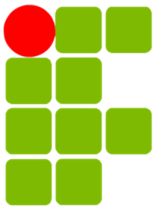


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus João Pessoa

Sistemas para Internet
Programação Orientada a Objetos

Fausto Maranhão Ayres

3
Fundamentos da POO

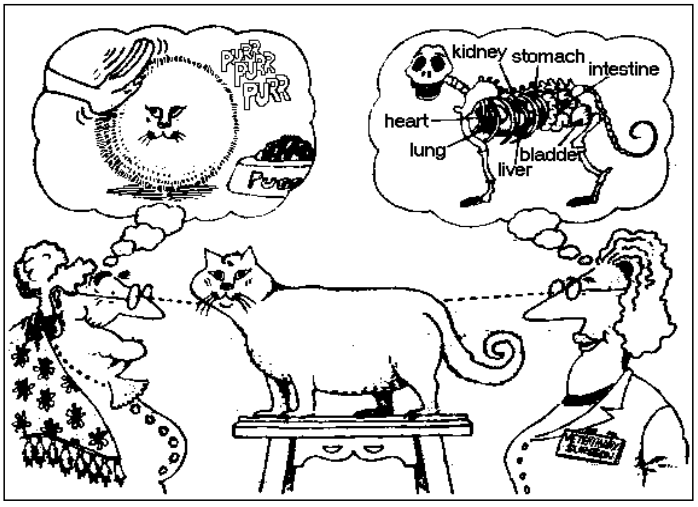
Paradigmas de Programação

Paradigmas de Programação

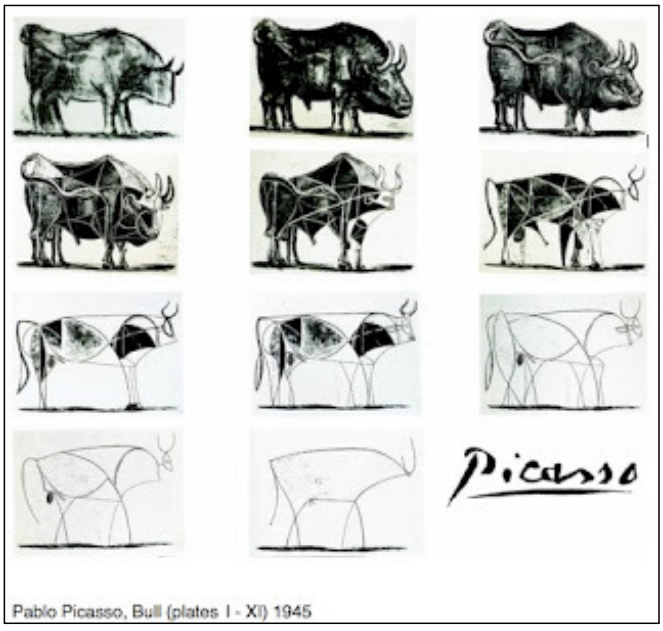
- **Programação orientada a procedimento**
- Programação orientada a função
- Programação orientada a lógica
- Programação orientada a *pattern matching*
- **Programação orientada a objeto**
- **etc**

Abstração

- Técnica para lidar com a **complexidade** de uma tarefa



Copyritght 1991 © Grady Booch



Pablo Picasso, Bull (plates I - XI) 1945

Abstração x Programação

Programação Procedimental

```
...
saldo1 = 0
saldo2 = 0
creditar(saldo1, 100)
debitar(saldo1, 20)
creditar(saldo2, 100)
transferir(saldo2, saldo1, 50)
print(saldo1)
print(saldo2)
```

POO

```
...
Conta conta1 = new Conta(...)
Conta conta2 = new Conta(...)
conta1.creditar(100)
conta1.debitar(20)
conta2.creditar(100)
conta2.transferir(conta1, 50)
println(conta1.getSaldo())
println(conta2.getSaldo())
```


Abstração x Programação

Programação	Linguagens
Orientado a procedimento	Fortran, Basic, Pascal, C, Cobol, Cliper ...
Orientado a objeto	Smaltalk, C++, Eiffel, Lua, Delphi, VB, Python, Java, C#, PHP, JavaScript, ...

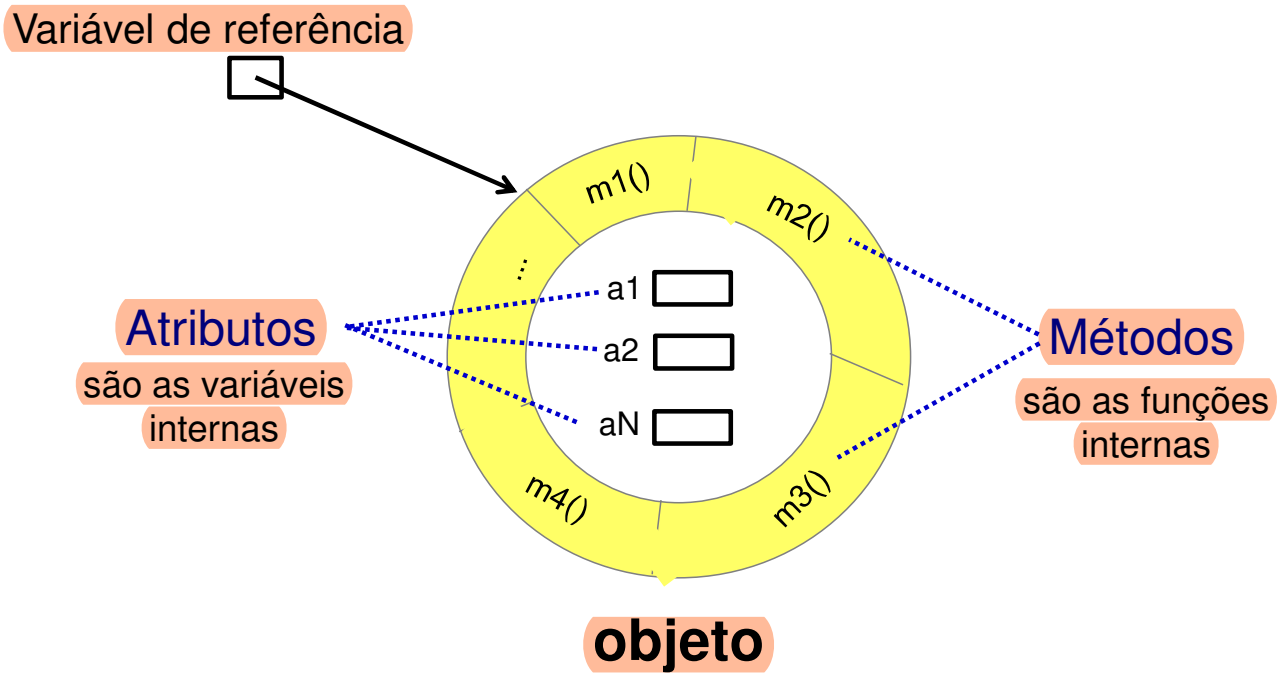
Classe e Objeto

Conceito

- Uma **classe** é a **descrição** de um *conjunto de objetos do mundo real* que possuem mesmos atributos (estrutura) e métodos (comportamento)

- **Objeto** é uma **instância** de uma classe

Representação do objeto



Etapas do desenvolvimento O.O.

Modelagem das classes

Especificar classes e seus relacionamentos

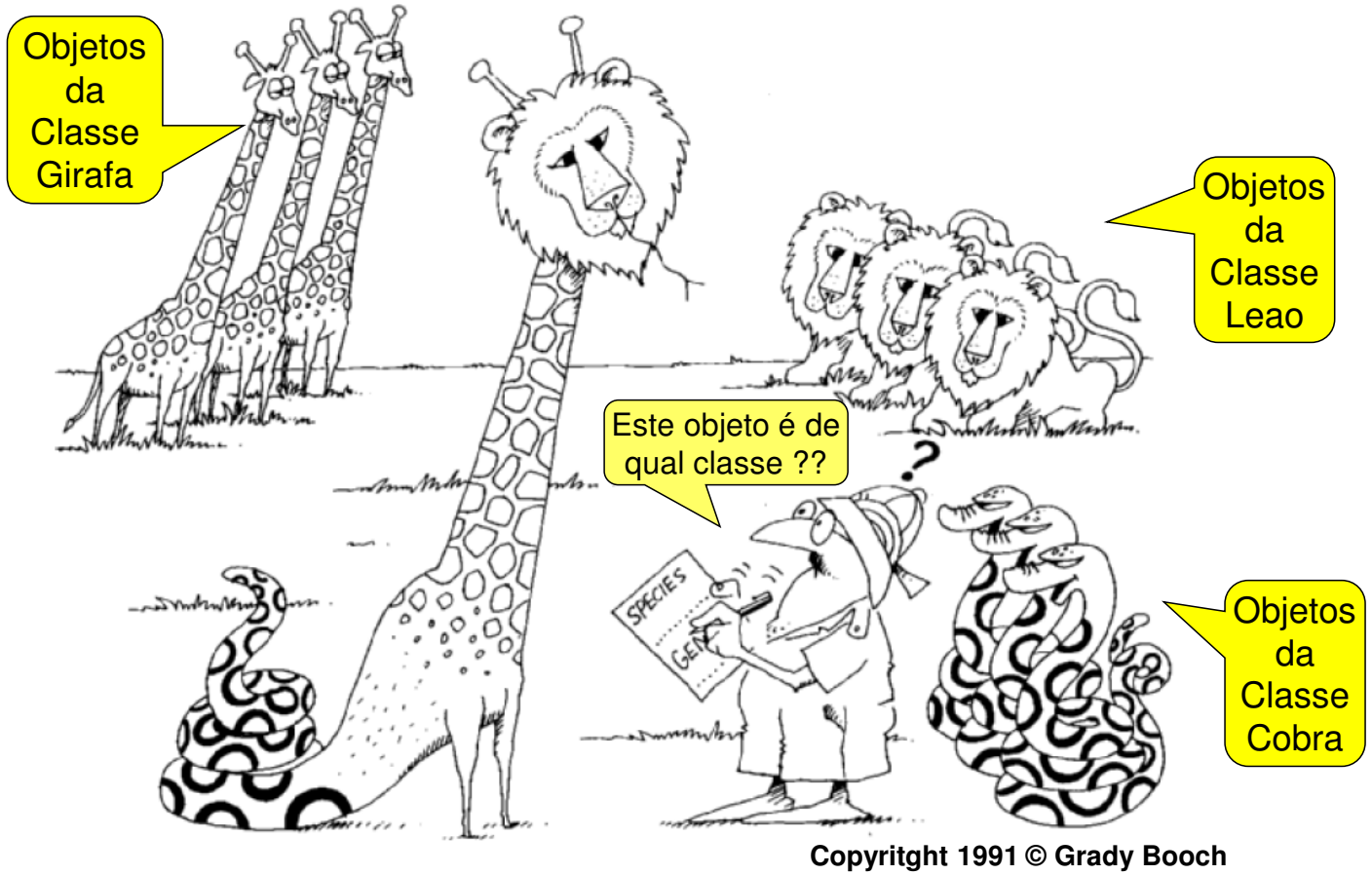
Implementação das classes

Usando uma linguagem orientada a objeto

Criação dos objetos

Teste e aplicações

Modelagem de classes (classificação)

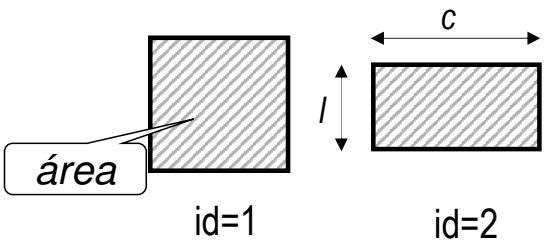


Copyritght 1991 © Grady Booch

Exemplo 1
Implementação da classe Retangulo

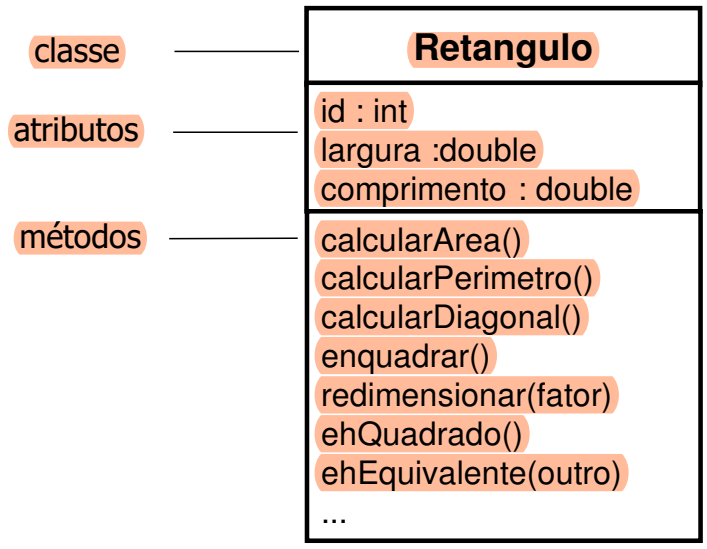
Modelagem da classe

- Figura geométrica de retângulo



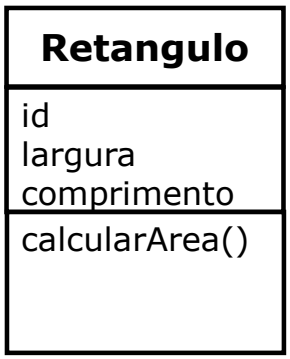
Objetos do mundo real

Diagrama UML



Implementação de Retangulo

UML



Retangulo.java

```
public class Retangulo
{ public int id;
  public double largura;
  public double comprimento;

  public double calcularArea(){
      return largura * comprimento;
  }
}
```

atributo

método

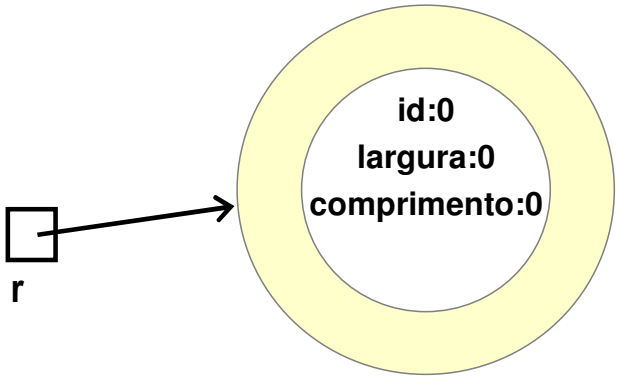
Criar objeto

```
Retangulo r = new Retangulo();
```

Construtor sem parametro

onde:

1. o operador **new** faz alocação dinâmica dos atributos
2. o método **construtor *default*** da classe, inicializa os atributos numéricos do objeto com 0
3. o endereço do objeto é armazenado na variável r

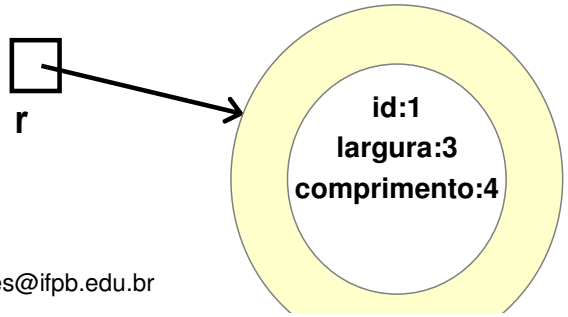


Atributos

Manipulação dos atributos

- Depois do objeto criado, pode-se manipular os atributos e chamar os métodos da classe

```
public class Teste{
    public static void main (String[] args) {
        Retangulo r = new Retangulo();
        r.id = 1;
        r.largura = 1;
        r.comprimento = 7;
        System.out.println(r.calcularArea());    //7.0
        r.largura = 3;
        r.comprimento = 4;
        System.out.println(r.calcularArea());    //12.0
    }
}
```



Adicionando construtores na classe

- Pode-se implementar vários métodos construtores com diferentes parâmetros, inclusive o construtor vazio

```
public class Retangulo {
    public int id;
    public double largura;
    public double comprimento;

    public Retangulo(int id, double largura, double comprimento) {
        this.id = id;
        this.largura = largura;
        this.comprimento = comprimento;
    }

    public Retangulo() { }

    public double calcularArea() {
        return largura * comprimento;
    }
```

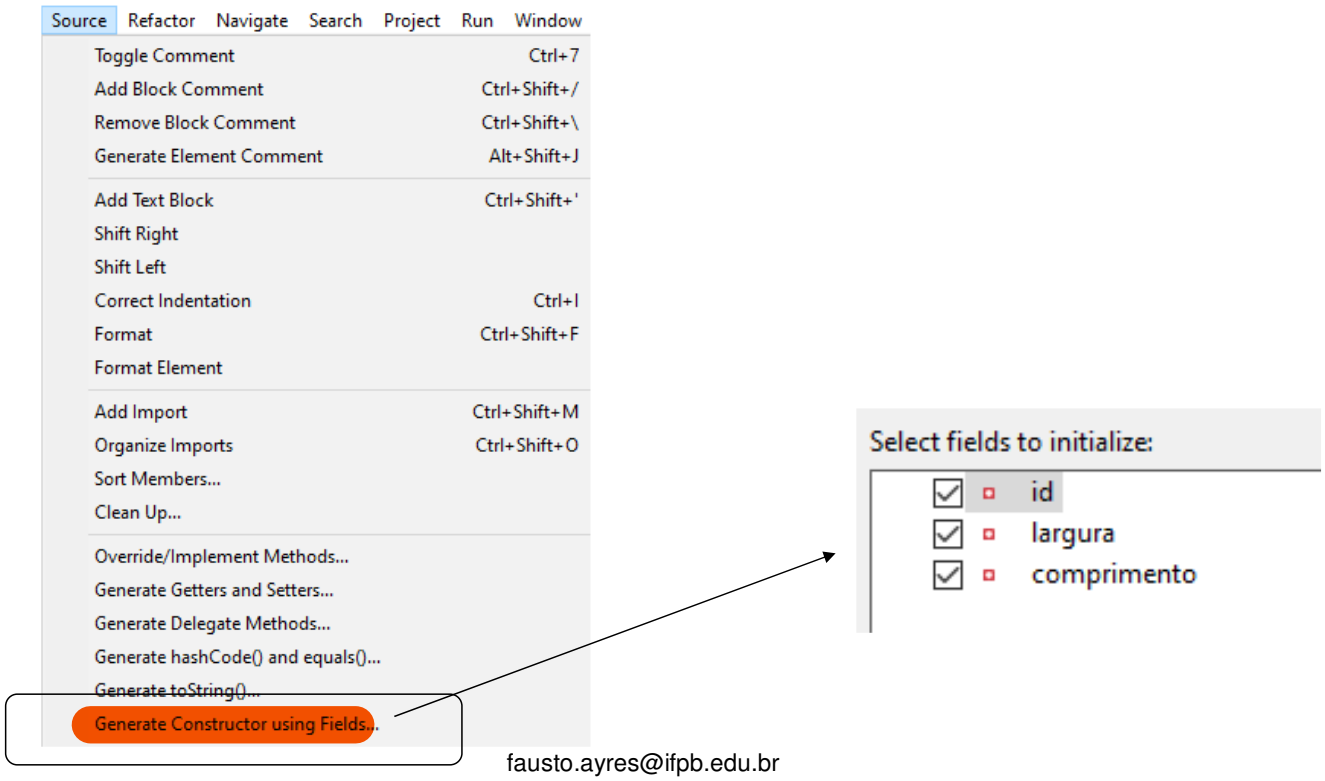
2 construtores

Usando os construtores

```
public class Teste1{
    public static void main (String[] args) {
        Retangulo r1 = new Retangulo(1, 3, 4);
        System.out.println(r1.calcularArea());  //12.0

        Retangulo r2 = new Retangulo();
        System.out.println(r2.calcularArea());  //0.0
    }
}
```


Dica eclipse

- É possível adicionar o construtor pelo eclipse:



Métodos

Chamada de métodos

- Individual

```
String nome;
nome = teclado.nextLine();
nome = nome.toUpperCase();
```

retorna String

retorna String

- Encadeada

```
nome = teclado.nextLine().toUpperCase();
```

retorna String

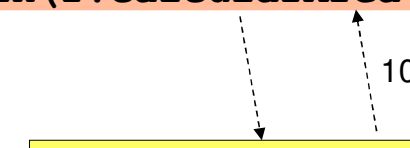
retorna String

```
double x = new Retangulo(1,3,4).calcularArea();
```

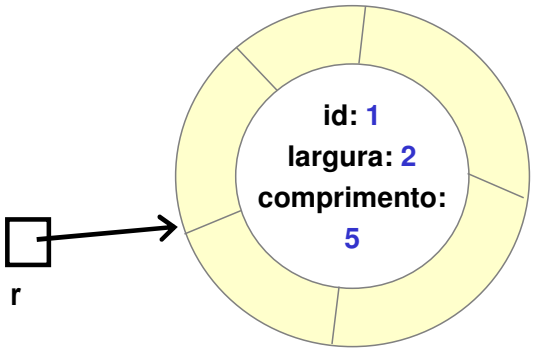
Chamada ao método calcularArea()

```
Retangulo r = new Retangulo(1,2,5);
System.out.println(r.calcularArea());
```

10




```
public double calcularArea(){
        return largura*comprimento;
}
```



Declaração de Método

- Os métodos se comportam como funções que recebem 0...N parâmetros e retornam (ou não) um resultado:

Modificador de acesso

tipo do resultado retornado

```
public tipo  nome  (lista de 0...N parâmetros)  {
    ...
    return resultado;
}
```

Não é necessário para o tipo **void**

Exemplo de método void (sem retorno)

- Executa uma tarefa sem retornar um resultado
  Ex:

```
public void enquadrar() {
    double media = (largura+comprimento) / 2;
    largura = media;
    comprimento = media;
}
```

Teste

```
Retangulo r = new Retangulo(1, 3, 5);
System.out.println(r.calcularArea());     //15.0
r.enquadrar();
System.out.println(r.calcularArea());     //16.0
```

exiba os atributos!

Exemplo de método boolean

```
public boolean ehQuadrado() {
    if (largura==comprimento)
        return true;
    else
        return false;
}
```

Teste

```
Retangulo r;
r = new Retangulo(1, 3, 3);
System.out.println(r.ehQuadrado()); // true
r = new Retangulo(2, 2, 7);
System.out.println(r.ehQuadrado()); // false
```

O método toString()

Converter o objeto para uma string

```
public String toString() {
    return  " id=" + id +
            " largura=" + largura +
            " comprimento=" + comprimento;
}
```

Teste

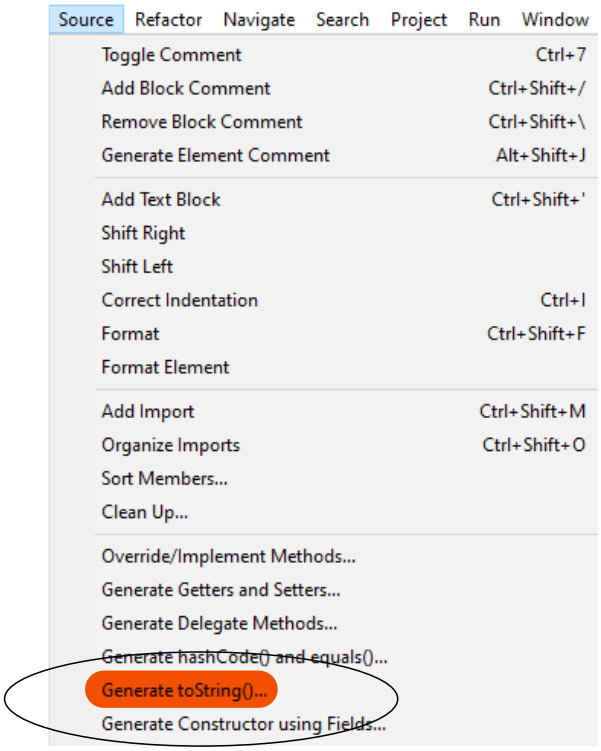
```
Retangulo r;
r = new Retangulo(1, 3, 5);
System.out.println(r);
```

chama automaticamente toString()

id=1 largura=3 comprimento=5

Dica eclipse

- É possível gerar toString() pelo eclipse



Exercícios

Exercício 1

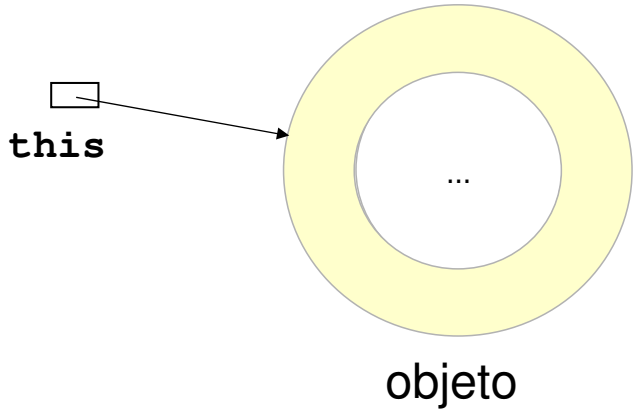
- Implemente a classe **Triangulo** com:
  - Atributos: base e altura
  - Métodos:
    - Construtor para inicializar os 2 atributos
    - calcularArea()
    - toString()

- Implemente a classe **TesteTriangulo** para criar um objeto Triangulo e aplicar os métodos sobre este objeto

Parâmetro implícito *this*

A variável **this**

- Contém a **referência** para o objeto envolvido na chamada de método



Uso da variável **this**

- É usada para distinguir um atributo do objeto de uma outra variável de mesmo nome

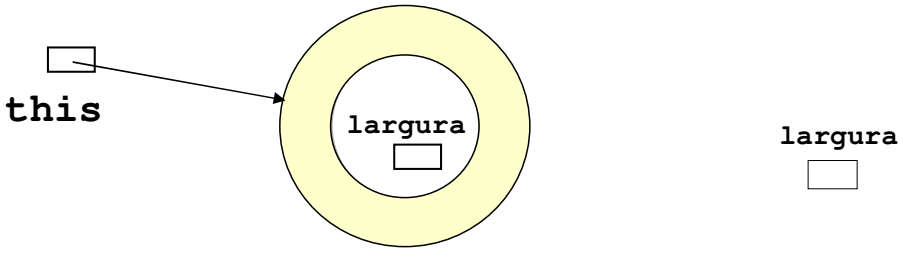
```
private double largura;
private double comprimento;

public Retangulo(double largura, 
    this.largura = largura;
```

atributo

parâmetro

atributo

parâmetro



Uso da variável **this**

- A variável **this** é implícita e pode ser ocultada no código-fonte.

```
public class Retangulo {
    public int id;
    public double largura;
    public double comprimento;

    public Retangulo(int id, double largura, double comprimento) {
        this.id = id;
        this.largura = largura;
        this.comprimento = comprimento;
    }

    public Retangulo() { }

    public double calcularArea() {
        return largura * comprimento;
    }
```

A variável **this** está implícita

Uso da variável **this**

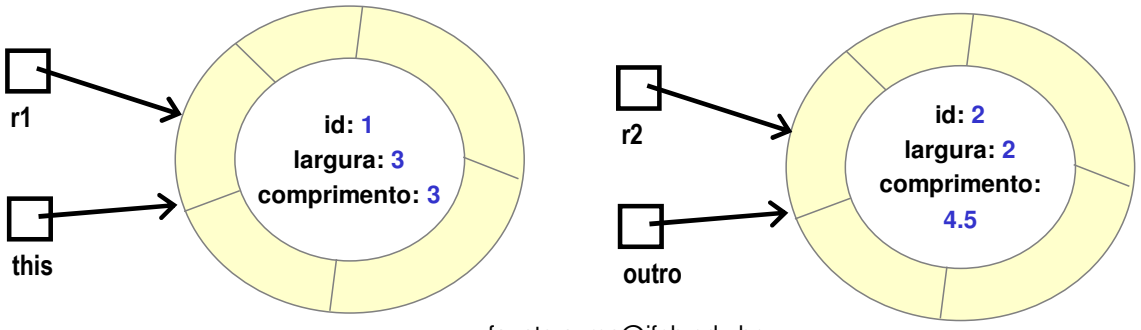
Ex: equivalência de área

```
public boolean ehEquivalente(Retangulo outro) {
    if (this.calcularArea() == outro.calcularArea())
        return true;
    else
        return false;
}
```

Compara a área dos dois objetos

Teste

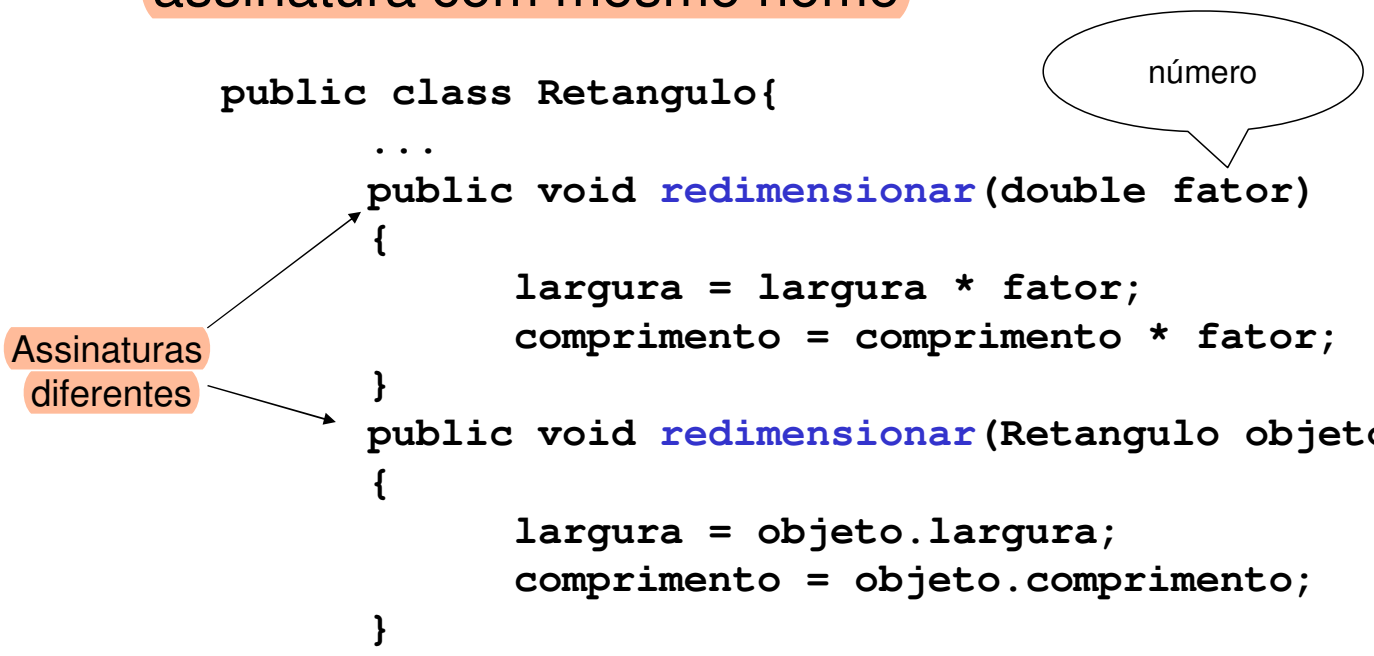
```
Retangulo r1,r2;
r1 = new Retangulo(1, 3, 3);
r2 = new Retangulo(2, 2, 4.5);
System.out.println(r1.ehEquivalente(r2)); // true
```



Sobrecarga

Sobrecarga de método

- Ocorre quando um método possui mais de uma assinatura com mesmo nome




```
public class Retangulo{
        ...
        public void redimensionar(double fator)
        {
                largura = largura * fator;
                comprimento = comprimento * fator;
        }
        public void redimensionar(Retangulo objeto)
        {
                largura = objeto.largura;
                comprimento = objeto.comprimento;
        }
```


Sobrecarga de método

Teste

```
Retangulo r1 = new Retangulo(1, 2, 4);
Retangulo r2 = new Retangulo(2, 3, 10);
System.out.println(r1.calcularArea()); //8
System.out.println(r2.calcularArea()); //30

r1.redimensionar(0.5);  //metade
System.out.println(r1.calcularArea()); //2

r2.redimensionar(2.0);  //dobro
System.out.println(r2.calcularArea()); //120

r2.redimensionar(r1);   //copia dados de r1
System.out.println(r2.calcularArea()); //2
```

Exercício 2

- Crie outro construtor de Retangulo com apenas um parâmetro, para inicializar a largura e o comprimento com o mesmo valor. Ex:

```
Retangulo r = new Retangulo(5);
System.out.println(r.calcularArea());  //25
```

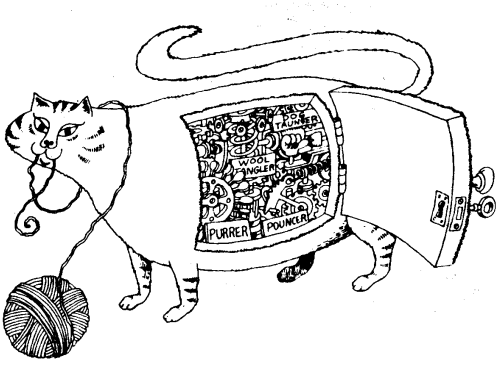

Encapsulamento

Acesso aos atributos públicos

- Os atributos declarados como ***public*** podem ser acessados diretamente pelo programa, mas isso **não é uma boa prática**, pois diminui a abstração de dados.
- Solução: Encapsulamento dos atributos

Encapsulamento dos atributos

- É o conceito da POO que impede o acesso direto aos atributos do objeto, visando aumentar a **abstração**



Fonte: Use a cabeça:Java

Modificador de acesso `private`

- Exemplo:

```
public class Retangulo {
  private double largura;
  private double comprimento;
   ...
```

`private` impede o acesso ao atributo fora de sua classe.

```
pubic static void main(...){
      Retangulo r1 = new Retangulo(1,5,30);
      r1.largura = 8;           //erro
      r1.comprimento = 10;      //erro
      System.out.println(r1.largura);      //erro
      System.out.println(r1.comprimento); //erro
      ...
```


Métodos get/set

- O acesso a um atributo deve ser feito sempre que possível via **getAtributo()** e **setAtributo()**

Ex:

```
Retangulo r1 = new Retangulo(1,5,30);
r1.setLargura(8);
r1.setComprimento(10);

System.out.println(r1.getLargura() );
System.out.println(r1.getComprimento() );
```

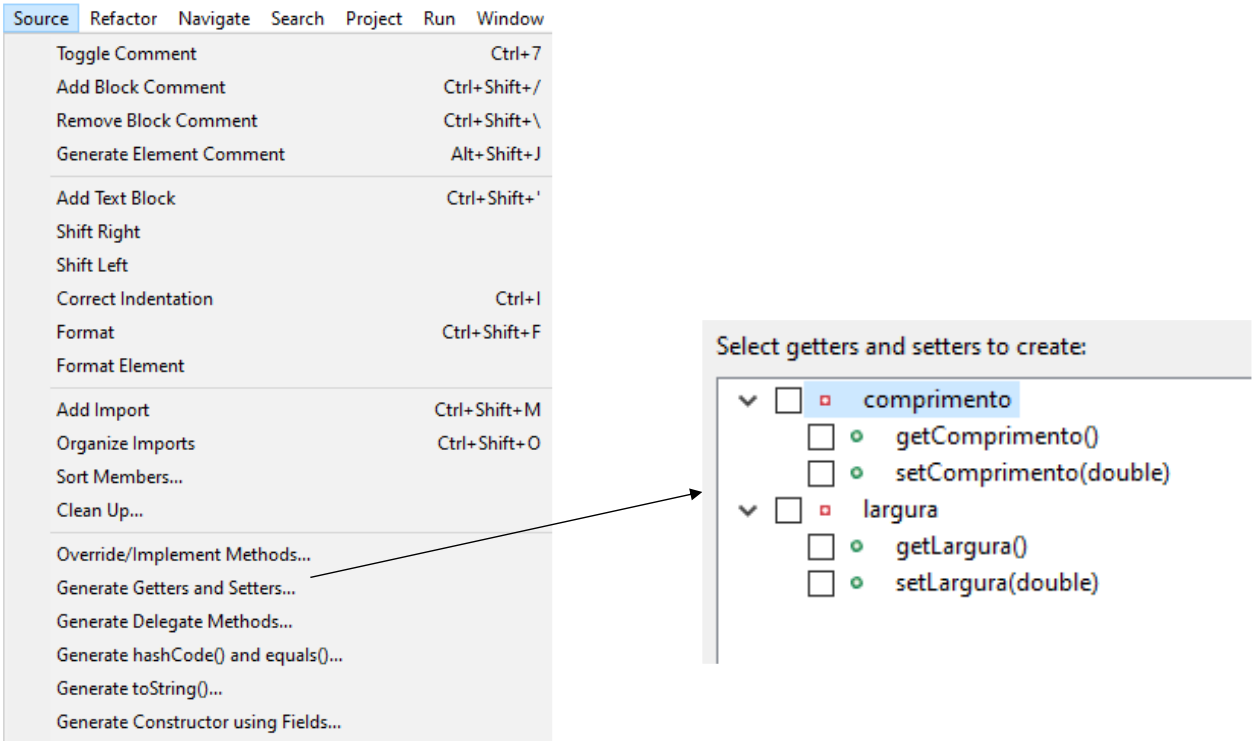
Implementação de get/set

- Retangulo

```
public class Retangulo{
  private double largura;
  private double comprimento;
  ...
  public double getLargura() {
      return this.largura;
  }
  public double getComprimento()  {
      return this.comprimento;
  }
  public void setLargura(double largura) {
      this.largura=largura;
  }
  public void setComprimento(double comprimento) {
      this.comprimento=comprimento;
  }
}
```


Dica eclipse

- É possível gerar get/set através do eclipse:



Exemplo de encapsulamento

- Obter os dados internos de uma matrícula de aluno

```
public static void main(String[] args) {
    Matricula matricula = new Matricula("20222370187");
    String ano = matricula.getAno();
    String periodo = matricula.getPeriodo();
    String codigoCurso = matricula.getCodigoCurso();
    String sequencia = matricula.getSequencia();

    System.out.println(ano);            //2022
    System.out.println(periodo);        //2
    System.out.println(codigoCurso);    //37
    System.out.println(sequencia);      //0187
}
```


Exemplo de encapsulamento

```
public class Matricula {
    private String matricula;

    public Matricula(String texto) {
        matricula = texto;
    }
    public String getAno() {
        return matricula.substring(0, 4) ;
    }
    public String getPeriodo() {
        return matricula.substring(4, 5) ;
    }
    public String getCodigoCurso() {
        return matricula.substring(5, 7) ;
    }
    public String getSequencia() {
        return matricula.substring(7) ;
    }
```

Outro exemplo de encapsulamento

- Criar uma janela desktop

```
import javax.swing.JFrame;
public class Janela {
    public static void main(String[] args) {
        JFrame janela = new JFrame();
        janela.setTitle("exemplo de janela");
        janela.setSize(500, 300);   //width, heigth
        janela.setResizable(false);
        janela.setVisible(true);
    }

}
```


Exercício 3

- A classe **Aluno**:
  - Atributos: nome, nota1, nota2 (0 a 100)
  - Métodos:
    - Construtor para inicializar nome e as 2 notas
    - getMedia() retorna a média aritmética (inteira)
    - getSituacao(), retorna “aprovado”, “reprovado” e “final” de acordo com a media
    - toString()

- A classe **TesteAluno**

Aluno a1 = new Aluno(“joao”, 100, 70) ;
System.out.println(a1);

Exemplo 2:
Implementação da classe Conta

Modelagem da classe

- conta de um banco digital

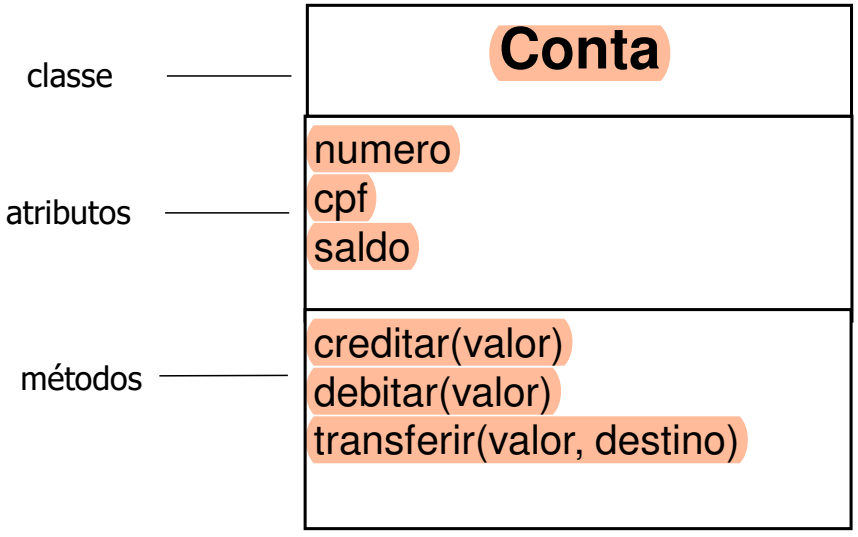


Diagrama de classe UML

Implementação

Conta
numero cpf saldo
creditar(valor) debitar(valor) transferir(…)

```
public class Conta {
        private String numero;
        private String cpf;
        private double saldo;

        public Conta(String numero, String cpf) {
                this.numero = numero;
                this.cpf = cpf;
                this.saldo = 0;
        }
        public void creditar(double valor) {
                saldo = saldo + valor;
        }
        public void debitar(double valor) {
                saldo = saldo - valor;
        }
        //...demais métodos
}
```

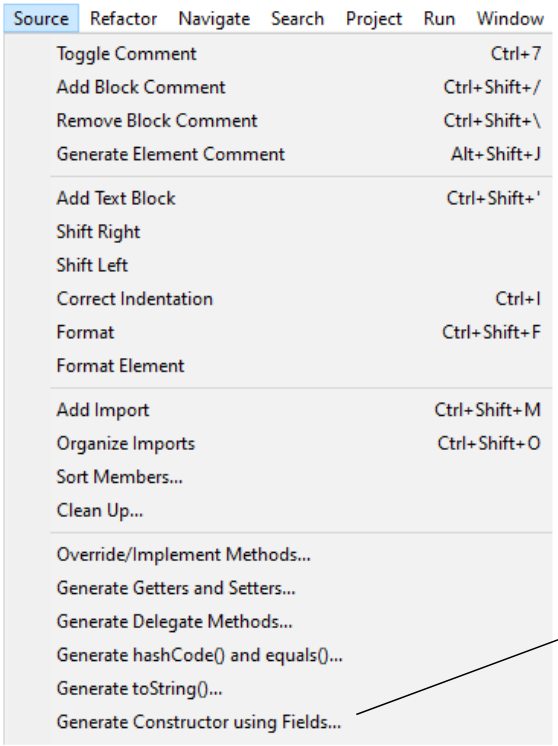
Atributos

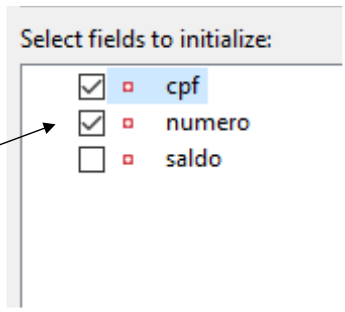
construtor

A variável *this* está implícita

Dica eclipse

- É possível inserir o construtor pelo eclipse:

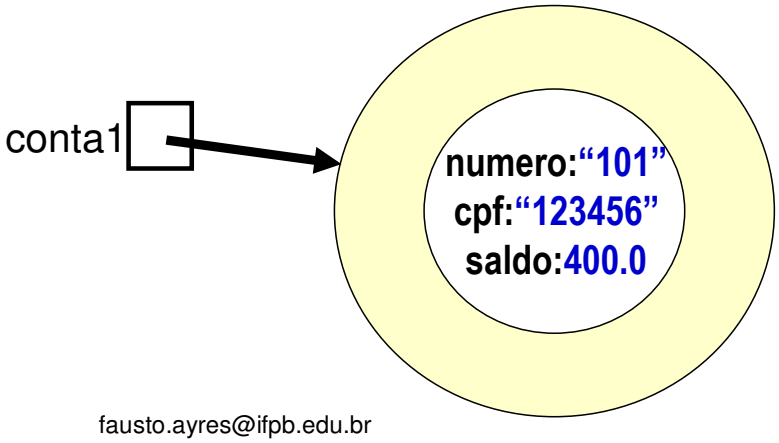




O saldo deve ser inicializado com 0

Criar objeto

```
public class TesteConta{

 public static void main (String[] args) {
   Conta conta1 = new Conta("101","123456");
   conta1.creditar(300.0);
   conta1.debitar(100.0);
   conta1.creditar(200.0);

   System.out.println(conta1.getSaldo());  //400
 }
}
```



Chamada ao método **creditar**

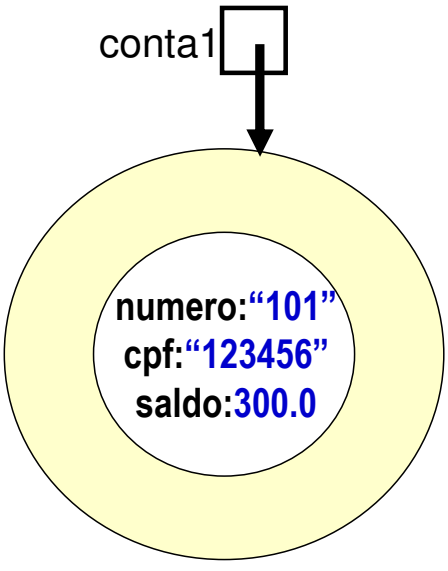
- O método creditar modifica o atributo saldo

Teste:

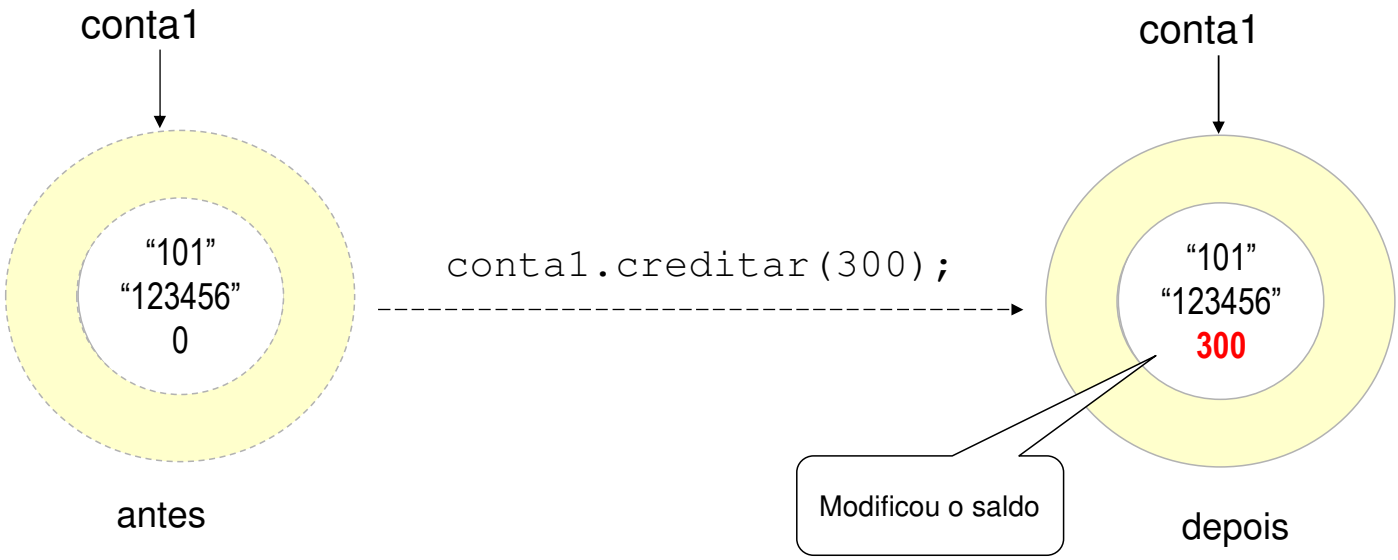
```
conta1.creditar(300);
```

```
public void creditar(double valor){
        saldo = saldo + valor;
}
```

saldo = 0 + 300



Chamada ao método **creditar**



Chamada ao método **debitar**

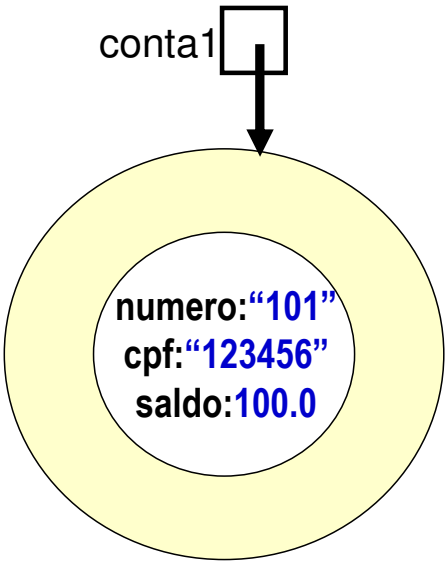
- O método debitar modifica o atributo saldo

Teste:

```
conta1.debitar(200);
```

```
public void debitar(double valor){
        saldo = saldo - valor;
}
```

saldo = 300 - 200



objeto após debitar

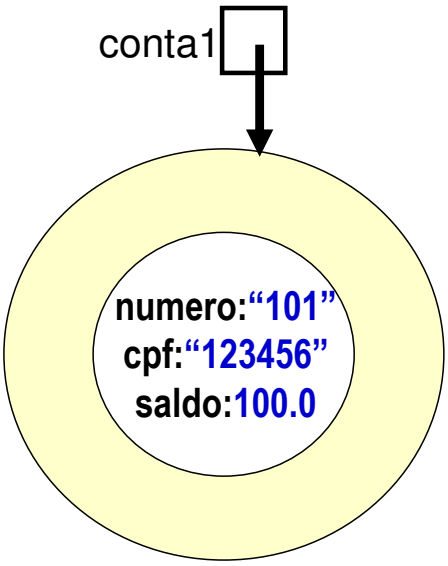
método **toString**

Converter o objeto para uma string

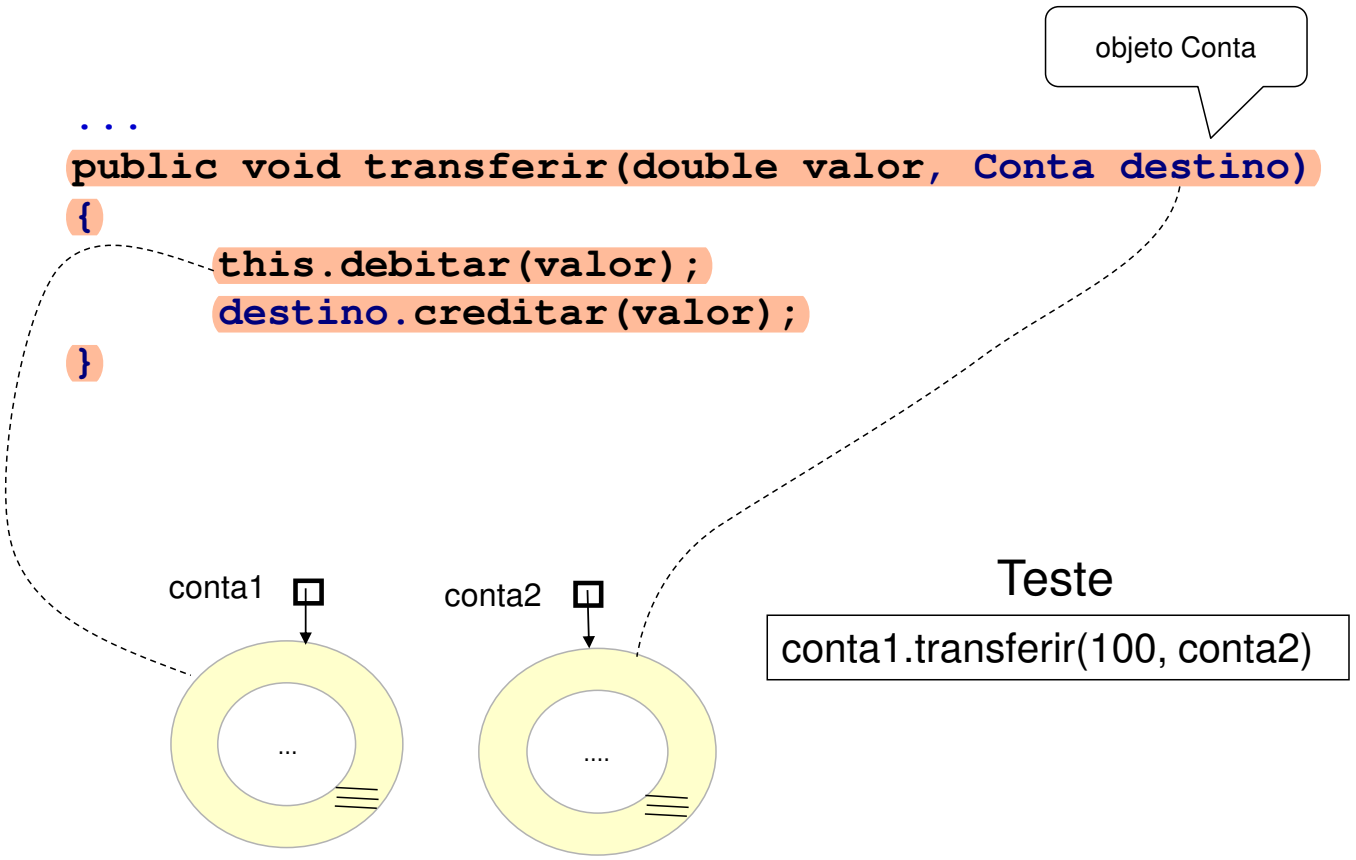
```
public String toString() {
    return  " numero=" + numero +
            " cpf=" + cpf +
            " saldo=" + saldo ;
}
```

```
System.out.println(conta1);
```

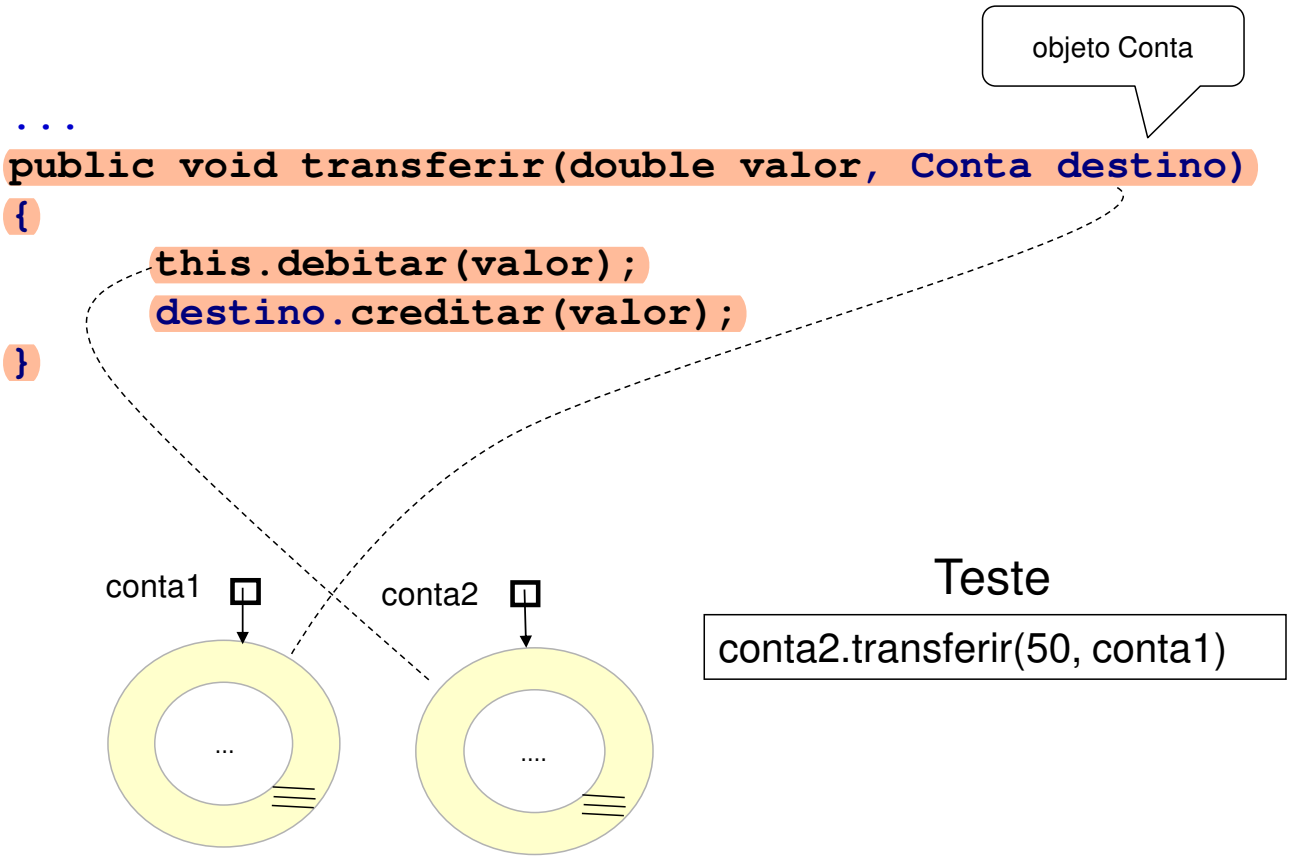
numero=101 cpf=123456 saldo=100



Uso da variável **this**



Uso da variável **this**



Tópicos adicionais

Visibilidade de atributos e métodos

- Os atributos e métodos podem ser declarados em **qualquer lugar** dentro da classe, pois são acessíveis em qualquer lugar da classe

Ex:

```
public class Conta {
        public void creditar(double valor) {...}
        public Conta(String n, String c) {...}
        public String numero;
        public String cpf;
        public void debitar(double valor) {...}
        public double saldo;
        ...
}
```


Inicialização default dos atributos

- Antes de executar o construtor, a JVM inicializa automaticamente todos os atributos do objeto com os seguintes valores *default:*

int	0
double	0.0
boolean	false
String	null (qualquer classe)

- Entretanto, pode-se inicializar manualmente

```
public class Conta {
        public String numero;
        public String cpf;
        public double saldo = 1000.0;
        ...
```

Inicialização pode ser feita aqui ou dentro do construtor

Argumentos variáveis (*varargs*)

- Pode-se chamar um método com quantidade variável de argumentos

```
conta1.creditar(100.0);                    //1
conta1.creditar(200.0, 50.0);              //2
conta1.creditar(100.0, 300.0, 100.0);//3
```

- Para tanto, o método deve ter o parâmetro declarado com a expressão "tipo..."

```
public class Conta {
      ...
      public void creditar(double... lista) {
            for(double valor : lista)
                  saldo = saldo + valor;
      }
```

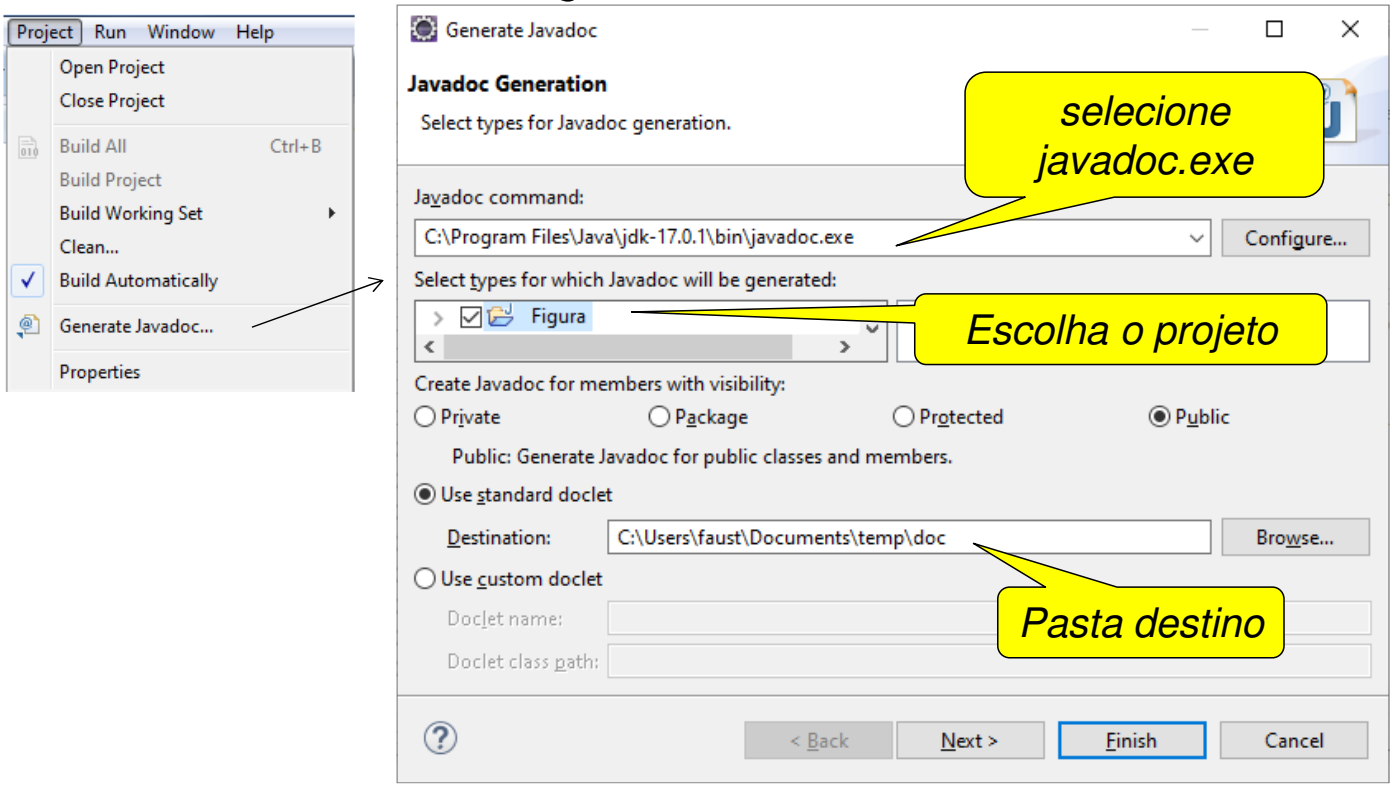
O parâmetro lista é um array

Tipos de modificadores de acesso

- Java possui 4 modificadores de acesso que determinam a forma de acesso a um atributo ou método:

Modificador	Permite acesso
private	apenas dentro da própria classe
public	em qualquer classe
protected	apenas na herança de classe
"ausente"	apenas dentro de um mesmo package (package private)

Documentação HTML das classes



São gerados arquivos HTML e o *index.html*

Exercícios

Exercicio 4

- Implemente o método **vazia()** na classe Conta que retorna *true* se o saldo estiver 0

```
public class TesteConta{

 public static void main (String[] args) {
      Conta conta1 = new Conta("333","123456");
      conta1.creditar(300.0);
      conta1.debitar(200.0);
      conta1.debitar(100.0);
      System.out.println(conta1.vazia()); //true
 }
}
```


Exercicio 5

- Implemente o método **clonar()** na classe Conta que retorna um objeto cópia do objeto original

```
public class TesteConta{

 public static void main (String[] args) {
      Conta conta1 = new Conta("333","123456");
      conta1.creditar(300.0);
      Conta conta2 = conta1.clonar();

      System.out.println(conta1); //333,123456,300
      System.out.println(conta2); //333,123456,300
 }
}
```

Exercício 6

- Crie a classe **AlunoFlex**, considerando
  - que o aluno possui várias notas (double... notas)
  - que o construtor vai receber uma quantidade *indeterminada* de notas, além do nome
  - Adaptar os demais métodos de Aluno

Teste

```
AlunoFlex a1 = new AlunoFlex(“joao”, 100, 70) ;
AlunoFlex a2 = new AlunoFlex(“maria”, 90, 50, 70, 80) ;
System.out.println(a1.getNome() + a1.getMedia() + a1.getSituacao());
System.out.println(a2.getNome() + a2.getMedia() + a2.getSituacao());
```


Exercício 7

- Criar classe **Produto**
  - Atributos: nome, estoque e preço
  - Métodos: Construtor , gets/sets, toString()

- Criar classe **Venda**
  - Atributos: data, nome, quantidade, valor
  - Métodos:
    - construtor(data, produto, quantidade)
      - Inicializa os atributos, calcula valor da venda e decrementa o estoque do produto
    - toString()

Teste

```
Produto p1 = new Produto("arroz", 10, 4.5);

System.out.println("\nestoque antes da venda");
System.out.println(p1);

Venda v1 = new Venda("31/08/2022", p1, 2);
System.out.println(v1);

System.out.println("\nestoque depois da venda");
System.out.println(p1);          //estoque 8
```


Ciclo de vida dos objetos

Ciclo de vida de um objeto

1. Instanciação: o objeto é criado na memória e passa a ser referenciado por uma variável de referência;
2. Uso: manipulamos o objeto através de seus métodos
3. Destruição: quando o objeto não é mais referenciado (acessível) ele torna-se elegível para a **coleta de lixo**

Coletor de Lixo:

serviço que limpa a memória ocupada pelos objetos inacessíveis (quando necessário)

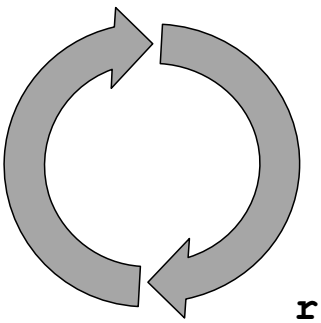
Esquema

Declaração da variável de referência (não existe objeto ainda)

```
Retangulo r;
```

Destruição (descarta referência)

```
r = null;
```



Instanciação (cria referência)

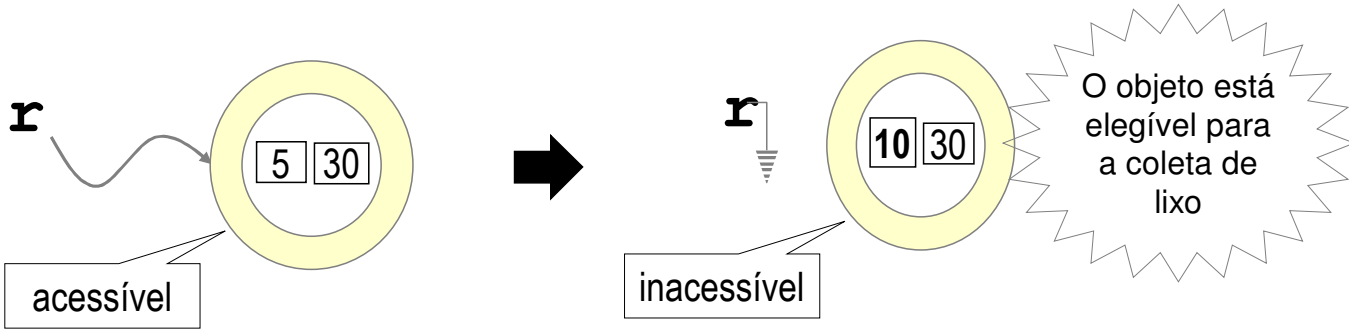
```
new Retangulo(1,5,30);
```

Uso (acessa referência)

```
r.setLargura(10);
```

Atribuição (guarda referência)

```
r = new Retangulo(5,30);
```



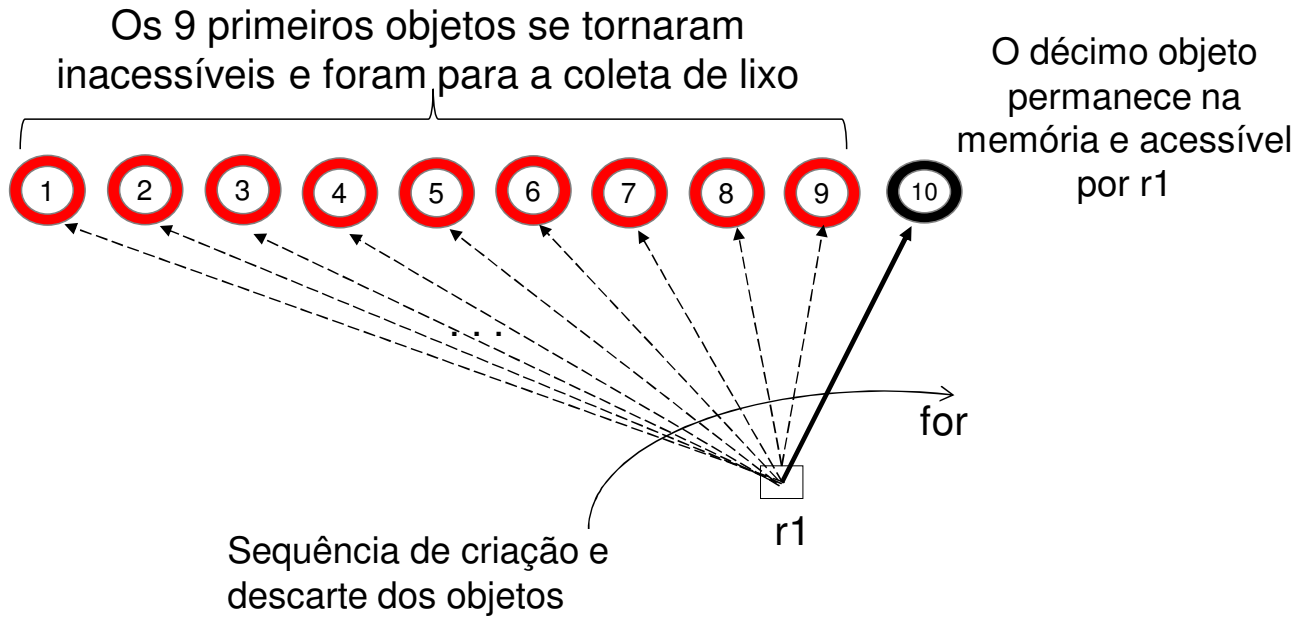
Exemplo

- O trecho de abaixo cria **10** objetos e descarta **9**

```
Retangulo r1 = null;
for(int i=1; i<=10; i++){
   r1 = new Retangulo(i,i,i);  //cria dez objetos
   System.out.println(r1.calcularArea());
}
System.out.println(r1.getId());    // 10
```

- No final do loop, apenas o último objeto (id=10) permanece na variável r1.
  - Onde estão os 9 primeiros objetos descartados?
  - Resposta: no lixo!

Criação e descarte dos objetos

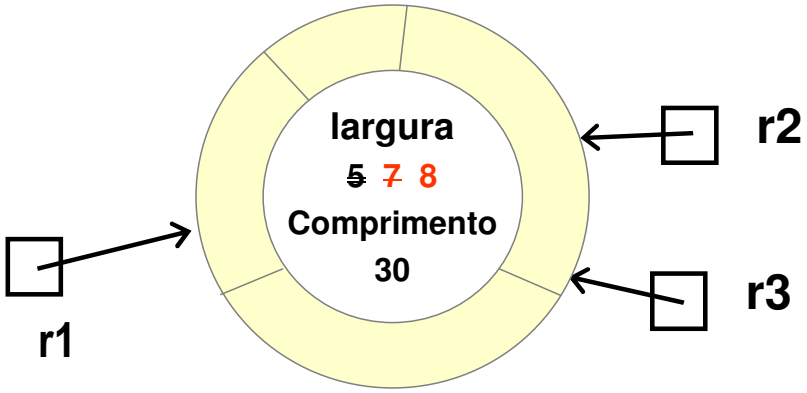


Várias variáveis para um objeto

- Pode-se ter várias referências para um objeto

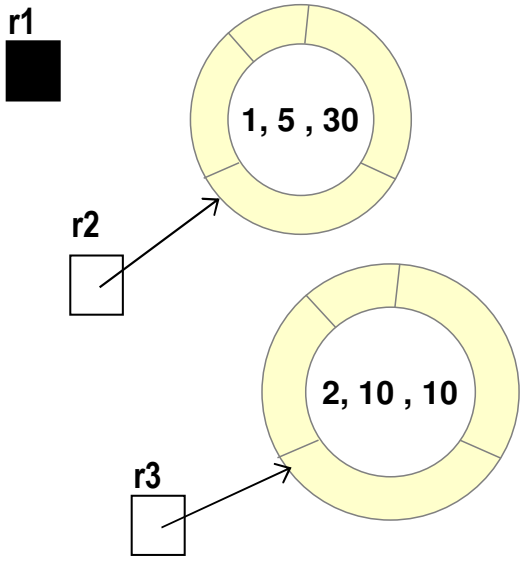
```
Retangulo r1,r2,r3;
r1 = new Retangulo(1,5,30);
r2=r1;
r3=r1;
r3.setLargura(7);
r2.setLargura(8);
...println(r1.calcularArea());   //240
...println(r2.calcularArea());   //240
...println(r3.calcularArea());   //240
```

3 variáveis
1 objeto



Exemplo

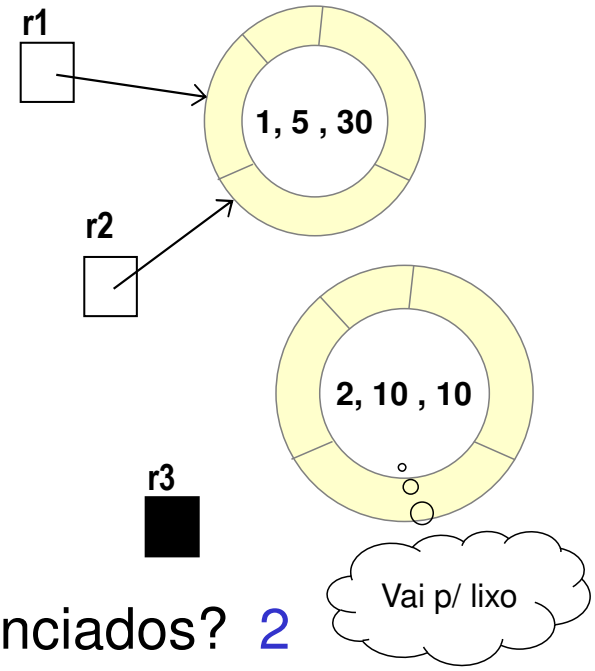
```
Retangulo r1,r2,r3;
r1 = new Retangulo(1, 5,30);
r2 = new Retangulo(2, 10,10);
r3=r2;
r2=r1;
r1=null;
```



- Quantos objetos foram instanciados? 2
- Quais objetos foram para a coleta de lixo? nenhum

Exemplo

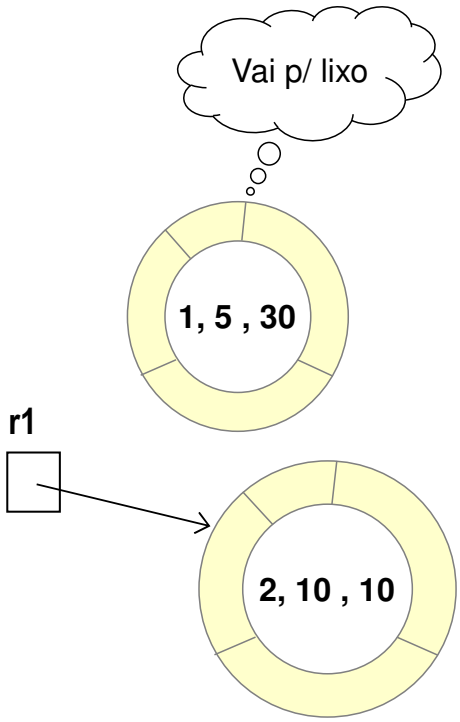
```
Retangulo r1,r2,r3;
r1 = new Retangulo(1,5,30);
r2 = new Retangulo(2,10,10);
r3=r2;
r2=r1;
r3=null;
```



- Quantos objetos foram instanciados? 2
- Quais objetos foram para a coleta de lixo? Id=2

Exemplo

```
Retangulo r1;
r1 = new Retangulo(1,5,30);
r1 = new Retangulo(2,10,10);
```



- Quantos objetos foram instanciados? 2
- Quais objetos foram para a coleta de lixo? id=1