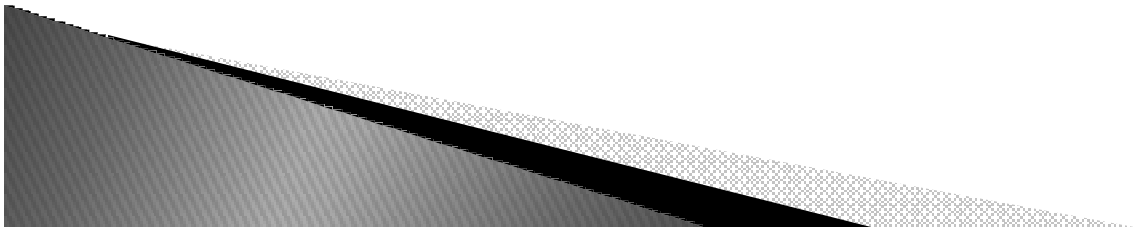
Introdução a Orientação a Objetos

Gustavo Callou
gcallou@gmail.com

Tópicos

- Conceitos de Orientação a Objetos
  - Objeto
  - Classe
  - Herança
  - Composição
  - Polimorfismo



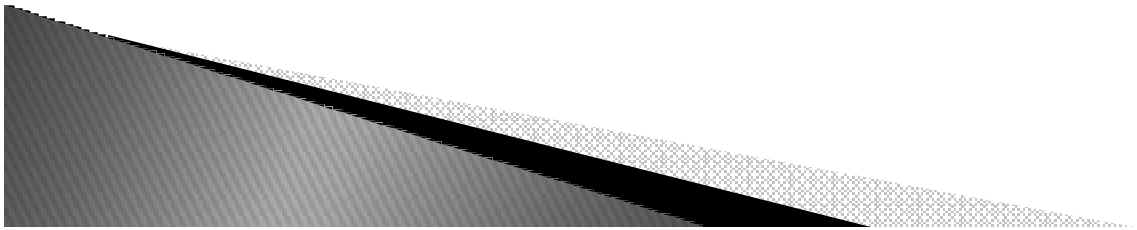
O paradigma da Orientação a Objetos

- *Um paradigma é uma forma de abordar um problema*.
- O paradigma da orientação a objetos surgiu no fim dos anos 60.
- Hoje em dia, praticamente suplantou o paradigma anterior, o *paradigma estruturado*...

O paradigma da Orientação a Objetos

- Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada **analogia biológica.**

- "Como seria um sistema de software que funcionasse como um ser vivo?



Analogia Biológica

- Cada “célula” interagiria com outras células através do envio de mensagens para realizar um objetivo comum.

- Adicionalmente, cada célula se comportaria como uma unidade autônoma.

Analogia Biológica

- De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si.

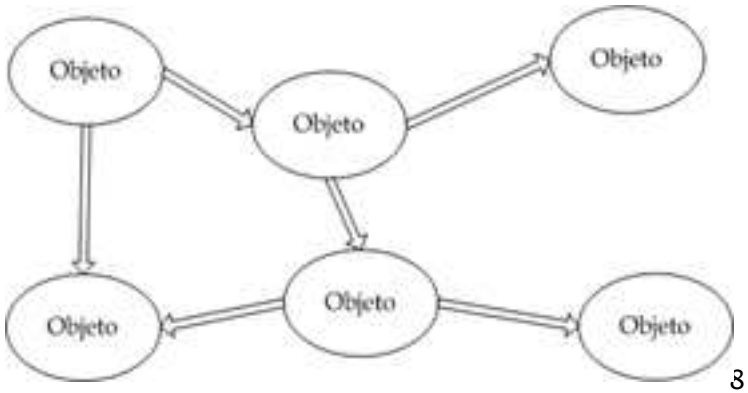
- Com isso, ele estabeleceu os princípios da orientação a objetos.

Orientação a Objetos – Princípios

- Tudo é um objeto.
- Pense em um objeto como uma super variável:
  - ele armazena dados, mas você também pode fazer requisições a esse objeto, pedindo que ele faça operações sobre si próprio.
- Em teoria, você pode representar qualquer elemento conceitual no problema que você está tentando resolver (cachorros, livros, sócios, empréstimos, etc.) como um objeto no seu programa.

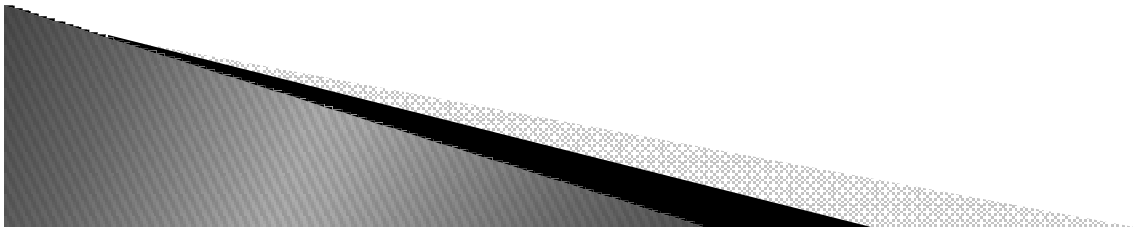
Orientação a Objetos – Princípios

- Um programa é uma coleção de objetos dizendo uns aos outros o que fazer.
- Para fazer uma requisição a um objeto você "manda uma mensagem" para este objeto. Mais concretamente, você pode pensar em uma mensagem como sendo uma chamada de um procedimento ou função pertencente a um objeto em particular.



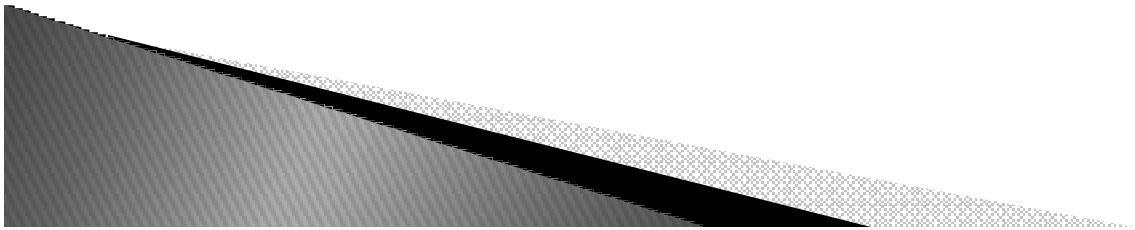
Orientação a Objetos – Princípios

- Um objeto pode ser composto por vários outros objetos
- Em outras palavras: você pode criar um novo tipo de objeto empacotando objetos existentes. Dessa forma, você pode adicionar complexidade a um programa e escondê-la por trás da simplicidade de uso dos objetos.



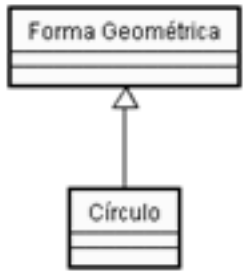
Orientação a Objetos – Princípios

- Todo objeto tem um tipo.
- Usando as palavras certas, cada objeto é uma instância de uma classe, onde classe é um sinônimo de tipo.
- A questão mais importante relativa a uma classe é “que mensagens eu posso enviar para uma instância dessa classe?”



Orientação a Objetos – Princípios

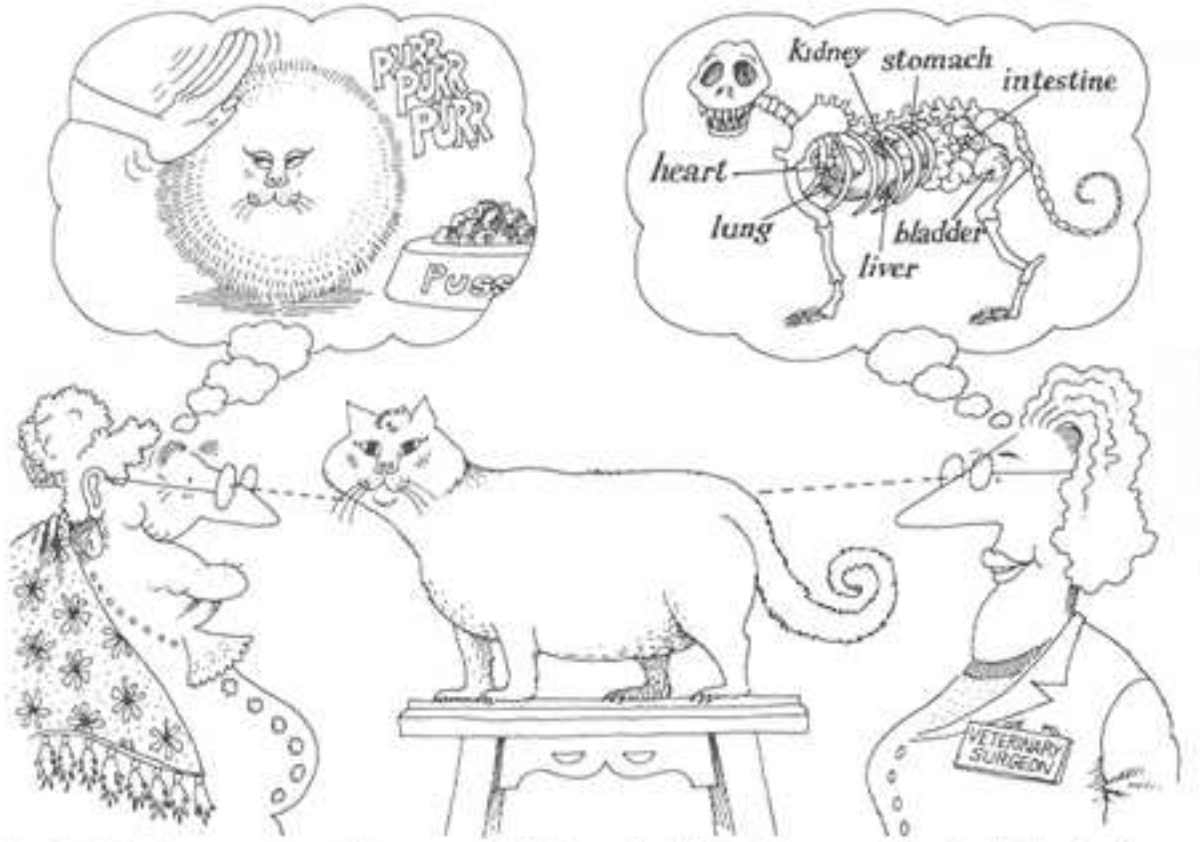
- Todos os objetos de um dado tipo podem receber as mesmas mensagens.
- Além disso, uma vez que, por exemplo, um objeto do tipo “círculo” é também um objeto do tipo “forma geométrica”, o objeto “círculo” aceita qualquer mensagem endereçada a uma “forma geométrica”. Essa capacidade de “subtituição” de um objeto por outro é um dos mais poderosos conceitos em orientação a objetos.



Objeto

- Definição:
  - Um objeto é qualquer coisa, real ou abstrata, sobre a qual armazenamos dados e realizamos operações que manipulam tais dados.
  - Unidade básica de modularização do sistema na obordagem OO.
  - Um objeto é composto por:
    - Atributos→ características ou propriedades que definem o objeto.
    - Comportamento→ conjunto de ações pré-definidas (métodos).

Abstração

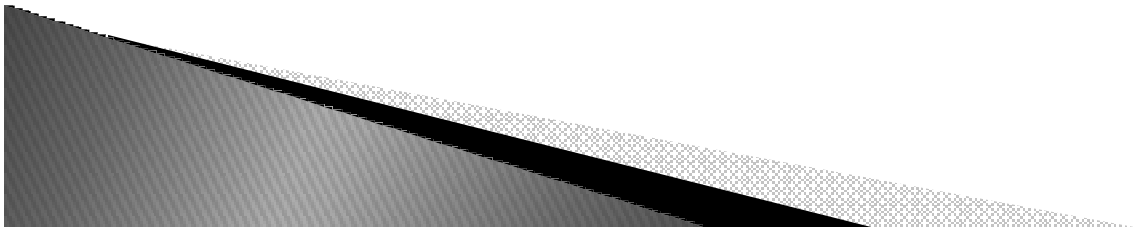


Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Abstração

- É o mecanismo que nos permite representar uma realidade complexa em termos de um modelo simplificado, de modo que detalhes irrelevantes possam ser suprimidos.

- Processo de filtragem de detalhes sem importância do objeto, para que apenas as características apropriadas que o descrevem permaneçam.



Abstração

- Três abstrações de um carro

Officina	Consumidor	Detran
Placa, conserto, pagamento, etc	Consumo (Km/l), Manutenção, Conforto	Identificação, Impostos, Placa,

Abstração

- Para processar algo do mundo real em um computador, temos que extrair as características essenciais.
- Esses dados que caracterizam o objeto são utilizados na representação no sistema.
- Um mesmo objeto, pode ser visualizado de formas distintas. Exemplo: Carro para a oficina, para o detran, para o consumidor.

Objetos – Exemplos

- Pássaro

Identidade: beija-flor

Características:

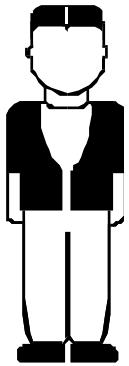
cores
forma do bico
tipo do vôo

Comportamento:

voar
piar

Exemplo

- Pessoa



Identidade: ‘Mário’

Características:

olhos pretos
nasceu em 16/02/70
pesa 70kg
mede 1,70m

Comportamento:

andar
falar
comer
rir

Exemplo

- Telefone

Identidade:: número 2576-0989

Características:

azul
2.4 GHz
tone

Comportamento:

tocar
discar

Exemplo

- Ônibus

Identidade: placa LXY 7684

Características:

cor amarela
30 assentos
a diesel

Comportamento:

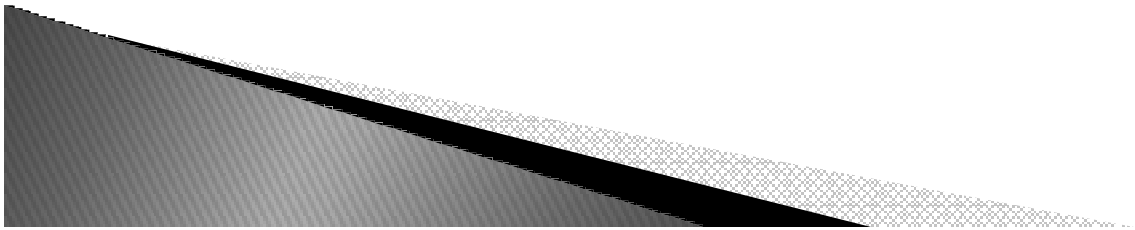
frear
andar
correr
buzinar
acelerar

Exercício

- Defina um computador PC segundo os princípios de Orientação a Objetos:

OO

- A expressão orientada a objetos→ significa que o aplicativo é organizado como uma coleção de objetos que incorporam tanto a estrutura como o comportamento dos dados.



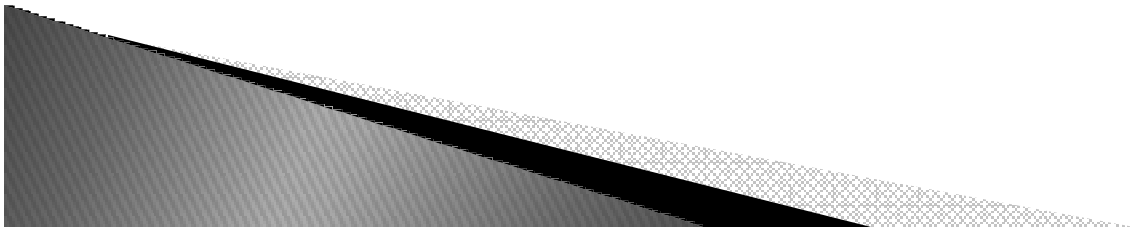
Sistema de Controle de Pizzarias

- Sistema que informatiza os pedidos de pizza em um restaurante.

- Objetos:
  - Pedido,
  - Cardápio,
  - Pizza,
  - Caixa,
  - Cliente,
  - Garçom,
  - Cozinheiro,
  - etc

Sistema de Controle de Pizzarias

- Cardápio → armazenar os preços e mantê-los atualizados.
- Pedido → processamento dos pedidos feitos pelos clientes.
- Caixa → computa a conta a ser paga pelo Cliente.

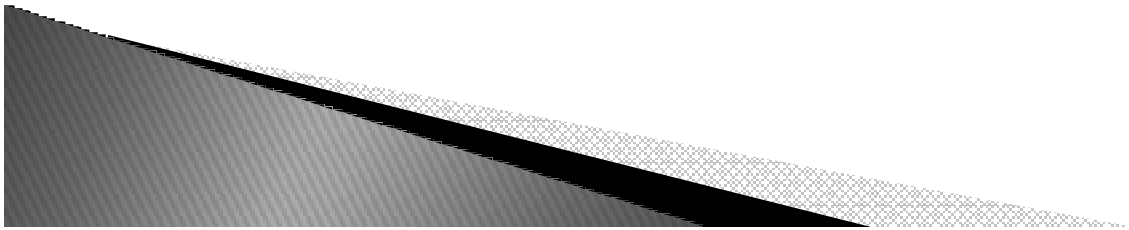


Sistema de Controle de Pizzarias

- Utilidade:
- Exemplo1:
  - Caso houvesse alteração no sistema para atender a necessidade de atualização de preços→ seria a responsabilidade do cardápio.
  - Assim, os demais objetos não sofreriam alteração
- Exemplo 2:
  - Caso a forma de calcular a conta fosse modificada (exemplo: gorjeta), o caixa seria refeito.
- OBS: Cada objeto tem a sua respectiva função

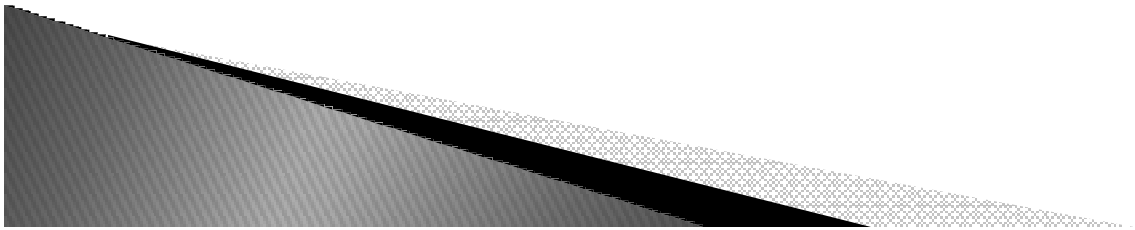
Classe

- Definição:
  - Abstrações utilizadas para representar um conjunto de objetos com c*aracterísticas e comportamento idênticos*
- Uma classe pode ser vista como uma “fábrica de objetos”



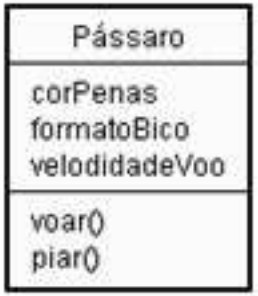
Classe

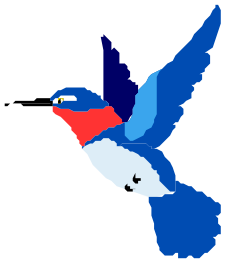
- Tecnicamente falando, objetos são "instâncias" em tempo de execução de uma classe
  - Todos os objetos são instâncias de alguma classe
  - Todos os objetos de uma classe são idênticos no que diz respeito a sua interface e implementação (*o que difere um objeto de outro é seu estado e sua identidade*)



Classe – Exemplo

- classe





Identidade::o beija-flor que vem ao meu jardim

Características:

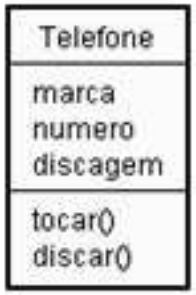
cor das penas: azuis
formato do bico: fino
velocidade de vôo: rápida

instância da classe (objeto)

Comportamento:

voar
piar

Classe – Exemplo



classe

Identidade: 'Telefone da minha casa'

Características:

marca: Siemens
número: 2576-0989
discagem: pulso

instância da classe (objeto)

Comportamento:

tocar
discar

Classe – Exemplo

classe

Identidade: 'Meu celular'

Características:

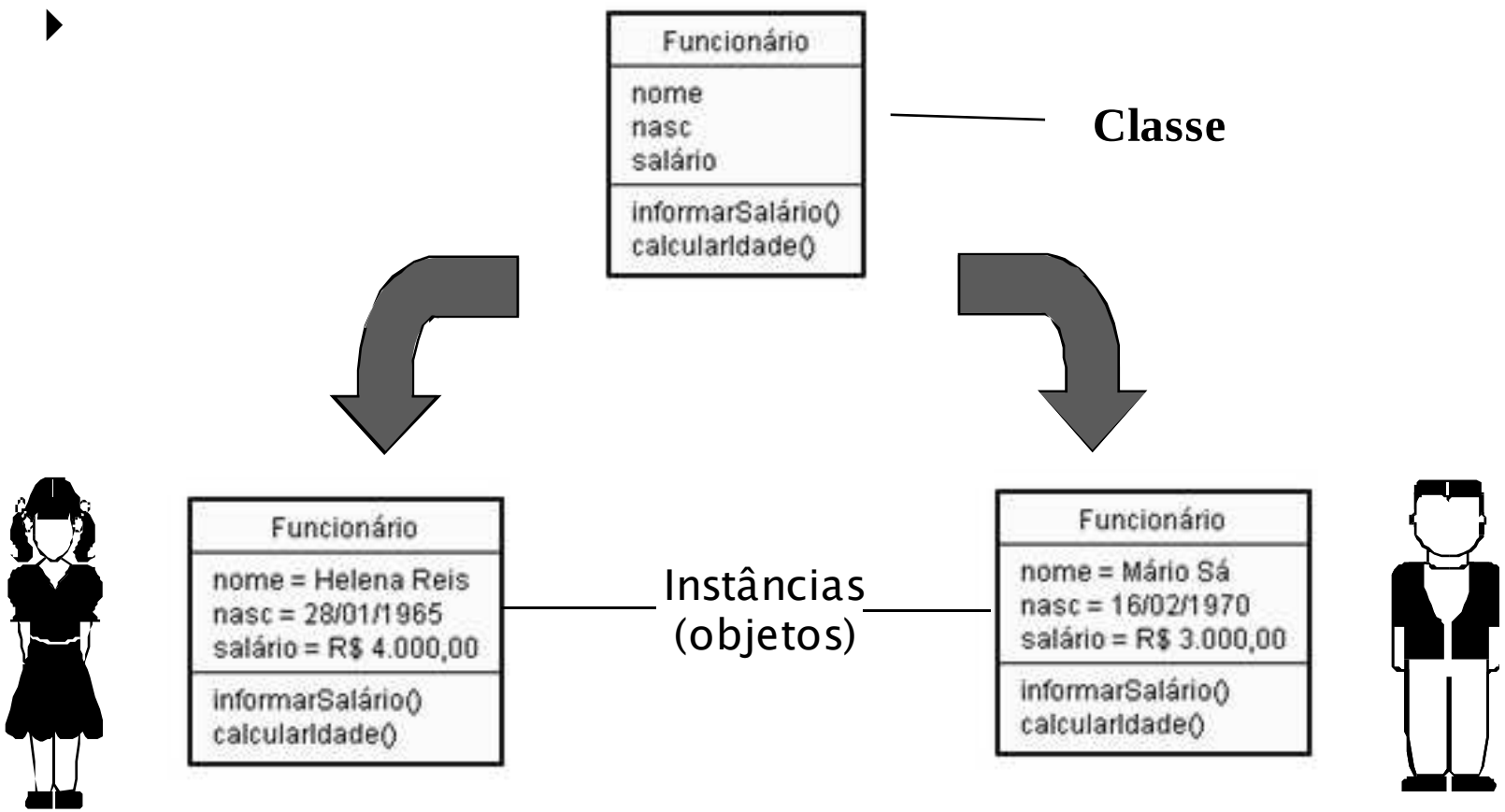
marca: Nokia
número: 99193467
discagem: tom

instância da classe (objeto)

Comportamento:

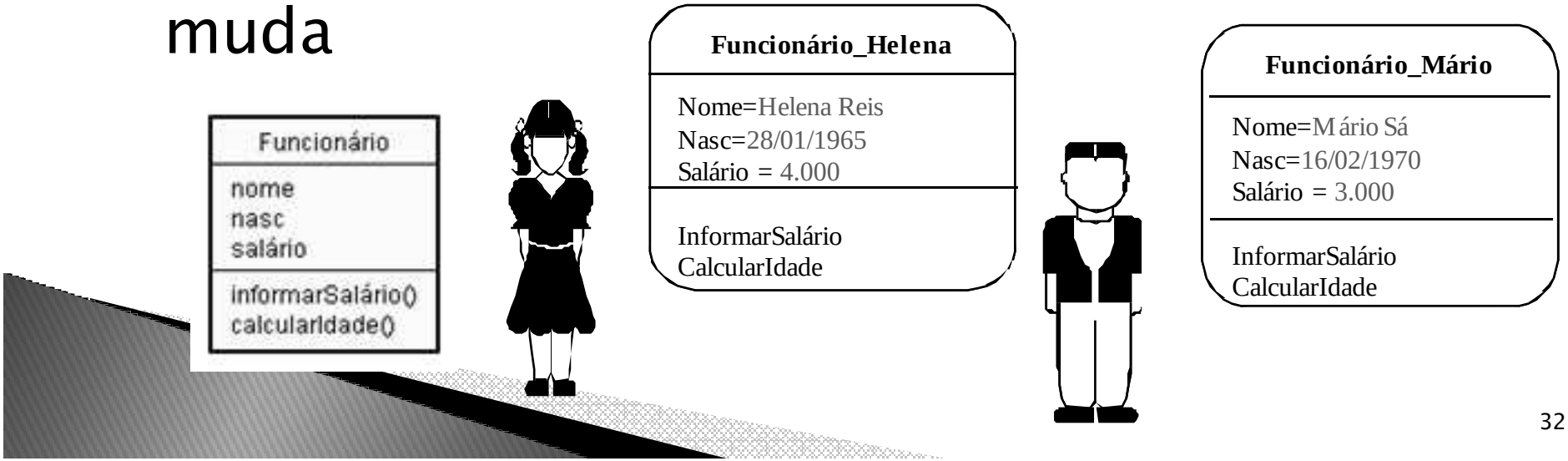
tocar
discar

Classes


Funcionário
nome
nasc
salário
informarSalário()
calcularIdade()
Classe
Funcionário
nome = Helena Reis
nasc = 28/01/1965
salário = R$ 4.000,00
informarSalário()
calcularIdade()
Instâncias
(objetos)
Funcionário
nome = Mário Sá
nasc = 16/02/1970
salário = R$ 3.000,00
informarSalário()
calcularIdade()

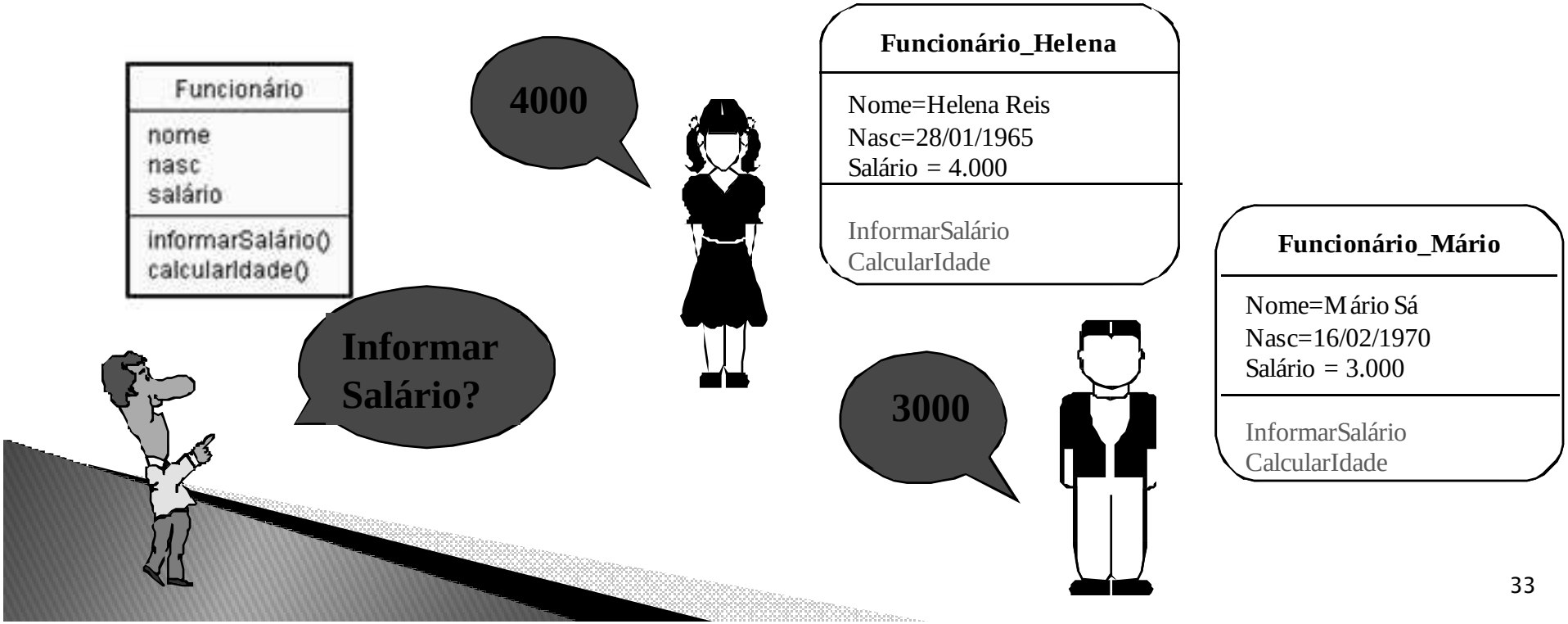
Classe – Atributos

- Descrevem as características das instâncias de uma classe
- Seus valores definem o estado do objeto
- O estado de um objeto pode mudar ao longo de sua existência
- A identidade de um objeto, contudo, nunca muda



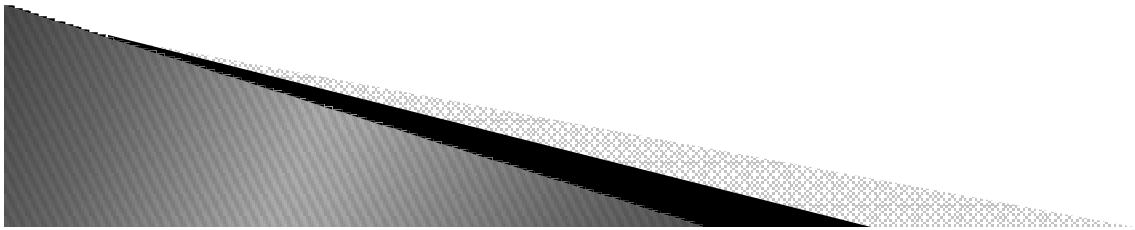
Classe – Serviços/Operações

- Representam o comportamento das instâncias de uma classe
- Correspondem ao protocolo ou ações das instâncias de uma classe



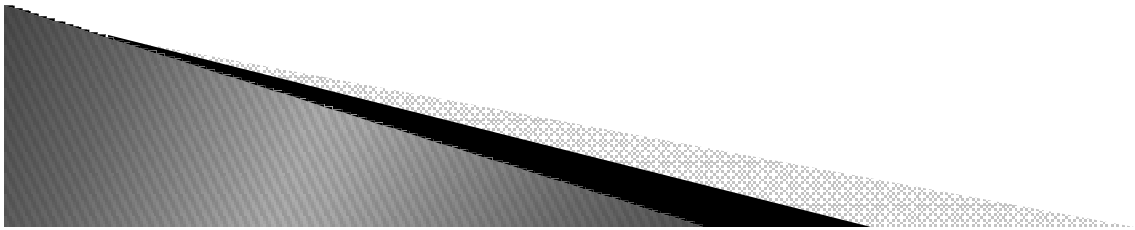
Classe – Sintaxe Básica

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```



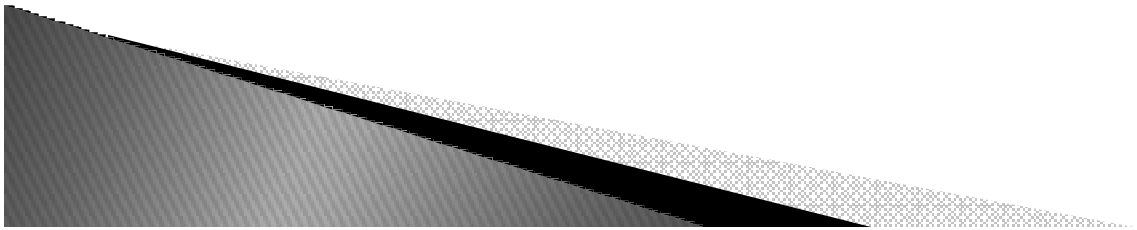
Classe - Exemplo

```
class MyClass:
    '"A simple example class'"
    i = 12345
    def f(self):
        return ('hello world‘)

x = MyClass()
print (x.f())
```



Classe – Exemplo 02

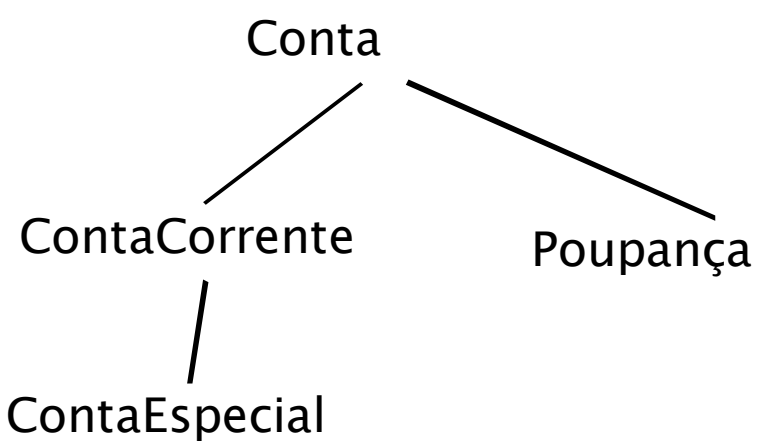
```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

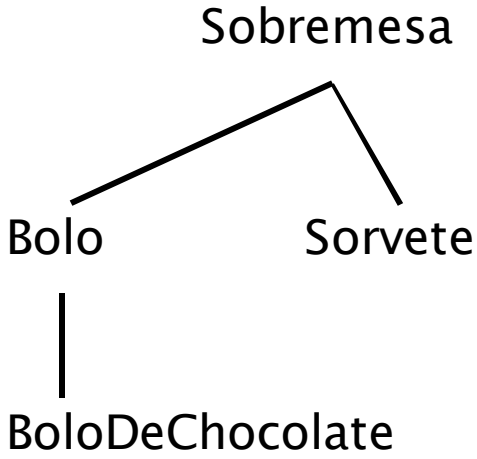


Generalização/Especialização

- Generalização é um processo que ajuda a identificar as classes principais do sistema.
- Ao identificar as partes comuns dos objetos, a generalização ajuda a reduzir as redundâncias, e promove a reutilização.
- O processo inverso a generalização é a especialização. A especialização foca na criação de classes mais individuais.

Generalização/Especialização





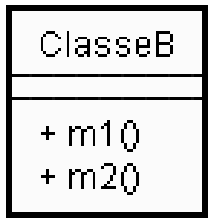
Herança

- O que é herança?
  - Herdar é derivar características de gerações precedentes.
  - No mundo OO, o termo é associado com uma das formas de reutilização de software.
  - Através da herança, novas classes podem ser derivadas das classes existentes.
  - A nova classe herda propriedades e métodos da classe base.
  - A nova classe também pode adicionar suas próprias propriedades e métodos

Herança

- Para que serve a herança?

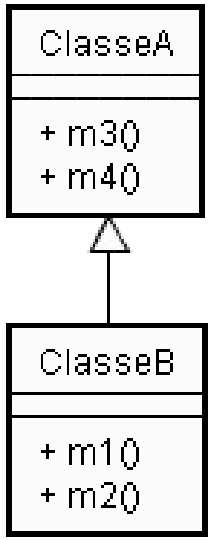
  Que métodos estão disponíveis através de uma referência para a ClasseB (isto é, um objeto)?



Herança

- Suponha agora que a classe ClasseB herda de ClasseA

  Que métodos estão agora disponíveis para uma referência da ClasseB (um objeto) ?



Herança

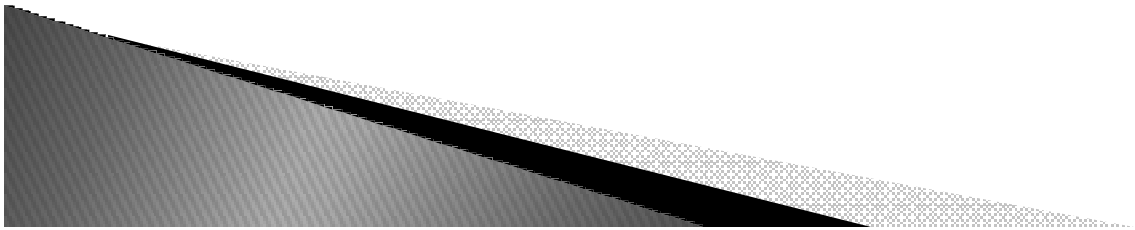
- Poderoso mecanismo para o reaproveitamento de código.
- O objeto objB tem agora disponíveis os métodos da ClasseA sem ser necessário reescrevê-los na ClasseB.

Herança

Um objeto da ClasseB também é um objeto da ClasseA.

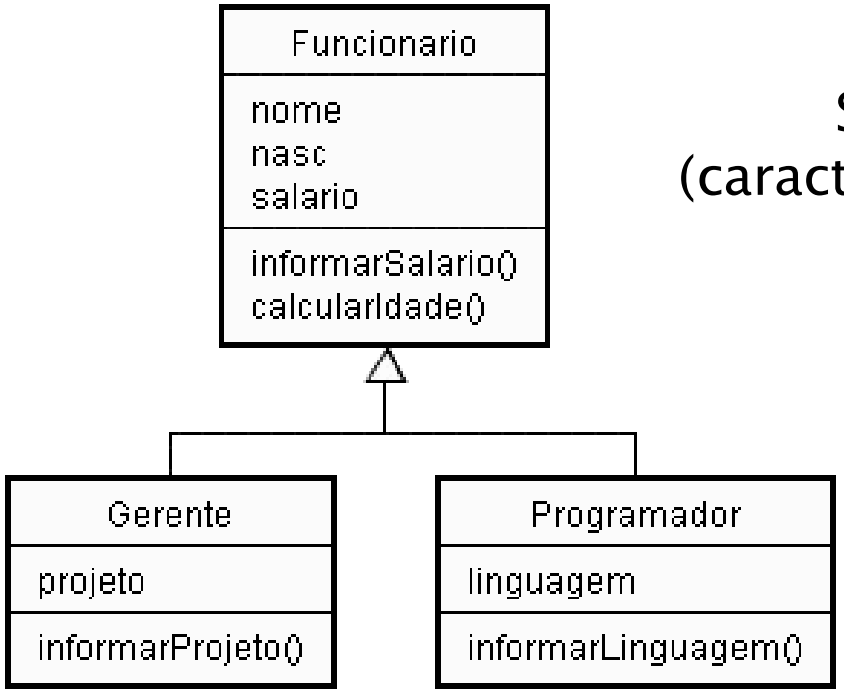
Facilita a manutenção do código: Os métodos não são replicados. Se for necessário alterar o código do método m3, basta alterá-lo em ClasseA.

ClasseB pode "recusar" parte da herança reimplementando os métodos herdados



Herança

- *Exemplo:*

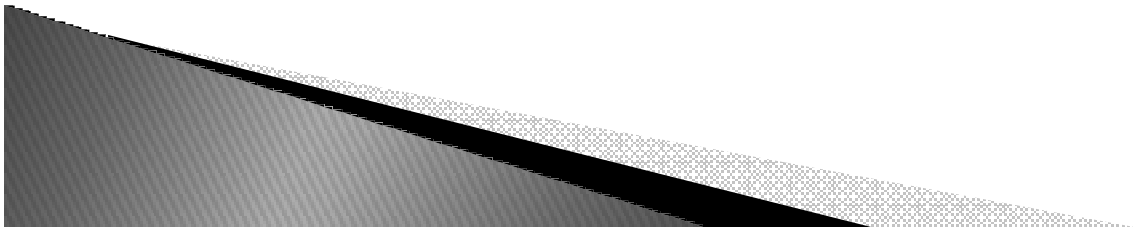


Superclasse
(características comuns)

Subclasses
(características
específicas)

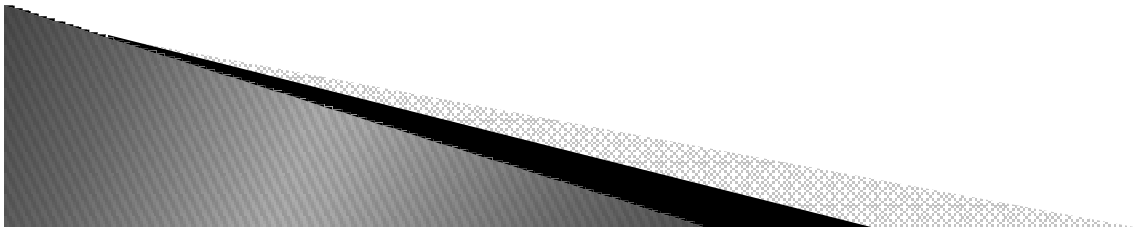
Herança – Sintaxe Básica

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```



Herança Multipla – Sintaxe Básica

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```



Encapsulamento

- Na terminologia da orientação a objetos, diz-se que um objeto possui uma *interface*.
- A interface de um objeto é o que ele conhece e o que ele sabe fazer, sem descrever *como* o objeto conhece ou faz.
- A interface de um objeto define os serviços que ele pode realizar e conseqüentemente as mensagens que ele recebe.

Encapsulamento

- Encapsulamento é a proteção dos atributos ou métodos de uma classe.
- Em Python existem somente o public e o private e eles são definidos no próprio nome do atributo ou método.
- Atributos ou métodos iniciados por no máximo dois sublinhados e terminados por um sublinhado são privados e todas as outras formas são públicas.

Encapsulamento - Exemplo

```
class A:
  a = 1 # atributo publico
  __b = 2 # atributo privado a class A

class B(A):
  __c = 3 # atributo privado a B

  def __init__(self):
    print self.a
    print self.__c

a = A()
print a.a # imprime 1

b = B()
print b.__b # Erro, pois __b é privado a classe A.
print b.__c # Erro, __c é um atributo privado, somente chamado pela classe.

print b._B__c # Imprime __c = 3, muito pouco utilizada, mas existe.
```

IMPORTANDO OBJETOS (CLASSES,FUNÇÕES, ETC.)

- Um módulo é o arquivo de código fonte propriamente dito (arquivo com extensão .py).
- Um módulo pode conter vários objetos, como classes, funções e variáveis.
- A importação desses objetos por outro módulo pode ser feita de duas maneiras:
  - from nome_modulo import nome_objeto
    - Para acessar o objeto precisamos apenas digitar o nome do objeto.
    - Exemplo: x = nome_objeto()
  - from nome_modulo import * importa todos os objetos do módulo.
    - Para acessar um objeto do módulo precisamos digitar o nome do módulo antes.
    - Ex.: x = nome_modulo.nome_objeto()

Construtor de Classe

- O construtor de uma classe é um método responsável por inicializar uma nova instância dessa classe (criar um objeto).

- Se ele não for definido o Python atribui a essa classe um construtor vazio (usado apenas para criar os objetos).

Exemplo:

```
class Pessoa:
  def __init__(self, nome, idade, genero): # O construtor recebe 3 parâmetros
    self.nome = nome # o atributo nome da instancia que está sendo criada
                     # recebe o valor do parâmetro nome, e assim por
    diante...
    self.idade = idade
    self.genero = genero
```

A classe pessoa tem três atributos (variáveis) que representam o estado de uma pessoa

Construtor de Classe

Para a classe pessoa, a criação de uma nova instância deve ser da seguinte forma:

- x = Pessoa(“Augusta”, 23, “F”)
- y = Pessoa(“Fábio”, 30, “M”)

x é uma instância (objeto) da classe Pessoa que tem nome Augusta, idade 23 e gênero F.

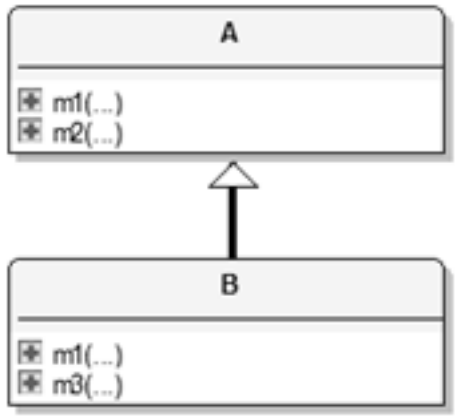
y é uma outra instância (objeto) da classe Pessoa que tem nome Fábio, idade 30 e gênero M.

Polimorfismo

- Significa que a mesma operação pode ter implementações distintas.
- Exemplo: Uma subclasse pode sobrepor a implementação de uma operação que ela herda de uma superclasse.
- Somente pode ser usado com a herança.

Polimorfismo

- Ocorre quando um classe possui um método com o mesmo nome e assinatura (paramêtros) de um método da superclasse.



Classe A	
operação	método invocado
m1()	A.m1()
m2()	A.m2()

Classe B	
operação	método invocado
m1()	B.m1()
m2()	A.m2()
m3()	B.m3()

Polimorfismo – Exemplo

```
#a.py

class A:
  a = 1
  b = 2
  def calcular(self):
      return (self.a+self.b)
```

```
#b.py

from a import A

class B(A):
  pass
```

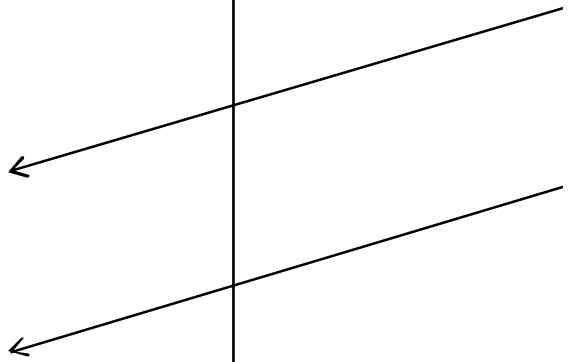
```
#c.py

from a import A
from b import B

a = A()
print (a.calcular()) # saída = ?

b = B()
print (b.calcular()) # saída = ?
```

Polimorfismo – Exemplo

```
#a.py

class A:
  a = 1
  b = 2
  def calcular(self):
      return (self.a+self.b)
```

```
#b.py

from a import A

class B(A):
  pass
```

```
#c.py

from a import A
from b import B

a = A()
print (a.calcular()) # imprime 3

b = B()
print (b.calcular()) # imprime 3
```

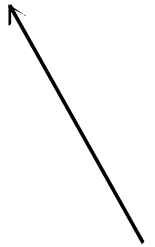


Polimorfismo – Exemplo

```
#a.py

class A:
  a = 1
  b = 2
  def calcular(self):
      return (self.a+self.b)
```

```
#b.py

from a import A

class B(A):
  def calcular(self):
      return (self.a*self.b)
```

```
#c.py

from a import A
from b import B

a = A()
print (a.calcular()) # saída = ?

b = B()
print (b.calcular()) # saída = ?
```



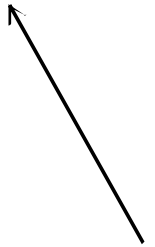
Polimorfismo

Polimorfismo – Exemplo

```
#a.py

class A:
  a = 1
  b = 2
  def calcular(self):
      return (self.a+self.b)
```

```
#b.py

from a import A

class B(A):
  def calcular(self):
      return (self.a*self.b)
```

```
#c.py

from a import A
from b import B

a = A()
print (a.calcular()) # saída = 3

b = B()
print (b.calcular()) # saída = 2
```



Polimorfismo

Exercícios

1)Descreva o que os objetos têm em comum:
a) bicicleta, carro, caminhão, avião, motocicleta
b) prego, parafuso, pino
c) tenda, caverna, barraco, celeiro, casa

2)Identifique classes nos seguintes sistemas:
Obs.: Instancie alguns objetos.
a. Esta sala de aula;
b. Um sistema de transporte urbano;
c. Um ecossistema;
d. Um sistema de estacionamento de veículos
e. Um sistema aéreo

3)Classes possuem propriedade (atributos). Identifique propriedades pertencentes à classe PESSOA nos seguintes sistemas:
a. Um sistema de controle de notas e freqüências ;
b. Um sistema de registro civil;
c. Um sistema de correio.

Exercícios

- 4 – Pensando em um sistema de controle para um banco que possui dois tipos de conta: conta simples (conta corrente) e conta poupança. Defina em termos de OO como podemos representar tal sistema (classes/atributos/métodos).

- 5 – Faça o programa da questão anterior.