

Programação

OO

II

I

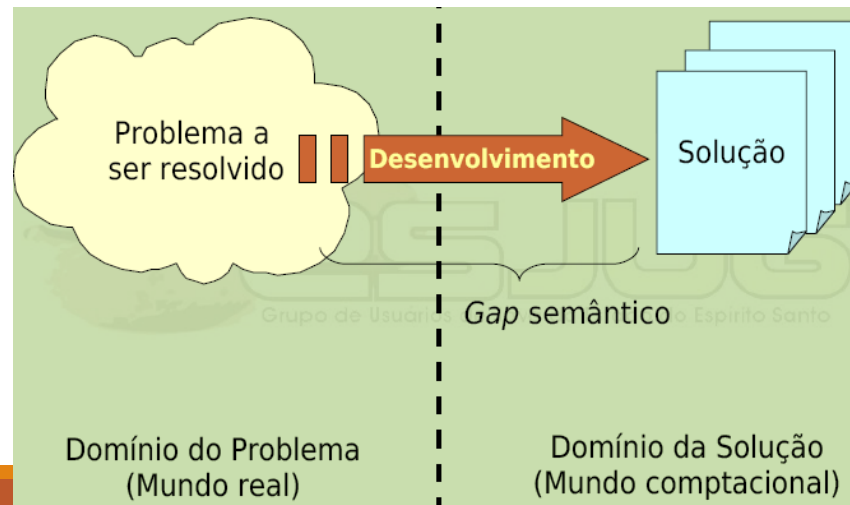
Parte I



Conceitos OO - Relembrar

Gap semântico

- Nossa mente é orientada a objetos;
- Se nossa LP também for OO, podemos diminuir o *gap* semântico:
 - Os recursos oferecidos são os mesmos, porém os mecanismos de abstração são mais poderosos;
 - Passamos a construir a solução em termos do domínio do problema (mundo real – objetos);
 - Usamos também objetos que não são parte do domínio do problema.





Conceitos OO - Relembrar

Exemplos de Objetos de software:

- Todo objeto no mundo real possui 3 características:
 - Estado;
 - Comportamento.
 - Identidade única

Objeto	Estado	Comportamento
Cão	Nome Raça Cor	Caçando Comendo Balançando o ramo
Bicicleta	Velocidade do pedal Marcha Velocidade da Bike	Freiar Acelerar Mudar de marcha

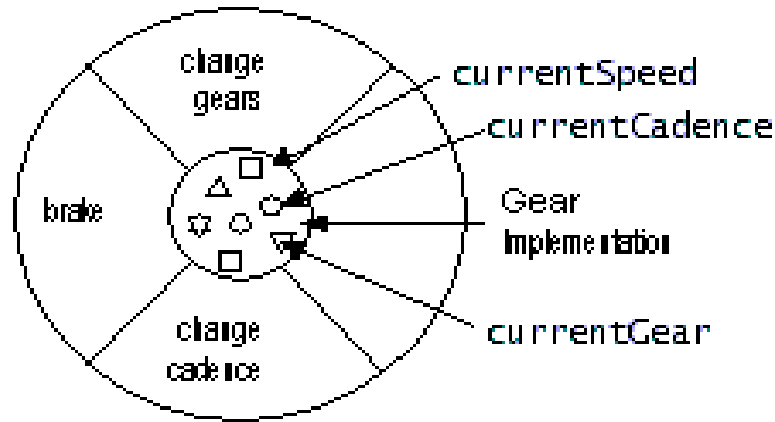


Conceitos OO - Relembrar

Classes

- Uma Classe é um Protótipo que define as variáveis e métodos comuns a todos os objetos de um determinado tipo.
- Uma classe define as propriedades (estado) e as operações (comportamento) que são comuns a um conjunto de objetos.
- Uma Classe define um tipo, a partir do qual todos os objetos são criados.

Classe
Bicicleta



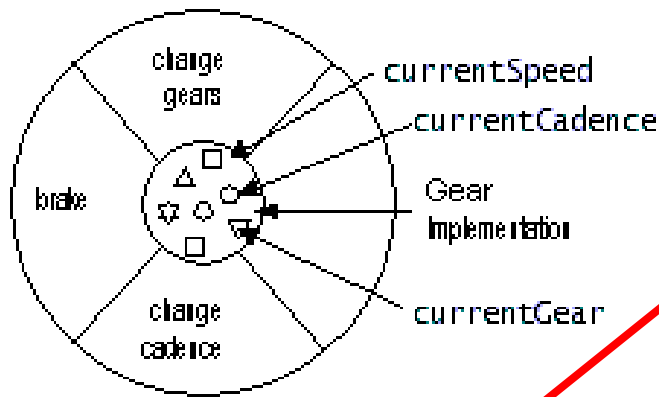


Conceitos OO - Relembrar

Instâncias:

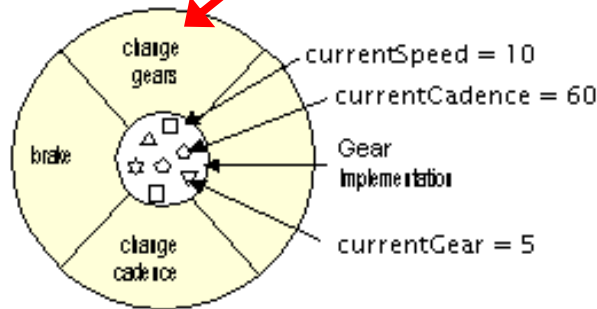
- Um objeto é uma instância/representação de uma classe.

Classe Bicicleta

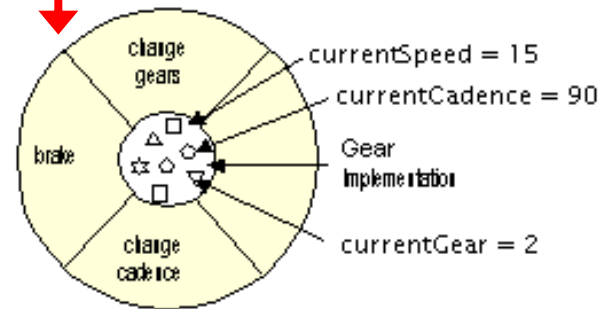


Em software, instâncias alocam um espaço de memória especificado do seu protótipo (CLASSE).

Objetos (instâncias) da classe Bicicleta



MyBike



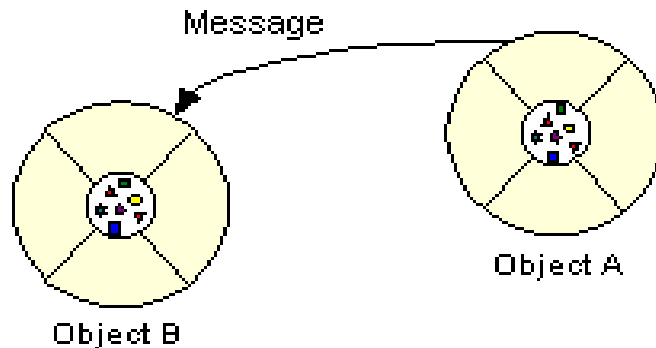
YourBike



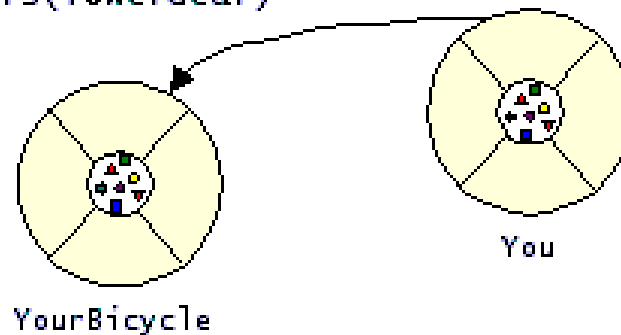
Conceitos OO - Relembrar

Mensagem

- As mensagens são o meio de comunicação entre objetos.



`changeGears(lowerGear)`



Você (*You*) solicita mudança de marcha para sua bicicleta (*YourBicycle*)

Informação necessárias:

- Objeto destino (*YourBicycle*);
- Método (*changeGears*);
- Parâmetro (*lowerGear*).



Estrutura de uma classe

Visualmente podemos modelar uma classe da seguinte forma:



- Todos aplicados de acordo com a sintaxe da linguagem Java
-Mas.....Que sintaxe é essa!!!



Declaração de Classes

Para declarar uma nova classe, utiliza-se a seguinte sintaxe:

```
<modificador-acesso> class <nome-classe> {  
    /* corpo da classe, com atributos e métodos */  
}
```

Ex:

```
class Ponto {  
    ...  
}
```

Estamos definindo
um tipo!!

- A classe cria um novo tipo, a partir do qual podemos criar e usar objetos.



Declaração de Variáveis

Nas classes podemos declarar variáveis/atributos e métodos:

- Variáveis definem o estado de um objeto, de acordo com seus atributos.
- Métodos define como é possível interagir com um objeto, de maneira a modificar o seu estado ou obter informações a seu respeito.

Para declarar um ATRIBUTO de classe deve-se atender a seguinte sintaxe:

```
<modificador de acesso><tipo><nome>;
```

- Exemplo:

```
class MinhaClasse {  
    int atributo1;  
    int atributo2;  
}
```

```
class Ponto {  
    float x, y;  
}
```



Declaração de Métodos

A declaração de um MÉTODO deve obedecer a seguinte sintaxe:

```
<modificador-acesso><tipo-retorno><nome> (<parametros>) {  
    /* código do método */ }
```

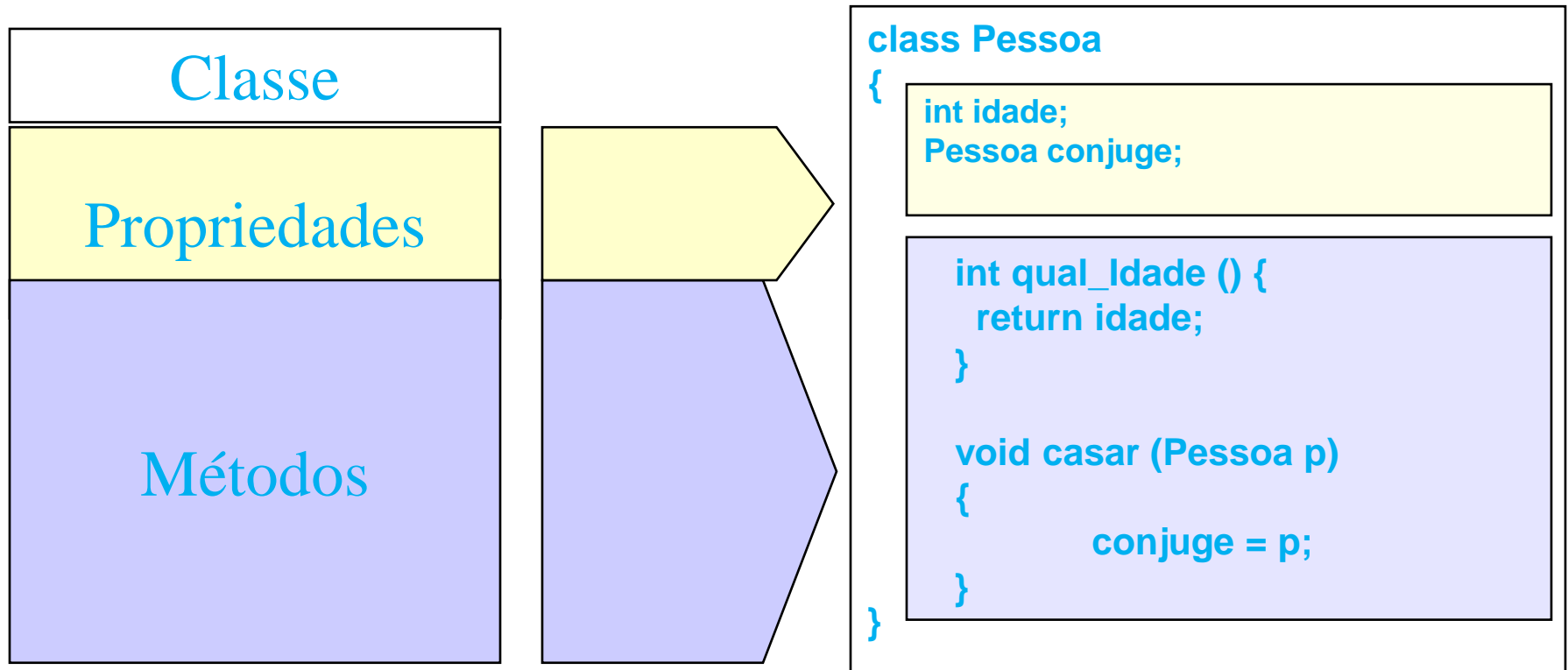
- Exemplo:

```
class Ponto {  
    int x, y;  
    void mover ( int dx, int dy) {  
        x += dx;  
        y += dy;  
    }  
}
```



Estrutura de uma classe

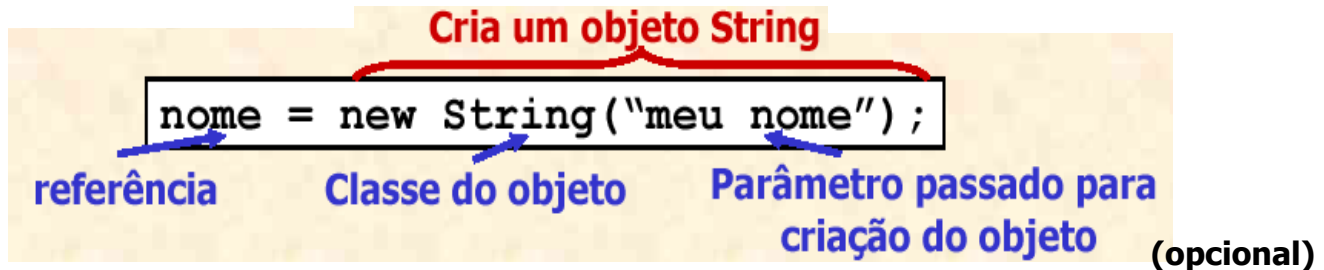
- Resumindo: No final a nossa estrutura ficará da seguinte forma....





Manipulação de Objetos em Java

Para criar um objeto em memória, ou instanciá-lo, precisamos utilizar o operador *new* da seguinte maneira:



Importante: Objetos Java são manipulados por referências.

- Internamente, o Java aponta para a posição da memória que representa a variável

Vamos entender isso melhor...



Instanciação de Objetos

Vamos analisar com mais detalhes... As figuras a seguir ilustram o processo de instanciação do objetos

- Exemplo:

```
class Editor {  
    public static void main (String arg[]) {  
        Ponto p;  
        int a = 2;  
    }  
}
```

```
class Ponto {  
    int x, y;  
    void mover ( int dx, int dy) {  
        x += dx;  
        y += dy;  
    }  
}
```

Declaração de uma
variável a do tipo
inteiro.

Declaração de uma
variável p do tipo da
classe Ponto.

2

p → null

X = ?
Y = ?



Instanciação de Objetos

Dada uma classe, pode ser criada uma nova instância desta classe usando o comando *new*.

- Aqui o objeto é criado em memória
- Exemplo:

```
class Editor {  
    public static void main (String arg[]) {  
        Ponto p = new Ponto();  
    }  
}
```

p

X = 0
Y = 0



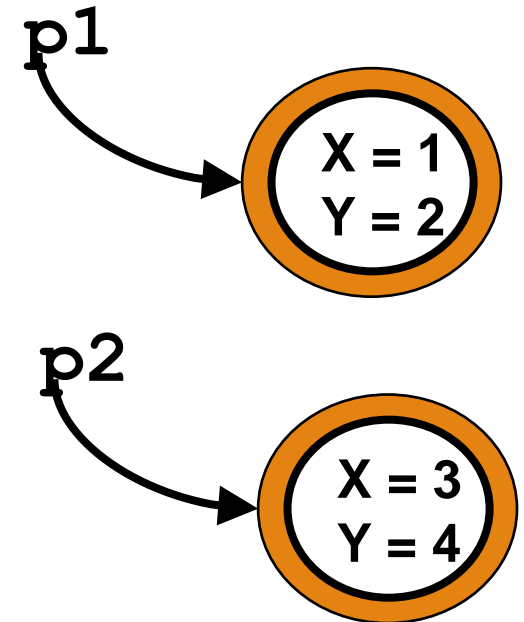
Manipulação de Objetos em Java

Acessando membros de uma classe - Variáveis

- As propriedades visíveis dos objetos podem ser manipuladas diretamente através do ponto (.):

```
Ponto p1 = new Ponto();  
p1.x = 1;  
p1.y = 2;  
// p1 representa o ponto (1,2)
```

```
Ponto p2 = new Ponto();  
p2.x = 3;  
p2.y = 4;  
// p2 representa o ponto (3,4)
```



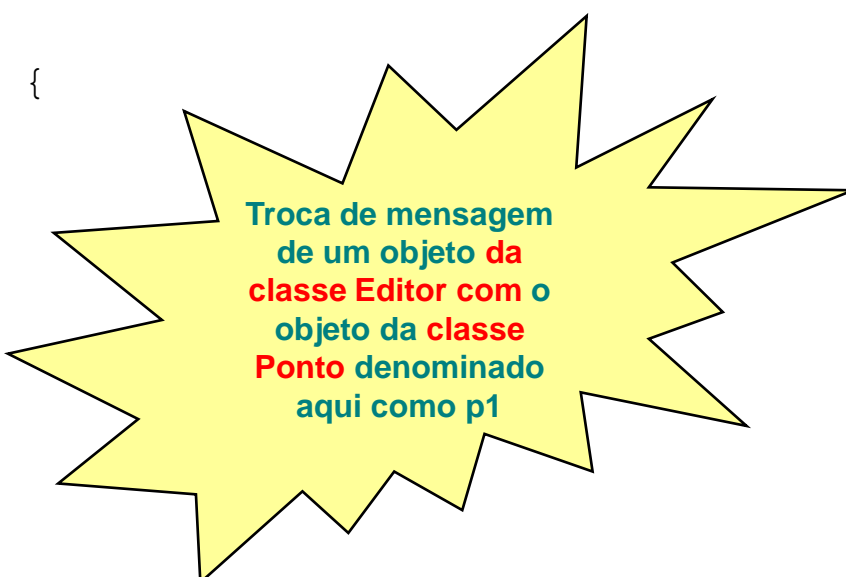


Manipulação de Objetos em Java

Acessando membros de uma classe – Métodos Operacionais

- A troca de mensagens entre os objetos é realizada através da chamada de métodos com passagem de parâmetros
- Mensagem entre Objetos

```
class Editor {  
    void criarPonto (int x1, int y1) {  
        Ponto p1 = new Ponto ();  
        p1.x = x1;  
        p1.y = y1;  
        p1.mover(1,2);  
    }  
}
```



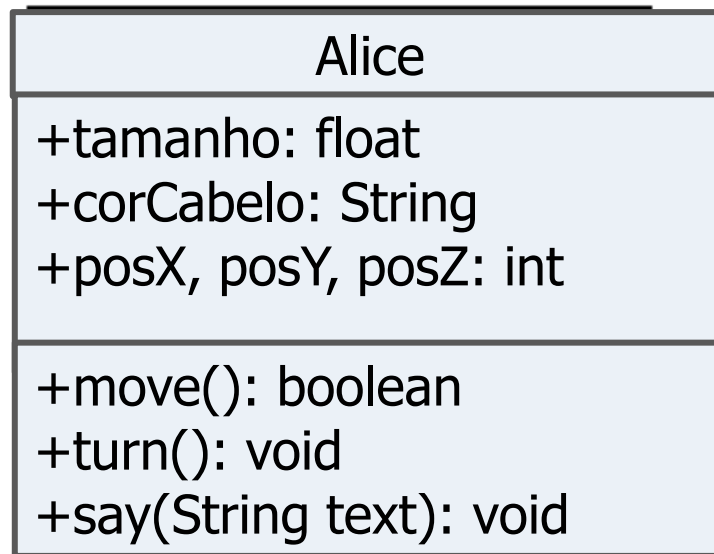
Troca de mensagem
de um objeto da
classe Editor com o
objeto da classe
Ponto denominado
aqui como p1



Estrutura de um programa Java - Exemplo junto

Modelagem

Seja a Representação gráfica da classe Alice:



- Qual seria uma codificação para esse modelo e uma classe executável?
Vamos implementar.



Sobrecarga de Métodos

Agora vale relembrar um conceito importante...

- Que impacta diretamente no uso dos nosso construtores...

É bastante utilizado quando definimos mais de um construtor para nossa classe!

Sobrecarregar (*overloading*) um método significa:

- Que em uma classe é possível definir dois ou mais métodos com o mesmo nome, porém com assinaturas diferentes;
- Mas o que determina a diferença das assinaturas?



Sobrecarga de Métodos Operacionais

O que são as assinaturas....???

- Assinatura diferente pode ser: número ou tipos de parâmetros diferentes.
 - A ordem dos tipos de parâmetro influi
- Java não considera o tipo de retorno para diferenciar entre os métodos sobrecarregados. Se tentar criar dois métodos com a mesma assinatura e tipo de retorno diferentes, a classe não será compilada.

- Exemplo:

```
public class Sobrecarga {  
    public void metodo1() {}  
    public void metodo1(int a) {}  
    public void metodo1(int a, int b) {}  
    public void metodo1(float a, int b) {}  
    public void metodo1(int a, float b) {}  
    public int metodo1() {} // ERRO:método repetido  
    public void metodo1(int b) {} // ERRO:método repetido  
}
```



Sobrecarga de Métodos Operacionais

Uma boa prática é usar a sobrecarga, somente, em métodos que possuam a mesma funcionalidade.

- Exemplo:

```
Class Ponto {  
    ...  
    void mover (int dx, int dy){  
        x += dx;  
        y += dy;  
    }  
  
    void mover (int raio, float ang){  
        x += raio*Math.cos(ang);  
        y += raio*Math.sen(ang);  
    }  
}
```

Mover tem o significado de mover algo, mas de forma diferentes.

No primeiro caso desloca o ponto de dx e dy.

No segunda caso usa regras de trigonometria para mover o ponto.



Construtores – Principal Característica

Um construtor é um método especial da classe, responsável exclusivamente por inicializar um objeto

- Devemos utilizar o construtor quando queremos atribuir valores aos atributos de um objeto no momento de sua criação

Para declarar um método construtor, devemos declarar um método com o mesmo nome da classe

- Um método construtor não possui valor de retorno.

O construtor padrão de uma classe não tem parâmetros

- Exemplo 1:

```
class Pilha {  
    /*atributos e métodos */  
  
    public Pilha() {  
        /* implementação do método construtor */  
        tamanho = 0;  
        vetor = new int [10 ];  
    }  
}
```



Método
construtor
default

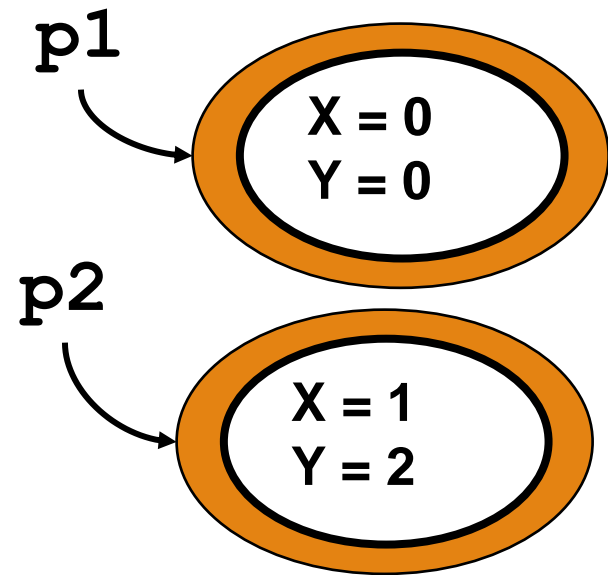


Sobrecarga de Métodos Construtores

Exemplo 2:

```
Class Ponto {  
    int x = 0;  
    int y = 0;  
  
    Ponto () { }  
    Ponto (int a, int b) {  
        x = a;  
        y = b;  
    }  
}
```

```
Ponto p1 = new Ponto();  
//p1 está em (0,0)  
Ponto p2 = new Ponto(1,2);  
//p2 está em (1,2)
```



A sobrecarga de construtores visa definir formas diferentes de criar e inicializar um objeto



Construtores

Qual a saída impressa na tela após a execução do código ao lado?...

Lembrar:

1º: é alocada memória para o objeto;

2º: o construtor é chamado.

```
1 class Rotulo {  
2     Rotulo (int marca) {  
3         System.out.println("Rotulo(" + marca + ")");  
4     }  
5 }  
6  
7 class Cartao {  
8     Rotulo t1 = new Rotulo(1);  
9     Rotulo t2 = new Rotulo(2);  
10    Rotulo t3 = new Rotulo(3);  
11  
12    Cartao() {  
13        System.out.println("Cartao()");  
14        t3 = new Rotulo(33);  
15    }  
16  
17    void f() {  
18        System.out.println("f()");  
19    }  
20 }  
21  
22  
23 public class OrdemDeInicializacao {  
24     public static void main (String[] args) {  
25         Cartao t = new Cartao();  
26         t.f();  
27     }  
28 }
```

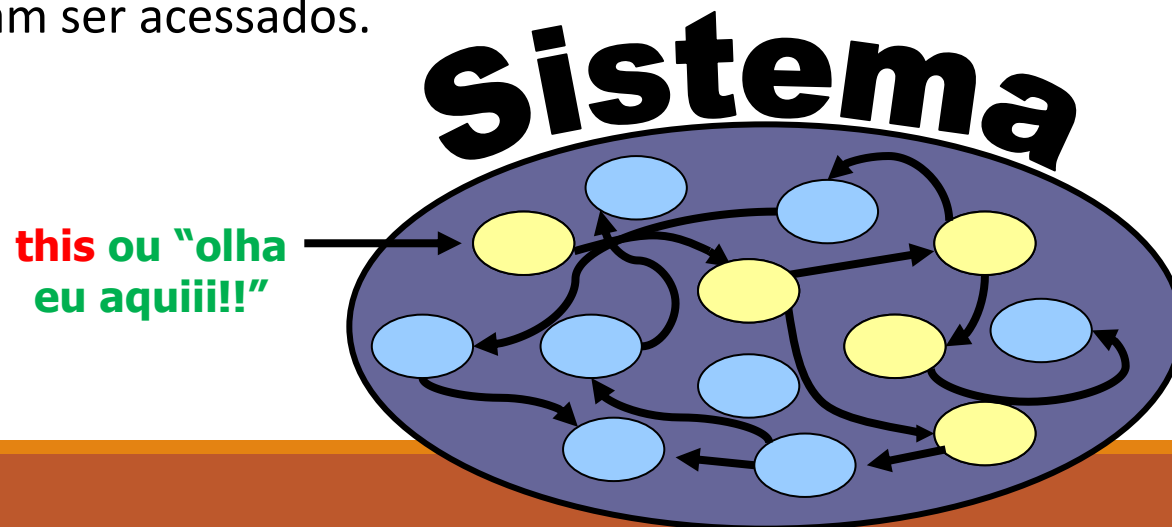



This

A referência *this* referencia o próprio objeto ao qual operação está sendo aplicada.

Indica o objeto sendo processado; o objeto corrente

A referência *this* também permite que os atributos e métodos da classe possam ser acessados.





This – Na Prática

Exemplo 1

```
class Ponto {  
    → int x = 0;  
      int y = 0;  
  
      Ponto (int x, int y) {  
          → this.x = x;  
            this.y = y;  
        }  
    }  
}
```

- this referenciando e diferenciando as variáveis (x e y)



This – Na Prática

Qual a saída impressa na tela após a execução do código abaixo?...

Exercícios

```
1 public class ExemploThis {  
2     int i = 0;  
3  
4     ExemploThis increment() {  
5         i++;  
6         return this;  
7     }  
8  
9     void print() {  
10        System.out.println("i = " + i);  
11    }  
12  
13    public static void main(String[] args) {  
14        ExemploThis x = new ExemploThis();  
15        x.increment().increment().increment().print();  
16    }  
17 }
```



Controle de Acesso

O controle de acesso a variáveis e métodos é feito por meio dos modificadores de visibilidade.

Seu principal objetivo é:

- Determinar a visibilidade de um determinado membro da classe com relação a outras classes; (*modificadores-acesso*)

Podem ser:

- Public, Private , Protected, Package

Inicialmente vamos falar sobre os Public, Private

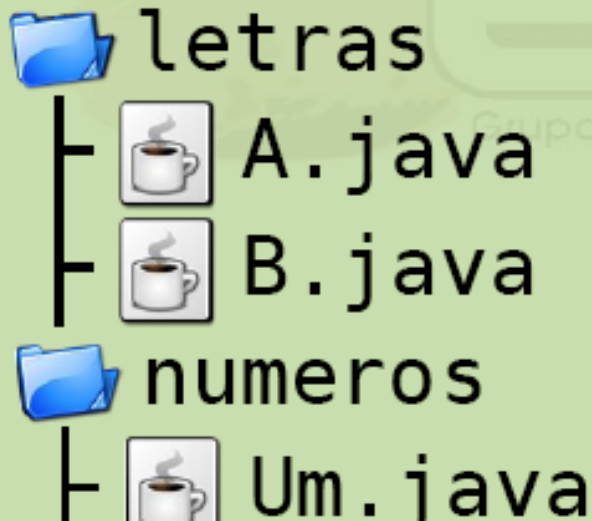


Controle de Acesso

Variáveis e métodos públicos - Public

```
public class A {  
    public int x = 10;  
    public void print() {  
        System.out.println(x);  
    }  
}
```

```
import letras.B;  
public class Um {  
    B b = new B();  
    public void g() {  
        b.f();  
    }  
}
```



```
public class B {  
    public A a = new A();  
    public void f() {  
        a.x = 15;  
        a.print();  
    }  
}
```



Controle de Acesso

Variáveis e métodos privados - Private

```
public class A {  
    private int x = 10;  
    private void print() {  
        System.out.println(x);  
    }  
    void incr() { x++; }  
}
```

```
import letras.B;  
public class Um {  
    B b = new B();  
    public void g() {  
        b.f();  
    }  
}
```

letras

- A.java
- B.java

numeros

- Um.java

```
public class B {  
    public A a = new A();  
    public void f() {  
        // Erro: a.x = 15;  
        // Erro: a.print();  
    }  
}
```



Controle de Acesso e Métodos Acessores

- Seja a modelagem..

Cliente

- nome:String
- cidade:String
- ativo:boolean
+setNome(nome:String):void
+getNome():String
+setCidade(cidade:String):void
+getCidade:String
+setAtivo(ativo:boolean):void
+getAtivo:Boolean

```
package Aula05;
```

```
public class Cliente {
```

```
// atributos do tipo Propriedade  
private String nome = "";  
private String cidade = "";  
private boolean ativo = false;
```

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

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

```
public void setCidade(String cidade) {  
    this.cidade = cidade;  
}
```

```
public String getCidade() {  
    return this.cidade;  
}
```

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

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

```
}
```



Membros Estáticos

Cada vez que um novo objeto é criado, é alocado um espaço para cada um de seus atributos

- Problema:
 - Como implementar, por exemplo, um atributo que funcione como uma variável global, de maneira que cada instância de objeto compartilhe e acesse o mesmo atributo em memória?
 - Para resolver este problema precisamos utilizar atributos estáticos

Atributos estáticos são declarados utilizando o modificador *static*

- Exemplo:
 - `public static double pi = 3.14;`



Atributos Estáticos

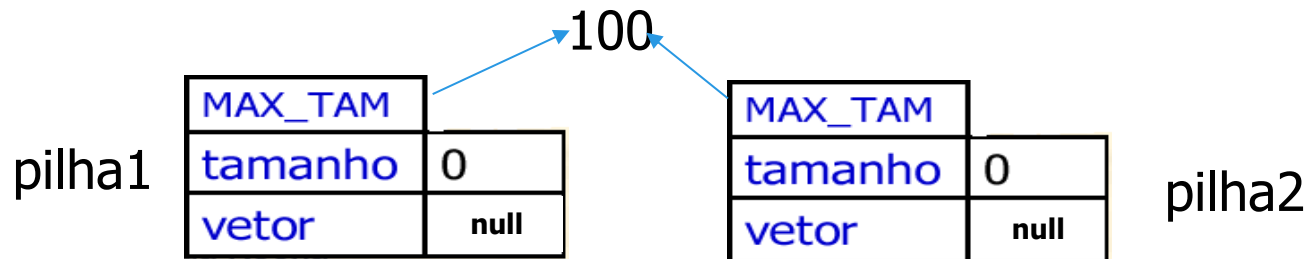
Vamos analisar um exemplo.. seja a seguinte declaração....

```
class Pilha {  
    private static int MAX_TAM =100;  
    private int tamanho;  
    private int [] vetor;  
}
```

Pilha pilha1 = new Pilha(); Pilha pilha2 = new Pilha();

Onde MAX_TAM armazena o máximo tamanho da pilha

Os objetos pilha1 e pilha2 terão as seguintes configurações de memória



- Se fizermos Pilha.MAX_TAM = 200, isso surte efeito em todos os objetos



Final

Outro comando interessante que podemos estudar é o comando *final*

Significa “Isto não pode ser mudado”;

- Pode ser usada em:
 - Dados (atributos / variáveis locais);
 - Métodos;
 - Classes.
- Objetivos:
 - Dependendo do contexto, o efeito é levemente diferente, mas no final temos uma **CONSTANTE!!!**



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Exercícios

Agora é hora de exercitar.....

Tente resolver os seguintes problemas...

- Em dupla
- Apresentar ao professor no final da aula
- Pontuação em Atividades em sala de aula...
- Faça o JAVADOC de todos os exercícios!!!



Exercício

Seja a classe Pessoa. Vamos implementar.

```
public class Pessoa  
{
```

```
    private String nome;  
    private int anoNasc;  
    private double altura;
```

Visível
Invisível

**Os métodos acessores
para nome poderiam ser:**

```
Pessoa (String nome, int anoNasc, double altura)
```

```
{  
    this.nome = nome;  
    this.anoNasc = anoNasc;  
    this.altura = altura;  
}  
public void Cresce(double m)  
{  
    altura = altura + m;  
}  
public int CalculaIdade()  
{/* implementação do algoritmo */  
}  
private int ObtemAnoCorrente()  
{/* código */  
}
```

```
public String getNome() // leitor/get  
{  
    return nome;  
}  
  
public void setNome(String nome) //modificador/set  
{  
    this.nome = nome;  
}
```

```
}
```

Calendar....

Atividade:

01: Implemente essa classe e crie uma outra classe (UsaPessoa) que instancie a classe Pessoa e acesse o atributo nome.

02: Implemente as funcionalidades que faltam.



Exercício

Crie uma classe chamada Calculus....

- Que tenha os métodos somar, subtrair, multiplicar e dividir
- Esses métodos sempre recebem 2 parâmetros
- E retornam a operação solicitada...

Logo depois crie uma classe programa UsaCalculus que utilize os métodos implementados

Obtenha os dados via Scanner



Exercício

Crie uma classe chamada Televisao de acordo com a representação abaixo.

Televisao
+marca:String +tamanho:int +tipo:String
+ligar():void +desligar():void +mudarCanal():void

- Os métodos simplesmente imprimem um texto (String) dizendo que eles fazem

Logo depois crie uma classe programa que utilize os atributos e métodos implementados



Exercício

Crie e implemente uma classe `PopulacaoBaratas` que simule o crescimento de uma população de baratas.

- A quantidade inicial da população de baratas é definido de forma Randômica. (pesquise o objeto `Random`...)
- O método `aumentaBaratas`, simula a proporção que a população de baratas vai se multiplicar.
- O método `spray` pulveriza as baratas com um inseticida e reduz a população em 10%.
- O método `getQtdBaratas` devolve o número atual de baratas.
- Implemente também uma classe que simule uma cozinha que tenha uma população de baratas
- Utilize a `aumentaBaratas`, utilize o `spray`, e imprima a contagem de baratas.



Exercício

Crie uma classe Pessoa com atributo Nome e Idade

Solicite ao usuário essas informações de um grupo de 5 pessoas.

- Pegue as informações via Scanner

Após o término da entrada, apresente:

- a média das idades,
- a maior idade,
- a menor idade,
- a quantidade de pessoas maior de idade.

Crie uma outra classe principal para usá-la



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Exercício

Verifique a validade de uma data de aniversário

- Solicite apenas o número do dia e do mês

Além de falar se a data está ok, informe também o nome do mês.

- Dica: meses com 30 dias: abril, junho, setembro e novembro.

Utilize a funcionalidade...



UNIVERSIDADE
VILA VELHA
ESPÍRITO SANTO

Exercício

Acrescente no exercício anterior a apresentação do signo do horóscopo da pessoa.

Inclua no exercício anterior a solicitação do ano de nascimento e também da data de hoje.

- A partir destas informações, apresente a idade atual desta pessoa.

Utilize a funcionalidade...