Herança Múltipla

Programação Orientada a Objetos

Python

Prof. Diógenes Furlan

Sumário

- Herança múltipla
- Algoritmo do Diamante
- Problemas na herança múltipla
- Interfaces

Herança Múltipla

- Um objeto **pode conter** “MUITOS” outros por composição.
- Na “herança múltipla” um objeto **quer ser** “muitos” outros.

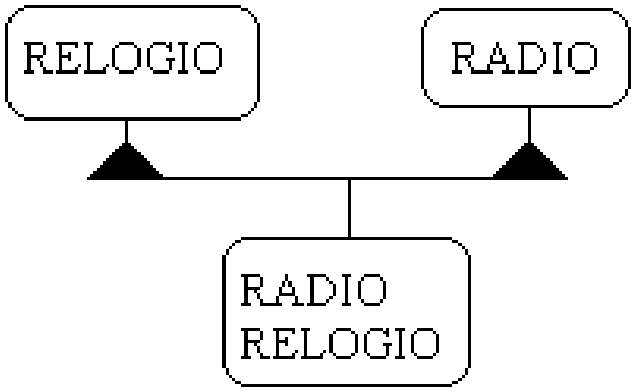
Herança Múltipla

- Uma classe pode “herdar” as características de mais de uma classe-base.

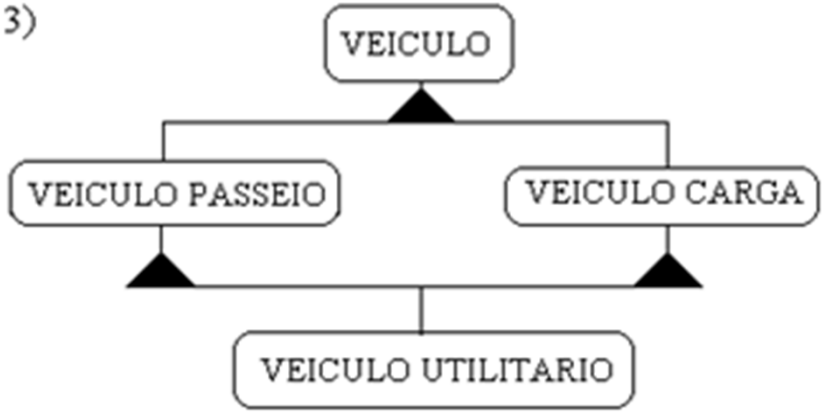
```
class X:
    pass

class Y
    pass

class Z (X,Y):
    pass
```

Exemplos de Herança Múltipla

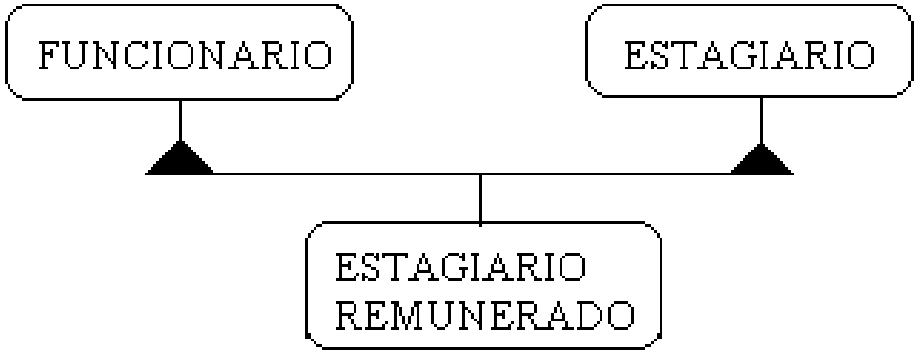
1)



3)



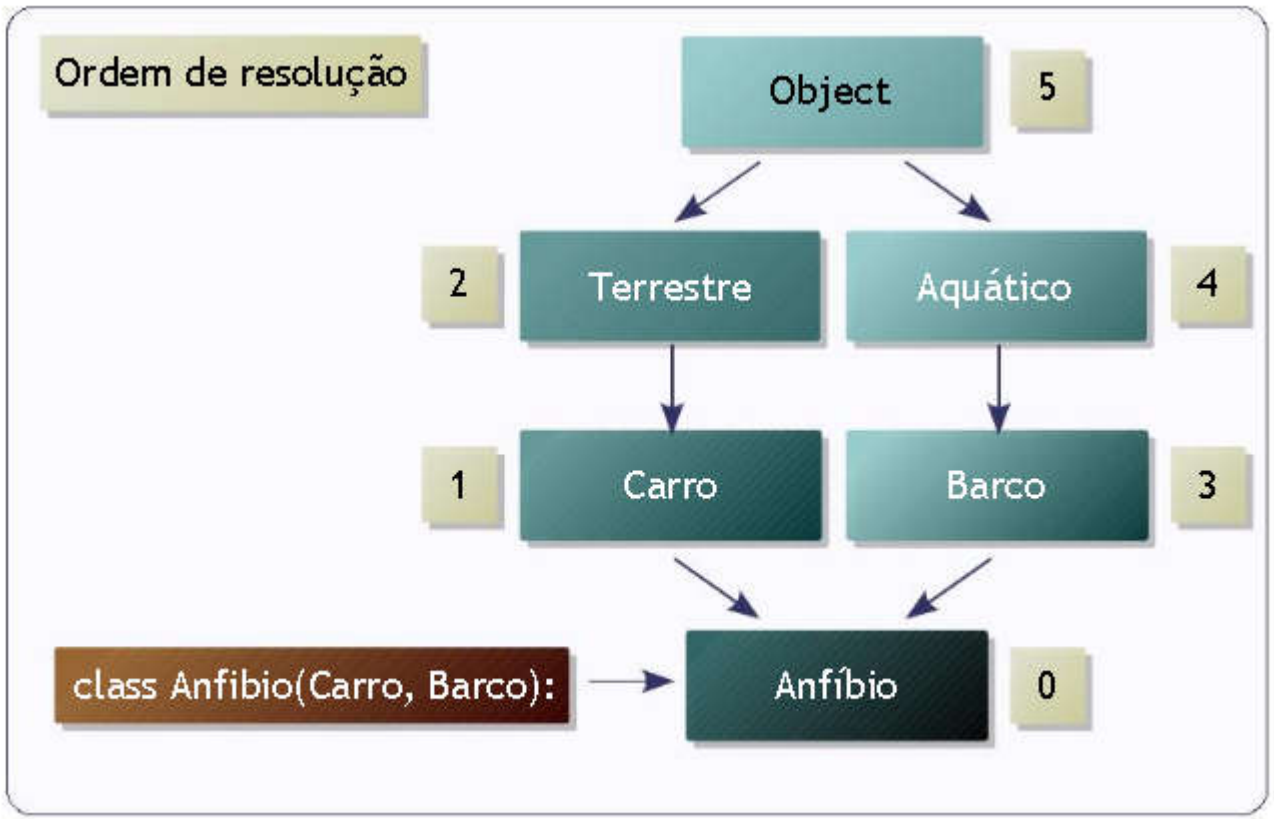
2)



Herança Múltipla

- Na herança múltipla, a nova classe deriva de várias classes já existentes.
- A diferença mais significativa em relação à herança simples é a ordem de resolução de métodos, que segue o chamado algoritmo diamante.
  - (em inglês, Method Resolution Order- MRO)

Algoritmo Diamante



Exemplo 1

```
class Terrestre(object):
    """ Classe de veículos terrestres """

    def __init__(self, velocidade=100):
        self.se_move_em_terra = True
        self.velocidade_em_terra = velocidade

    def __str__(self):
        aux = "\nse_move_em_terra = "
        aux += str(self.se_move_em_terra)
        aux += "\nvelocidade_em_terra = "
        aux += str(self.velocidade_em_terra)
        return aux
```

Exemplo 1

```
class Aquatico(object):
    """ Classe de veículos aquaticos """

    def __init__(self, velocidade=5):
        self.se_move_na_agua = True
        self.velocidade_agua = velocidade

    def __str__(self):
        aux = "\nse_move_na_água = "
        aux += str(self.se_move_na_agua)
        aux += "\nvelocidade_água = "
        aux += str(self.velocidade_agua)
        return aux
```

Exemplo 1

```
# main
t1 = Terrestre(110)
a1 = Aquatico(30)
print(t1)
print(a1)
```

```
se_move_em_terra = True
velocidade_em_terra = 110

se_move_na_água = True
velocidade_água = 30
```

Exemplo 1

```
class Carro(Terrestre):

    def __init__(self, veloc=120, pistoes=4):
        self.rodas = 4
        self.pistoes = pistoes
        Terrestre.__init__(self, veloc)

    def __str__(self):
        aux = "\nrodas = " + str(self.rodas)
        aux += "\npistões = " + str(self.pistoes)
        aux += super().__str__()
        return aux
```

Exemplo 1

```
class Barco(Aquatico):

    def __init__(self, veloc=6, helices=1):
        self.helices = helices
        Aquatico.__init__(self, veloc)

    def __str__(self):
        aux = "\nhélices = " + str(self.helices)
        aux += super().__str__()
        return aux
```

Exemplo 1

```
# main
c1 = Carro(130,6)
b1 = Barco(35,3)
print(c1)
print(b1)
```

```
rodas = 4
pistões = 6
se_move_em_terra = True
velocidade_em_terra = 130

hélices = 3
se_move_na_água = True
velocidade_água = 35
```

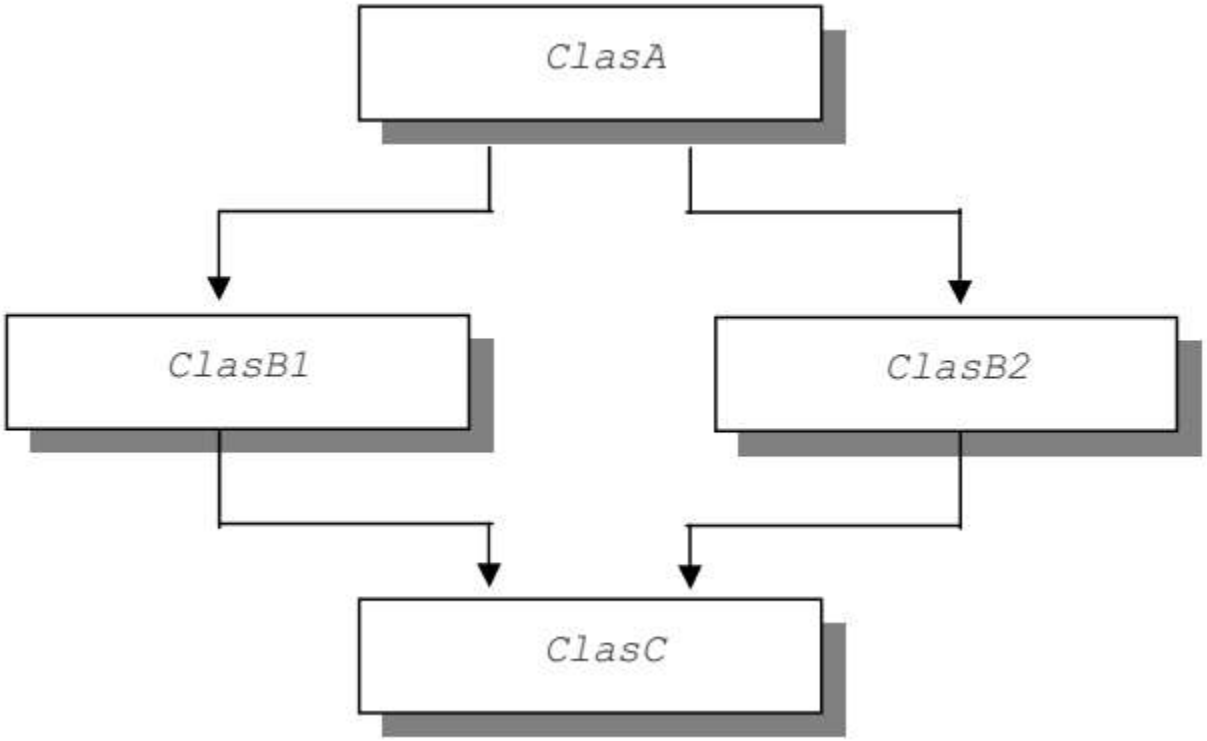
Exemplo 1

```
class Anfibio(Carro, Barco):

    def __init__(self, vt=80, va=4, pi=6, he=2):
        # É preciso evocar o __init__ de cada classe pai
        Carro.__init__(self, veloc=vt, pistoes=pi)
        Barco.__init__(self, veloc=va, helices=he)

    def __str__(self):
        aux = Carro.__str__(self)
        aux += Barco.__str__(self)
        return aux

# main
anf1 = Anfibio()
print(anf1)
```

Exemplo 1

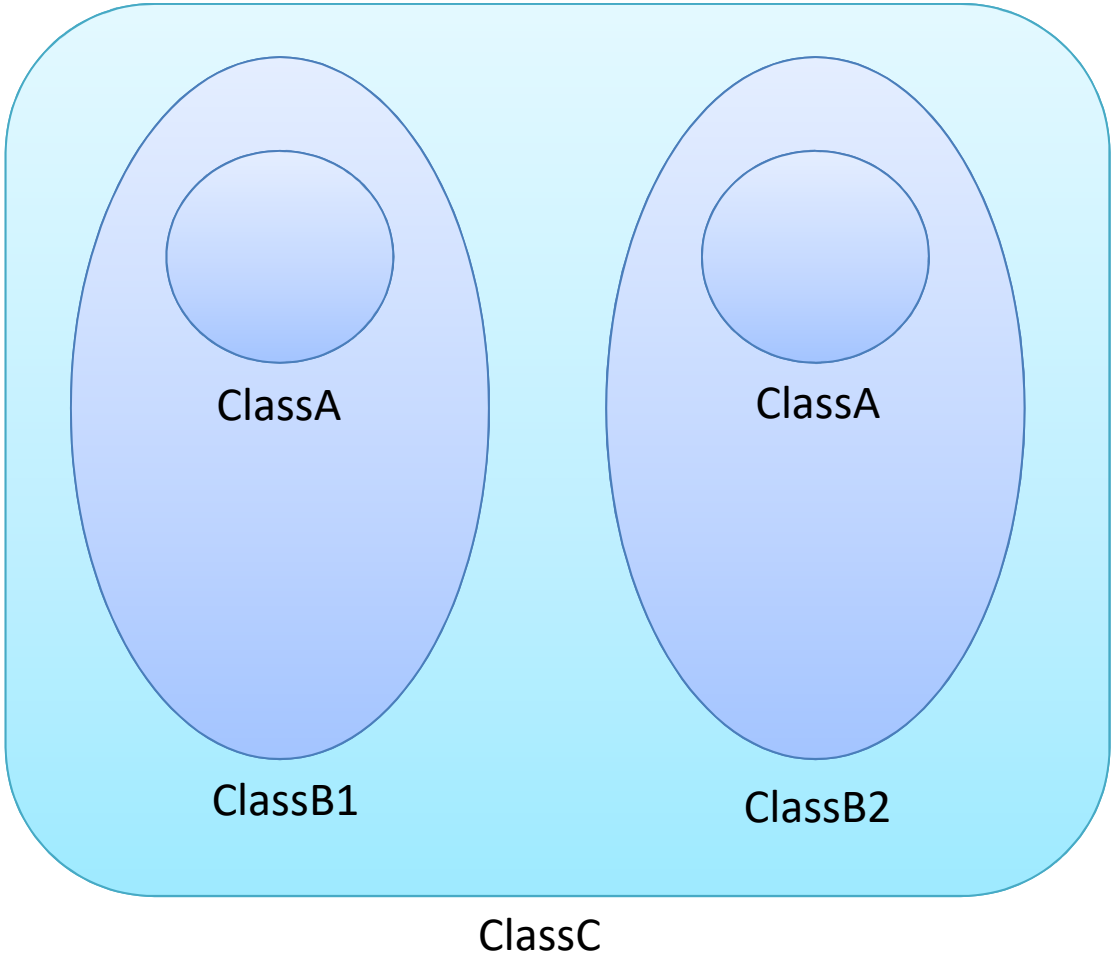
```
rodas = 4
pistões = 6
se_move_em_terra = True
velocidade_em_terra = 80
hélices = 2
se_move_na_água = True
velocidade_água = 4
```

Exemplo 2

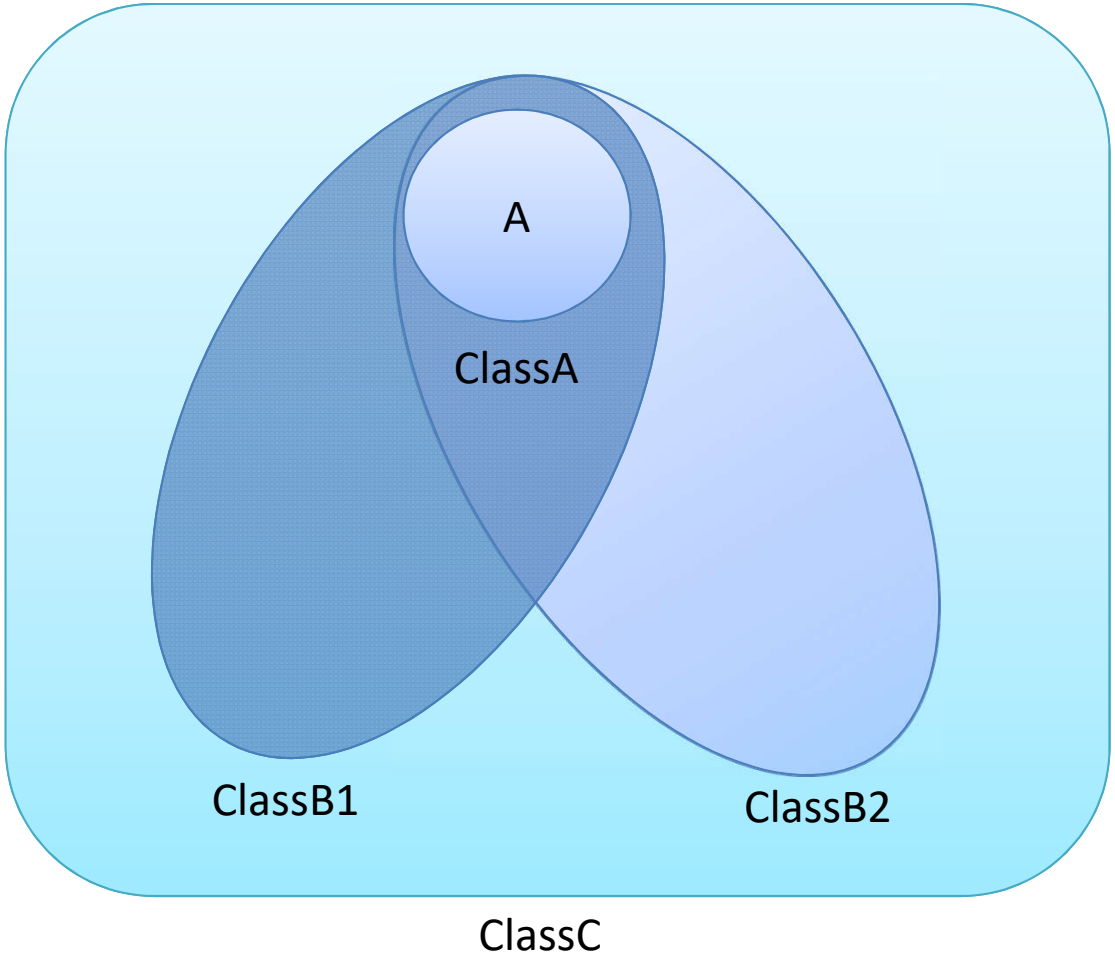


FONTE: Manssour I. H. – Herança Múltipla

Problema na HM



Solução



Exemplo 2

```
class ClasseA:
    id = 'A'
    def __init__(self):
        self.nome = "Classe A"

    def __str__(self):
        return f'Nome da classe: {self.nome}'
```

Exemplo 2

```
class ClasseB1(ClasseA):
    id = 'B1'
    def alterar(self):
        self.nome = "Classe B1"

    def pai(self):
        return 'O pai da classe ' + str(type(self)) + ' é ' + super().id

class ClasseB2(ClasseA):
    id = 'B2'
    def alterar(self):
        self.nome = "Classe B2"

    def pai(self):
        return 'O pai da classe ' + str(type(self)) + ' é ' + super().id
```

Exemplo 2

```
class ClasseC(ClasseB1,ClasseB2):
    id = 'C'

    def pai(self):
        return 'O pai da classe ' + str(type(self)) + ' é ' + super().id

# main
c = ClasseC()
print( c )
c.alterar()
print( c )
```

MRO:
[<class '__main__.ClasseC'>,
 <class '__main__.ClasseB1'>,
 <class '__main__.ClasseB2'>,
 <class '__main__.ClasseA'>,
 <class 'object'>]

Nome da classe: Classe A
Nome da classe: Classe B1

Exemplo 2

```
# main
c = ClasseC()
b1 = ClasseB1()
b2 = ClasseB2()
print( c.pai() )
print( b1.pai() )
print( b2.pai() )
```

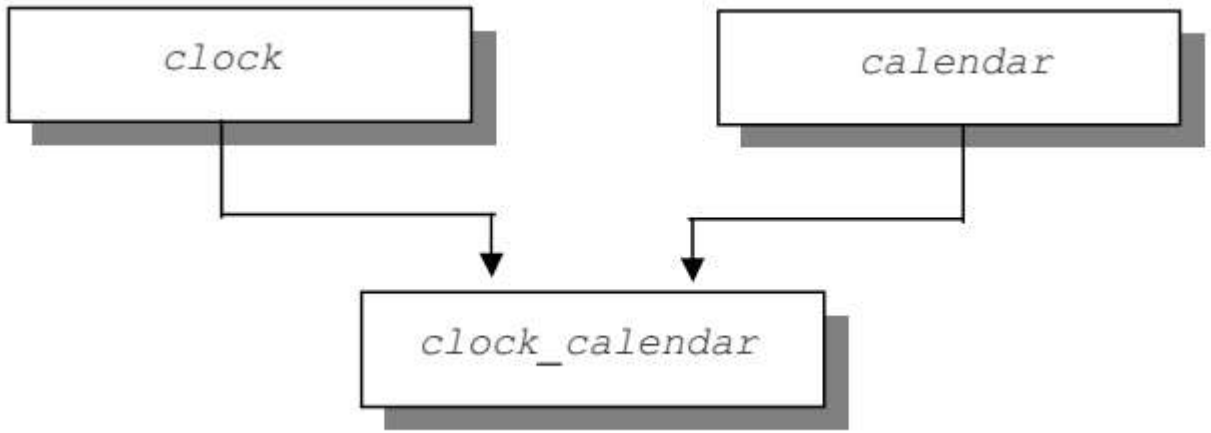
MRO:
[<class '__main__.ClasseC'>,
 <class '__main__.ClasseB1'>,
 <class '__main__.ClasseB2'>,
 <class '__main__.ClasseA'>,
 <class 'object'>]

O pai da classe <class '__main__.ClasseC'> é B1
O pai da classe <class '__main__.ClasseB1'> é A
O pai da classe <class '__main__.ClasseB2'> é A

Apague o método pai() em ClasseC
Depois apague o método pai() em ClasseB1

EXERCÍCIO 1

Exercício 1



Exercício 1

- Classe CLOCK:
  - as variáveis "hora", "min" e "seg" guardam, respectivamente, as horas, os minutos e os segundos.
  - A variável "is_pm" é false para A.M. e true para P.M.
  - O construtor "clock()" cria um novo relógio e determina a hora de acordo com os parâmetros recebidos;
  - "set_clock()" seta a hora corrente para a hora passada como parâmetro;
  - "read_clock()" retorna a hora corrente através de seus parâmetros de referência;
  - "advance()“ avança o relógio em um segundo.

Exercício 1

- Classe CALENDAR:
  - As variáveis “dia", “mês" e “ano" guardam a data, isto é, dia, mês, e ano, respectivamente.
  - O construtor "calendar()" cria um calendário e determina a data de acordo com os parâmetros recebidos;
  - "set_calendar()" seta um calendário existente para a data passada como parâmetro;
  - "read_calendar()" retorna a data corrente através de seus parâmetros de referência;
  - "advance()" avança a data em um dia.

Exercício 1

- Classe CLOCK_CALENDAR:
  - Para inicializar o objeto "clock_calendar", o construtor "clock_calendar()" precisa invocar os construtores de "clock" e "calendar".
- As funções membro
  - "set_clock()“
  - "read_clock()“
  - "set_calendar()“
  - "read_calendar()“
- são todas herdadas pela classe "clock_calendar" e trabalham exatamente da mesma maneira tanto para os objetos de "clock_calendar" como para "clock" ou "calendar".

Exercício 1

- Entretanto, todas as três classes possuem uma função "advance()";
  - A versão de "advance()" declarada em "clock_calendar" sobrescreve aquela herdada de "clock()" e "calendar()".
  - Entretanto, as funções herdadas "clock.advance()" e "calendar.advance()" podem ser usadas na definição de "clock_calendar.advance()“.