Métodos

Definição, utilização, parâmetros e variáveis locais e globais dos métodos

Page

- Um método não é mais que um bloco de código
- Esse bloco de código é referenciado através de um nome - o nome do método
- Sempre que o compilador encontra o nome do método executa o boloco de código correspondente
- Para distinguir um método de uma variável (ambos têm nome) os métodos usam ()

- Já utilizamos métodos anteriormente: (`printf`, `println`, `size()`, `substring`)
- Isso significa que quando se escreve `printf(…)`, o compilador executa o código correspondente ao método `printf`

Definição

- Basta definir um bloco de código e, antes desse bloco, colocar o nome que indentificará esse método

```java
static void primeiroMetodo( ) {
    // colocar aqui o código do método
    System.out.println("Este é o meu primeiro método");
}
```

Utilização

- Basta escrever o nome que identifica esse método

```java
static void primeiroMetodo( ){
    System.out.println("Este é o meu primeiro método");
}
```

```java
public static void main( String []args ) {
    primeiroMetodo( );
    System.out.println("Posso chamá-lo quantas vezes quiser");
    primeiroMetodo( );
    primeiroMetodo( );
}

// OUTPUT:
    // Este é o meu primeiro método
    // Posso chamá-lo quantas vezes quiser
    // Este é o meu primeiro método
    // Este é o meu primeiro método
```

- Reparar que no exemplo o mai é, ele próprio , um método
    - Isso permite-nos concluir que um método pode chamar outros métodos

Os métodos permitem...

- Não repetir o mesmo código vezes sem conta
- Reduzir a complexidade do código
    - Para isso os métodos devem ter nome:
        - que indiquem o que o método faz
        - curtos mas relevantes
        - normalmente são verbos
    - É mais fácil elaborar um método que faça uma dada tarefa que escrever um programa completo
    - Uma abordagem é pegar num programa e dividi-lo em vários métodos

```java
static void menuPrincipal( ){
    String menu = "1- Fazer Isto\n" +
                  "2- Fazer Aquilo\n" +
                  "3- Fazer Aqueloutro\n" +
                  "0- Sair";
    int opcao = 1;
    while( opcao != 0 ){
        System.out.println( menu );
        opcao = input.nextInt();
            switch( opcao ){
                case 1: fazerIsto(); break;
```

```
                case 2: fazerAquilo(); break
                case 3: fazerAqueloutro(); break;
                case 0: break;
                default: System.out.println("Opção inválida");
            }
    }
}
public static void main( String []args ) {
    menuPrincipal();
}
```

- Quando se usa um método não se precisa de saber como funciona, basta saber o que faz
- Para isso um método deve desempenhar uma única tarefa

- Reduzir o número de alterações num programa
  - Se for necessário alterar qualquer coisa no código basta alterar no método
  - Sem métodos seria necessário alterar em todo o lado em que o código estaria escrito

Parâmetros

- Mas os métodos podem sair ainda mais úteis:
  - Se usarem parâmetros

```
Java
public static void main(String []args ) {
    for( int i=0; i < 32; i++)
        System.out.print("*");
        System.out.println( "Este programa foi feito por mim"
);
        System.out.println( " e por ele" );
        for( int i=0; i < 32; i++)
            System.out.print("*");
            System.out.println("\n\n");

        for( int i=0; i < 32; i++)
            System.out.print("*");

    System.out.println( "Para a disciplina de Programação I" );
    for( int i=0; i < 32; i++)
        System.out.print("*");
```

```
}

// OUTPUT:
    // ******************************
    // Este programa foi feito por mim
    // e por ele
    // ******************************
    // ******************************
    // Para a disciplina de Programação I
    // ******************************
```

- Ficava melhor assim:

```
Java
static void desenhaLinha(){
    for( int i=0; i < 32; i++)
        System.out.print("*");
}

public static void main(String []args ) {
    desenhaLinha();
    System.out.println( "Este programa foi feito por mim" );
    System.out.println( " e por ele" );
    desenhaLinha();

    System.out.println("\n\n");

    desenhaLinha();
    System.out.println( "Para a disciplina de Programação I" );
    desenhaLinha();
}

// OUTPUT:
    // ******************************
    // Este programa foi feito por mim
    // e por ele
    // ******************************
    // ******************************
    // Para a disciplina de Programação I
    // ******************************
```

- O ideal era que o método desenhasse o número de asteriscos que se pretende

- Isso é possível - indicando que o método tem um parâmetro: o número de asteriscos pretendido

```java
static void desenhaLinha( int nAsteriscos ){
    for( int i=0; i < nAsteriscos; i++)
        System.out.print('*');
}

public static void main(String []args ) {
    desenhaLinha( 32 );
    System.out.println( "Este programa foi feito por mim" );
    System.out.println( " e por ele" );
    desenhaLinha( 32 );

    System.out.println("\n\n");

    desenhaLinha( 35 );
    System.out.println( "Para a disciplina de Programação I" );
    desenhaLinha( 35 );
}

// OUTPUT:
    // ********************************
    // Este programa foi feito por mim
    // e por ele
    // ********************************
    // ***********************************
    // Para a disciplina de Programação I
    // ***********************************
```

Declaração

- Um método pode ter tantos parâmetros quantos se quiser
- Para declarar um parâmetro tem-se de:
    - Indicar qual o seu tipo
    - Indicar o nome pelo qual vai ser conhecido dentro do método
- Os parâmetros são separados por vírgulas
- A sintaxe é: `nomeMetodo( tipo1 param1, tipo2 param2, …, tipoN paramN )`

Utilização

- Quando se chama o método é necessário indicar o valor para cada um dos parâmetros:
    - Pode ser uma constante
    - Pode ser uma variável
    - Pode ser uma expressão
    - Tem é de ser do tipo definido pelo método

```java
desenhaLinha( 32 ); // chamado com uma constante
desenhaLinha( nChars ); // chamado com uma variável
desenhaLinha( grande? 35: 32 ); // chamado com uma expressão
```

⚠️ A ordem de colocação tem de ser igual à ordem de declaração

- Quando se cria um método devem-se usar parâmetros de modo a que o método seja geral:

```java
static void desenhaLinha( int nAsteriscos, char ch ){
    for( int i=0; i < nAsteriscos; i++)
    System.out.print( ch );
}

public static void main(String []args ) {
    desenhaLinha( 32, '*' );
    System.out.println( "Este programa foi feito por mim" );
    System.out.println( " e por ele" );
    desenhaLinha( 32, '*' );

    System.out.println("\n\n");

    desenhaLinha( 35, '-' );
    System.out.println( "Para a disciplina de Programação I" );
    desenhaLinha( 35, '-' );
}

// OUTPUT:
    // *********************************
```

```
    // Este programa foi feito por mim
    // e por ele
    // ******************************
    // --------------------------------
    // Para a disciplina de Programação I
    // --------------------------------
```

Constituição:

- Além de executar tarefas os métodos também podem retornar um valor
- Essa indicação dá-se com a sintaxe: `tipoRetorno nomeMetodo ( parametros )`, mas para já vamos usar a sintaxe `static tipoRetorno ( parametros )`
- Um método é então constituído por:
    - Um cabeçalho
        - Onde se define o tipo de retorno, nome e parâmetros
    - Um corpo
        - Onde se coloca o código

```java
TipoRetorno nomeMetodo( parâmetros ) { // cabeçalho do método
    // corpo do Método
}
```

- Um método que não retorne nenhum valor deve ser o tipo de retorno void (vazio)

```java
static void desenhaLinha( int nAsteriscos, char ch ){
    for( int i=0; i < nAsteriscos; i++)
        System.out.print( ch );
}
```

Return

- Para retornar um valor usa-se a instrução return seguida do valor: `return valorRetornar`
- O `valorRetornar` pode ser:

- Uma constante
- Uma expressão
- Uma variável

```java
static double perimetroCirculo( double raio ) {
    return 2 * Math.PI * raio; // perimetro = 2*PI*Raio
}

public static void main( String []args ) {
    double raio = 2.3;
    double perimetro = perimetroCirculo( raio );
    printf("Um circulo de raio %f tem um perímetro %f", raio,
perimetro );

    raio = 4.5;
    perimetro = perimetroCirculo( raio );
    printf("Um circulo de raio %f tem um perímetro de %f", raio,
perimetro );

    perimetro = perimetroCirculo( 1.5 );
    printf("Um circulo de raio %f tem um perímetro de %f", 1.5,
perimetro );
}
```

- Assim que a instrução `return` é executada o métoido termina a sua execução
- A instrução `return` pode ser colocada em qualquer parte do código do método
- A instrução return pode ser usada sem qualquer valor
    - Em métodos `void` quando se pretende sair sem executar o código restante do método (situação de erro, por exemplo)

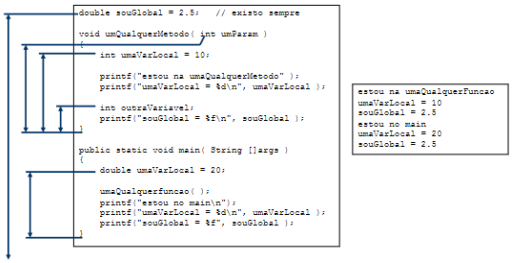
Variáveis Locais e Globais

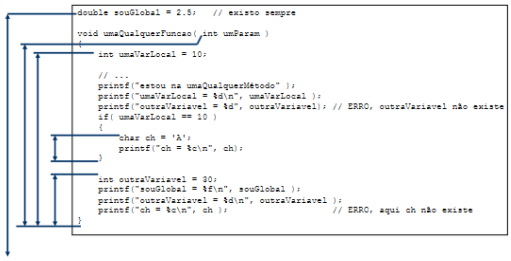
- Uma variável diz-se local se for só conhecida num dado bloco de código
    - São variáveis que são declaradas dentro de um bloco de código (entre {e})

- Uma variável diz-se global se for conhecida por todo o código
    - São declaradas fora de qualquer método

Scope de variáveis

- O scope de uma variável é o espaço de código em que a variável é reconhecida
- As variáveis globais têm um scope global
- O scope de uma variável local é desde a declaração até ao final do bloco onde é declarada





Overload de métodos

- Onde declarar os métodos num programa?
    - A linguagem Java exige que os métodos sejam declarados sempre dentro de uma classe
    - E não permite que sejam declaradas dentro de blocos de código
        - Por exemplo não se podem declarar métodos dentro de outros métodos

- Podem-se ter métodos com o mesmo nome? ✅
- Ao facto de 2 ou mais métodos terem o mesmo nome chama-se overload de métodos
- Como se distinguem então métodos com o mesmo nome?
    - Tipo de parâmetros
    - Número de parâmetros
        - Basta que um deles seja diferente que já dá para distinguir o método

⚠️ Se 2 métodos têm o mesmo nome eles devem fazer a mesma tarefa

Java

```java
void print ( char c );
void print ( int i );
void print ( float f );
void print ( double d );

char umChar = 'A';
print ( umChar );           // chamada ao método print ( char c )
print ( 200 );              // chamada ao método print ( int i )
print ( 2.56 );             // chamada ao método print ( double d )
```

Java

```java
static void print( int array[] ){
    print( array, 0, array.length );
}

static void print( int array[], int ini, int fim ){
    for( int i=ini; i < fim; i++ )
        System.out.println( "[" + i + "] = " + array[i] );
}
```

Java

```java
public static void main( String []args ) {
    int diasMeses[] = { 31, 28, 31, 30, …, 30, 31 };

    print( diasMeses ); // chama o método que imprime o array todo
    print( diasMeses, 5, 8 ); // chama o método que imprime parte do
array
}
```

- O overload de métodos permite assim que 2 métodos que fazem a mesma tarefa não tenham nomes diferentes só porque usam parâmetros diferentes
    - Isto reduz muito a complexidade

- O valor de retorno não serve para distinguir 2 métodos

Java

```java
int metodo( int a ); // são o MESMO método
float metodo( int a );

metodo( 10 ); // Ambiguidade: 1º ou 2º método?
```