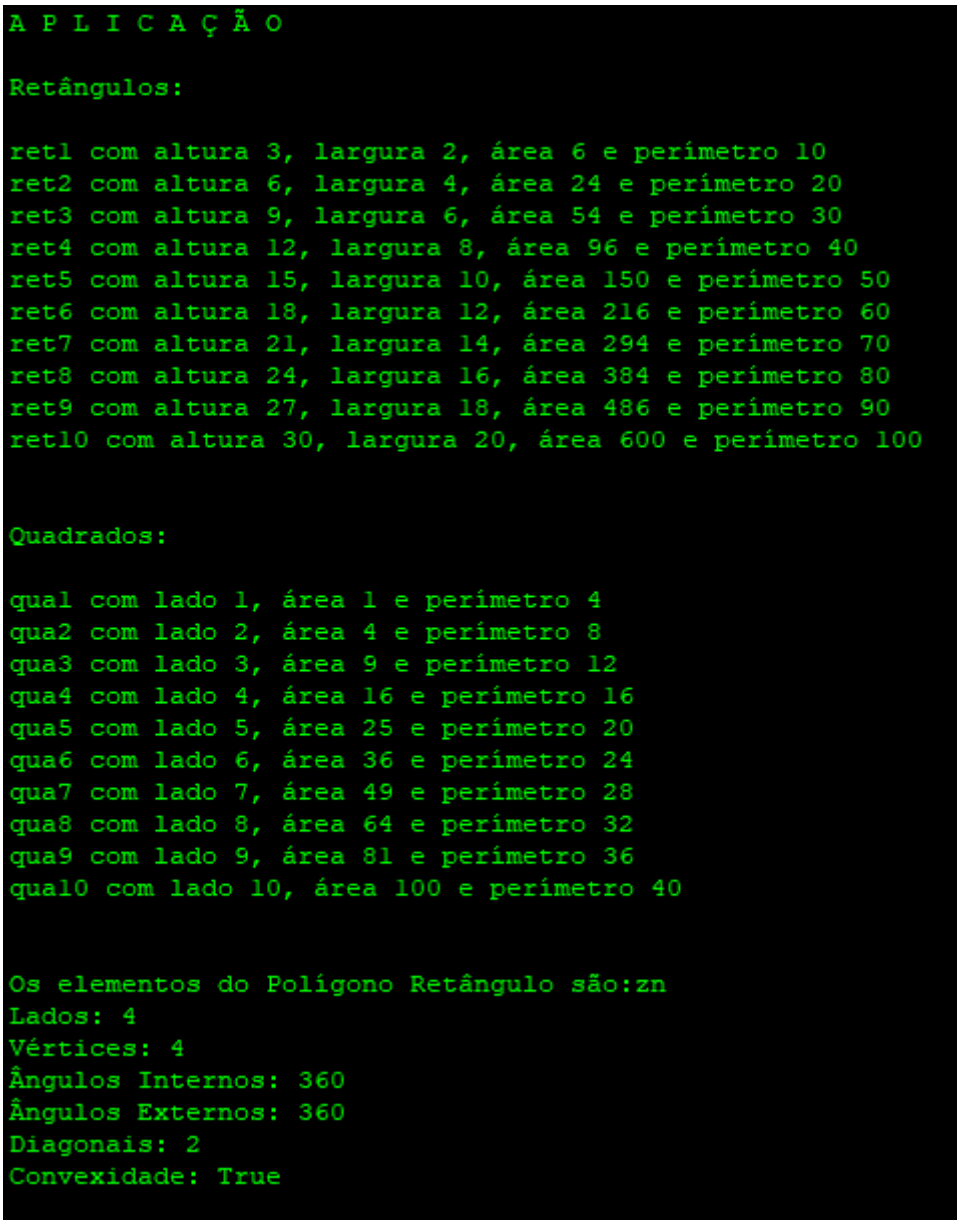
```
def perimetro(self)->int:
        return 2*(self.__altura + self.__largura)

## Herança
class Quadrado(Retangulo):
    ## Atributos da Classe
    contador = 0
    ## Polimorfismo de Sobrescrita
    def __init__(self, lado:int):
        Retangulo.__init__(self, lado, lado)
        self.__lado = lado
        Quadrado.contador += 1

    ## Método Getter
    def getLado(self)->int:
        return  self.__lado

## Aplicação
def main():
    print("A P L I C A Ç Ã O\n")

    ## Listas das Instâncias de Retangulo e Quadrado
    listaDeRetangulos = []
    listaDeQuadrados = []

    ## Inicialização de 10 Retângulos e 10 Quadrados
    for i in range(1, 11):
        listaDeRetangulos.append(Retangulo(i+2*i, i+i))
        listaDeQuadrados.append(Quadrado(i))

    ## Imprime cada Retângulo com seus atributos
    print("Retângulos:\n")
    for i in range(1, 11):
        print("ret{:d} com altura {:d}, ".format(i, listaDeRetangulos[i-1].getAltura()) +
              "largura {:d}, ".format(listaDeRetangulos[i-1].getLargura()) +
              "área {:d} e perímetro {:d}".format(listaDeRetangulos[i-1].area(),
                                        listaDeRetangulos[i-1].perimetro()))

    ## Imprime cada Quadrado com seus atributos
    print("\n\nQuadrados:\n")
    for i in range(1, 11):
        print("qua{:d} com lado {:d}, ".format(i, listaDeQuadrados[i-1].getLado()) +
              "área {:d} e perímetro {:d}".format(listaDeQuadrados[i-1].area(),
                                        listaDeQuadrados[i-1].perimetro()))
```

```
    ## Lista de Elementos do Retângulo
    elementosRet = []
    elementosRet.append(listaDeRetangulos[0].numeroLados())
    elementosRet.append(listaDeRetangulos[0].numeroVertices())
    elementosRet.append(listaDeRetangulos[0].angulosInternos())
    elementosRet.append(listaDeRetangulos[0].angulosExternos())
    elementosRet.append(listaDeRetangulos[0].numeroDiagonais())
    elementosRet.append(listaDeRetangulos[0].ehConvexo())

    ## Imprime o elemento seguido de seu valor específico para Retângulos
    print("\n\nOs elementos do Polígono Retângulo são:zn")
    for i in range(0, 6):
        print(Poligono.elementos[i] + ': ' + str(elementosRet[i]))


## Testes
alt1 = 2
larg1 = 3
lado1 = 4
r1 = Retangulo(alt1, larg1)
q1 = Quadrado(lado1)

## Testa o Método getAltura()
def testaGetAltura():
    print("Teste 1: getAltura()\n" +
          "A altura de r1 ({:d}) deve ser ".format(r1.getAltura()) +
          "igual a {:d}.".format(alt1))
    if r1.getAltura() == alt1:
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")

## Testa o Método getLargura()
def testaGetLargura():
    print("Teste 2: getLargura()\n"+
          "A altura de r1 ({:d}) deve ser ".format(r1.getLargura())  +
          "igual a {:d}.".format(larg1))
    if r1.getLargura() == larg1:
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")
```

```
## Testa o Método area()
def testaArea():
    print("Teste 3: area()\n" +
          "A área de r1 ({:d}) deve ser igual ao ".format(r1.area()) +
          "produto de {:d} com {:d}, ".format(alt1, larg1) +
          "que é {:d}.".format(alt1*larg1))
    if r1.area() == alt1*larg1:
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")

## Testa o Método perimetro()
def testaPerimetro():
    print("Teste 4: perimetro()\n" +
          "O perímetro de r1 ({:d}) deve ser igual à ".format(r1.perimetro()) +
          "soma dos lados de r1, ou seja, {:d}.".format(2*(alt1+larg1)))
    if r1.perimetro() == 2*(alt1+larg1):
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")

## Testa a Herança Simples
def testaHerancaSimples():
    print("Teste 5: Herança Simples\n" +
          "O método isinstance() com os parâmetros:\n" +
          "r1(objeto) e Poligono(interface) " +
          "precisa indicar 1 como booleano para True.")
    ## Testa se Retangulo é subclasse de Polígono
    if isinstance(r1, Poligono):
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")

## Testa a Herança Múltipla
def testaHerancaMultipla():
    print("Teste 6: Herança Múltipla\n" +
          "O método isinstance() com os parâmetros:\n" +
          "r1(objeto), Poligono(interface) e r1(objeto), Retangulo(classe), \n" +
          "precisa indicar 1 como booleano para True.")
    ## Testa se a classe quadrado é subclasse de Poligono e Retangulo
    if isinstance(q1, Poligono) and (q1, Retangulo):
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")
```

```
## Testa a Mutabilidade do Atributo da Classe
def testaAtributoClasse():
    print("Teste 7: Mutabilidade do Atributo da Classe\n" +
          "A classe Retangulo tem " +
          "{:d} instâncias ativas. \n" .format(Retangulo.contador) +
          "Se criarmos mais uma instância, o contador precisará " +
          "ser incrementado de 1.")
    aux = Retangulo.contador
    r2 = Retangulo(larg1, alt1)
    if Retangulo.contador == aux+1:
        print("PASSOU!!\n\n")
    else:
        print("FALHOU!!\n\n")

## Chamada para todos os Testes
def testes():
    print("\n\nT E S T E S\n")
    print("Retângulo r1\naltura: {:d}\n".format(alt1) +
          "largura: {:d}\n".format(larg1))
    print("Quadrado q1\nlado: {:d}\n".format(lado1))
    testaGetAltura()
    testaGetLargura()
    testaArea()
    testaPerimetro()
    testaHerancaSimples()
    testaHerancaMultipla()
    testaAtributoClasse()

main()
testes()
```

Captura de tela da Aplicação:



Captura de tela dos Testes:

```
T E S T E S

Retângulo r1
altura: 2
largura: 3

Quadrado q1
lado: 4

Teste 1: getAltura()
A altura de r1 (2) deve ser igual a 2.
PASSOU!!


Teste 2: getLargura()
A altura de r1 (3) deve ser igual a 3.
PASSOU!!


Teste 3: area()
A área de r1 (6) deve ser igual ao produto de 2 com 3, que é 6.
PASSOU!!


Teste 4: perimetro()
O perímetro de r1 (10) deve ser igual à soma dos lados de r1, ou seja, 10.
PASSOU!!


Teste 5: Herança Simples
O método isinstance() com os parâmetros:
r1(objeto) e Poligono(interface) precisa indicar 1 como booleano para True.
PASSOU!!


Teste 6: Herança Múltipla
O método isinstance() com os parâmetros:
r1(objeto), Poligono(interface) e r1(objeto), Retangulo(classe),
precisa indicar 1 como booleano para True.
PASSOU!!


Teste 7: Mutabilidade do Atributo da Classe
A classe Retangulo tem 22 instâncias ativas.
Se criarmos mais uma instância, o contador precisará ser incrementado de 1.
PASSOU!!
```

Conclusão

A programação orientada a objetos apresenta vantagens em relação à programação estruturada. Ela está mais focada nos dados do que na estrutura de processo e lógica. Segue uma abordagem ascendente e tem a capacidade de resolver qualquer programa complexo, enquanto a estruturada, por sua vez, consegue apenas resolver programas moderadamente complexos. O paradigma orientado a objetos suporta herança, encapsulamento (ocultar dados), abstração, polimorfismo, entre outras características vistas acima nos conceitos desse paradigma. Por conta disso tudo, a programação orientada a objeto tem uma menor dependência de função e acaba sendo mais flexível.

BIBLIOGRAFIA

1. Roteiro de Estudo - UNIDADE 4 - https://uva.instructure.com/courses/17142/files/2560127?module_item_id=195029&fd_cookie_set=1
2. https://stackoverflow.com/
3. https://www.tutorialspoint.com
4. Material sobre Orientação a Objeto do Prof. José Augusto Sapienza Ramos - https://sites.google.com/view/comp2-ufrj-sapienza-20172
5. Material sobre Orientação a Objeto do Prof Vinicius Gusmão P. De Sá - https://github.com/vigusmao