Orientação a Objetos com Java

3ª aula

Prof. Douglas Oliveira
douglas.oliveira@prof.infnet.edu.br

Construtores

- Quando criamos um objeto:
  1. Precisamos definir os valores de seus atributos;
  2. Evitar que os objetos sejam criado com valores de atributos inválidos.
- No nosso exemplo, não queremos um Retangulo com largura e altura igual a zero!
- Para definir como as instâncias de uma classe podem ser criadas usamos construtores. Assim, podemos dizer que Construtores são métodos especiais usados para criar uma instância de uma classe (objeto).
- Podem existir várias formas diferentes de criarmos um objeto. Assim, podemos ter vários construtores diferentes para uma mesma classe.

Construtores

- Sintaxe básica de construtores:

```
public NomeDaClasse ([parâmetros]) [throws exceções]
{
    [ corpo do construtor ]
}
```

- Observe que:
  - ✔ Construtores devem ter, obrigatoriamente, o mesmo nome da classe.
  - ✔ Construtores não tem nenhum tipo de retorno.
  - ✔ Construtores podem receber parâmetros que serão usados na inicialização do objeto.

Construtores

- Quais construtores podemos definir para a classe Retangulo ? Lembre-se que a instância deve ficar em um estado consistente.

Construtores

- Quais construtores podemos definir para a classe Retangulo ? Lembre-se que a instância deve ficar em um estado consistente.

1. Construir um retângulo definindo sua posição (x, y), largura e altura.

- Alguma outra ideia ?

Construtores

- Construtor para a classe Retângulo:

```
package principal;
public class Retangulo {
  private int x;
  private int y;
  private int largura;
  private int altura;
  public Retangulo(int x, int y, int largura, int altura) {
    this.x = x;
    this.y = y;
    this.largura = largura;
    this.altura = altura;
  }
}
```

Inicializa todos os atributos do **Retangulo**.
O **this** é usado para referenciar o próprio objeto. Dessa forma diferenciamos o **atributo largura** do **parâmetro largura**.

Construtores

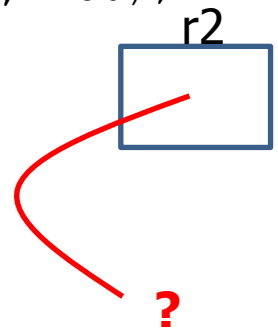
- Como ocorre a criação do objeto com operador new e construtor ?

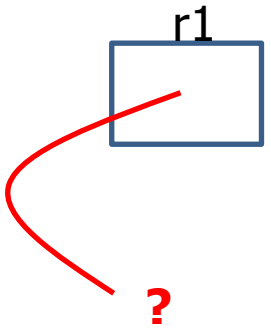
```
Retangulo r1, r2;

r1 = new Retangulo(10, 20, 100, 200);

r2 = new Retangulo(5, 15, 50, 150);
```

Quando definimos as variáveis os objetos ainda não existem.





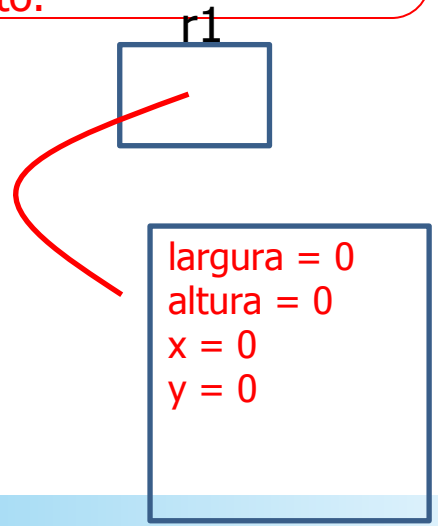
Construtores

- Como ocorre a criação do objeto com operador new e construtor ?

```
Retangulo r1, r2;

r1 = new Retangulo(10, 20, 100, 200);
```

O operador **new** cria o **objeto** e, em seguida, inicializa os atributos do objeto.



Construtores

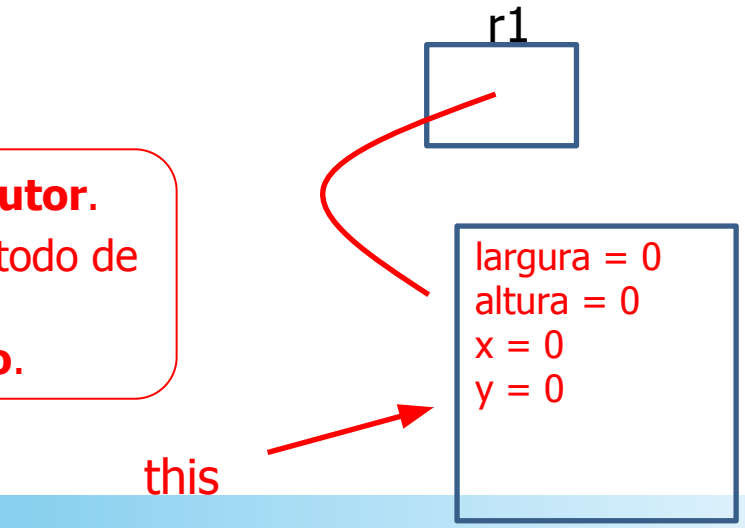
- Como ocorre a criação do objeto com operador new e construtor ?

```
Retangulo r1, r2;

r1 = new Retangulo(10, 20, 100, 200);
```

Depois é chamado o **construtor**.

Quando executamos um método de um objeto, a referência **this** referencia **o próprio objeto**.



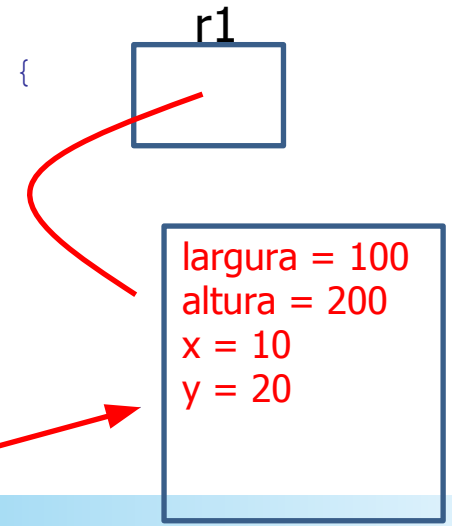
Construtores

- Como ocorre a criação do objeto com operador new e construtor ?

```
Retangulo r1, r2;

r1 = new Retangulo(10, 20, 100, 200);

public Retangulo(int x, int y, int largura, int altura) {
    this.x = x;
    this.y = y;
    this.largura = largura;
    this.altura = altura;
}
```



A referência **this** é usada para informar que estamos acessando o **atributo largura** ao invés do **parâmetro largura**.

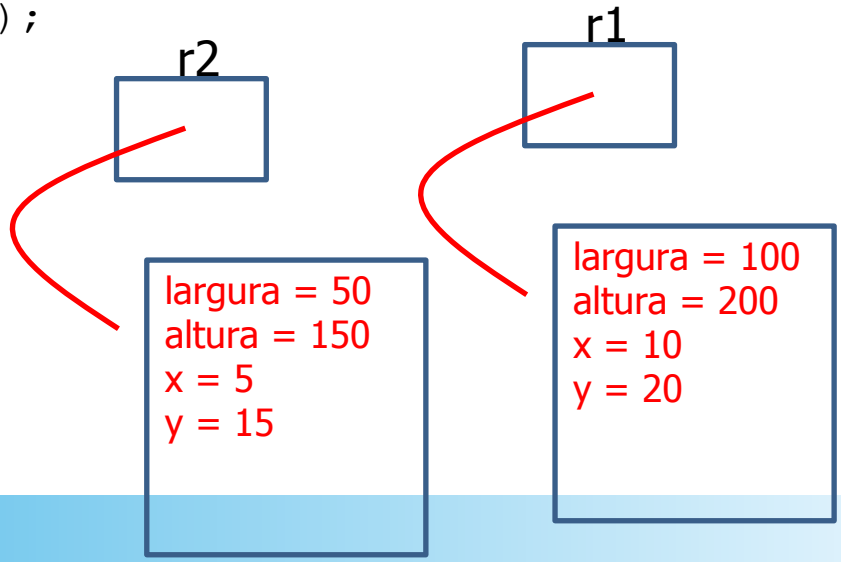
Construtores

- Como ocorre a criação do objeto com operador new e construtor ?

```
Retangulo r1, r2;
r1 = new Retangulo(10, 20, 100, 200);

r2 = new Retangulo(5, 15, 50, 150);
```

Quando criamos outro Retangulo, todo esse processo se repete.



Construtores

- Podemos definir mais de um construtor para a mesma classe, desde que eles tenham assinaturas diferentes;

```
package principal;
public class Retangulo {
  private int largura;
  private int altura;
  private int x;
  private int y;
  public Retangulo(int x, int y, int largura, int altura) {
    this.x = x;
    this.y = y;
    this.largura = largura;
    this.altura = altura;
  }
  public Retangulo(int x, int y, int lado) {
    this.x = x;
    this.y = y;
    this.largura = lado;
    this.altura = lado;
```

Os dois construtores são válidos porque tem **assinaturas** diferentes, ou seja, a quantidade ou tipo dos parâmetros são diferentes.

Exercícios

- Exercício 16: leia do usuário a posição (x, y) e os valores de largura e altura. Em seguida, crie um novo Retângulo usando essas informações.

Construtores

- Resumo sobre Construtores:
  - ✔ São métodos especiais usados para inicializar a instância com os valores desejados.
  - ✔ Construtores não retornam nenhum valor.
  - ✔ Ao criar um construtor, nós definimos que parâmetros deverão ser passados para que o objeto seja criado de forma coerente.
  - ✔ Podemos criar vários construtores para uma mesma classe.
  - ✔ Construtores não podem ser chamados diretamente pelo nosso código.
  - ✔ Construtores são chamados automaticamente pela JVM, de acordo com o tipo dos parâmetros que estamos passando no operador new.
  - ✔ Construtores são chamados após a inicialização dos atributos.
  - ✔ O construtor sem parâmetros é chamado de construtor *default*.
  - ✔ Quando não definimos nenhum construtor para a classe, o Java gera automaticamente um construtor *default*.

Métodos de Acesso

- Se os atributos são privados, como lemos ou alteramos os seus valores?

  ✔ Resposta: métodos de acesso !

Métodos de Acesso

- Métodos de acesso são usados para ler ou alterar atributos e usam os seguintes padrões:

```
public tipo  get<nome do atributo>()
{
      return atributo;
}
public void  set<nome do atributo>(tipo novoValor)
 {
      atributo = novoValor;
}
```

Métodos de Acesso

- Exemplo: métodos get e set para o atributo largura.

```
public int getLagura() {
    return largura;
}
public void setLargura(int largura) {
    this.largura = largura;
}
```

Métodos de Acesso

- Exercício 17: altere o exercício 16 e crie métodos de acesso para a classe Retangulo. Use esses métodos de acesso para alterar os atributos do retângulo.

Métodos de Acesso

```
package principal;
public class Retangulo {
  private int x, y, largura, altura;

  public int getLargura() { return largura; }
  public int getAltura()  { return altura;  }
  public int getX() { return x; }
  public int getY() { return y; }

  public void setLargura(int larg) { if (larg > 0) largura = larg; }

  public void setAltura(int alt) { if (alt > 0) altura = alt; }

  public void setX(int x) { this.x = x; }

  public void setY(int y) { this.y = y; }
}
```

Os atributos são privados, então precisamos dos métodos de acesso para permitir que estes sejam lidos ou alterados.

Repare que, com os métodos de acesso, podemos impedir que alterações inválidas sejam executadas.

Exercícios

- Exercícios 18: altere o exercício 17 e acrescente as chamadas para os métodos definidos.

Sobrecarga de Métodos

- Java permite que tenhamos na mesma classe métodos com o mesmo nome, mas parâmetros diferentes. Isso é chamado sobrecarga de métodos.

- No exemplo da classe Retângulo, podemos definir outro método redimensionar().

Sobrecarga de Métodos

- Exemplo:

```
public class Retangulo {
  public void redimensionar( float sx, float sy) {
    if (sx > 0 && sy > 0) {
      largura = (int) (sx / 100 * largura);
      altura  = (int) (sy / 100 * altura);
    }
  }
  public void redimensionar( int larg, int alt) {
    if (larg > 0 && alt > 0) {
      largura = larg;
      altura  = alt;
    }
  }
}
```

O Java não confunde os dois métodos porque eles tem **assinaturas diferentes !**

Assinatura diferente significa que os métodos têm:

- quantidades de parâmetros diferentes,
- tipos dos parâmetros diferentes ou
- ordem dos parâmetros diferente

Sobrecarga de Métodos

- Importante: o Java não leva em conta o tipo de retorno dos métodos para diferencia-los, apenas a quantidade, a ordem e os tipos dos parâmetros. Exemplo:

```
public class Retangulo {
  public void redimensionar(float dx, float dy) { ... }
  public void redimensionar(int larg, int alt) { ... }

  public boolean redimensionar(int larg, int alt) { ... }
}
```

Aqui teremos um **erro de compilação**, porque já existe um método redimensionar que recebe um tipo float. Apesar de um método não retornar nada e o outro retornar um boolean, os métodos são considerados **iguais**.

Exercícios

- Exercício 19: implemente a classe Círculo :
  a) Crie um nova classe com o nome Círculo no mesmo projeto onde está a classe Retângulo.
  b) Defina os atributos.
  c) Defina um construtor.
  d) Defina os métodos de acesso.
  e) Defina os demais métodos (os mesmos métodos da classe Retângulo)
  f) Leia do usuário os valores de x, y e raio e crie um objeto da classe Círculo. Em seguida, chame alguns métodos da classe Círculo.

Construtor Cópia

- Imagine o seguinte código:

```
int a, b;
a = 10;

b = a;
```

A variável **b** é uma **cópia** de **a**.

Se alterarmos o valor de **b** não vamos alterar o valor de **a**.

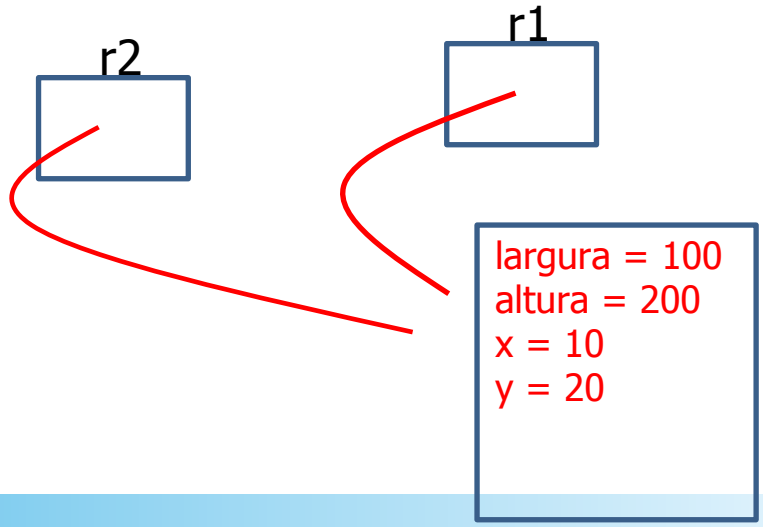
Construtor Cópia

- Com objetos essa situação é diferente:

```
Retangulo r1, r2;
r1 = new Retangulo(10, 20, 100, 200);
r2 = r1;
```

r2 vai referenciar o **mesmo objeto** que **r1**.

Um objeto só é criado quando usamos o operador **new** com o construtor da classe.

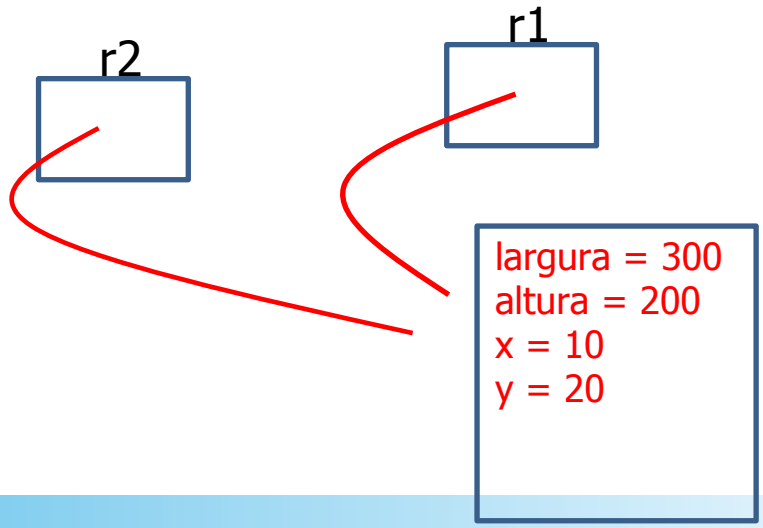


Construtor Cópia

- Com objetos essa situação é diferente:

```
Retangulo r1, r2;
r1 = new Retangulo(10, 20, 100, 200);
r2 = r1;
r2.setLargura(300);
```

Quando alteramos o objeto referenciado por **r2**, estamos alterando o **mesmo objeto** referenciado por **r1**.



Construtor Cópia

- Quando precisamos clonar um objeto, ou seja, criar um novo objeto idêntico a um outro que já existe, precisamos implementar um construtor cópia.
- Como o nome já diz, um construtor cópia tem a finalidade de criar um objeto como cópia de outro objeto.
- Para definir um construtor cópia devemos criar um construtor que recebe como parâmetro um objeto da própria classe.
- Dessa forma podemos copiar todos os atributos de um objeto para o novo objeto.

Construtor Cópia

- No caso das classes Retangulo e Circulo, definidas anteriormente, o construtor cópia pode ser definido como:

```
public class Retangulo{
  private int largura;
  private int altura;
  private int x;
  private int y;
  public Retangulo(Retangulo outro)
  {
    largura = outro.getLargura();
    altura  = outro.getAltura();
    x       = outro.getX();
    y       = outro.getY();
  }
}
```

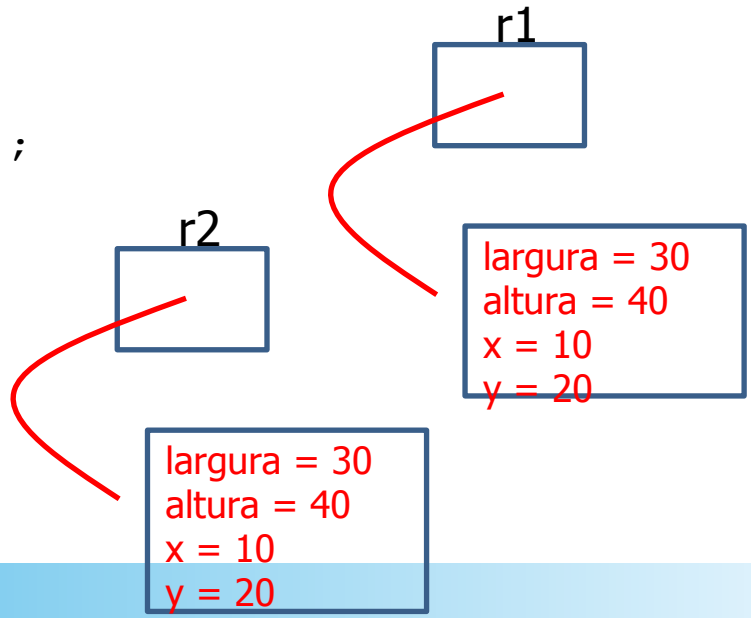
```
public class Circulo {
  private int x;
  private int y;
  private int raio;

  public Circulo(Circulo outro)
  {
    x    = outro.getX();
    y    = outro.getY();
    raio = outro.getRaio();
  }
}
```

Construtor Cópia

- Para usar o construtor cópia, basta passar o objeto que será copiado:

```
public class EditorGrafico {
  public static void main(String[] args) {
    Retangulo r1, r2;
    Circulo c1, c2;
    r1 = new Retangulo(10, 20, 30, 40);
    r2 = new Retangulo(r1);
    c1 = new Circulo(5, 15, 50);
    c2 = new Circulo(c1);
  }
```



Exercícios

- Exercício 20: crie uma classe Sequencia com atributos inteiros inicial e final. Crie um método para imprimir a sequência de números de inicial até final, inclusive. A impressão da sequência pode ser de 1 em 1 (valor *default*) ou de p em p. Exemplos:
  a) Sequência de 2 a 6: 2 3 4 5 6
  b) Sequência de 2 a 10 com salto 2: 2 4 6 8 10
  c) Sequência de 0 a 15 com salto 3: 0 3 6 9 12 15
  d) Sequência de 0 a 10 com salto 4: 0 4 8

Exercícios

- Exercício 21: crie uma classe ContaCorrente para representar uma conta-corrente, com métodos para depositar uma quantia, sacar uma quantia e obter o saldo da conta. Para cada saque será debitada também uma taxa de operação no valor de R$ 1,50. Repare que o saque só poderá ser efetuado ser houver saldo suficiente para a quantia solicitada mais a taxa da operação.

Exercícios

- Exercício 22: crie uma classe Data para representar uma data. Para criar uma data é obrigatório informar dia, mês e ano. Crie, também, três métodos:
  a) ehValida() que deverá retornar *true* se a data for válida ou *false* caso contrário.
  b) ehBissexto() que deverá retornar *true* se a data for válida e o ano for bissexto ou *false* caso contrário.
  c) imprime() que deverá imprimir a data com o separador *default* "/" ou com um separador definido pelo usuário. Caso a data seja inválida, o método deverá imprimir "INVÁLIDA".

  Ano bissexto é aquele que é múltiplo de 4 e não é múltiplo de 100 **OU** aquele que é múltiplo de 400.