# Programação Orientada a Objetos

## Encapsulamento

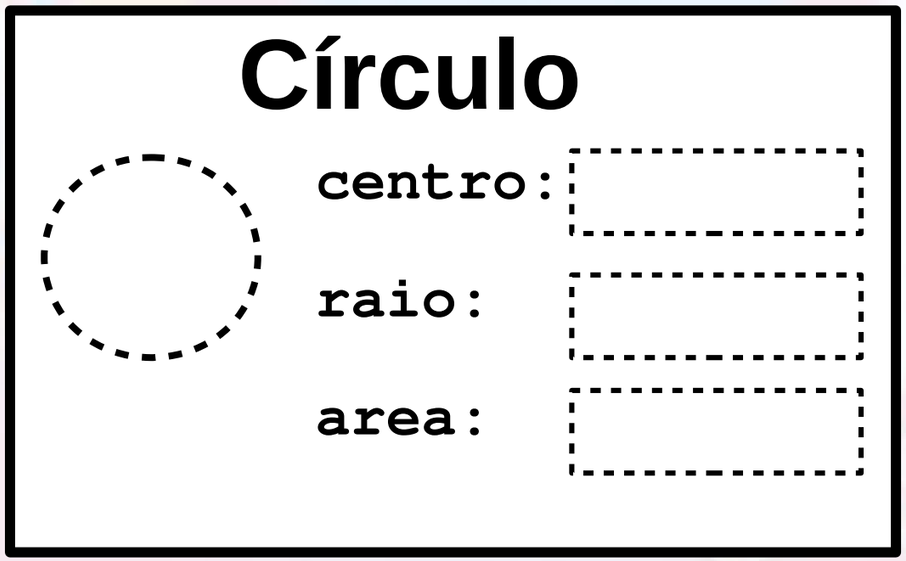
André Santanchè
Laboratory of Information Systems – LIS
Instituto de Computação – UNICAMP
Abril 2020

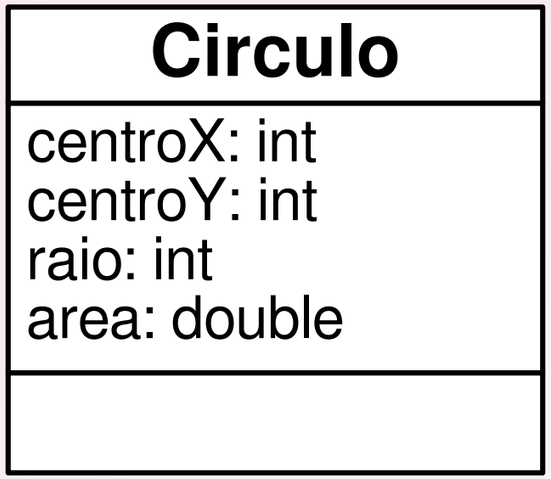
Picture by Doug Wheller [http://www.flickr.com/photos/doug88888/7032440831/]

# Retomando os Círculos

# Área como Atributo

- Projeto inicial de manter o cálculo da área em um atributo.
- Cálculo feito no construtor.



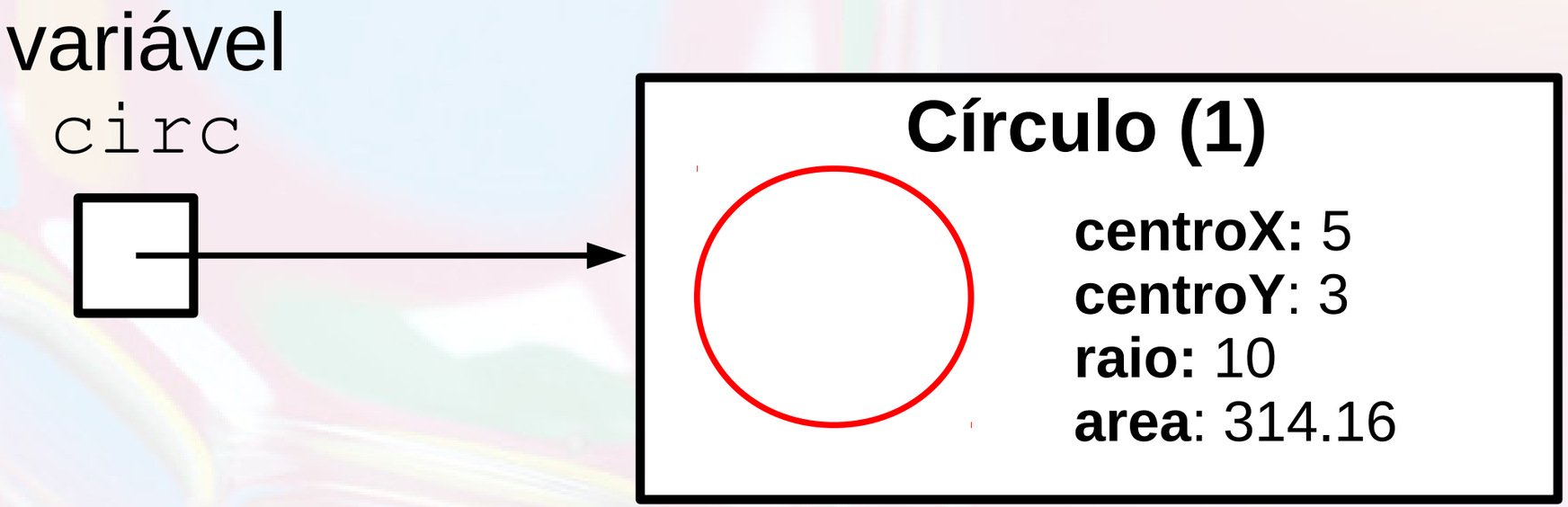
# Área como Atributo

```
public class Circulo {
    int centroX, centroY;
    int raio;
    double area;

    Circulo(int centroX, int centroY, int raio) {
        this.centroX = centroX;
        this.centroY = centroY;
        this.raio = raio;
        this.area = Math.PI * raio * raio;
    }
}
```

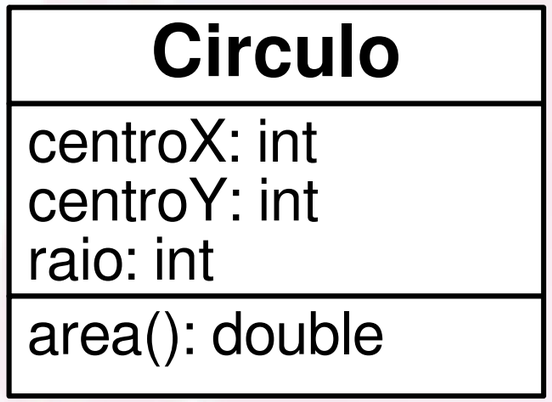
# Área como Atributo

```
System.out.println(circ.area);
```



# Transformando Atributo em Método

- Como manter a consistência se o raio for modificado?
- Projeto resolve realizar o cálculo sobre demanda através de método.

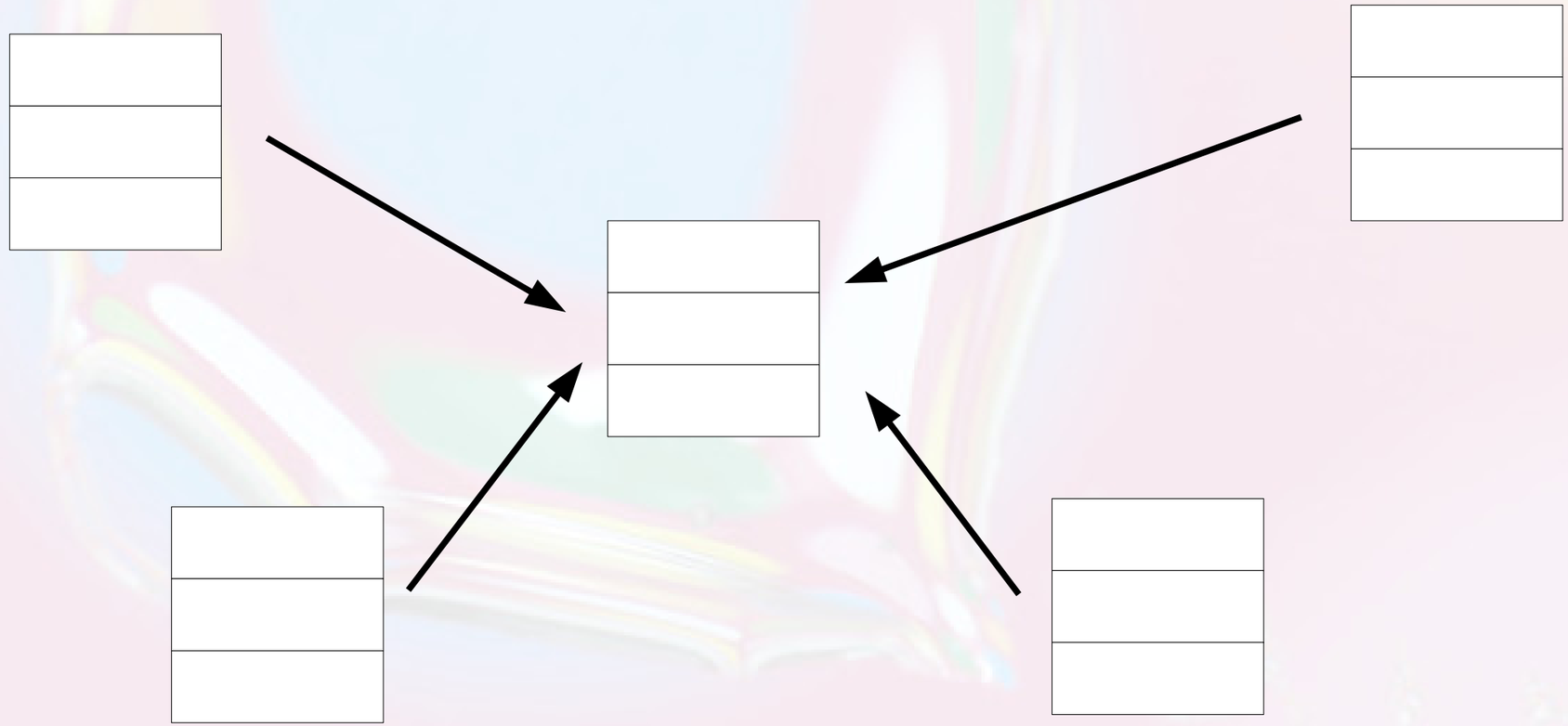


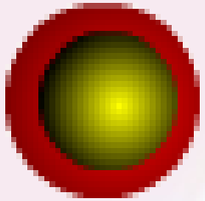
# Transformando Atributo em Método

```
public class Circulo {
    int centroX, centroY;
    int raio;
    Circulo(int centroX, int centroY, int raio) {
        this.centroX = centroX;
        this.centroY = centroY;
        this.raio = raio;
    }
    double area() {
        return Math.PI * raio * raio;
    }
}
```

# Transformação de Atributo em Método

- O que acontece com todos os objetos externos que usavam o objetos da classe `Circulo`?





# Princípios do Paradigma
# Encapsulamento

- "Objetos do mundo real encapsulam em si os próprios atributos, quer sejam descritivos, partes componentes ou funções.” (Meyer, 1997)
- Objetos e mensagens
  - “Na programação orientada a objetos, os objetos comunicam-se entre si através de mensagens. A única coisa que um objeto conhece sobre outro objeto é a sua interface de comunicação. Os dados e a lógica de cada objeto são mantidos escondidos dos outros objetos. Em outras palavras, a interface encapsula o código e os dados do objeto”. (IBM)

# Encapsulamento
# Interface x Implementação

- **Interface**: descreve como os objetos da classe se relacionam com outros objetos externos.
- **Implementação**: dados e código que implementam o comportamento dos objetos da classe; esta parte não é visível externamente.

# Encapsulamento
# Níveis de Acesso

- **Privada**: não visível por classe/objetos externos; visível apenas dentro da classe ou objetos da classe onde é definido.
- **Pública**: completamente visível para qualquer classe/objeto interno ou externo.
- **Protegida**: visível apenas dentro da classe/objetos da classe e para seus herdeiros; não visível a classes/objetos externos.

# Encapsulamento em POO

- Diretrizes para classe e respectivos objetos
  - atributos – privados
  - métodos - podem ser públicos
    - métodos serão privados se não se quiser expô-los publicamente
- O objeto torna-se uma **caixa preta** em que só aparece o suficiente para que objetos externos possam utilizá-lo.
- Detalhes de implementação ficam escondidos do público.

# Encapsulamento em UML

- Visibilidade:
  - + público
  - - privado
  - # protegido

| **Circulo** |
| --- |
| - centroX: int<br>- centroY: int<br>- raio: int |
| + «constructor» Circulo(centroX: int, centroY: int, raio: int)<br>+ area(): double |

# Encapsulamento em Java

- **Privada (private)**: não visível por classe/objetos externos; visível apenas dentro da classe ou objetos da classe onde é definido.
- **Pública (public)**: completamente visível para qualquer classe/objeto interno ou externo.
- **Protegida (protected)**: visível apenas dentro da classe/objetos da classe, para seus herdeiros **e classes/objetos do mesmo pacote**; não visível a classes/objetos externos.
- **Pacote (padrão)**: visível apenas dentro da classe/objetos da classe e pelas classes/objetos que estão no mesmo pacote.

# Encapsulamento em Java

```
public class Circulo {
    private int centroX, centroY;
    private int raio;
    public Circulo(int centroX, int centroY, int raio) {
        this.centroX = centroX;
        this.centroY = centroY;
        this.raio = raio;
    }
    public double area() {
        return Math.PI * raio * raio;
    }
}
```

# Propriedades

# Componente JavaBean

- Componentes são unidades de software auto-contidas e reusáveis que podem ser compostas visualmente em componentes compostos, applets, aplicações, e servlets usando ferramentas visuais de construção de aplicações.” (Sun, 2006)

- Tradução do Inglês: “Components are self-contained, reusable software units that can be visually assembled into composite components, applets, applications, and servlets using visual application builder tools.” (Sun, 2006)

# Propriedades

- Expõem indiretamente atributos para classes/objetos externos através de métodos.
- Métodos podem ser mantidos/adaptados com mudanças de atributos:
  - Interface se mantém mesmo com mudanças externas.
- Permitem consistência perante mudanças.

# Propriedades em Java

- Expostas através de métodos:
  - prefixo “get” → leitura
  - prefixo “set” → modificação

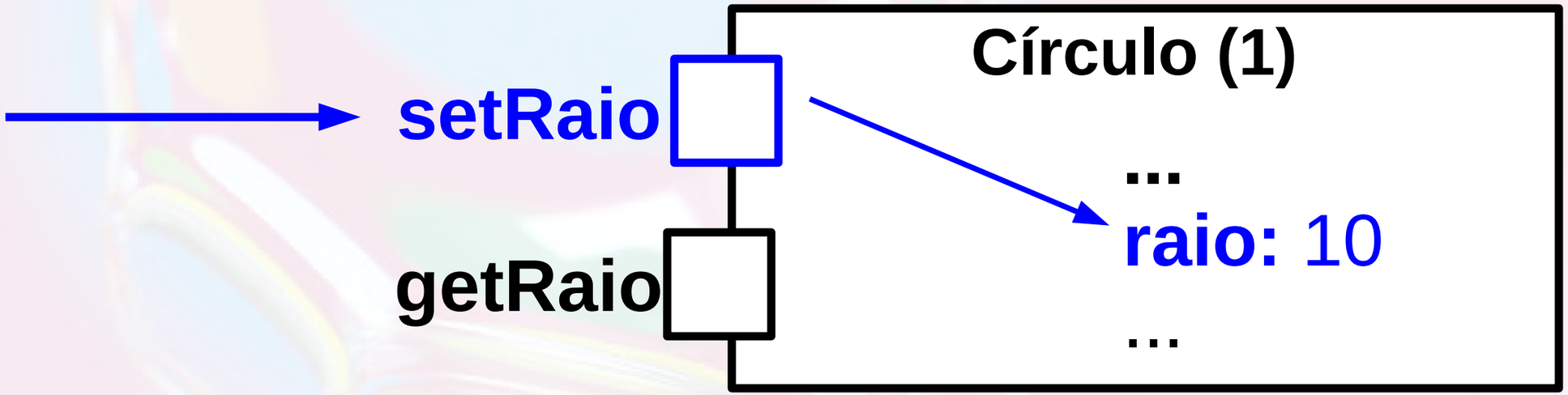
```
private int raio;

public int getRaio() {
    return raio;
}

public void setRaio(int raio) {
    this.raio = raio;
}
```

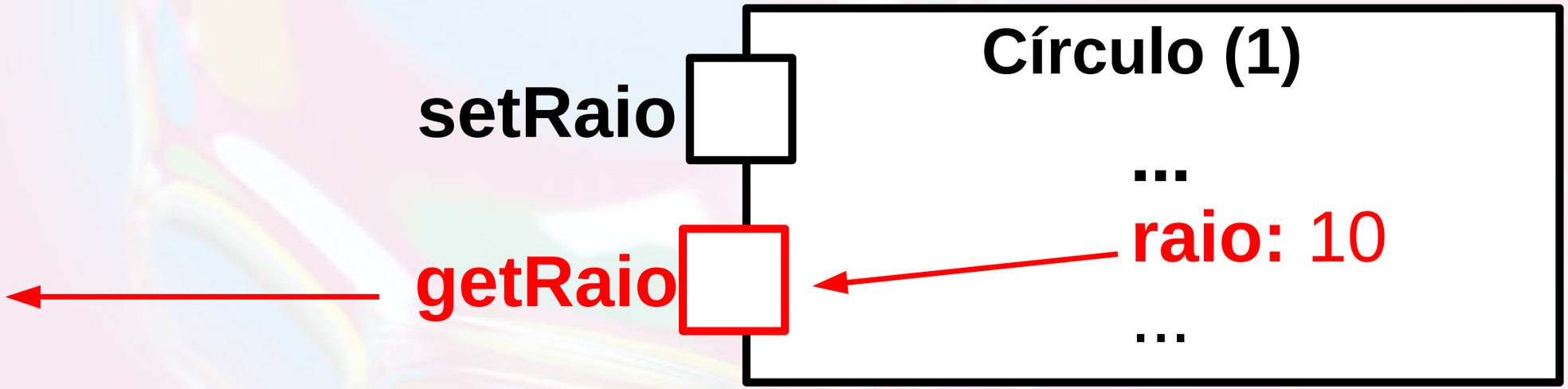
# Atribuindo valor a uma Propriedade

```
circ.setRaio(10);
```



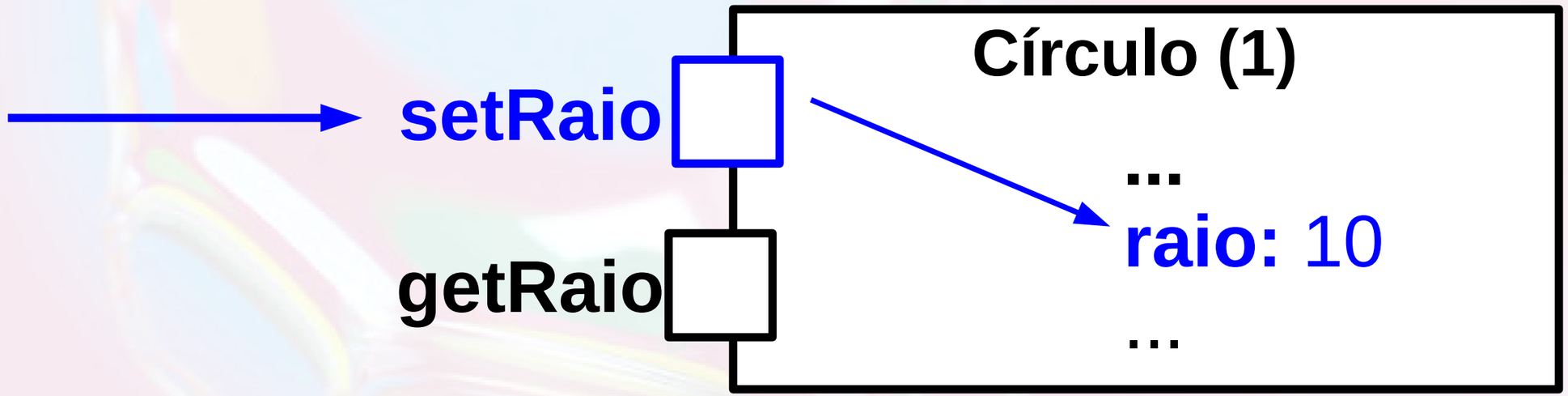
# Recuperando valor de uma Propriedade

```
circ.getRaio()
```



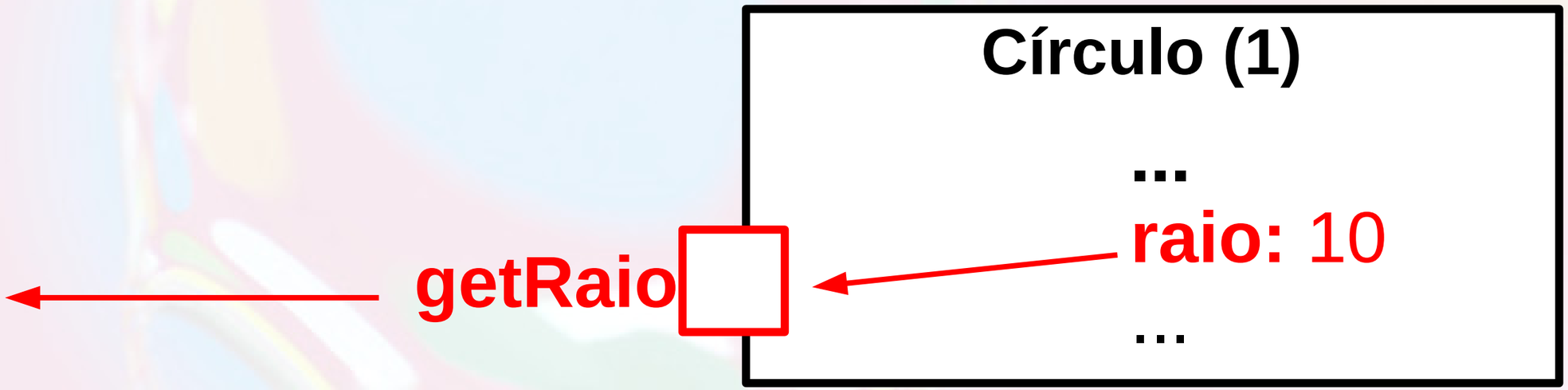
# Consistência

```
public void setRaio(int raio) {
    if (raio > 0)
        this.raio = raio;
}
```



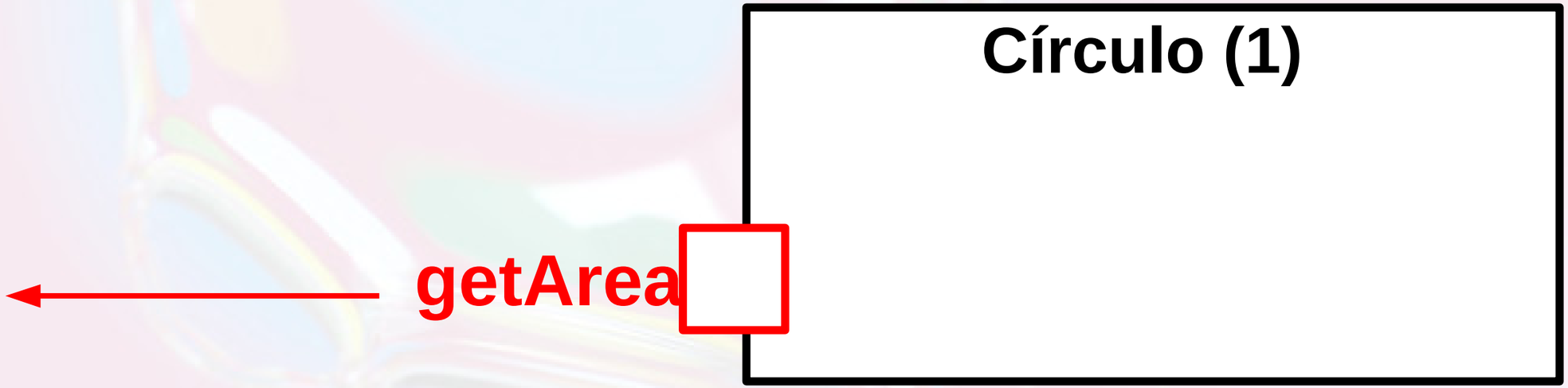
# Propriedades Somente Leitura

- Não definem o método “set”



# Propriedades sem Atributo correspondente

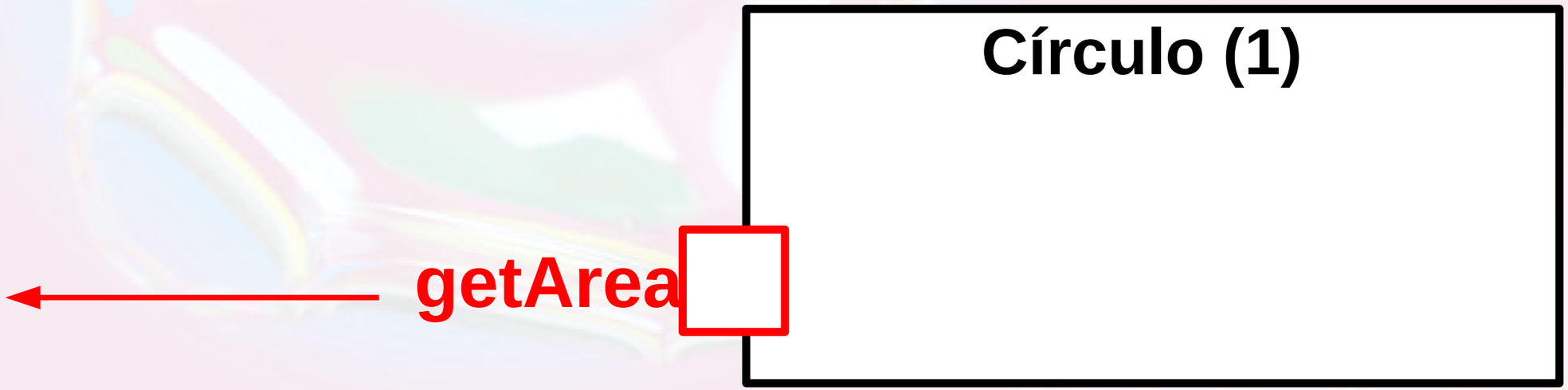
```
public double getArea() {
    return Math.PI * raio * raio;
}
```



# Propriedades sem Atributo correspondente

- Propriedades não estão obrigadas a estar associadas a atributos.

```
public double getArea() {
    return Math.PI * raio * raio;
}
```



André Santanchè
http://www.ic.unicamp.br/~santanche

# Licença

- Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença.
- Mais detalhes sobre a referida licença Creative Commons veja no link: http://creativecommons.org/licenses/by-nc-sa/3.0/
- Agradecimento a Doug Wheller [http://www.flickr.com/photos/doug88888/] por sua fotografia “Water drop” usada na capa e nos fundos, disponível em [http://www.flickr.com/photos/doug88888/7032440831/]

vide licença específica da fotografia.