Python para Aplicações em Eletrônica

Aula 05: Programação Orientada a Objetos

Programação Orientada a Objetos

- A programação orientada a objetos (POO) é um paradigma de programação
- O código é planejado como um conjunto de objetos que interagem entre si.
- Principais vantagens:
  - Abstração (identidade, características e ações dos objetos)
  - Encapsulamento
  - Herança
  - Polimorfismo

Conceitos Básicos

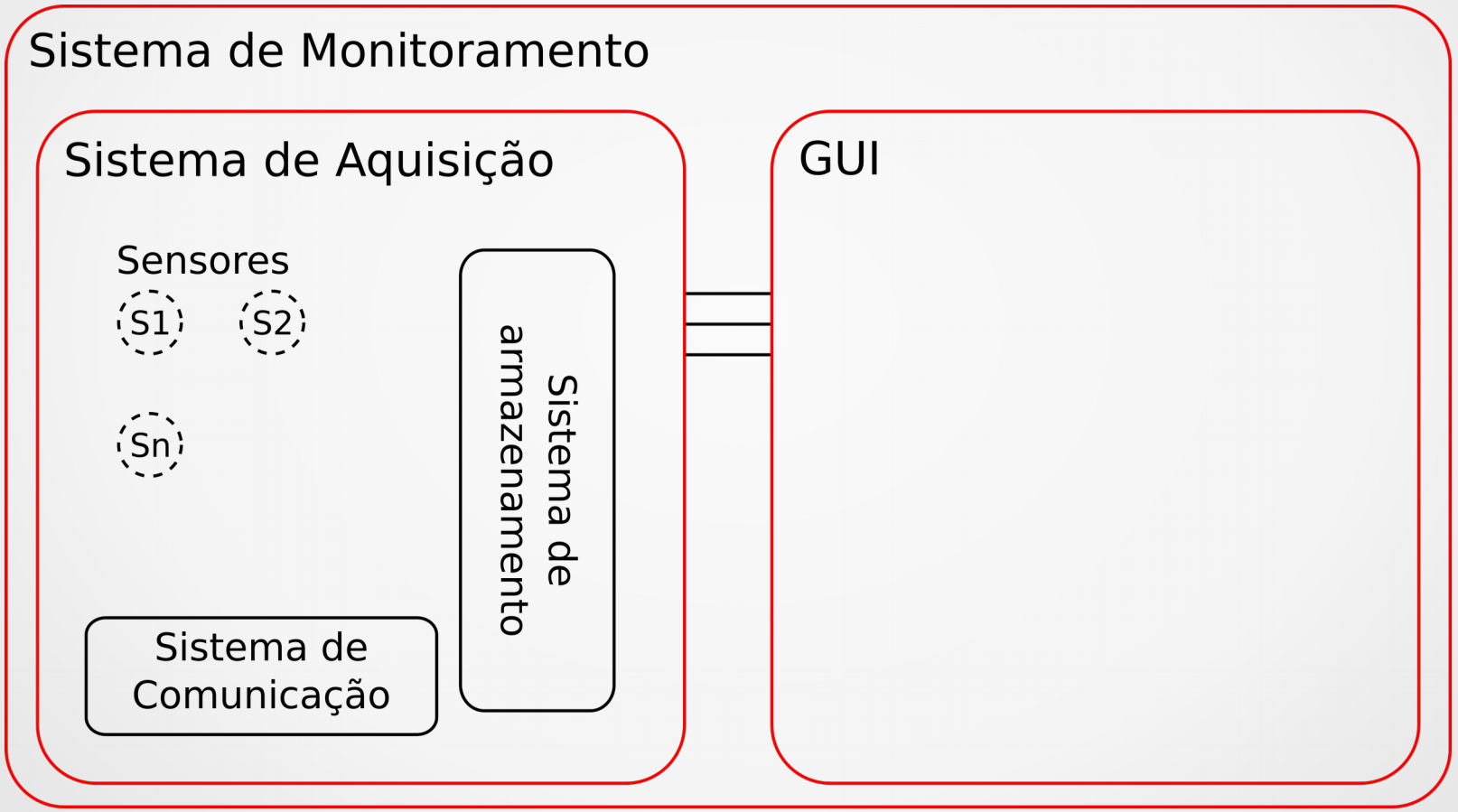
- Classes, atributos, métodos, objetos.
- Exemplo: Classe Pokemon

Pokemon
Name: Pikachu
Type: Electric
Health: 70
attack()
dodge()
evolve()

Fields

Methods

Exemplo Motivador



Principais elementos: Uso básico

```
1 class Caixa:
2     def __init__(self, c,l,h):
3         self.c = c
4         self.l = l
5         self.h = h
6 
7     def volume(self):
8         return self.c*self.l*self.h
9 
10    def area(self):
11        return 2*self.c*self.l +2*self.c*self.h + 2*self.l*self.h
12 
13 cx1 = Caixa(2,3,4)
14 cx2 = Caixa(10,3,3)
15 
16 print('caixa {}: Volume {} e Area {}'.format(1,cx1.volume(), cx1.area()))
17 print('caixa {}: Volume {} e Area {}'.format(2,cx2.volume(), cx2.area()))
```

```
caixa 1: Volume 24 e Area 52
caixa 2: Volume 90 e Area 138
```

Principais elementos: Uso básico

- **`__init__()`**: construtor da classe.
- **`self`**: usado para se referir aos atributos e métodos do objeto.

Principais elementos: Uso básico

```
 1 class Caixa:
 2     n_cx=0
 3     def __init__(self, c,l,h):
 4         self.c = c
 5         self.l = l
 6         self.h = h
 7 
 8         Caixa.n_cx+=1
 9         self.__name = 'caixa '+ str(Caixa.n_cx)
10 
11     def volume(self):
12         return self.c*self.l*self.h
13 
14     def area(self):
15         return 2*self.c*self.l +2*self.c*self.h + 2*self.l*self.h
16 
17     def get_name(self):
18         return self.__name
19 
20 
21 cx1 = Caixa(2,3,4)
22 cx2 = Caixa(10,3,3)
23 cx3 = Caixa(10,3,3)
24 
25 print('Numero de caixas: {}'.format(Caixa.n_cx))
26 print('caixa {}: H {} Volume {} e Area {}'.format(cx2.get_name(),cx2.h,cx2.volume(), cx2.area()))
27 print('caixa {}: Volume {} e Area {}'.format(cx2.__name,cx2.h, cx2.volume(), cx2.area()))
```

```
Numero de caixas: 3
caixa caixa 2: H 3 Volume 90 e Area 138
Traceback (most recent call last):
  File "ex02.py", line 27, in <module>
    print('caixa {}: Volume {} e Area {}'.format(cx2.__name,cx2.h, cx2.volume(), cx2.area()))
AttributeError: Caixa instance has no attribute '__name'
```

Principais elementos: Uso básico

- Variáveis de classe: Se refere à classe e não a um objeto em particular.
- Visibilidade: É possível tornar um atributo inacessível externamente. Basta iniciar seu nome com “__”(ex: **__id**).
- A visibilidade é uma das ferramentas de encapsulamento.
- Evita a alteração “de qualquer jeito” dos valores de atributos.
- É necessário definir métodos para acessar os atributos.
- Isso aumenta a segurança.

Herança

- O mecanismo de herança permite definir uma nova classe que herda os atributos e métodos de uma classe existente.
- Evita retrabalho na criação a atualização de códigos.
- É muito útil para permitir a particularização de objetos.
  - Ex: Caixa (classe pai); CaixaBombom, CaixaJoias, etc (classes filhas)
- Ajuda a manter a consistência, por exemplo, na manipulação de dados.

Herança

```
1 class Caixa:
2     def __init__(self):
3         print('Criando uma caixa')
4 
5     def set_dimensions(self,c,l,h):
6         self.c = c
7         self.l = l
8         self.h = h
9 
10    def volume(self):
11        return self.c*self.l*self.h
12 
13    def area(self):
14        return 2*self.c*self.l +2*self.c*self.h + 2*self.l*self.h
15 
16 class CaixaBombom(Caixa):
17    def __init__(self, qt):
18        self.qt = qt
19 
20 cx = Caixa()
21 cx.set_dimensions(2,3,4)
22 cxBB = CaixaBombom(100)
23 cxBB.set_dimensions(5,3,4)
24 
25 print('caixa Normal: Volume {} e Area {}'.format(cx.volume(), cx.area()))
26 print('caixa de Bombom: Quantidade {} Volume {} e Area {}'.format(cxBB.qt, cxBB.volume(), cxBB.area()))
```

```
Criando uma caixa
caixa Normal: Volume 24 e Area 52
caixa de Bombom: Quantidade 100 Volume 60 e Area 94
```

Herança: Sobrescrita de Métodos

- O mecanismo de herança permite sobrescrever métodos (polimorfismo).
- Corresponde a redefinição de um método já criado, no contexto de uma classe filha.

Herança: Sobrescrita de Métodos

```
 1 class Caixa:
 2     def __init__(self,c,l,h):
 3         self.c = c
 4         self.l = l
 5         self.h = h
 6 
 7     def volume(self):
 8         return self.c*self.l*self.h
 9 
10     def area(self):
11         return 2*self.c*self.l +2*self.c*self.h + 2*self.l*self.h
12 
13 class CaixaBombom(Caixa):
14     def __init__(self, c,l,h,qt):
15         self.c = c
16         self.l = l
17         self.h = h
18         self.qt = qt
19 
20     def area(self):
21         return self.c*self.l
22 
23 cx = Caixa(2,3,4)
24 cxBB = CaixaBombom(5,3,4,100)
25 
26 print('caixa Normal: Volume {} e Area {}'.format(cx.volume(), cx.area()))
27 print('caixa de Bombom: Quantidade {} Volume {} e Area {}'.format(cxBB.qt, cxBB.volume(), cxBB.area()))
```

```
caixa Normal: Volume 24 e Area 52
caixa de Bombom: Quantidade 100 Volume 60 e Area 15
```

Tópicos Especiais

Uso de Vários Arquivos

Arquivo Modulo.py

```python
class Caixa:
    def __init__(self,c,l,h):
        self.c = c
        self.l = l
        self.h = h

    def volume(self):
        return self.c*self.l*self.h

    def area(self):
        return 2*self.c*self.l +2*self.c*self.h + 2*self.l*self.h
```

```python
from Modulo import Caixa

class CaixaBombom(Caixa):
    def __init__(self, c,l,h,qt):
        self.c = c
        self.l = l
        self.h = h
        self.qt = qt

    def area(self):
        return self.c*self.l

cx = Caixa(2,3,4)
cxBB = CaixaBombom(5,3,4,100)

print('caixa Normal: Volume {} e Area {}'.format(cx.volume(), cx.area()))
print('caixa de Bombom: Quantidade {} Volume {} e Area {}'.format(cxBB.qt, cxBB.volume(), cxBB.area()))
```

Exercícios

Exercícios

1) Criar uma classe sensor (atributos: id, valor, incerteza, etc.; métodos: *getters* e *setters*, “medição”, “detecção de erro”, etc).
2) Criar classes filhas (sensor de vazão, sensor de nível, etc) e redefinir o método de “medição”).
   - Implemente um esquema para geração de id unico

Exercícios

3) Criar um classe que realize o armazenamento e acesso a dados dos sensores (o acesso pode ser por número de amostras, por exemplo)

Exercícios

4) Crie uma classe “sistema de comunicação” para auxiliar a obtenção de dados da placa de aquisição (referente a medição de cada sensor).
   - Sua classe pode simular a obtenção de dados de um microcontrolador ou obter do arduino.
   - A ideia é prezar pelo encapsulamento. Antes de programar, tente fazer um diagrama que indique como os objetos irão interagir.

Exercícios

5) Implemente a comunicação com arduino na classe “sistema de comunicação”.

- Projete sua classe (e o código do arduino) para permitir varredura (coletar os dados de todos os sensores de uma vez) e leitura individual (retorna apenas a medição do sensor solicitado).