# EN05170 - Programação de Computadores II

## Introdução a Programação Orientada a Objetos - Exercícios

Prof. Dr. Lidio Mauro Lima de Campos

limadecampos@gmail.com

**Universidade Federal do Pará – UFPA**
**ICEN**
**PPGCC**

# Agenda

- Introduzir a programação orientada a objetos, POO.
- Como utilizar this
- Encapsulamento, modificadores de acesso
- Passagem de objetos para método
- Métodos que retornam objetos
- Sobrecarga de métodos
- Propriedades Estáticas, métodos estáticos, iniciadores estáticos.
- Exercícios

# Orientação a Objetos

- **A palavra chave this**
- **Java** inclue um valor de referência especial, chamado **this**, que é usado dentro de qualquer método para se referir ao objeto corrente.
- **this** permite referir-se diretamente ao objeto.

# Orientação a Objetos

- **A palavra chave this**

```
class Ponto
{
 int x,y;
 void init(int x,int y)
 {
   this.x=x;this.y=y;
 }}
```

O uso de this em tal contexto pode causar confusão e alguns programadores tomam cuidado para não usar as variáveis locais e os nomes dos parâmetros formais que ocultam as variáveis de instância.

# Orientação a Objetos

- **A palavra chave this**

```
class DoisPontos{
 public static void main(String args[])
  {
    Ponto p1=new Ponto();
    Ponto p2=new Ponto();
    p1.init(10,20);p2.init(42,99);
    System.out.println("x="+p1.x+"y="+p1.y);
    System.out.println("x="+p2.x+"y="+p2.y);
  }}
```

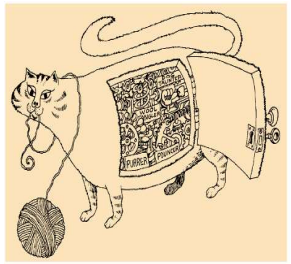
# OO-Uso da palavra da palavra chave de this

```
class Ilustracao
{ int instanceVar = 5;
  static int staticVar = 10; // declarando uma variavel estatica

  void Scaler()
  { //Variáveis especificas do Método Scaler()
    int instanceVar = 20; int staticVar = 40;

    // referindo-se as variáveis de instancia da classe e variaveis estaticas
    this.instanceVar = 50; this.staticVar = 100;

    // imprimindo as variaveis de instancia da classe e variavel estatica.
    System.out.println("Valor da variavel de instancia: " + this.instanceVar);
    System.out.println("Valor da variavel estatica : " + this.staticVar);

    // imprindo as variáveis especificas do método.
    System.out.println("instanceVar dentro do método : " + instanceVar);
    System.out.println("staticVar dentro do metodo: " + staticVar);
  }
}
```

# Encapsulamento

- Encapsulamento, consiste em separar os aspectos externos de um objeto, que são acessíveis para outros objetos, dos detalhes internos de implementação do objeto [Rumbaugh];
- Qualquer mecanismo que nos permita “esconder” a implementação do objeto fazendo com que outros componentes do sistema não tenham conhecimento do conteúdo interno dos dados armazenados no objeto

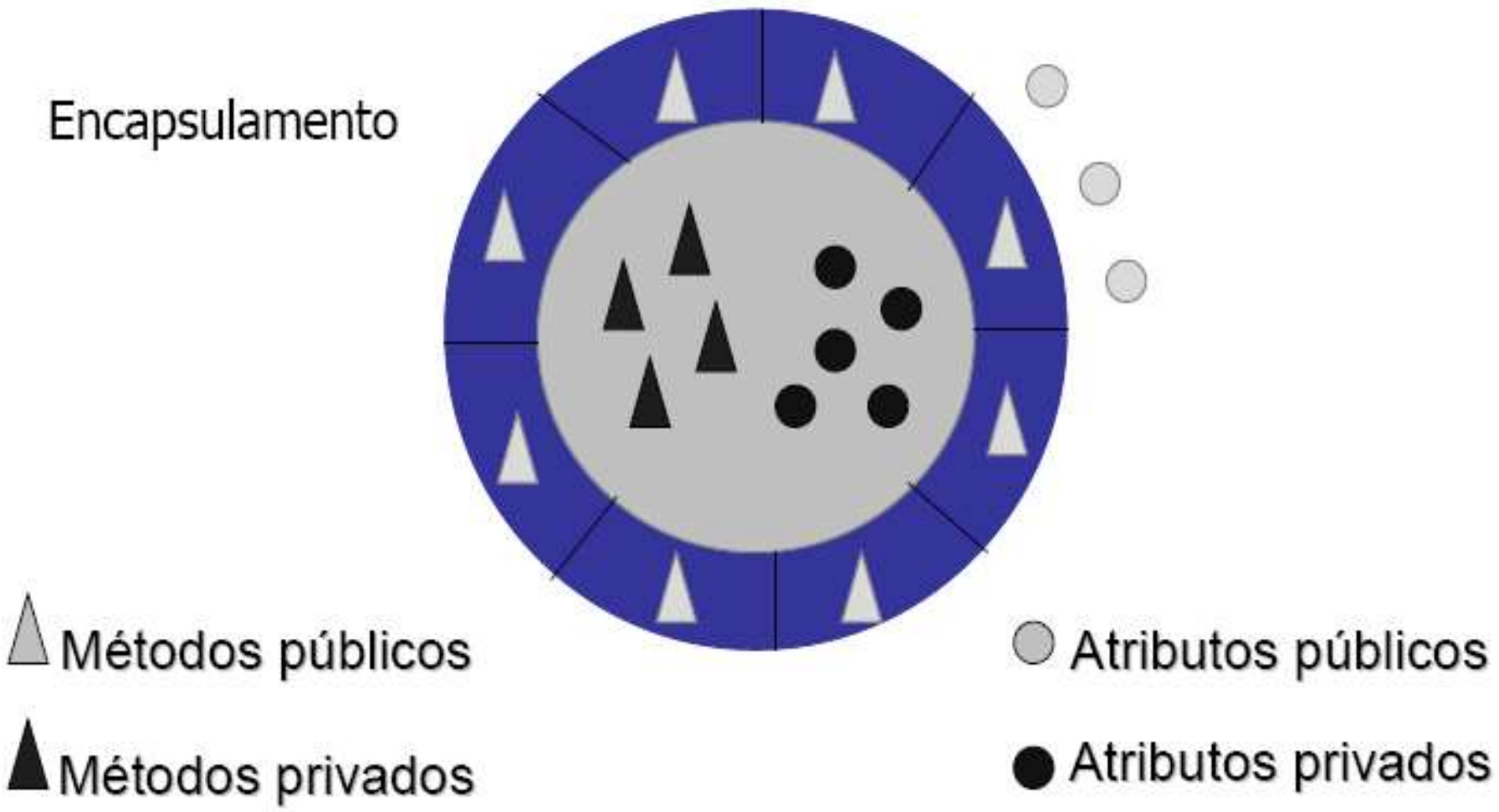
  [Yourdon];



# Encapsulamento

- Informação escondida;
- A comunicação entre objetos é feita através de mensagens enviadas para as operações;
- A implementação de um objeto pode ser mudada sem afetar as aplicações que o usam;
- Facilidade de manutenção;

# Encapsulamento



# Modificadores de acesso

| Especificador | Nível | Indica que o campo ou método |
|---|---|---|
| public | publico | Pode ser usado livremente pelas instancias de classes |
| | pacote | Só pode ser usado por classes e instâncias dentro do mesmo pacote. |
| protected | Protegido | Só pode ser usado na implementação de subclasse e instâncias dentro do mesmo pacote |
| private | privado | Não pode ser usado fora da implementação da própria classe. |

# Encapsulamento - Visibilidade

- Consequências de tornar um atributo privado
- Tentar acessar um componente privado (de fora da classe) resulta em erro de compilação;
- Mas como torná-lo acessível apenas para consulta (leitura)?
- Isto é possível definindo-se um método que retorna o atributo (na própria classe onde o atributo se encontra);

# Modificadores de acesso

```
class MyClass
{
  private int alpha; //acesso privado  public int beta; // acesso publico
  int gamma; // acesso padrão publico

   /* Métodos para acessar alpha.  Não há problema em um membro de
   uma classe acessar um membro privado da mesma classe.  */

 void setAlpha(int a)
  { alpha = a; }

  int getAlpha()
 {  return alpha; }
}
```

# Modificadores de acesso

```
class AccessDemo
{  public static void main(String args[])
   {
    MyClass ob = new MyClass();
   /* Accesso a alpha só é permitido por intermédio de seus métodos acessadores */
   ob.setAlpha(-99);
   System.out.println("ob.alpha is " + ob.getAlpha());
   // Voce não pode acessar alpha da forma abaixo:
   //ob.alpha = 10; // Errado! alpha é privado!
    ob.setAlpha(10);
   // Essas linhas estão corretas, pois beta e gamma são publicos.
   ob.beta = 88;
   ob.gamma = 99;
   }
}
```

# Encapsulamento - Visibilidade

- Todas as vezes que você quiser ler uma propriedade defina um método

  *getNomePropriedade*

- Caso o atributo seja boolean use

  *isNomePropriedade*

- Para alterar uma propriedade declare um método

  *setNomePropriedade*

Esse padrão (get, set ) é apenas uma convenção de nome.

# Modificadores de acesso

```
public class Cliente {
   private int codigo=0; private String nome=""; private boolean ativo=true;

   public void setCodigo(int codigo)
   {this.codigo=codigo;}

    public int getCodigo()
   {return this.codigo;}

   public void setNome(String nome)
   {this.nome=nome;}

   public String getNome()
   {return this.nome;}

   public void setAtivo(boolean ativo)
   {this.ativo=ativo;}

   public boolean isAtivo()
   {return this.ativo;}
}
```

# Modificadores de acesso

```
public class TestaCliente{

public static void main(String[] args)
{
        Cliente c=new Cliente();
        c.setCodigo(1000);
        c.setNome("LIDIO");
        c.setAtivo(true);
        System.out.println(c.getCodigo());
        System.out.println(c.getNome());
        System.out.println(c.isAtivo());
  }
  }
```

# Modificadores de Acesso

```
public class MyDate
{ private int day; private int month; private int year;
 public int getDay()
 { return day; }
 public void setDay( int newDay )
 { day = newDay; }
 public int getMonth()
 { return month; }
 public void setMonth( int newMonth )
 { month = newMonth; }
...}
```

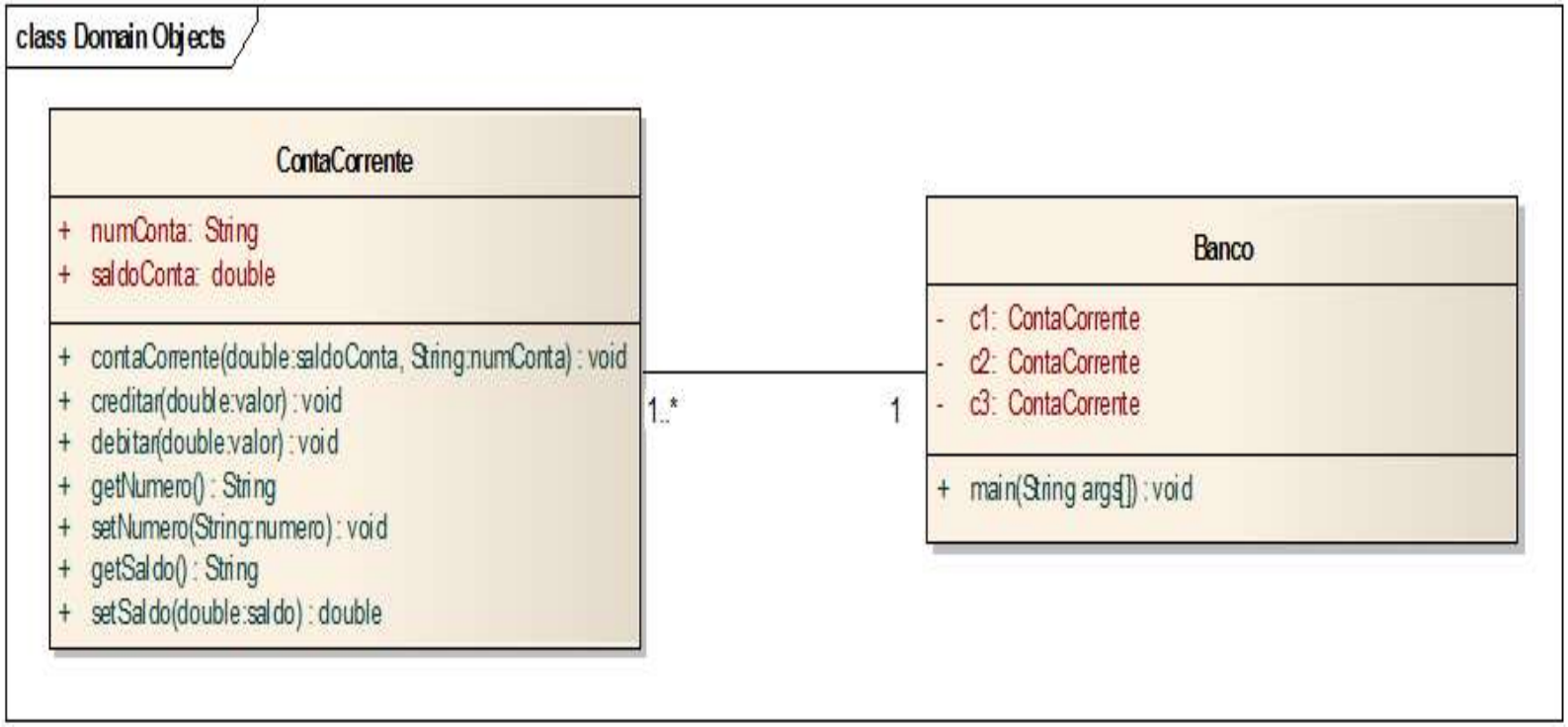
# Encapsulamento

- Dica
- É uma boa prática de programação definir as propriedades de objetos sempre como private.

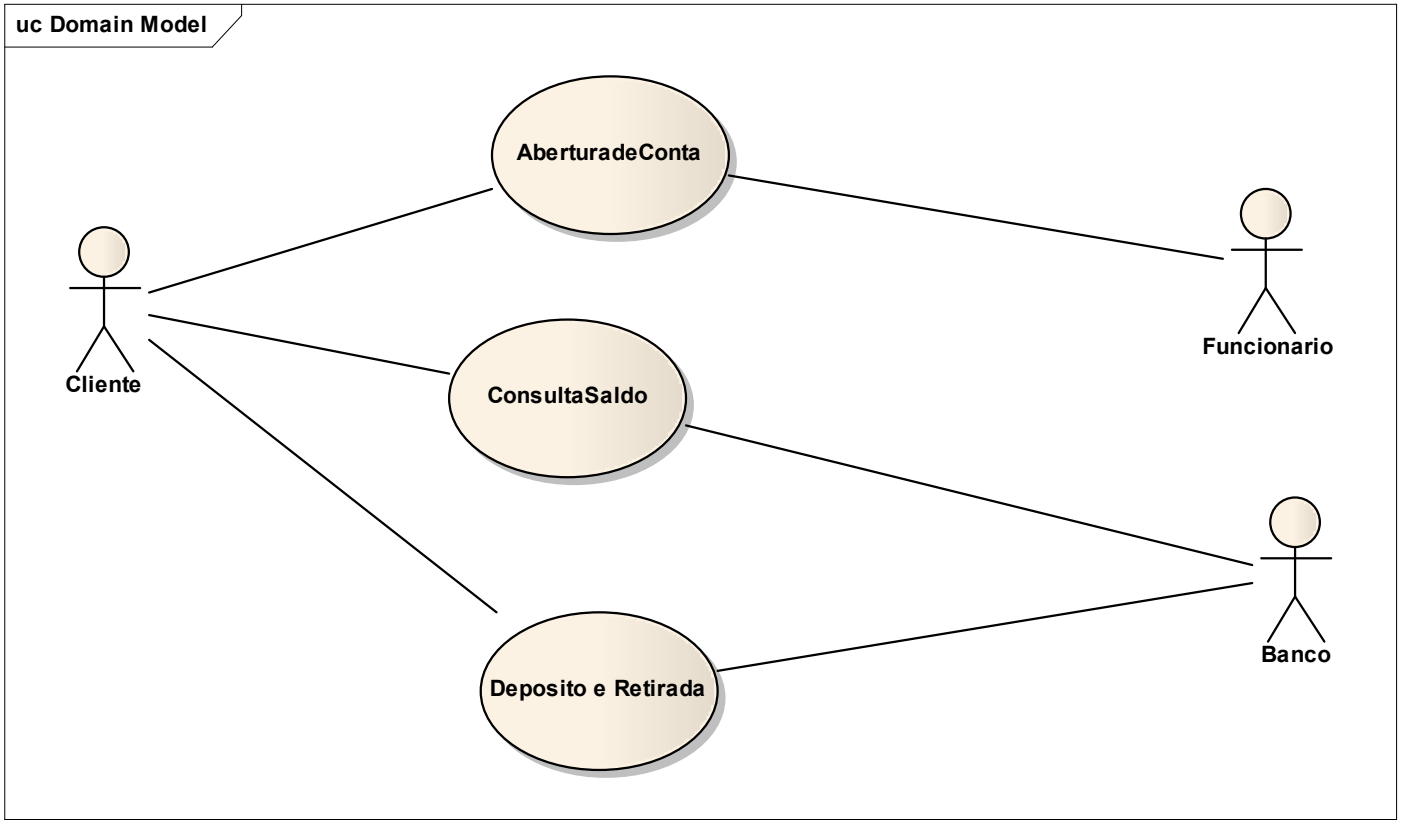
# Exercício - Estudo de Caso 2

- Altere a classe ContaCorrente declarando os atributos saldo e numero como private e definindo os métodos setSaldo, getNumero e setNumero, getSaldo.

# D.Classes UML



# D. Caso de Uso (Resumido)



# Estudo de Caso 2

```
class Conta_Corrente
{ private String numero; private double saldo;
  public Conta_Corrente(String numConta,double saldoConta)
  { numero=numConta; saldo=saldoConta; }

  public void creditar(double valor)
  {saldo=saldo+valor; }
  public void debitar(double valor)
  { if (saldo >= valor)
    {saldo = saldo - valor;} }

  public double getSaldo()
  { return saldo; }

  public String getNumero()
  { return numero; }

  public void setNumero(String n)
  {numero = n; }
  public void setSaldo(double s)
  { saldo = s; }
}
```

# Estudo de Caso 2

```
public class Banco {
 public static void main(String args[])
  {
   ContaCorrente c1=new ContaCorrente("1001",1090);
   ContaCorrente c2=new ContaCorrente("1002",1290);
   ContaCorrente c3=new ContaCorrente("1003",1340);
   ContaCorrente c4=new ContaCorrente("1004",1870);
   c1.creditar(100);
   c2.creditar(2100);
   c3.debitar(100);
   c4.debitar(2100);
   System.out.println("Numero Conta:"+c1.getNumero()+" Saldo="+c1.getSaldo());
   System.out.println("Numero Conta:"+c2.getNumero()+" Saldo="+c2.getSaldo());
   System.out.println("Numero Conta:"+c3.getNumero()+" Saldo="+c3.getSaldo());
   System.out.println("Numero Conta:"+c4.getNumero()+" Saldo="+c4.getSaldo());
  c4.setNumero("3100");
  System.out.println("Numero Conta:"+c4.getNumero()+" Saldo="+c4.getSaldo());
 }
}
```

# Passando objetos para métodos

```
class Paralelepipedo
{  int a, b, c;   int volume;

   Paralelepipedo(int i, int j, int k)
   { a = i;  b = j;  c = k;
   }
   // Retorna verdadeiro  se os objetos tem a mesma dimensão
   boolean mesmoParalelepipedo(Paralelepipedo ob)
   {return (ob.a == a) & (ob.b == b) & (ob.c == c); }

  // Retorna verdadeiro se os objetos possuem o mesmo volume.
   boolean mesmoVolume(Paralelepipedo ob)
  {  return ob.volume == volume;  }
}
```

# Passando objetos para métodos

```
class ParalelepipedoDemo
 public static void main(String args[])
 {
    Paralelepipedo ob1 = new Paralelepipedo(10, 2, 5);
    Paralelepipedo  ob2 = new Paralelepipedo(10, 2, 5);
    Paralelepipedo ob3 = new Paralelepipedo (4, 5, 5);

    System.out.println("ob1 tem a mesma dimensão do ob2: " +ob1.mesmoParalelepipedo(ob2));
    System.out.println("ob1 tem a mesma dimensão do ob3: " +ob1.mesmoParalelepipedo(ob3));
    System.out.println("ob1 tem a mesma dimensão do ob3: " + ob1.mesmoVolume(ob3));
  }
}
```

# Retornando Objetos

```
class ErrorMsg
{ String msgs[] = { "Output Error", "Input Error", "Disk Full", "Index Out-Of-Bounds"};

  // Return the error message.
  String getErrorMsg(int i)
  { if(i >=0 & i < msgs.length)
      return msgs[i];
    else
      return "Invalid Error Code";
  }
}
```

# Retornando Objetos

```
class ErrMsg {
  public static void main(String args[]) {
    ErrorMsg err = new ErrorMsg();

    System.out.println(err.getErrorMsg(2));
    System.out.println(err.getErrorMsg(19));
  }
}
```

Saída: Disk Full

Invalid Error Code

# Sobrecarga de Métodos

```
class Overload {
 void ovlDemo()
 {System.out.println(“Sem parametros");}
  void ovlDemo(int a) {System.out.println(“Um parametro: " + a);}

  int ovlDemo(int a, int b) { System.out.println(“Dois parametros: " + a + " " + b);
    return a + b;
  }
  double ovlDemo(double a, double b)
  {
    System.out.println(“Dois parâmetros Double: "+a + " "+ b);
    return a + b;
  }
}
```

# Sobrecarga de Métodos

```
class OverloadDemo
{  public static void main(String args[])
   {Overload ob = new Overload();
      int resI; double resD;
      ob.ovlDemo();
      System.out.println();
      ob.ovlDemo(2);
      System.out.println();
      resI = ob.ovlDemo(4, 6);
     System.out.println("Resultado de ob.ovlDemo(4, 6): " +resI);
      System.out.println();
      resD = ob.ovlDemo(1.1, 2.32);
      System.out.println("Resultado de ob.ovlDemo(1.1, 2.2): "+resD);
  }
}
```

# Cuidado na Sobrecarga de Métodos

```
void ovlDemo(int a)
{ System.out.println(“um parâmetro”+a);}

int ovlDemo(int a)
{
   System.out.println(“um parâmetro”+a);
   return a*a;
}
```

- Os tipos de retorno não podem ser usados para diferenciar métodos sobrecarregados.
- Erro não é correto usar dois métodos ovlDemo(int) mesmo que os tipos de retorno sejam diferentes.

# Propriedades Estáticas

- Propriedades estáticas são compartilhadas por todas as instâncias de uma classe.
- Normalmente um membro de uma classe é acessado por intermédio de um objeto de sua classe.
- No entanto, é possível criar um membro (estático) para ser usado por conta própria, sem referência a uma instância específica.

# Propriedades Estáticas

```
class StaticDemo
{
  int x; // uma variável de instância comum
  static int y; // uma variável estática

  // Retorna  a soma da variável de instancia x
  // e a variável estática y.
  int sum()
  {
    return x + y;
  }
}
```

# Propriedades Estáticas

```
class SDemo {
  public static void main(String args[]) {
    StaticDemo ob1 = new StaticDemo();
    StaticDemo ob2 = new StaticDemo();

    // Cada objeto tem sua própria cópia de uma variável de instância
    ob1.x = 10; ob2.x = 20;
    System.out.println("Claro, ob1.x e ob2.x " + "sao independentes.");
    System.out.println("ob1.x:"+ob1.x + "\nob2.x:"+ob2.x);
    System.out.println();

    // Cada objeto compartilha uma cópia de uma variável estática.
    System.out.println("A variável estática é compartilhada.");
    StaticDemo.y = 19; //Acesso ao atributo estatico
    System.out.println("Mude  StaticDemo.y para 19.");

    System.out.println("ob1.sum(): " + ob1.sum());
    System.out.println("ob2.sum(): " + ob2.sum());
    System.out.println();
  }
}
```

# Métodos Estáticos

- Métodos também podem ser static;
- A diferença entre um método static e um método comum é que o método static é chamado com o uso do nome de sua classe, sem nenhum objeto dela ser criado.
- Já utilizamos o método sqrt(), que em Java é um método static da classe padrão Math.

# Métodos Estáticos

```
class StaticMeth
 {
  static int val = 1024; // uma variável estática
  // um método estático
  static int valDiv2()
 {
    return val/2;
  }
}
```

# Métodos Estáticos

```
class SDemo2
{
  public static void main(String args[]) {

    System.out.println(“Valor e " + StaticMeth.val);
    System.out.println("StaticMeth.valDiv2(): "+StaticMeth.valDiv2());

    StaticMeth.val = 4;
    System.out.println(“Valor e " + StaticMeth.val);
    System.out.println("StaticMeth.valDiv2(): "+StaticMeth.valDiv2());
   }
}
```

# Inicializadores Estáticos

- Uma classe pode conter um bloco *static* que não pertence a nenhum método;

- Esse código é executado apenas uma vez quando a classe é lida

# Inicializadores Estáticos

```
class StaticBlock
{
   static double rootOf2;
   static double rootOf3;
   static
   { System.out.println(“Dentro do bloco estatico");
     rootOf2 = Math.sqrt(2.0);
     rootOf3 = Math.sqrt(3.0);
   }

   StaticBlock(String msg)
   {
    System.out.println(msg);
   }
}
```

# Inicializadores Estáticos

```
class SDemo3
{
  public static void main(String args[])
  {
     StaticBlock ob = new StaticBlock(“Dentro do Construtor");
     System.out.println(“Raiz quadrada de 2 e " +StaticBlock.rootOf2);
     System.out.println(“Raiz quadrada de 3 e "+StaticBlock.rootOf3);
   }
}
//Dentro do bloco estático
//Dentro do construtor
//Raiz quadrada de 2 e : 1.41
//Raiz quadrada de 3 e: 1.73
```

- Referências

- Peter Jandl Junior. **Java - Guia do Programador: Atualizado Para Java 16 - 4ª edição. NOVATEC. 2021.**
- Herbert Schildt. Java: a referência completa. ALTA BOOKS Editora. 2020.