Introdução à Programação

Criando Classes

Classes em Java

```
public class MinhaJanela extends Window {
        Button botao;
        TextField campoResultado;

        public MinhaJanela() {
                botao = new Button(“Maiuscula”);
                campoResultado = new TextField();
                this.include(botao,250,65);
                this.include(campoResultado,100,90);
                this.setTitle("Minha Janela");
                this.setSize(500,400);
        }

        public void clickEvent() {
                String s = campoResultado.getText();
                campoResultado.setText(s.toUpper());
        }
}
```

Atributos

Construtor

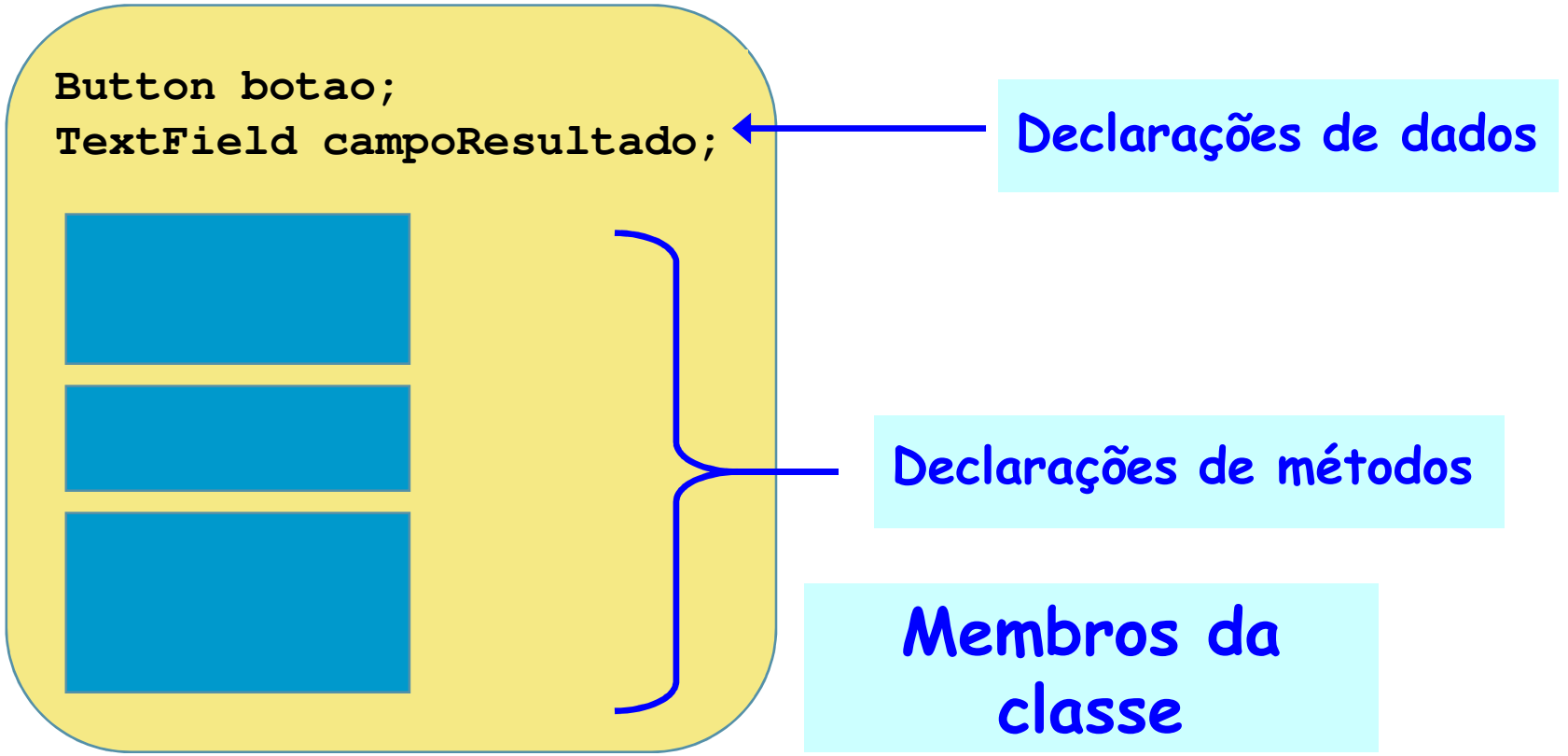
Método

Tópicos da Aula

- Hoje vamos criar as nossas próprias classes
  - Estrutura de uma classe
  - Definindo atributos
  - Escrevendo métodos
  - Passagem de parâmetros
  - Aliasing
  - Métodos x Construtores
  - Atributos x Variáveis Locais
- Depois iremos adicionar comportamento aos componentes gráficos
  - Eventos

Classes

- Classes podem conter declarações de dados e métodos



Definindo Classes em Java

```
ModificadorDeAcesso class NomeDaClasse {

    CorpoDaClasse

}
```

Uma classe em Java pode ser definida com um **modificador de acesso**
- private, public, protected

Veremos em breve!

O corpo de um classe em Java pode conter
- atributos
- métodos
- construtores
- e ainda outras classes...

Definindo Atributos em Java

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

   ...
}
```

Tipo do atributo

Nome do atributo

Cada atributo tem um tipo específico, que caracteriza as propriedades dos objetos da classe

Definindo Atributos em Java

```
public class Conta {
    String numero,digito;
    double saldo = 0;
   ...
}
```

Indica mudança de valor

Valor para inicialização

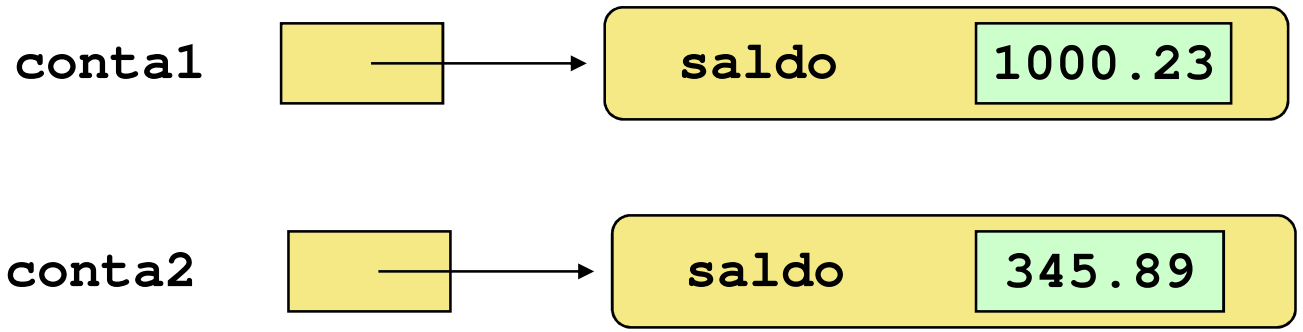
- Vários atributos de um mesmo tipo podem ser declarados conjuntamente
- Podemos especificar que um atributo deve ser inicializado com um valor específico

Considerações sobre Atributos

- Atributos em Java são armazenados em variáveis
- Caso atributos não sejam inicializados, Java atribui valores padrões que dependem do tipo do atributo
- Atributos podem ser declarados com *modificadores de acesso*
  - *public, protected, private*
- As variáveis que armazenam os atributos de um objeto existem enquanto o objeto existir
- Quando se cria 2 objetos do tipo Conta, por exemplo, cada um terá variáveis com os mesmos nomes para armazenar atributos, mas em espaços de memória diferentes (endereços diferentes)

Atributos

- Se criarmos 2 objetos do tipo Conta, as definições dos atributos são iguais, mas os valores são diferentes



Cada objeto mantém sua própria variável saldo, e assim seu próprio estado

Definindo Métodos em Java

```
class Conta {
   ...

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

Nome

parâmetros do método

tipo de retorno

corpo do método

Cabeçalho do método

Definindo métodos em Java

- O cabeçalho do método pode conter *modificadores de acesso*
- O tipo do valor a ser retornado pelo método
- Nome do método
- Lista, que pode ser vazia, indicando o tipo e o nome dos parâmetros a serem recebidos pelo método
- Exceções que podem ser levantadas pelo método

Métodos que retornam resultados

```
public class Conta {
   String   numero, digito;
   double   saldo;

   String getNumero() {

      return numero;
   }
   double getSaldo() {

      return saldo;
   }
   ...
```

Os métodos que retornam valores como resultado usam o comando **return**

Comando `return`

`return expressão`

- Um método que não tem valor para retornar tem o tipo de retorno ***void***
  - Neste caso, o uso do comando *return* é opcional

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

Pode ser omitido

- Para executar este comando o computador:
  - avalia **`expressão`**, obtendo um valor
  - devolve este valor como resultado, terminando a execução do método no qual ele se encontra

O Corpo do método...

- Contém instruções que determinam as ações a serem realizadas pelo método
- Estas instruções podem
  - declarar variáveis
  - realizar simples atualizações dos atributos de um objeto
  - retornar valores
  - executar ações mais complexas como se comunicar com outros objetos, repetir comandos, etc.

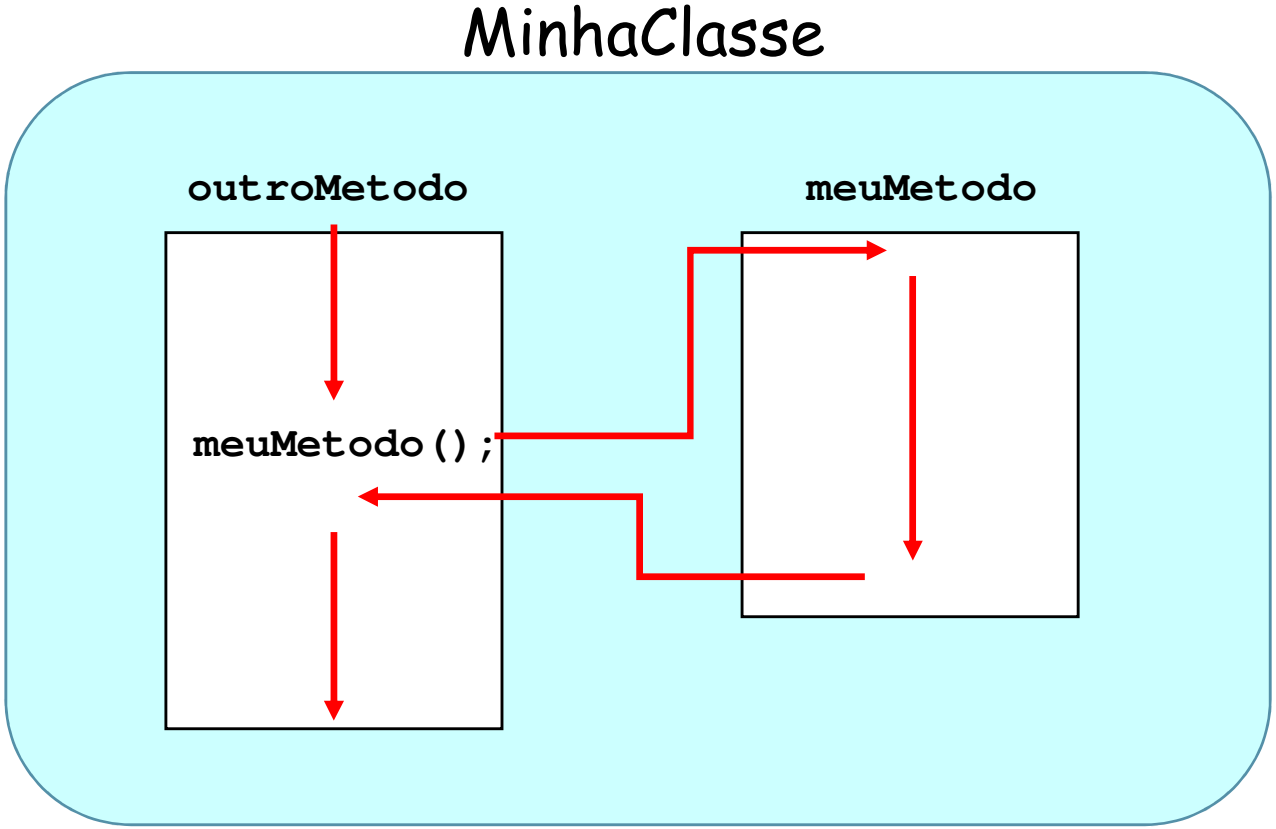
Veremos em breve!

Considerações sobre Métodos

- Uma definição de um método especifica a seqüência de instruções que serão executadas (**fluxo de controle**) quando este método for chamado (invocado)
- Quando um método é chamado, o fluxo de controle do programa pula para o método e executa o código que está nele
- Quando o método termina de ser executado, o fluxo de controle do programa volta para a instrução logo após a chamada do método
- Uma chamada a um método pode ou não retornar um valor
  - Depende da definição do método

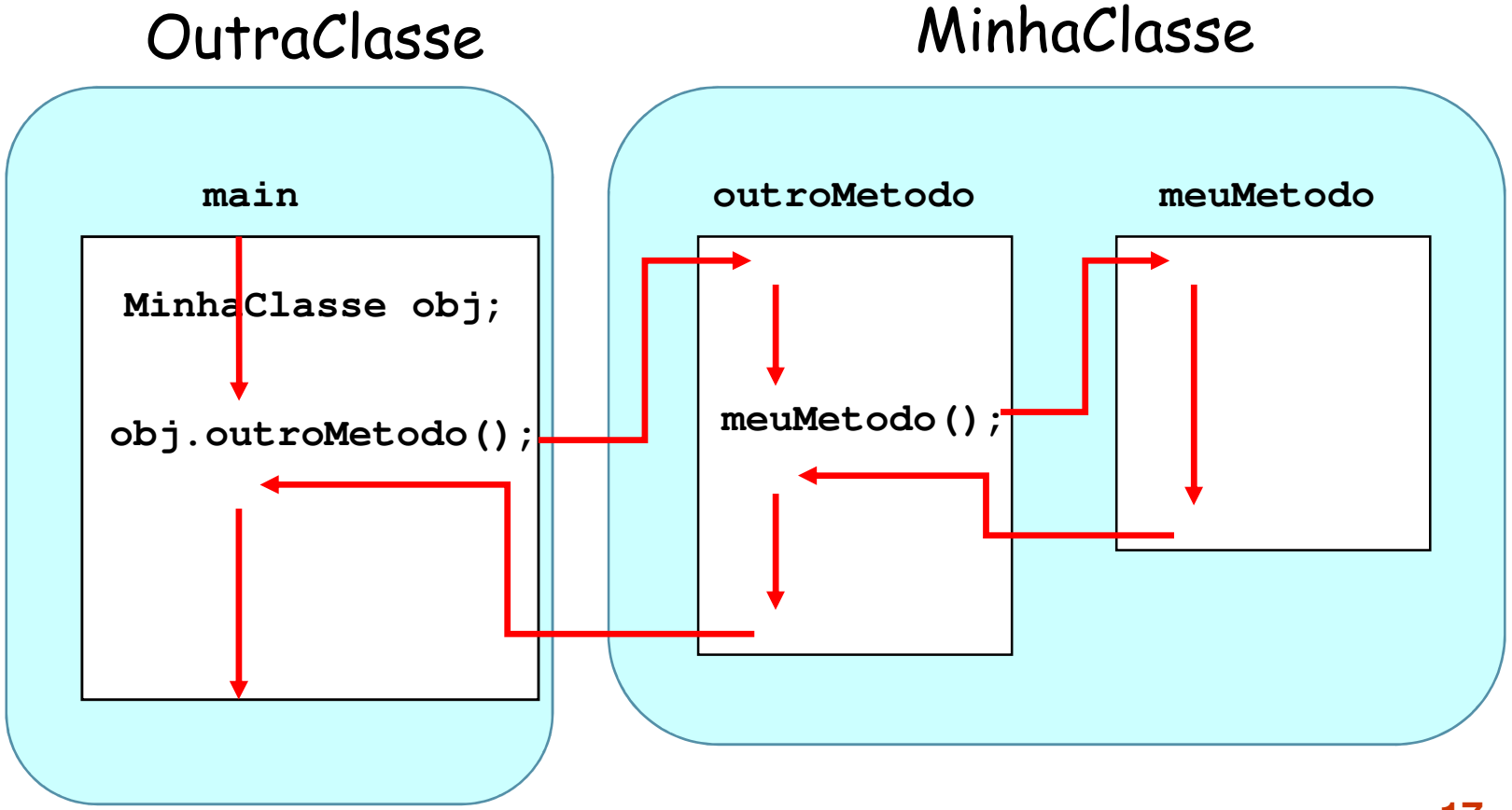
Fluxo de Controle de um Método

- Se o método chamado pertencer a mesma classe



Fluxo de Controle de um Método

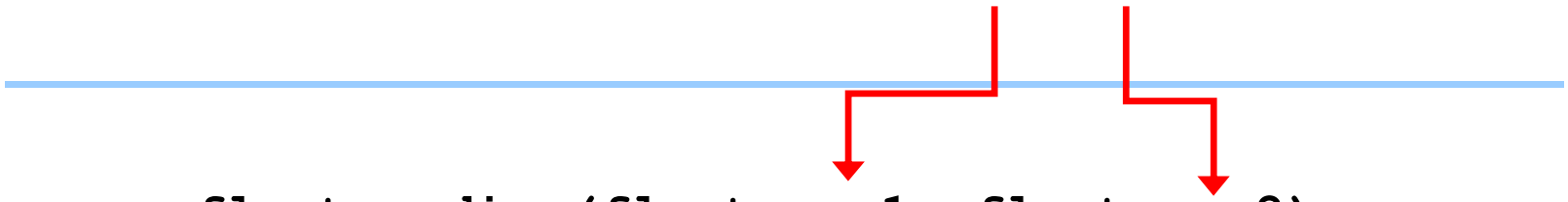
- Se o método chamado pertencer a outra classe



Parâmetros de um Método

- Quando um método é chamado, os argumentos da chamada são copiados para os parâmetros (formais) presentes no cabeçalho do método

```
float valor = obj.media (30,40);
```



```
float media (float num1, float num2)
{
    float result = (num1 + num2)/2;
    return result;
}
```

Passagem de parâmetro

Em Java, a passagem de parâmetro é por valor: o valor da expressão é avaliado primeiro e depois passado para o método chamado

A avaliação é feita da esquerda para a direita

Passagem de parâmetro por valor

```
public class PassagemPorValor {
  MiniJavaSystem  system =new MiniJavaSystem();
  void incrementa(int x)  {
    x = x + 1;
    system.println(x);
  }
}
```

```
PassagemPorValor p;
p = new PassagemPorValor();
int y = 1;
system.println(y);          saída:1
p.incrementa(y);            saída:2
system.println(y);          saída:1
```

não altera o valor de y

Referências são valores!

```
public class Referencia {
    void redefine(Button a)   {
      Button b = new Button();
      a.setText("Abrir");
      a = b;
      a.setText("Fechar");
    }
}
```

não altera o valor de c

```
Referencia r = new Referencia();
Button c = new Button("Calc");
r.redefine(c);
```

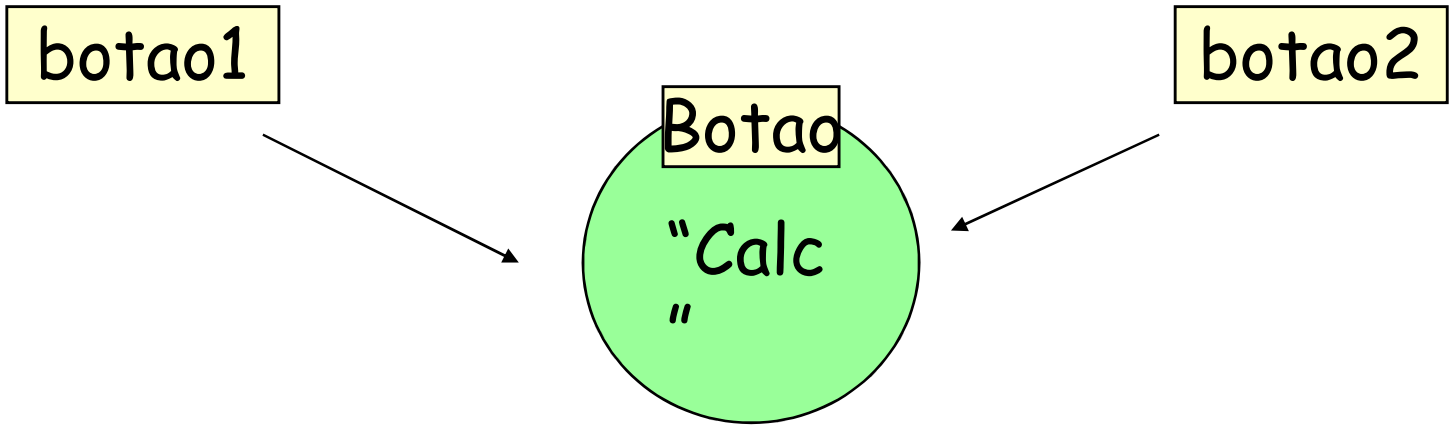
Altera o estado do objeto referenciado por c!

Qual é o *text* de c?

"Abrir"

Aliasing

- É o mecanismo através do qual, duas ou mais variáveis referenciam um mesmo objeto.



Aliasing

- Button botao1 = new Button();
- Button botao2 = botao1;

- botao1.setText("abc");
- botao2.setText("xyz");

Ao final, o objeto apontado tanto por botao1 quanto por botao2 terá o mesmo texto: "xyz"

Aliasing na Passagem de Parâmetros

```
public class Referencia {
    void redefine(Button a) {
      a.setText("Abrir");

    }
}
```

Parâmetro **a** e variável local **c** guardam a mesma referência
Aliasing!

```
public class Teste {
     public static void main(String[] args){
          Referencia r = new Referencia();
          Button c = new Button("Calc");
          r.redefine(c);
     }
}
```

Construtores x Métodos

- Diferentemente de um método normal, um construtor não possui um tipo de retorno
- Um **erro** comum é colocar *void* como tipo de retorno de um construtor
- Construtores só são chamados no ato de criação de um objeto
  - Podem fazer algumas inicializações
- Um construtor em Java tem necessariamente o mesmo nome da classe
- Não é obrigatório a definição de um construtor para uma classe
- Toda classe tem um construtor padrão sem parâmetros

Escopo de Variáveis

- O escopo de uma variável define a área do programa onde esta variável pode ser referenciada
- Variáveis que são declaradas no nível da classe (atributos), ou seja fora dos métodos, podem ser referenciadas por todos os métodos ou construtores da classe
- Variáveis que são declaradas dentro de um método ou construtor só podem ser referenciadas dentro deste método ou construtor
  - São chamadas de **variáveis locais**

Escopo de Variáveis

- Pode existir uma variável local a um método com mesmo nome e tipo de um atributo, neste caso ao se referir ao nome da variável dentro do método, estar-se-á acessando a variável local

```
class Conta {
  String   numero, digito;
      ...
  String getNumero() {
      return numero;
  }
  String getDigito() {
      return digito;
  }
  String getNumeroCompleto() {
      String numero = getNumero();
      numero = numero + "-"+getDigito();
      return numero;
  }
  ...
```

Declaração de variável local

Referência à variável local

As Variáveis Locais...

Têm a mesma capacidade de armazenamento que os atributos mas

- São declaradas dentro de um método ou construtor
- Só existem durante a execução do método ou construtor
- Não são inicializadas automaticamente
- Não podem ser declaradas com *modificadores de acesso*

As Variáveis Locais Servem para...

- Armazenar resultados temporários que serão utilizados depois
- Armazenar resultados que serão utilizados mais de uma vez
- Melhorar a legibilidade do método ou construtor, quebrando uma expressão grande em expressões menores

A Variável `this`

- A palavra reservada *this* representa uma variável que serve para um objeto se auto-referenciar
- Só pode ser lida, não se pode atribuir um valor a ela
- Contém a referência para o objeto no qual um dado método está sendo executado (o método no qual ela aparece)

O Valor da Variável `this`

```
public class MinhaJanela extends Window {...
  void copiaCor(MinhaJanela j) {
    Color cor = j.getBackground()
    this.setBackground(cor);

  }
}
```

```
public class Teste {...
  public static void main(String[] args) {
    MinhaJanela x = new MinhaJanela();
    MinhaJanela y = new MinhaJanela();
    Color corY = new Color(250,0,30);
    y.setBackground(cor);
    x.copiaCor(y);
  }
}
```

O Valor da Variável `this`

```
public class MinhaJanela extends Window {
   void copiaCor(MinhaJanela j) {
      Color cor = j.getBackground()
      this.setBackground(cor);

   }
}
```

Na chamada **`x.copiaCor(y)`** a variável **`this`** conterá a referência armazenada em **`x`**, e **`j`** conterá a referência armazenada em **`y`**

O Valor da Variável `this`

- A variável this pode ser também utilizada para fazer a distinção entre atributos e variáveis locais ou parâmetros

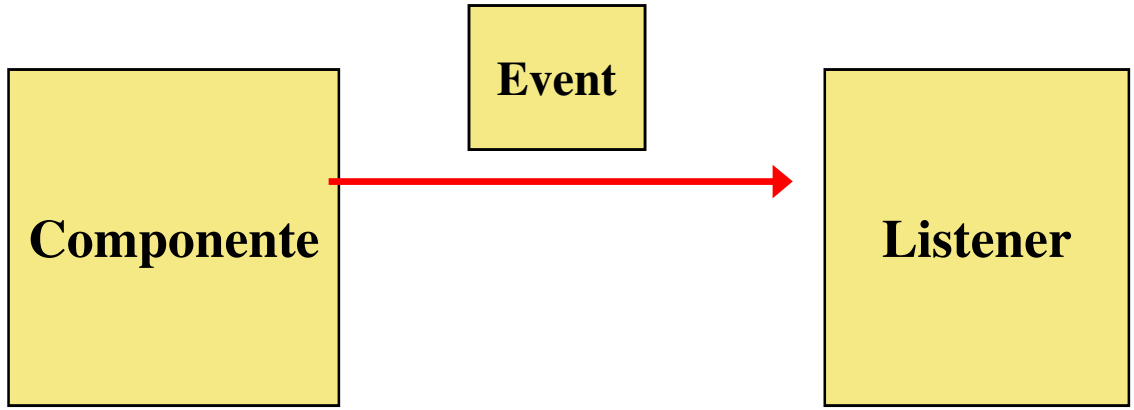
```
public Conta ( String numero,
               double saldo)
{
   this.numero = numero;

   this.saldo = saldo;

}
```

Atributos **Parâmetros**

Eventos

- Uma GUI em Java geralmente têm 3 tipos de objetos
  - Componentes
    - Botão, Janela, Campo de texto, etc
  - Eventos
    - Clique ou movimento do mouse, clique de um botão, etc
  - Listeners
    - Objetos que escutam eventos e tomam alguma ação
- Um **evento** é um objeto em Java

Eventos e Listeners



Um componente pode gerar um evento

Um listener é definido para responder ao evento

Quando um evento ocorre, um componente chama o método apropriado do listener, passando o objeto que representa o evento

Resumindo ...

- Definição de classes
  - Definindo atributos
  - Escrevendo métodos
  - Atributos x Variáveis Locais
  - Passagem de Parâmetros
  - Aliasing
  - Variável this
- Adicionando comportamento aos componentes gráficos
  - Eventos