

METODOLOGIA

- › Interprete o documento calmamente e com atenção.
- › Acompanhe a execução do exercício no seu computador.
- › Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- › Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- › Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

CONTEÚDO PROGRAMÁTICO

1. [Introdução](#)
2. [Parâmetros](#)
3. [Tipo de retorno](#)
4. [Âmbito das variáveis](#)

1. Introdução

Métodos são blocos de código que permitem executar uma determinada operação. Isto subentende que esta operação é executada várias vezes no nosso programa. Através dos métodos podemos reutilizar um determinado bloco de instruções várias vezes e em posições diferentes no programa. Para além desta vantagem, o uso de métodos facilita a leitura do código assim como a sua manutenção. Podemos também chamar "função" a um método.

Passemos à análise de um exemplo desta funcionalidade:

› **Crie** a classe `Metodos.java` e **insira** o seguinte código:

```
public class Metodos {  
    public static void main(String[] args) {  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
  
        System.out.println("\n Imprimir mais um quadrado \n");  
  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
  
        System.out.println("\n Imprimir mais um quadrado \n");  
  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
        System.out.println("####");  
    }  
}
```

› **Compile e teste** a classe

📄 O código `\n` força uma quebra de linha no texto apresentado na consola.

Nesta classe estamos apenas a escrever diretamente no ecrã o que seria uma representação em texto de três quadrados. O código repete-se várias vezes o que significa que podemos transformá-lo num método com um nome sugestivo. Desta maneira escrevemos menos código e tudo fica mais legível.

› **Altere** o código da **classe** de acordo com o indicado a seguir:

```
public class Metodos {
    public static void quadrado(){
        System.out.println("####");
        System.out.println("####");
        System.out.println("####");
        System.out.println("####");
    }

    public static void main(String[] args) {
        quadrado();
        System.out.println("\n Imprimir mais um quadrado \n");
        quadrado();
        System.out.println("\n Imprimir mais um quadrado \n");
        quadrado();
    }
}
```

› **Execute** a classe

📄 Verifique que o funcionamento do programa continua igual ao anterior.

Repare na simplicidade de escrita no código dentro do bloco `main`. Agora é claramente visível o que está a ser feito, isto se o nome dado ao método for sugestivo em relação ao que faz.

Para executar um método basta colocar o seu nome seguido de parênteses curvos:

```
quadrado();
```

Para além de ser mais simples de ler, é muito mais fácil de adicionar noutros sítios do programa. No exemplo acima mencionado, se for necessário mostrar mais um quadrado, é bastante mais simples.

› **Adicione** antes do fim do `main` o seguinte código e **observe** o resultado:

```
System.out.println("\n Imprimir mais um quadrado \n");  
quadrado();
```

Apenas com estas duas linhas o programa imprime agora mais um quadrado. Em termos de manutenção também é claramente mais fácil e rápido. Imaginemos que agora era necessário que este quadrado escrevesse mais uma linha no final. Seria apenas necessário atualizar a função `quadrado`.

› **Altere** o método `quadrado` de forma a ficar com o seguinte código:

```
public static void quadrado(){  
    System.out.println("#####");  
    System.out.println("#####");  
    System.out.println("#####");  
    System.out.println("#####");  
}
```

```
System.out.println("####");  
}
```

- **Confirme** que agora todos os quadrados são escritos da forma correta, ou seja, quadrados **5 x 5**

2. Parâmetros

À medida que começamos a usar métodos, reparamos que, por vezes, o código que se repete é bastante semelhante, mas não exatamente igual. Desta forma, é necessário dar alguma informação ao método para que este possa alterar um pouco o seu comportamento. A esta informação chamamos parâmetros, que em concreto são um conjunto de valores de um determinado tipo. Em termos genéricos a sintaxe de uma função é representada da seguinte forma:

```
public static void nomemetodo(int parametro1, string parametro2)
```

The diagram shows the following callouts:

- 1: `public`
- 2: `static`
- 3: `void`
- 4: `nomemetodo`
- 5: `int`
- 6: `parametro1`

No código acima os marcadores assinalados representam as seguintes informações:

- **1 - Visibilidade do método**
- **2 - Modificador de acesso**
- **3 - Tipo de retorno:** Refere o tipo de valor que o método devolve. Em muitos dos métodos, o tipo devolvido, que não foi ainda abordado, é o `void`. Este tipo corresponde ao vazio, o que significa que deve ser utilizado quando o método não devolve qualquer valor. Para os restantes casos devem utilizar-se os tipos que vimos até agora, assim como `int`, `string`, `double`, `float`, etc.
- **4 - Nome do método**

- **5- Tipo do parâmetro:** Refere os tipos dos parâmetros do método. Nestes são válidos todos os tipos menos o `void`.
- **6 - Nome do parâmetro**

Cada método tem a quantidade de parâmetros que for necessária ou relevante para o problema em questão. Isto significa que não existe nenhum número específico de parâmetros a cumprir. A **visibilidade do método** assim como **modificador de acesso** serão abordados nos módulos seguintes.

De seguida exemplificam-se alguns tipos de métodos:

```
public static void mostraDobro (int numero)
public static int soma(int operando1, int operando2)
public static double divisao3Numeros(double numero1, double numero2, double numero3)
public static string maiusculas(string textooriginal)
```

Vamos agora utilizar o método `quadrado` criado anteriormente para explorar o conceito de parâmetros. Imagine que agora era preciso mostrar dois quadrados no fim mas com um caratere diferente. A abordagem mais normal seria de construir uma nova função que mostrasse o mesmo quadrado mas com outro caratere.

› **Adicione** na mesma classe este **novo método**:

```
public static void quadrado2(){
    System.out.println("&&&&");
    System.out.println("&&&&");
    System.out.println("&&&&");
    System.out.println("&&&&");
    System.out.println("&&&&");
}
```

- › Agora, no fim do método `main`, **adicione** o seguinte código:

```
System.out.println("\n Imprimir mais um quadrado com caratere diferente \n");
quadrado2();
8
System.out.println("\n Imprimir mais um quadrado com caratere diferente \n");
quadrado2();
```

- › **Compile e confirme** que a classe imprime mais dois quadrados com caratere diferente

O código que adicionámos resolve o problema em questão, no entanto, cada vez que seja necessário imprimir um quadrado com um caratere diferente, será necessário adicionar um novo método. A solução passa por alterar o método `quadrado` de forma a que este receba também o caratere a ser impresso no ecrã.

- › **Elimine** o código da **área 7 e 8**

- › **Substitua** o código do método `main` e do método `quadrado` de acordo com o seguinte:

```
9  public class Metodos {
10      public static void quadrado(char caratere){
        System.out.println(" " + caratere + caratere + caratere + caratere + caratere);
        System.out.println(" " + caratere + caratere + caratere + caratere + caratere);
        System.out.println(" " + caratere + caratere + caratere + caratere + caratere);
        System.out.println(" " + caratere + caratere + caratere + caratere + caratere);
        System.out.println(" " + caratere + caratere + caratere + caratere + caratere);
    }
```

```
public static void main(String[] args) {  
    quadrado('#');  
    System.out.println("\n Imprimir mais um quadrado \n");  
  
    quadrado('#');  
    System.out.println("\n Imprimir mais um quadrado \n");  
  
    quadrado('#');  
    System.out.println("\n Imprimir mais um quadrado \n");  
  
    quadrado('#');  
}  
}
```

› Compile e teste a classe

📄 Confirme que esta classe se comporta de forma igual ao que estava antes.

Neste momento, apesar de estar a ser escrito um `quadrado` apenas com um tipo de caratere, o método `quadrado` já recebe a informação do caratere, através do **nome dentro do método 9** e respetivo **tipo 10**. Note que, quando é chamado o método `quadrado` no `main`, é indicado qual o parâmetro a ser recebido:

```
quadrado('#');
```

Tendo isto em mente, agora para escrever um quadrado com um caratere diferente é apenas necessário invocar o método `quadrado` passando um caratere diferente.

› Adicione o respetivo código antes do fim do método `main`:


```
System.out.println("\n Imprimir mais um quadrado diferente\n");  
quadrado( '$' );  
  
System.out.println("\n Imprimir mais um quadrado diferente\n");  
quadrado( '%' );
```

› **Compile e execute** a classe

☐ Confirme que os dois últimos quadrados são impressos com caracteres diferentes.

☐ Certifique-se que compreendeu em pleno a noção de parâmetros de um método. Em qualquer caso de dúvida não hesite em chamar o formador.

3. Tipo de retorno

Existem vários casos em que é necessário que um método devolva um valor. Isto acontece porque a lógica associada ao método que produz um determinado resultado, não deve estar encarregue de o utilizar. Vamos abordar um exemplo concreto desta situação, começando por construir um método que soma dois números.

› **Crie** a classe `MetodosRetorno.java`

› **Insira** o seguinte código:

```
import java.util.Scanner;

public class MetodosRetorno {
    public static int soma(int oper1, int oper2){
        int resultadofinal = oper1 + oper2;

        return resultadofinal;
    }

    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        System.out.println("Insira o primeiro operando");
        int operando1 = teclado.nextInt();

        System.out.println("Insira o segundo operando");
        int operando2 = teclado.nextInt();

        int resultado = soma(operando1,operando2);

        System.out.println("A soma entre " + operando1 + " e " + operando2 + " é " + resultado);
    }
}
```

› Compile e execute a classe

Neste momento foi referida uma nova palavra reservada, a instrução `return`.

```
return resultadofinal;
```

Esta instrução faz com que o método termine a sua execução e seja devolvido o valor da expressão à sua frente. Após o término do método, o valor que este devolve é colocado no sítio da sua chamada.

```
int resultado = soma(operando1,operando2) ;
```

No exemplo acima, o valor retornado pelo método `soma` é colocado no **local de chamada do método 11** e, de seguida guardado na variável `resultado`.

⚠ Quando são passadas variáveis como parâmetros a um método, o seu valor é copiado para os respetivos parâmetros. Isto significa que qualquer alteração ao valor dos parâmetros dentro do método não altera o valor da variável que foi passada como parâmetro.

De forma a tornar mais evidente esta situação, vamos comprovar com um pequeno exemplo:

› Crie a classe `MetodosParametros.java`

› Insira nessa classe o seguinte código:

```
import java.util.Scanner;

public class MetodosParametros {
    public static void soma(int oper1, int oper2, int resultado ){
        resultado = oper1 + oper2;
    }

    public static void main(String[] args) {

        Scanner teclado = new Scanner(System.in);
```

```
System.out.println("Insira o primeiro operando");
int operando1 = teclado.nextInt();

System.out.println("Insira o segundo operando");
int operando2 = teclado.nextInt();

int res = 0;

soma(operando1,operando2, res );

System.out.println("O resultado da soma dos dois numeros e " + res);
}
}
```

› Compile e teste o programa

O resultado é sempre zero, correto? Isto acontece porque o valor dos parâmetros é copiado para dentro do método ao invés de ser referida a variável original. Analisando mais em detalhe, quando é chamado o método passando a variável `res` como terceiro parâmetro:

```
soma(operando1,operando2, res );
```

É criada uma cópia desta variável que irá ser usada dentro do método resultado:

```
public static void soma(int oper1, int oper2, int resultado ){
```

Como é uma cópia e não a própria variável, a sua alteração dentro do método não influencia a variável exterior. Por isso dizemos que no Java a passagem de parâmetros é sempre feita por cópia.

Tendo isto em mente, a forma de corrigir este método para ter o funcionamento esperado, será retornar o valor do resultado, assim como foi feito na

classe `MetodosRetorno.java`.

4. Âmbito das variáveis

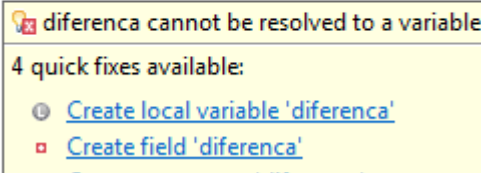
Este ponto refere uma regra importante da programação, que define o tempo de vida de cada variável e passa muitas vezes despercebida. O primeiro ponto desta regra é que as variáveis não existem em toda a duração do programa, sendo que muitas delas são criadas a meio de um método (como o `main`, por exemplo). Sendo assim, quando são eliminadas (inatingíveis)? **A regra diz que uma variável existe apenas no bloco de código onde foi declarada, sendo este definido por um par de chavetas `{}`.**

Vamos ver exemplos deste conceito. Considere o exemplo seguinte:

```
if (saldo > 0){  
    int diferenca = saldo - valor_inicial;  
}  
  
System.out.println("A diferenca é " + diferenca);
```

No código acima escrito foi cometido um erro que refere o âmbito da variável `diferenca`. Como esta foi declarada dentro do `if` (e respetivo bloco de código definido pelas chavetas) esta existe apenas dentro do respetivo bloco. Por este motivo, ao tentar aceder a esta variável fora do `if`, é gerado um erro de âmbito. O compilador mostra o erro indicando que naquele ponto a variável `diferenca` é desconhecida:

```
    int diferenca = saldo - valor_inicial;  
}  
  
System.out.println("A diferenca é " + diferenca);
```



The screenshot shows an IDE error message for the variable `diferenca`. The message is: "diferenca cannot be resolved to a variable". Below the message, it says "4 quick fixes available:" and lists two options: "Create local variable 'diferenca'" and "Create field 'diferenca'".

Este exemplo foi definido com um bloco de código `if` mas, no entanto, isto passa-se com qualquer bloco de código. Exemplos disso são ciclos `for`, `while` e métodos.

Para corrigir o código anterior era necessário declarar a variável diferença noutra local para que esta não deixasse de existir após o `if`:

```
int diferenca = 0;

if (saldo > 0){
    diferenca = saldo - valor_inicial;
}

System.out.println("A diferenca é " + diferenca);
```

Agora, como a variável `diferenca` é declarada fora do `if`, podemos dizer que alterámos o seu âmbito. Desta forma ela não é eliminada após o fim do `if`.