Computação II

Departamento de Ciência da Computação – UFRJ

Aulas 2 e 3

Professora: *Fernanda Duarte Vilela Reis de Oliveira*

2019/01

Tópicos

1. Comparação entre Paradigmas
2. Orientação a Objeto:
   a. Definindo uma classe;
   b. Métodos e atributos;
   c. Construtor da classe;
   d. Herança e polimorfismo;
   e. Atributos e métodos privados;
   f. Sobrecarga de operadores.

1. Comparação entre Paradigmas

O paradigma de programação é o estilo que programamos – define a forma como vamos pensar para resolver um problema.

Procedural

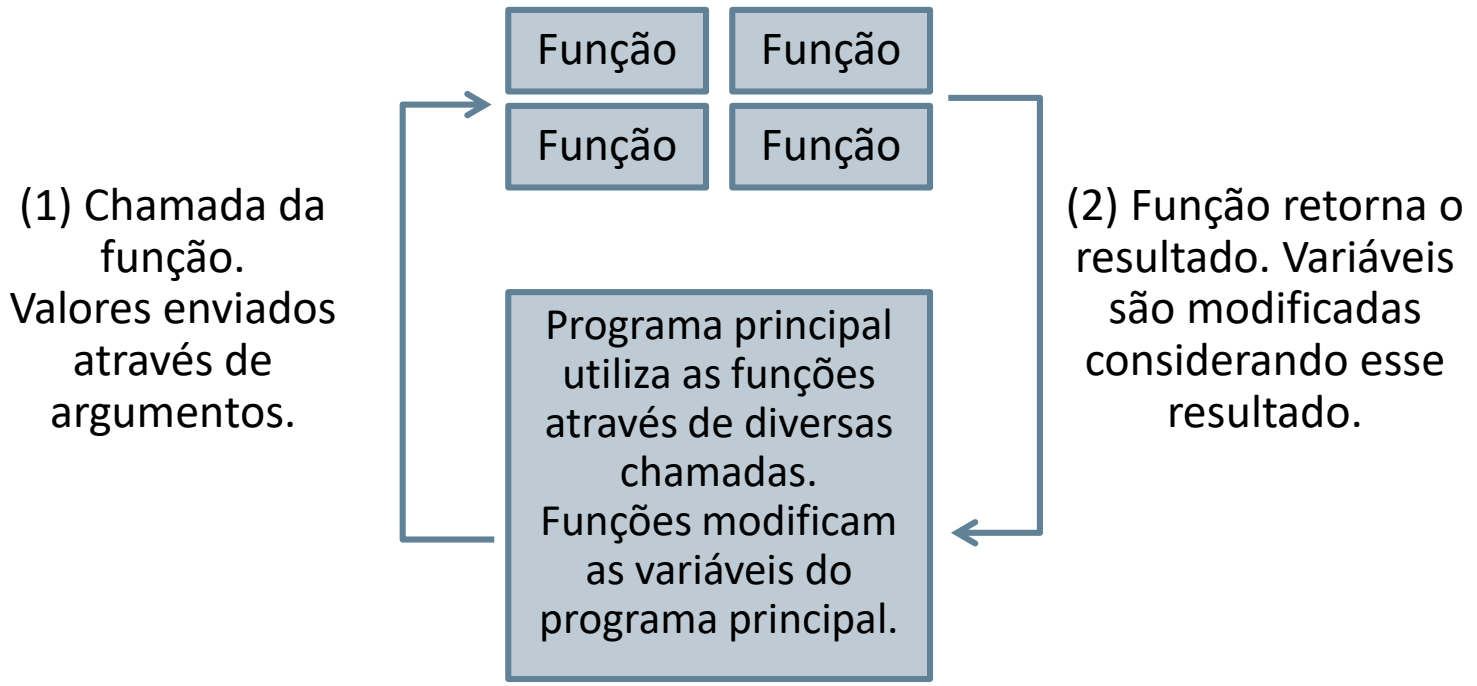
- Unidade de modularização: **rotinas** (funções).
- Programa principal cria variáveis.
- Funções recebem argumentos e retornam resultados;
- Variáveis são modificadas dependendo do valor de retorno das funções.

Orientado a Objeto

- Unidade de modularização: **classes**.
- Programa principal cria instâncias da classe.
- Uma classe possui atributos e métodos.
- Métodos podem modificar instâncias e/ou retornar valores.

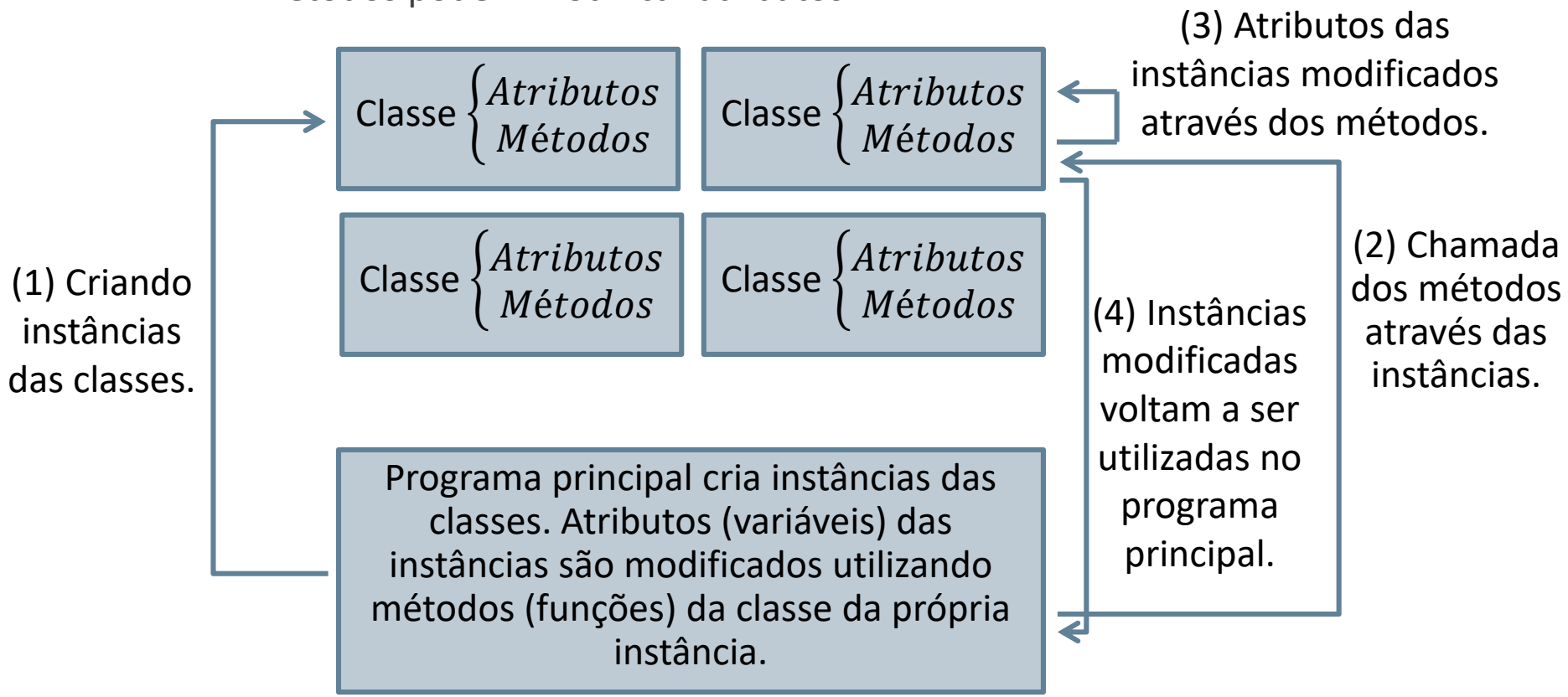
1. Comparação entre Paradigmas

- Paradigma procedural:



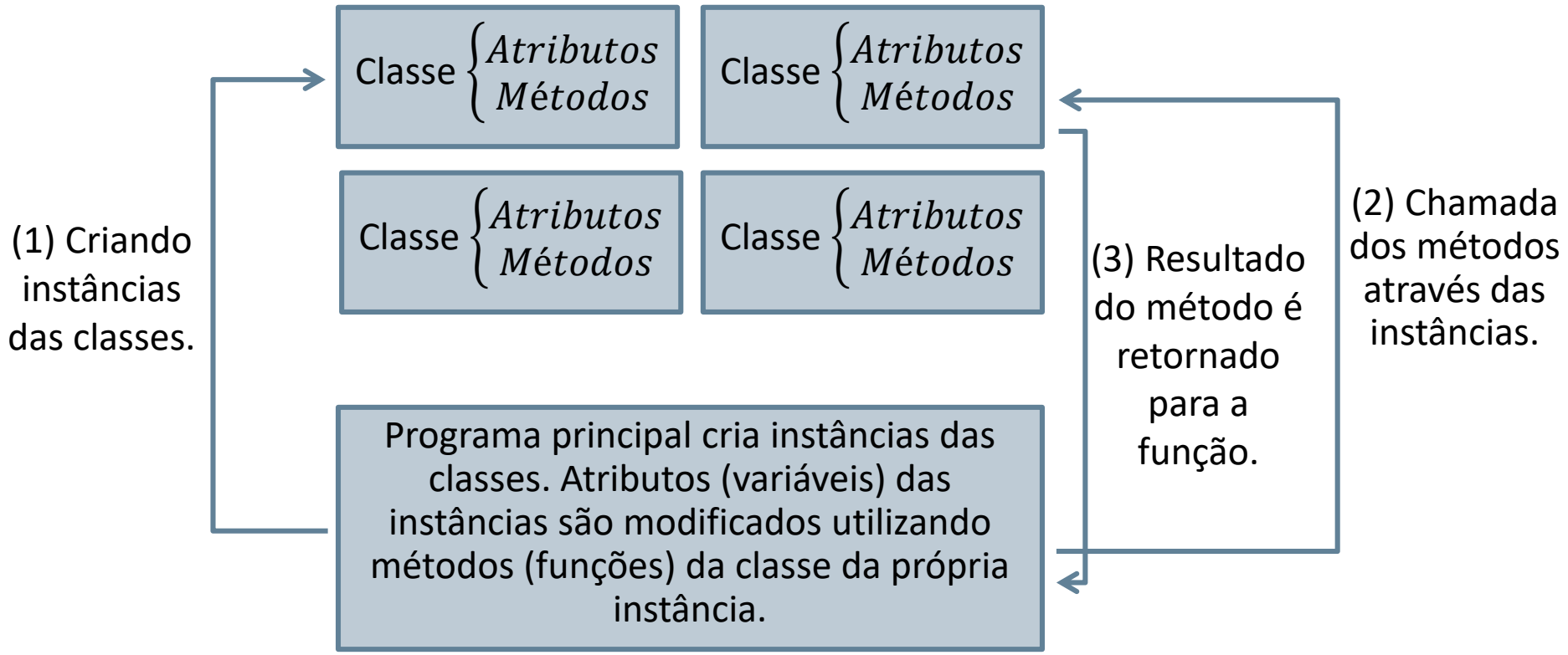
1. Comparação entre Paradigmas

- Paradigma de orientação a objeto:
  - Métodos podem modificar atributos



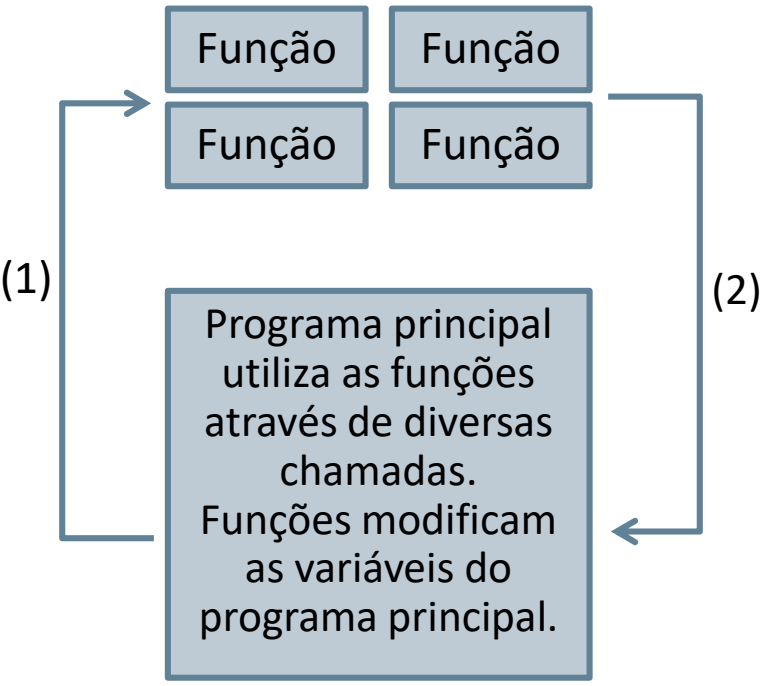
1. Comparação entre Paradigmas

- Paradigma de orientação a objeto:
  - Métodos também podem retornar valores

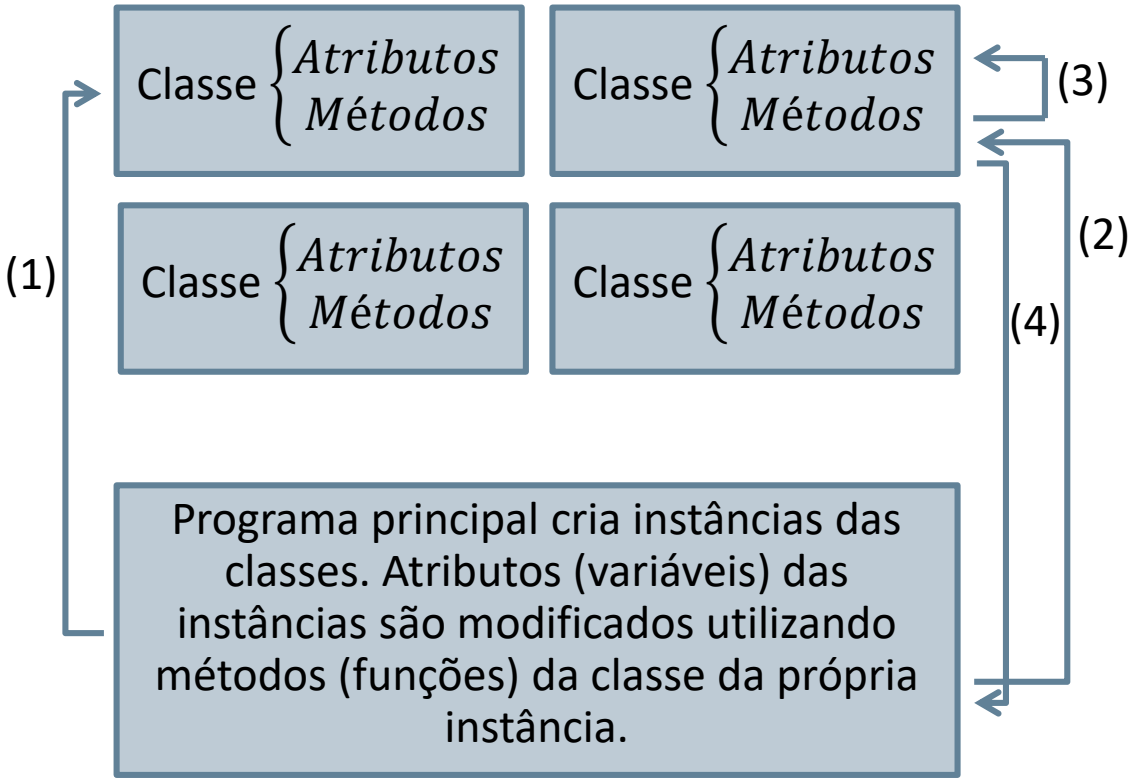


1. Comparação entre Paradigmas

Procedural



Orientado a Objeto



2. Orientação a Objetos

- Quatro conceitos importantes:
  - **Abstração**
  - **Encapsulamento**
  - **Herança**
  - **Polimorfismo**

2. Orientação a Objetos

- Abstração:
  - Abstração é uma representação de uma identidade que contém somente as características mais importantes.
  - Tem como objetivo simplificar a programação, pois permite que o programador mantenha o foco somente em características importantes.
  - A entidade é descrita através de um modelo matemático das suas operações, entradas e saídas
  - Uma instância de um tipo abstrato de dado é um objeto.

Referência: R. Sebesta – Concepts of Programming Languages

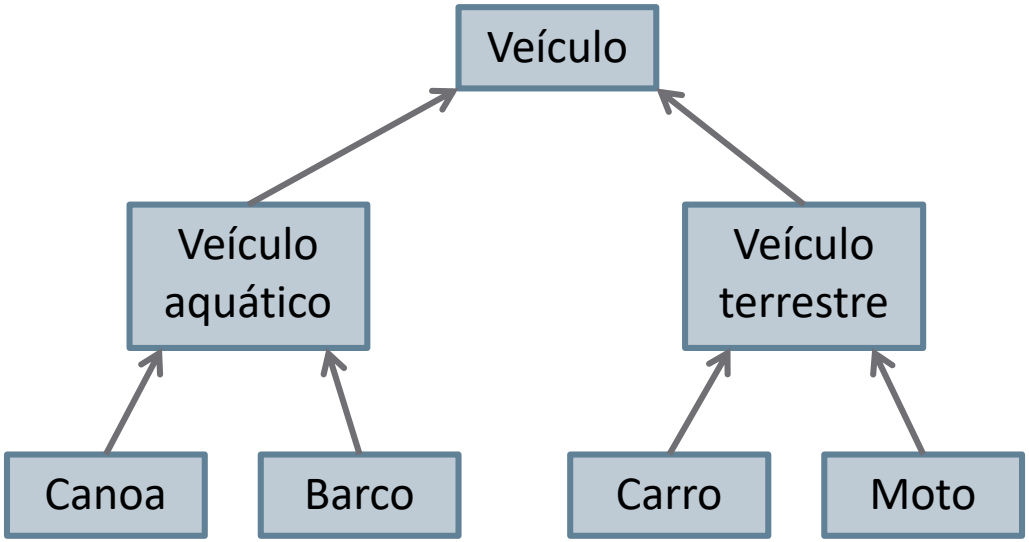
2. Orientação a Objetos

- Encapsulamento:
  - Objetos são entidades que encapsulam dados e operações.
  - Tais objetos são caixa pretas para o usuário – o usuário só tem acesso aos dados através das operações permitidas pelo objeto.
  - Os dados estão escondidos, e dessa forma protegidos, do usuário.

Referência: C. Ghezzi et al – Programming Language Concepts

2. Orientação a Objetos

- Herança:
  - Permite que novas classes sejam criadas a partir de outras.
  - A classe herdeira possui os mesmos atributos e métodos da classe pai.

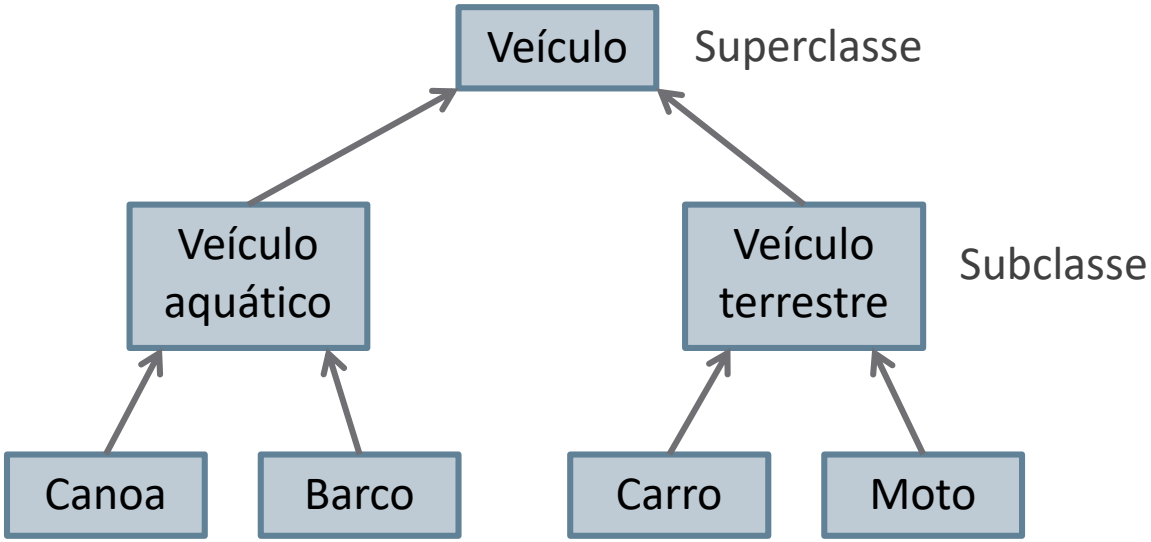


Objeto mais abaixo herda características de todos os objetos acima.

Herança direta do objeto imediatamente acima. **Herança indireta** de todos os demais objetos acima.

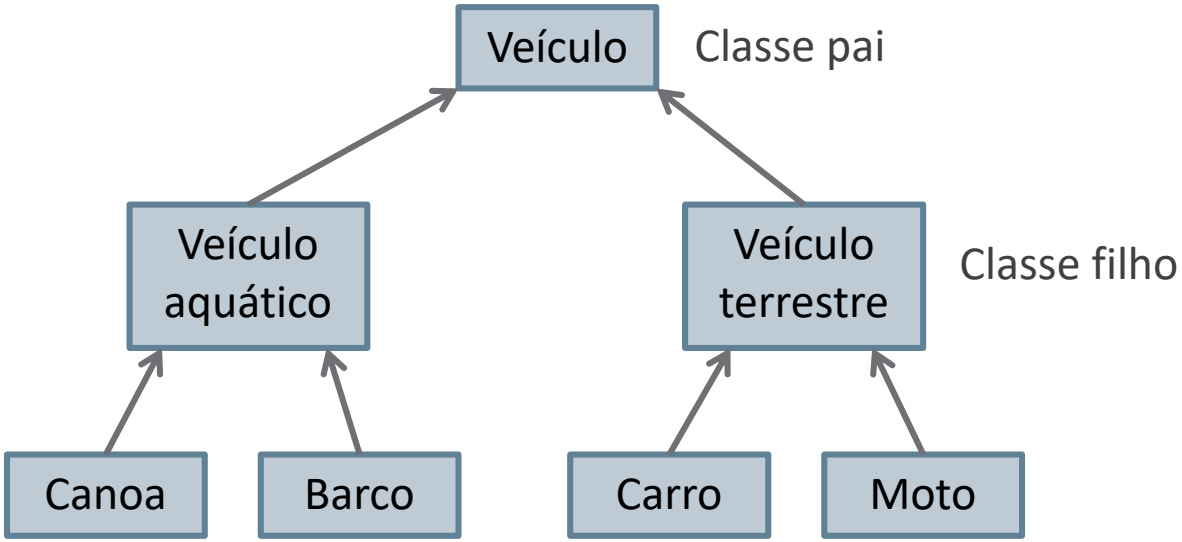
2. Orientação a Objetos

- Herança:
  - Permite que novas classes sejam criadas a partir de outras.
  - A classe herdeira possui os mesmos atributos e métodos da classe pai.



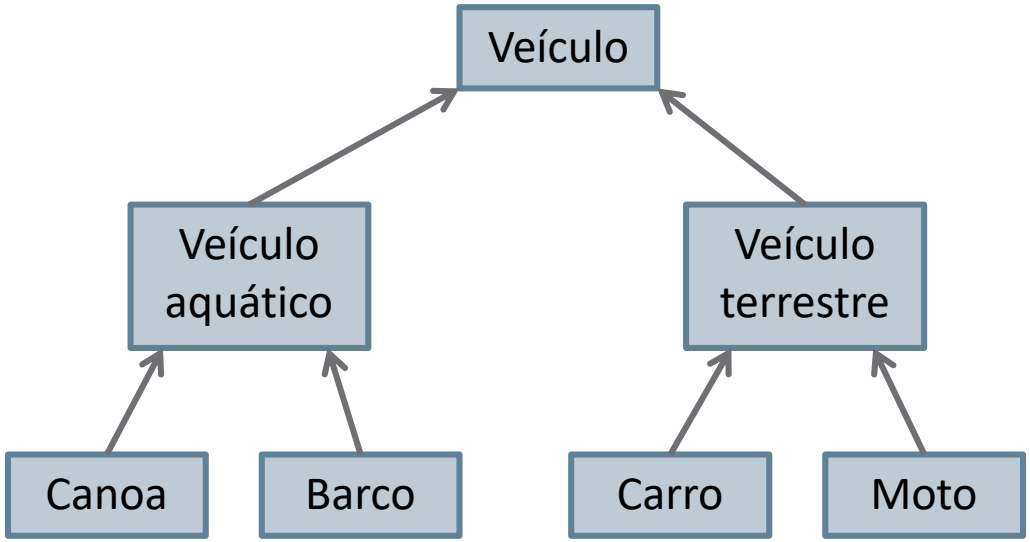
2. Orientação a Objetos

- Herança:
  - Permite que novas classes sejam criadas a partir de outras.
  - A classe herdeira possui os mesmos atributos e métodos da classe pai.



2. Orientação a Objetos

- Herança:
  - Permite que novas classes sejam criadas a partir de outras.
  - A classe herdeira possui os mesmos atributos e métodos da classe pai.

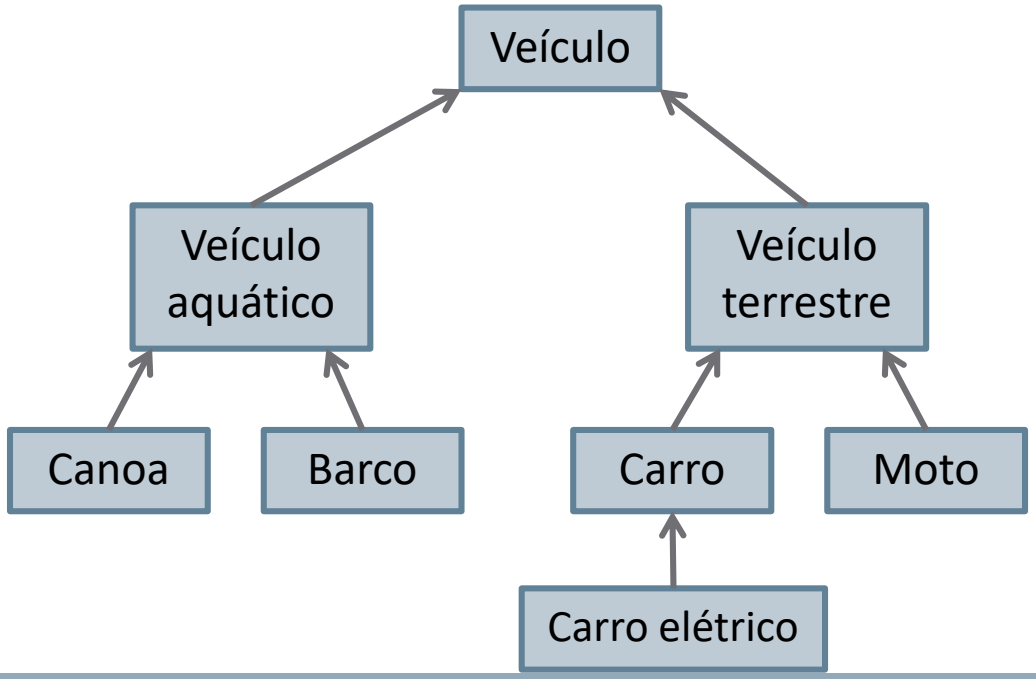


A classe veículo terrestre herda de veículo -> veículo terrestre possui todas as características de veículo.

A classe carro herda de veículo terrestre, que por sua vez herda de veículo -> carro possui todas as características de veículo terrestre e todas as características de veículo.

2. Orientação a Objetos

- Polimorfismo:
  - Permite que classes herdeiras criem novos métodos e atributos.
  - Permite que classes herdeiras modifiquem métodos e atributos herdados – basta sobrescrever o método/atributo.



Carro elétrico herda reabastecer de Carro, mas desejamos modificar esse método, pois Carro precisa de gasolina para reabastecer e Carro elétrico precisa ser carregado.

2. Orientação a Objetos

- Principais vantagens da programação orientada a objetos:

  - Representação mais próxima do mundo real – objeto contém características e ações.

  - Reutilização de código – herança permite que todo o código escrito em uma classe seja aproveitado em outra e polimorfismo permite modificações para adequar a classe ao novo objeto desejado.

2. Orientação a Objetos

- Exemplo:

Objeto: teclado

Atributos:

- Marca
- Categoria
- Número de teclas
- Ligado

Métodos:

- Ligar
- Desligar
- Tocar tecla N
- Tocar teclas N1, N2, ... Nn

2. Orientação a Objetos

- Exemplo:

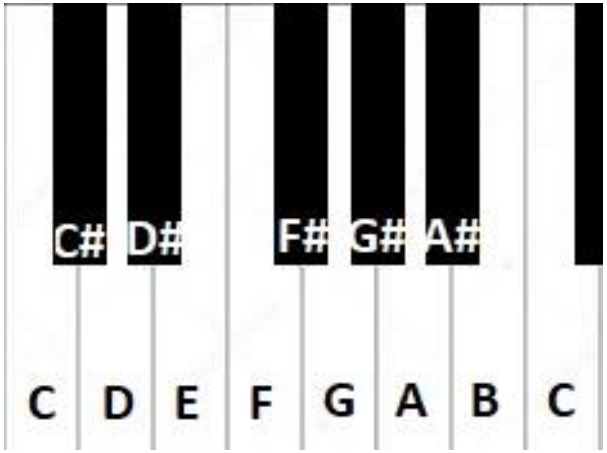
Objeto: teclado

Teclado
Atributos: - Marca - Categoria - Número de teclas - Ligado
Métodos: + Ligar + Desligar + Tocar tecla N + Tocar teclas N1, N2, ... Nn

2. Orientação a Objetos

- Exemplo:

Objeto: teclado



Teclado
Atributos: - Marca - Categoria - Número de teclas - Ligado
Métodos: + Ligar + Desligar + Tocar tecla N + Tocar teclas N1, N2, ... Nn

2. Orientação a Objetos

- Definindo a classe teclado em Python:

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

2. Orientação a Objetos

- Definindo a classe teclado em Python:

Definindo uma classe

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

2. Orientação a Objetos

- Definindo a classe teclado em Python:

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

Atributos

2. Orientação a Objetos

- Definindo a classe teclado em Python:

Métodos

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

2. Orientação a Objetos

- Definindo a classe teclado em Python:

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

Self é uma referência para o objeto atual.

2. Orientação a Objetos

- Definindo a classe teclado em Python:

```
class teclado():
    marca = 'Yamaha'
    modelo = 'PSR E363'
    categoria = 'Arranjador'
    nTeclas = 61
    Ligado = False
    def ligar(self):
        self.Ligado = True
    def desligar(self):
        self.Ligado = False
    def tocarTecla(self,N):
        if self.Ligado:
            if 0<=N<61:
                notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                print(notas[N%len(notas)])
    def tocarTeclas(self,teclas):
        notasTeclas = []
        if self.Ligado:
            for tecla in teclas:
                if 0<=tecla<61:
                    notas = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
                    notasTeclas.append(notas[tecla%len(notas)])
            print(notasTeclas)
```

Ao utilizar self.Ligado estamos dizendo que "Ligado" é um atributo do objeto.

2. Orientação a Objetos

- Atributos da Classe VS Atributos da Instância:
  - O atributo da classe é definido através de uma atribuição, fora de qualquer método.
  - O atributo da classe deve ser utilizado (para leitura e escrita) através de uma referência para a classe (nome_da_classe.atributo).
  - O atributo da instância é definido através de uma atribuição dentro de um método utilizando uma referência para a instância (self.).
  - O atributo da instância deve ser utilizado (para leitura e escrita) através de uma referência para a instância (self.atributo).

2. Orientação a Objetos

- Exemplo:
  - Classe que define um contato:

```
class Contato():
    numeroContatos = 0
    def iniciarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
        Contato.numeroContatos += 1
    def modificarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
    def imprimirContato(self):
        print('Nome: ',self.nome)
        print('Telefone: ',self.telefone)
```

2. Orientação a Objetos

- Exemplo:
  - Classe que define um contato:

```
class Contato():
    numeroContatos = 0
    def iniciarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
        Contato.numeroContatos += 1
    def modificarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
    def imprimirContato(self):
        print('Nome: ',self.nome)
        print('Telefone: ',self.telefone)
```

Atributo da classe

Atributos da instância

2. Orientação a Objetos

- Exemplo:
  - Classe que define um contato:

```
class Contato():
    numeroContatos = 0
    def iniciarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
        Contato.numeroContatos += 1
    def modificarContato(self):
        self.nome = input('Digite o nome do contato: ')
        self.telefone = input('Digite o telefone do contato: ')
    def imprimirContato(self):
        print('Nome: ',self.nome)
        print('Telefone: ',self.telefone)
```

Modificando o atributo da classe

Sempre que iniciarContato for utilizado, o atributo da classe é modificado. Logo, a modificação é feita para todas as instâncias da classe.

2. Orientação a Objetos

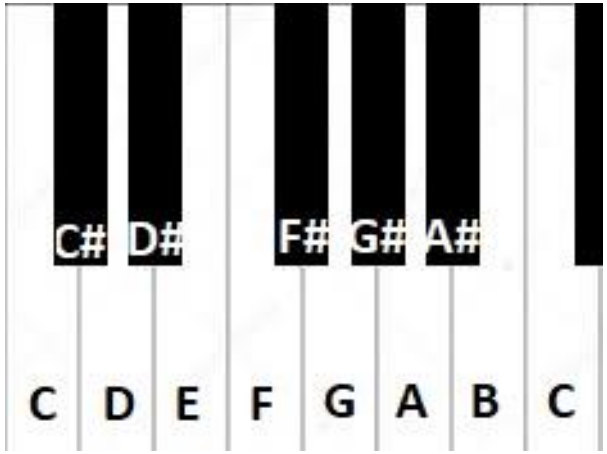
- Exemplo:
  - Faça o seguinte teste com a Classe Contato:

```
>>> c1 = Contato()
>>> c1.iniciarContato()
Digite o nome do contato: Fernanda
Digite o telefone do contato: 973396398
>>> c1.numeroContatos
1
>>> Contato.numeroContatos
1
>>> c2 = Contato()
>>> c2.iniciarContato()
Digite o nome do contato: Pedro
Digite o telefone do contato: 998301780
>>> c1.numeroContatos
2
>>> c2.numeroContatos
2
>>> Contato.numeroContatos
2
```

Veja que numeroContatos é modificado tanto para c1 quanto para c2

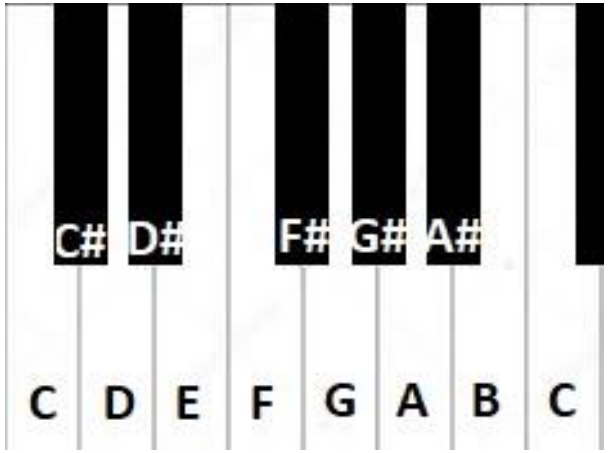
2. Orientação a Objetos

- Conceito de música – Tom e Semitom:
  - Intervalos musicais, a distância entre duas notas.
  - Semitom: distância entre duas teclas adjacentes do teclado - C-C# ; C#-D; D-D#; D#-E; E-F; F-F# ...
  - Tom: dois semitons C-D ; C#-D#; D-E; D#-F; E-F#; F-G ...



2. Orientação a Objetos

- Conceito de música – a Escala Maior:
  - Sequência de notas que obedece à seguinte ordem de intervalos:
    - Tom - Tom - Semitom - Tom - Tom - Tom – Semitom
  - Exemplos:
    - Escala de Dó Maior – C ; D ; E ; F ; G ; A ; B ; C
    - Escala de Ré Maior – D ; E ; F# ; G ; A ; B ; C# ; D
  - É a partir da escala maior que são formados:
    - O acorde maior é formado pela fundamental, terça e quinta da escala maior referente à nota fundamental. Acorde de dó maior – C ; E ; G.



2. Orientação a Objetos

- Exercício:
  - Criar uma classe para ajudar principiantes no estudo da escala maior de música. A classe deve possuir dois métodos:
    - Método que retorna a sequência de sete notas de uma escala maior dada a nota fundamental.
    - Método que retorna as três notas que formam um acorde dada a nota fundamental.

2. Orientação a Objetos

```
class modoMaior():
    notas = ['C','C#',\
             'D','D#',\
             'E','F',\
             'F#','G',\
             'G#','A',\
             'A#','B']
    sequencia = [2,2,1,2,2,2,1]

    def escalaMaior(self,nota):
        escala = [nota]
        posicao = self.notas.index(nota)
        for passo in self.sequencia:
            escala.append(self.notas[posicao+passo-len(self.notas)])
            posicao = posicao + passo
        return escala
    def acordeMaior(self,nota):
        escala = self.escalaMaior(nota)
        return escala[0],escala[2],escala[4]
```

2. Orientação a Objetos

```
class modoMaior():
    notas = ['C','C#',\
             'D','D#',\
             'E','F',\
             'F#','G',\
             'G#','A',\
             'A#','B']
    sequencia = [2,2,1,2,2,2,1]
```

Atributos

Métodos

```
    def escalaMaior(self,nota):
        escala = [nota]
        posicao = self.notas.index(nota)
        for passo in self.sequencia:
            escala.append(self.notas[posicao+passo-len(self.notas)])
            posicao = posicao + passo
        return escala
    def acordeMaior(self,nota):
        escala = self.escalaMaior(nota)
        return escala[0],escala[2],escala[4]
```

2. Orientação a Objetos

```
class modoMaior():
    notas = ['C','C#',\
             'D','D#',\
             'E','F',\
             'F#','G',\
             'G#','A',\
             'A#','B']
    sequencia = [2,2,1,2,2,2,1]

    def escalaMaior(self,nota):
        escala = [nota]
        posicao = self.notas.index(nota)
        for passo in self.sequencia:
            escala.append(self.notas[posicao+passo-len(self.notas)])
            posicao = posicao + passo
        return escala
    def acordeMaior(self,nota):
        escala = self.escalaMaior(nota)
        return escala[0],escala[2],escala[4]
```

Utilização dos atributos dentro dos métodos: sempre através de uma referência ao objeto atual ou de uma referência à classe.

self.nome_do_atributo

nome_da_classe.nome_do_atributo

2. Orientação a Objetos

Definindo uma classe

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome

    def getNome(self):
        return self.nome

    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario

    def getSalario(self):
        return self.salario
```

2. Orientação a Objetos

Métodos (funções) da classe

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome

    def getNome(self):
        return self.nome

    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario

    def getSalario(self):
        return self.salario
```

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome

    def getNome(self):
        return self.nome

    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario

    def getSalario(self):
        return self.salario
```

Métodos sempre recebem self (referência ao objeto da classe) como primeiro argumento

2. Orientação a Objetos

Atributos (variáveis) da classe

```
class trabalhador():
    def   init   (self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome

    def getNome(self):
        return self.nome

    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario

    def getSalario(self):
        return self.salario
```

2. Orientação a Objetos

Construtor classe

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome

    def getNome(self):
        return self.nome

    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario

    def getSalario(self):
        return self.salario
```

2. Orientação a Objetos

- Construtor da classe:
  - Método chamado automaticamente quando uma instância da classe é criada;
  - Utilizado para inicializar os atributos da classe;
  - Atributos definidos dentro de métodos são criados somente quando o método é chamado. O construtor é uma maneira de criar e inicializar os atributos assim que uma instância da classe é criada.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        if salario == '':
            self.salario = salario
        elif salario > float(self.salario):
            self.salario = salario
```

Herança: engenheiro é subclasse de trabalhador.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        if salario == '':
            self.salario = salario
        elif salario > float(self.salario):
            self.salario = salario
```

engenheiro herda todos os atributos e métodos de *trabalhador*

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        if salario == '':
            self.salario = salario
        elif salario > float(self.salario):
            self.salario = salario
```

Sobrescrevendo um método:

Definir um método com o mesmo nome.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        if salario == '':
            self.salario = salario
        elif salario > float(self.salario):
            self.salario = salario
```

Sobrescrevendo um método (polimorfismo): definir um método com o mesmo nome de um método da superclasse.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        if salario == '':
            self.salario = salario
        elif salario > float(self.salario):
            self.salario = salario
```

Orientação a objetos permite o aproveitamento de código e a customização do código.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        trabalhador.__init__(self,nome,salario)
        self.__idade = idade
```

Utilizando métodos da superclasse dentro da subclasse.

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        trabalhador.__init__(self,nome,salario)
        self.__idade = idade
```

Atributos e métodos privados: _ _

2. Orientação a Objetos

```
class trabalhador():
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        trabalhador.__init__(self,nome,salario)
        self.__idade = idade
```

Não são totalmente privados no Python.

Permitem um controle maior do código: modificar atributos de maneira controlada.

Guardam informação que não podem estar disponível para o usuário: senha, criptografia...

2. Orientação a Objetos

```
class trabalhador(object):
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        super(engenheiro,self).__init__(nome,salario)
        self.__idade = idade
```

Outra forma de utilizar métodos da superclasse dentro da subclasse: função super.

2. Orientação a Objetos

```
class trabalhador(object):
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        super(engenheiro,self).__init__(nome,salario)
        self.__idade = idade
```

Outra forma de utilizar métodos da superclasse dentro da subclasse: função super.

A função super é definida na superclasse *object* do Python. Por padrão, no Python 3 *object* é herdada por todas as classes.

No Python 2 é necessário herdar de object para utilizar super.

2. Orientação a Objetos

```
class trabalhador(object):
    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario
    def setNome(self):
        nome = raw_input('Digite o nome: ')
        self.nome = nome
    def getNome(self):
        return self.nome
    def setSalario(self):
        salario = raw_input('Digite o salario: ')
        self.salario = salario
    def getSalario(self):
        return self.salario
class engenheiro(trabalhador):
    def __init__(self,nome='',salario='',idade=''):
        super(engenheiro,self).__init__(nome,salario)
        self.__idade = idade
```

É comum colocar object como classe herdada mesmo no Python 3 para tornar a herança explícita.

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []

    def adicionarTrab(self):
        novoTrabalhador = trabalhador()
        novoTrabalhador.setNome()
        novoTrabalhador.setSalario()
        self.__listaTrabalhadores.append(novoTrabalhador)
    def mostrarTrabalhadores(self):
        nTrab = 1
        for i in self.__listaTrabalhadores:
            print '%i.'%nTrab
            print 'Nome: ', i.getNome()
            print 'Salario: ', i.getSalario()
            nTrab +=1
```

Classe que utiliza instâncias associadas a outra classe.

2. Orientação a Objetos

Sobrecarga de operadores: sobrescrever métodos construídos no Python.

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

in

len

[ ]

+

-

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

todosTrab = trabalhadores()

t = trabalhador()

t in todosTrab

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

```
todosTrab = trabalhadores()
len(todosTrab)
```

2. Orientação a Objetos

todosTrab = trabalhadores()

todosTrab[0]

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

todosTrab1 = trabalhadores()

todosTrab2 = trabalhadores()

todosTrab1 + todosTrab2

2. Orientação a Objetos

```
class trabalhadores():
    def __init__(self):
        self.__listaTrabalhadores = []
    def __contains__(self, trab):
        for i in self.__listaTrabalhadores:
            if trab.getNome() == i.getNome() and trab.getSalario() == i.getSalario():
                return True
        return False
    def __len__(self):
        return len(self.__listaTrabalhadores)
    def __getitem__(self,index):
            return self.__listaTrabalhadores[index].getNome(),\
        self.__listaTrabalhadores[index].getSalario()
    def __add__(self,other):
        for i in other.__listaTrabalhadores:
            if i not in self:
                self.__listaTrabalhadores.append(i)
    def __sub__(self,other):
        ind = 0
        for i in other.__listaTrabalhadores:
            if i in self:
                i.getNome()
                self.__listaTrabalhadores.pop(ind)
            ind = ind+1
```

todosTrab1 = trabalhadores()

todosTrab2 = trabalhadores()

todosTrab1 - todosTrab2

2. Orientação a Objetos

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __setattr__(self,attrName,value):
        if attrName == 'nome':
            self.__dict__[attrName] = value
            self.__email = value
        elif attrName == 'salario':
            self.__dict__[attrName] = value
        elif attrName == 'salarioInicial':
            self.__dict__[attrName] = value
        elif attrName == '_trabalhador__email':
            self.__dict__[attrName] = value.lower() + '@email.com'

    def getEmail(self):
        return self.__email
```

2. Orientação a Objetos

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __setattr__(self,attrName,value):
        if attrName == 'nome':
            self.__dict__[attrName] = value
            self.__email = value
        elif attrName == 'salario':
            self.__dict__[attrName] = value
        elif attrName == 'salarioInicial':
            self.__dict__[attrName] = value
        elif attrName == '_trabalhador__email':
            self.__dict__[attrName] = value.lower() + '@email.com'

    def getEmail(self):
        return self.__email
```

Email é definido através do __setattr__ . Sempre que o nome mudar, o e-mail também vai mudar, mas de forma controlada.

=

2. Orientação a Objetos

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __setattr__(self,attrName,value):
        if attrName == 'nome':
            self.__dict__[attrName] = value
            self.__email = value
        elif attrName == 'salario':
            self.__dict__[attrName] = value
        elif attrName == 'salarioInicial':
            self.__dict__[attrName] = value
        elif attrName == '_trabalhador__email':
            self.__dict__[attrName] = value.lower() + '@email.com'

    def getEmail(self):
        return self.__email
```

Se o atributo não estiver em __setattr__ , utilizar objeto.atributo = ‘ ’ não funciona =

2. Orientação a Objetos

salarioInicial é atributo de trabalhador, não somente das instâncias associadas a trabalhador.

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __setattr__(self,attrName,value):
        if attrName == 'nome':
            self.__dict__[attrName] = value
            self.__email = value
        elif attrName == 'salario':
            self.__dict__[attrName] = value
        elif attrName == 'salarioInicial':
            self.__dict__[attrName] = value
        elif attrName == '_trabalhador__email':
            self.__dict__[attrName] = value.lower() + '@email.com'

    def getEmail(self):
        return self.__email
```

2. Orientação a Objetos

t1 = trabalhador()

t2 = trabalhador()

trabalhador.salarioInicial = ‘3000’

t1e t2 são modificados

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __setattr__(self,attrName,value):
        if attrName == 'nome':
            self.__dict__[attrName] = value
            self.__email = value
        elif attrName == 'salario':
            self.__dict__[attrName] = value
        elif attrName == 'salarioInicial':
            self.__dict__[attrName] = value
        elif attrName == '_trabalhador__email':
            self.__dict__[attrName] = value.lower() + '@email.com'

    def getEmail(self):
        return self.__email
```

2. Orientação a Objetos

```
class trabalhador(object):

    salarioInicial = '1500'

    def __init__(self,nome='',salario=''):
        self.nome = nome
        self.salario = salario

    def __repr__(self):
        return 'trabalhador(%s,%s)'%(self.nome,self.salario)

    def __str__(self):
        return '%s,%s'%(self.nome,self.salario)
```

Representação do objeto sem ambiguidade.

Representação do objeto para o usuário.