Encapsulamento

Programação Orientada a Objetos

Python

Prof. Diógenes Furlan

Sumário

- Construtores
- Destrutores
- Modificadores de Acesso

Construtores

- **Construtor** de classe é um método especial em POO
- É chamado automaticamente quando um objeto é criado
- Tem as funções de:
  - inicializar os atributos da classe
  - realizar operações necessárias para preparar o objeto para uso
- Não são obrigatórios.
- Podem ser vazios.

Classe Carro

- O construtor em Python é um método chamado __init__
  - dois underlines

```
class Carro:
    def __init__(self, marca, modelo, ano):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano

# Criando uma instância da classe Carro
meu_carro = Carro("Toyota", "Corolla", 2022)
```

Classe Monitor

- Classes podem ser definidas de forma bem simples apenas com o uso de construtores

```
class Monitor1:
    def __init__(self, x, y):
        self.stLigado = False
        self.resolucaoX = x
        self.resolucaoY = y

#main
m = Monitor1(800, 600)
```

Destrutores

- O **destrutor** é um método especial em programação que é chamado automaticamente quando um objeto é destruído ou liberado da memória.
- Tem as funções de:
  - realizar limpeza
  - liberar recursos associados ao objeto

- Python não possui suporte a destrutor
  - utiliza "garbage collector"

Destrutores (C++)

```
#include <iostream>
using namespace std;

class Exemplo {
public:
   // Construtor
   Exemplo() {
      cout << "Construtor chamado" << endl;
   }

   // Destrutor
   ~Exemplo() {
      cout << "Destrutor chamado" << endl;
   }
};

int main() {
   // Criando uma instância da classe Exemplo
   Exemplo exemplo;
   // O destrutor é chamado quando a instância 'exemplo' sair do escopo
   return 0;
}
```

Classe Monitor

- Porém a programação de métodos “garante” a existência de certos comportamentos

- Monitor:
  - ligar
  - desligar
  - alterar resolução
  - verificar o status

Classe Monitor

```
class Monitor2:
    def ligar(self):
        self.stLigado = True;

    def desligar(self):
        self.stLigado = False;

    def alterarResolução(self, x:int, y: int):
        self.resoluçãoX = x
        self.resoluçãoY = y

    def status(self) -> bool:
        return self.stLigado
```

Classe Monitor

```
# main
m = Monitor2()
m.ligar()
print( m.status() )
m.desligar()
print( m.status() )
```

Comportamento desejado

Como garantir que o usuário da classe utilize os métodos criados??

```
# main
m = Monitor2()
m.stLigado = True
print( m.status() )
m.stLigado = False
print( m.status() )
```

Comportamento indesejado

Pergunta da Aula

Em relação ao conteúdo interno da classe, como controlar (limitar) o que é visto ou utilizado pelo usuário da classe?

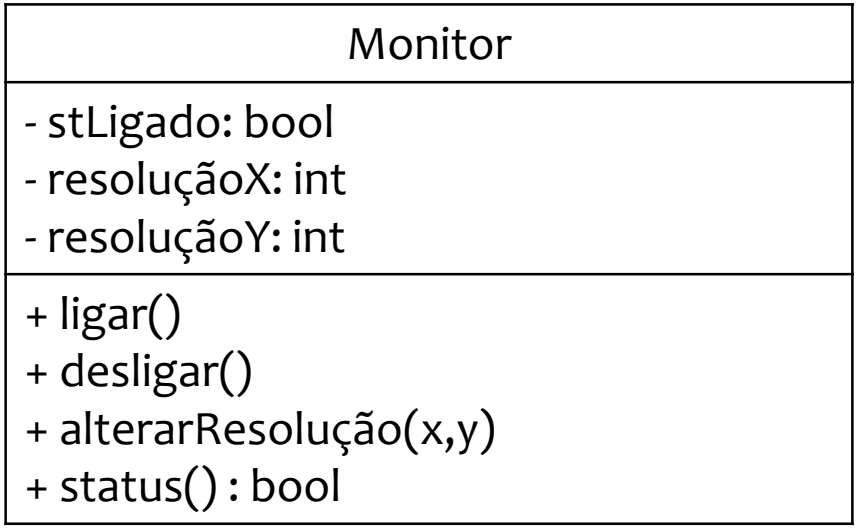
Visibilidade

- É a forma com que os elementos de uma classe (atributos e métodos) podem ser vistos e utilizados externamente
  - Privado
    - somente no interior da classe
  - Protegido
    - somente no interior da classe, e de suas herdeiras
  - Público
    - dentro e fora da classe, de forma livre

Diagrama UML

Notação UML para visibilidade

```
+ public
# protected
- private
```



Classe Monitor

- Atributos privados em Python são declarados iniciando com 2 underlines (__)

```
class Monitor3:
    def ligar(self):
        self.__stLigado = True;

    def desligar(self):
        self.__stLigado = False;

    def alterarResolução(self, x:int, y: int):
        self.resoluçãoX = x
        self.resoluçãoY = y

    def status(self) -> bool:
        return self.__stLigado
```

Visibilidade no Python

- Métodos privados são declarados iniciando com 2 underlines (__)
- Atributos privados são declarados iniciando com 2 underlines (__)

Classe Monitor

```
class Monitor4:
    def ligar(self):
        self.__stLigado = True;
        self.__log('Ligando')

    def desligar(self):
        self.__stLigado = False;
        self.__log('Desligando')

    def alterarResolução(self, x:int, y: int):
        self.resoluçãoX = x
        self.resoluçãoY = y
        self.__log('Alterando resolução')

    def status(self) -> bool:
        return self.__stLigado
        self.__log('Verificando status')

    def __log(self, msg):
        print(msg)
```

Main

- Teste e remova a chamada ao método __log
  - pois ele é privado, só pode ser utilizado dentro da classe

```
# main
m = Monitor4()
m.ligar()
m.desligar()
m.alterarResolução(800,600)
print( m.status() )
m.__log('escreva')
```

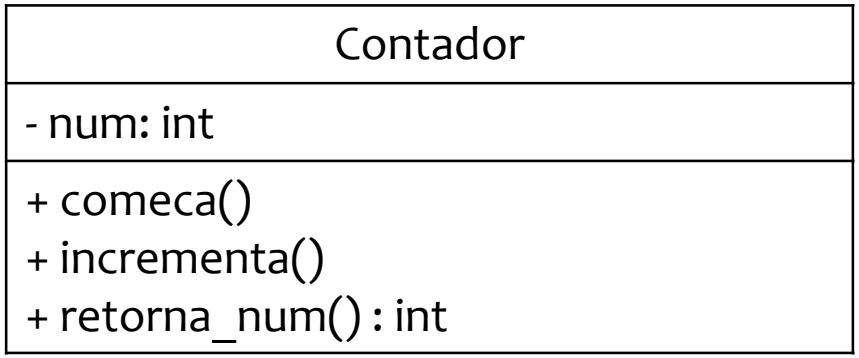
Tarefa 1

- Adicionar um contador de utilizações no monitor.
  - adicionar um atributo
  - incrementar o contador ao monitor ser ligado
  - adicionar um método para mostrar este contador

- Adicionar um método mostraResolução()

Tarefa 2

- Adicionar atributos privados na classe Contador da aula passada



Encapsulamento no Java

- Testar em
  - https://www.onlinegdb.com/online_java_compiler

Classe Monitor

```
public class Monitor {
    private boolean stLigado = false;
    private int resolucaoX;
    private int resolucaoY;

    public Monitor(int x, int y) {
        this.resolucaoX = x;
        this.resolucaoY = y;
    }

    public void ligar() {
        this.stLigado = true;
        this.log("Ligando");
    }

    public void desligar() {
        this.stLigado = false;
        this.log("Desligando");
    }
```

Classe Monitor

```
    public void alterarResolucao(int x, int y) {
        this.resolucaoX = x;
        this.resolucaoY = y;
        this.log("Alterando resolução");
    }

    public boolean status() {
        this.log("Verificando status");
        return this.stLigado;
    }

    private void log(String msg) {
        System.out.println(msg);
    }

    public static void main(String[] args) {
        Monitor m = new Monitor(1920, 1080); // cria objeto
        m.ligar();
        m.desligar();
        m.alterarResolucao(800, 600);
        System.out.println(m.status());
    }
}
```

EXERCÍCIOS

Exercício 1

- Programe a classe Fração

Fração
- num: int - den: int
+ Fração(num:int, den:int) + soma(Fração,Fração) + mostra() + toReal() : double

Exercício 2

- Encapsular as seguintes classes das aulas passadas:
  - Caixa
  - Contador