

Interação entre objetos

Criando objetos cooperadores

Abstração

- **Problema da complexidade**
 - Dividir para conquistar
- **Abstração é a habilidade de ignorar detalhes sobre as partes para concentrar a atenção no nível mais alto de um problema.**
- **Exemplo de um problema:**
 - Engenheiros projetando um novo carro

Abstração em software

- Identificação de subcomponentes que podemos programar como entidades independentes.
- Em POO, esses componentes e subcomponentes são objetos.

Modularização

- **Modularização** é o processo de dividir um todo em partes bem definidas, que podem ser construídas e examinadas separadamente e que interagem de uma maneira bem definida.
- A modularização e a abstração complementam-se reciprocamente.

Modularização no exemplo da universidade

- Estudantes e professores precisam fornecer um endereço para contato.
- Vamos agora adicionar um endereço para classes Estudante e Professor.
- Pode ser uma variável de instância do tipo String
- Mas para melhor decomposição, podemos definir uma outra classe de nome Endereço.

Implementação — Endereco

```
class Endereco:
    """attributes:
        __numero
        __complemento
        __nome
        __cidade
        __codPostal
    """
```

*Construtor e
métodos omitidos.*

Código-fonte: Endereco

```
def __init__(self,num):  
    self.__numero = num  
    self.__complemento = 0
```

```
def incremento(self):  
    self.__complemento = self.__complemento + 1
```

Implementação — Estudante

```
class Estudante:
    """attributes:
        __endereco - classe Endereco
        __nome
        __matricula
        __creditos
    """
```

*Construtor e
métodos omitidos.*

Implementação — Estudante

```
def __init__(self, end, nom, mat, cre) :  
  
    self.__endereco = end  
    self.__nome = nom  
    self.__matricula = mat  
    self.__creditos = cred  
  
métodos omitidos.
```

Implementação – main.py

```
end1 = Endereco(61)
```

```
aluno1=Estudante(end1,'Ian',9294,0)
```

```
aluno2=Estudante(Endereco(62),'Ad',3112,0)
```

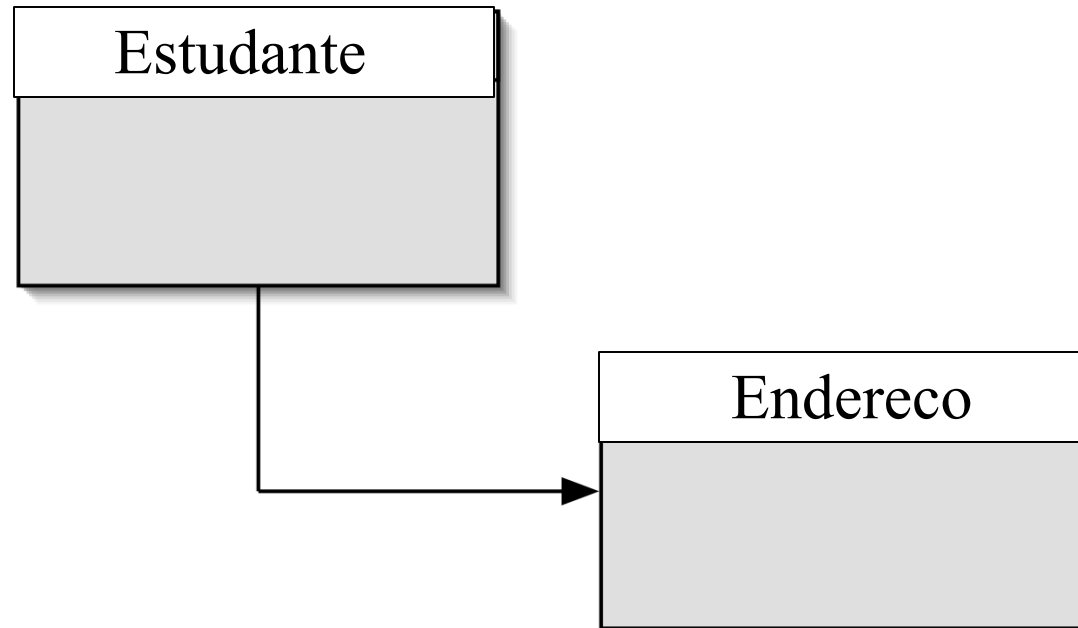
Dois imports no programa principal: um para Endereco e outro para Estudante

Outra implementação — Estudante

```
def __init__(self, nom, mat, cre) :  
  
    self.__endereco = Endereco(61)  
    self.__nome = nom  
    self.__matricula = mat  
    self.__creditos = cred
```

métodos omitidos.

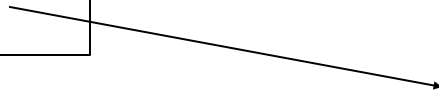
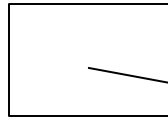
Diagrama de classes



Visualização estática

Tipos primitivos *versus* tipos de objeto (1)

`SomeObject obj`



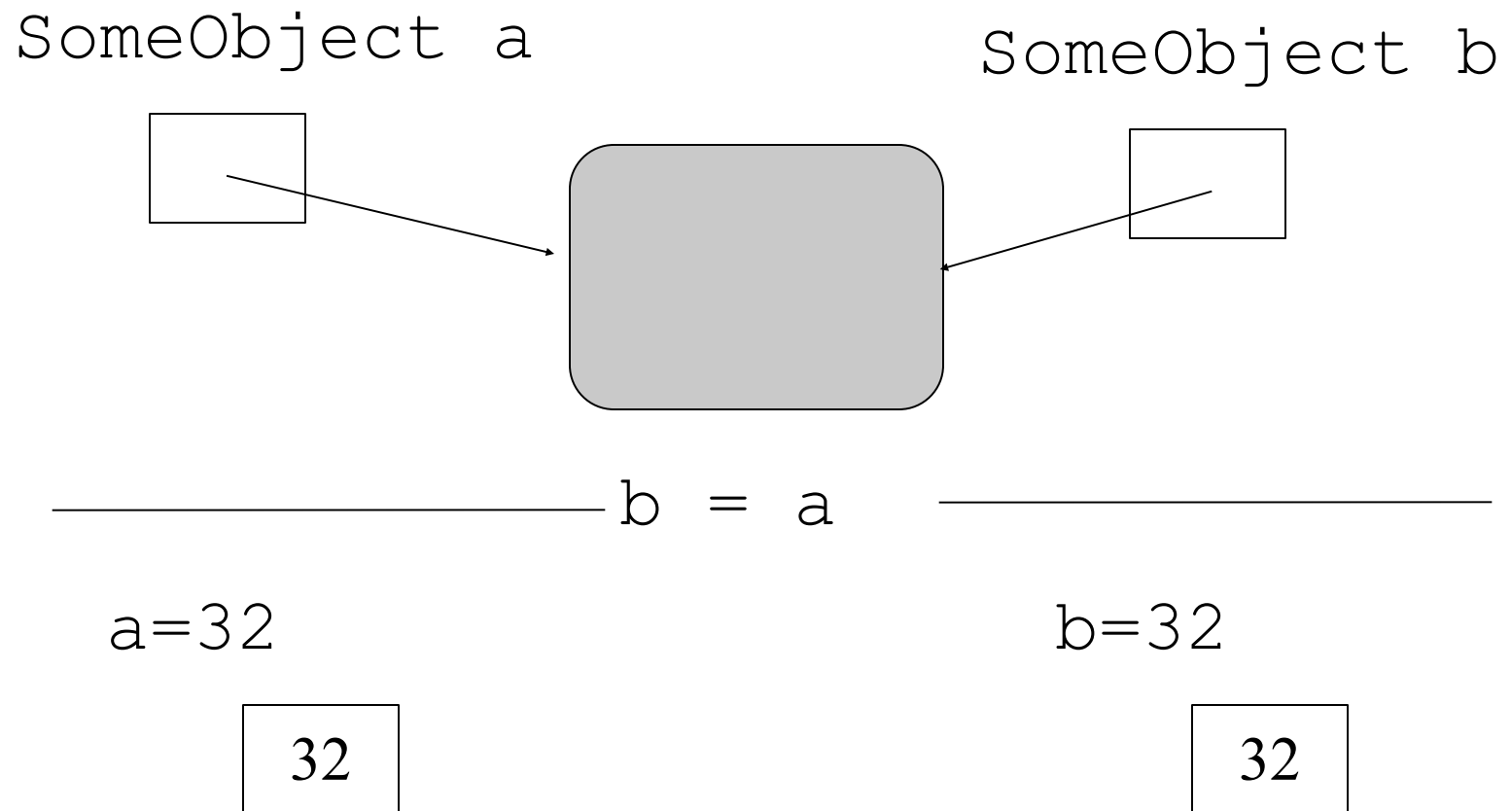
Tipo de objeto

`i = 32`



tipo primitivo

Tipos primitivos *versus* tipos de objetos (2)



Incremento linear

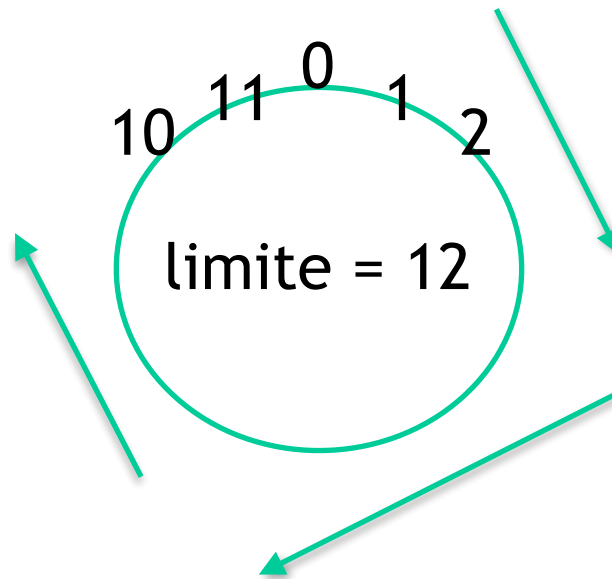
```
def increment(self):  
    self.__value = self.__value + 1
```

Operador de módulo

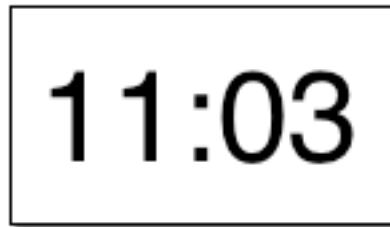
- Operador de módulo (%) calcula o resto de uma divisão de inteiros
- O resultado da expressão $(27 \% 4)$ seria 3

Incremento circular

```
def increment(self):  
    self.__value = (self.__value + 1) % self.__limit
```



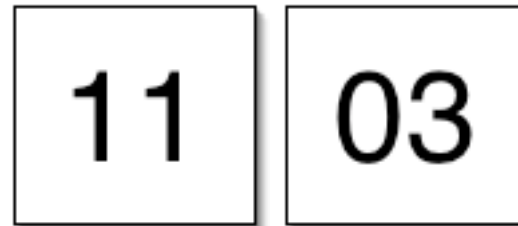
Modularização no exemplo de um relógio



11:03

Um mostrador de número de quatro dígitos?

Ou um mostrador de número de dois dígitos?

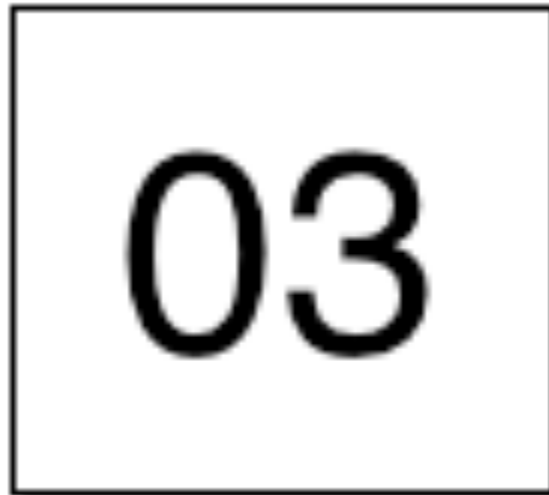


11 03

Modularização no exemplo do relógio

- Dois mostradores de dois dígitos cada um
 - Um par para horas e um par para os minutos
- Mostrador das horas:
 - Inicia em 0 e volta para 0 quando alcança o valor 23
- Mostrador de minutos
 - Inicia em 0 e retorna quando alcança 59

Mostrador de dois dígitos



Classe NumberDisplay

Implementação — NumberDisplay

```
class NumberDisplay:
    """
        __limit
        __value

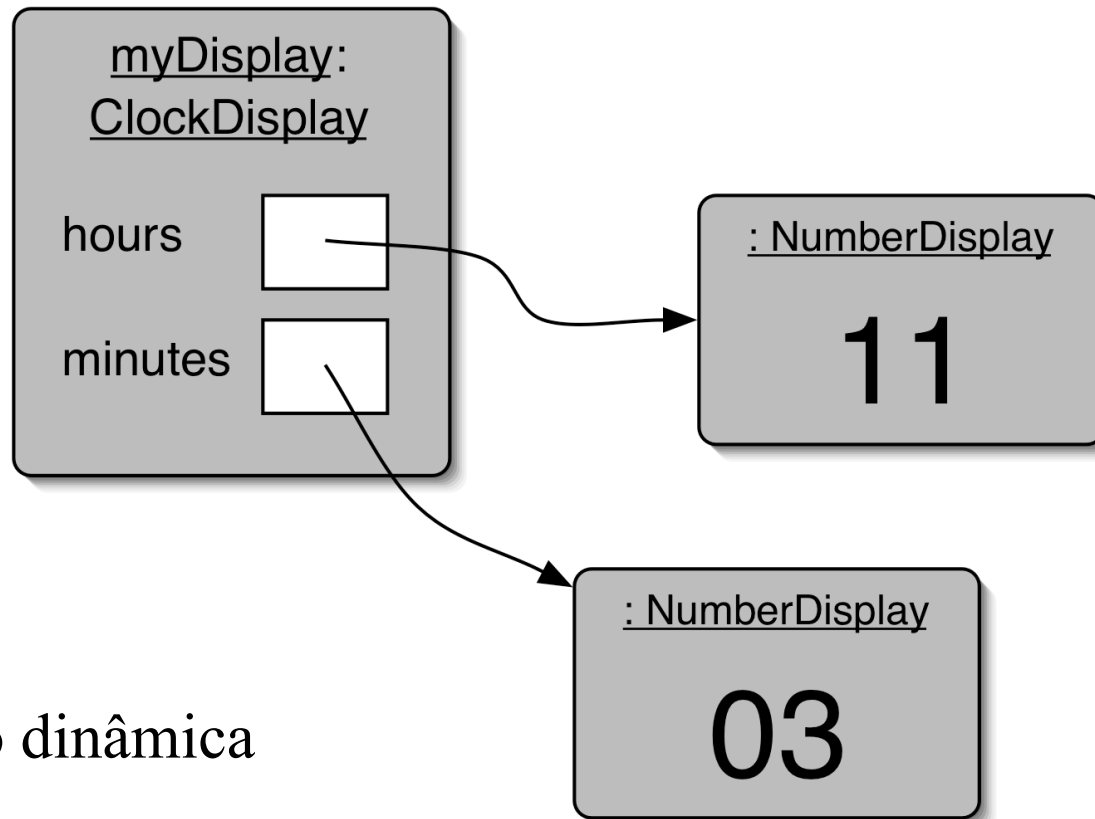
        Construtor e
        métodos omitidos.
    """
```

Implementação — ClockDisplay

```
class ClockDisplay:
    """ attributes:
        __hours - classe NumberDisplay
        __minutes - classe NumberDisplay

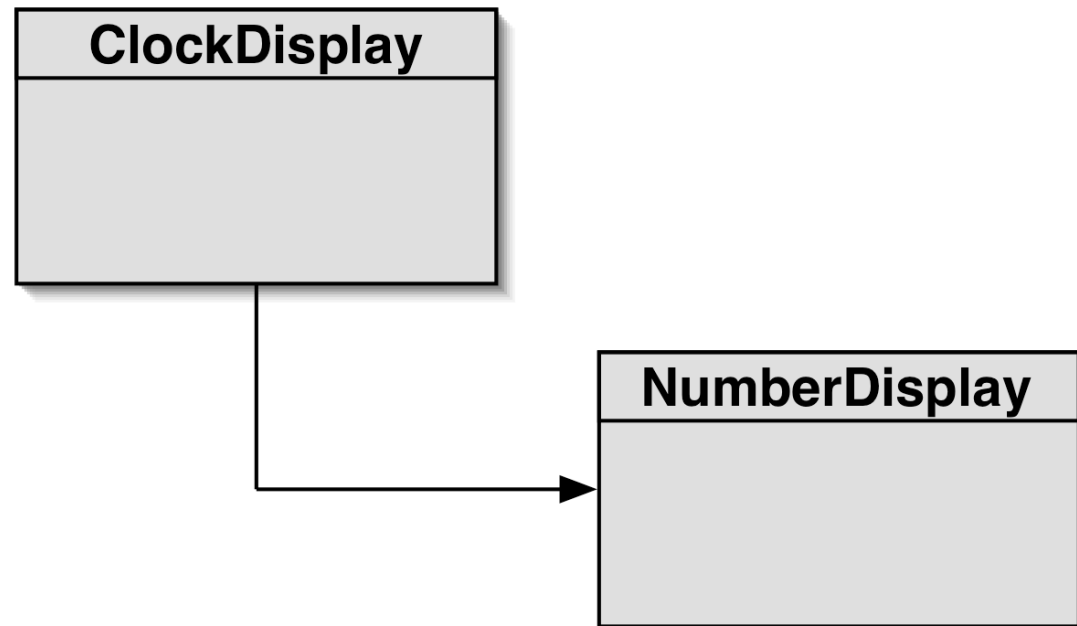
        Construtor e
        métodos omitidos.
    """
```

Diagrama de objetos



Visualização dinâmica

Diagrama de classes



Visualização estática

Código-fonte: NumberDisplay (1)

```
def __init__(self, rollOverLimit):  
  
    self.__limit = rollOverLimit  
    self.__value = 0  
  
def increment(self):  
  
    self.__value = (self.__value + 1) % self.__limit
```

Código-fonte: NumberDisplay (2)

```
def getDisplayValue(self):  
    if(self.__value < 10):  
        return "0" + str(self.__value)  
    else:  
        return "" + str(self.__value)
```

09

16

Concatenação de string

- + é o operador de concatenação de string em Python
- Exemplos:
- 'POO' + 'com Python' -> 'POOcom Python'
- 'Resposta: ' + str(42) -> 'Resposta: 42'
- return "0" + str(self.__value)
- return "" + str(self.__value)

Objetos criando objetos (1)

```
class ClockDisplay:
    """
    __hours - classe NumberDisplay
    __minutes - classe NumberDisplay
    __displayString - string
    """
    def __init__(self):

        self.__hours = NumberDisplay(24)
        self.__minutes= NumberDisplay(60)
        self.__updateDisplay()
```

Chamadas de método

```
def timeTick(self):  
  
    self.__minutes.increment()  
    if(self.__minutes.getValue() == 0)  
        # acaba de voltar a zero!  
        self.__hours.increment()  
  
    self.__updateDisplay()
```

Método interno

```
"""
```

```
    Atualiza a string interna que  
    representa o mostrador.
```

```
"""
```

```
def __updateDisplay(self):
```

```
    self.__displayString = self.__hours.getDisplayValue() + ":"  
        + self.__minutes.getDisplayValue()
```

16:40

Objetos criando objetos (2)

Na classe Endereco:

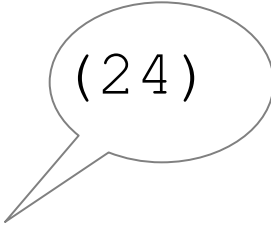
```
def __init__(self, num)
```



parâmetro formal

Na classe Estudante:

```
numeroe = Endereco (24)
```



parâmetro real

Múltiplos construtores

- Em Python, uma classe só tem um método construtor.
- Mas é possível determinar o valor **None** para os parâmetros opcionais e dentro do corpo da função verificar se eles foram passados ou não
- ```
def __init__(self, arg0=None, arg1=None):
```



# Múltiplos construtores

- Também é possível determinar um valor default para os parâmetros opcionais e utilizar outro valor sobrepondo o valor default
- `def __init__(self, arg0=param1, arg1):`

# Chamadas de método (1)

Chamada de método externo

`objeto.nomeDoMetodo( lista-de-parâmetros )`

Chamada de método interno

`nomeDoMétodo ( lista-de-parâmetros )`

# Chamadas de método (2)

- Chamadas de método interno

`__updateDisplay()`

...

## Chamadas de método externo

`self.__minutes.increment()`

# A palavra-chave **self**

**Sobrecarga de nome** ocorre quando um mesmo nome é utilizado para duas entidades diferentes

Por exemplo, atributos e parâmetros de entrada com o mesmo nome

Um parâmetro e um atributo que compartilham um nome não é um problema em Python

A palavra-chave **self** é utilizada para fazer a distinção

A expressão **self** referencia o objeto atual

# A palavra-chave `self`

```
def __init__(self, numero):
```

```
 self.numero = numero
```

```
 self.complemento = 0
```

```
outros métodos
```

# A palavra-chave `self`

```
class Aquecedor :
```

```
 def __init__(self, xmax, xmin) :
```

```
 self.xmax = xmax
```

```
 self.xmin = xmin
```

```
 self.incr = 5.0
```

```
 self.temperatura = 15.0
```

```
outros métodos
```

# Resumo dos conceitos (1)

- **abstração** Abstração é a capacidade de ignorar detalhes de partes para focalizar a atenção em um nível mais elevado de um problema.
- **modularização** A modularização é o processo de dividir um todo em partes bem definidas, que podem ser construídas e examinadas separadamente e que interagem de maneiras bem definidas.
- **classes definem tipos** Um nome de classe pode ser utilizado como o tipo para uma variável. Variáveis que têm uma classe como seu tipo podem referenciar objetos dessa classe

# Resumo dos conceitos (2)

- **diagrama de classes** O diagrama de classes mostra as classes de uma aplicação e os relacionamentos entre elas.
- **diagrama de objetos** O diagrama de objetos mostra os objetos e seus relacionamentos em tempo de execução.
- **referências de objeto** Variáveis do tipo objeto armazenam referências para objetos.
- **criação de objetos** Os objetos podem criar outros objetos.



# Resumo dos conceitos (3)

- **tipo primitivo** Os tipos primitivos em Python são os tipos não-objeto. Os mais comuns são int, float e bool.
- **sobrecarga** Uma classe não pode conter mais de um método com mesmo nome. Use parâmetros opcionais em Python

# Resumo dos conceitos (4)

- **chamada de método interno** Os métodos podem chamar outros métodos da mesma classe como parte de sua implementação.
- **chamada de método externo** Os métodos podem chamar métodos de outros objetos utilizando notação de ponto.