Criação de métodos em Java

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Introdução

- Java assim como C permite a criação de funções, mas em Java as funções se chamam métodos, isso porque método um método é uma função associada a uma classe, como C não possui classes, C não tem métodos.

- Implementamos até agora apenas o método main, que é o ponto de entrada de execução de qualquer programa Java.

Declaração de um método

- Um método pode ser declarado da seguinte forma:

```
qualificador tipo-do-retorno nome-do-método( [lista-de-argumentos] )
{
    código do corpo
}
```

Declaração de um método

- Qualificador (modificador) : define a visibilidade do método. Existem as seguintes possibilidades:
  - ***public**:* o método é visível por qualquer classe.
  - ***private**:* o método é visível apenas na própria classe.
  - ***protected**:* o método é visível pela própria classe, por suas subclasses e pelas classes de mesmo pacote.
  - A não colocação do qualificador deixa o método com visibilidade **default** (ou package). Nesse caso o método é visível pela própria classe e por classes do mesmo pacote.

Obs. Vamos analisar melhor as visibilidades de métodos posteriormente, entretanto é interessante ressaltar que comumente métodos são public.

Declaração de um método

- tipo-de-retorno: um método pode ter como retorno qualquer tipo primitivo (int, float, etc), um vetor, um objeto qualquer(instância de uma classe) ou void (não retorna).
- nome-do-método: é o identificador utilizado para o método. Por padrão métodos em Java devem começar com letras minúsculas (normalmente verbos). Se outras palavras forem necessárias então elas devem iniciar com maiúsculas. Ex: gravarArquivoTexto, salvar, imprimirNome.
- lista-de-argumentos: parâmetros passados para o método de forma que ele faça o que foi projetado para fazer.
- código do corpo: códigos Java que determinam o comportamento do método.

Noções de OO

- Analisemos o seguinte trecho de código:

```
Cachorro rex;
rex = new Cachorro();
```

Cachorro é uma classe que determina o comportamento de todos os cachorros.

rex é um Cachorro específico, é uma instância de Cachorro, é um objeto do tipo Cachorro.

Métodos estáticos

- São métodos de classe, ou seja, são métodos chamados sem a necessidade de um objeto da classe (nesse caso ou não há sentido em existir um objeto da classe para acionar o método ou o método se relaciona de forma geral com todos os objetos da classe, ou seja, acaba por ser, de fato, algo relacionado a classe e não a um objeto em específico). Suponhamos que existe o método estático imprimirQuantidadeCachorros. Ele será chamado da seguinte forma:

```
...
Cachorro.imprimirQuantidadeCachorros();
...
```

Declaração de métodos estáticos

- É feito utilizando a palavra static entre o qualificador e o tipo-do-retorno. Ou seja:

qualificador **static** tipo-do-retorno nome-do-método([lista-de-argumentos])

{

código do corpo

}

Ex:

public **static** void imprimir()

{

System.out.println("teste");

}

Exemplo – declaração de métodos

```
package criacaometodos01;

public class CriacaoMetodos01 {

    public static void tela(String texto) {
        System.out.println("O texto informado é: ");
        System.out.println(texto);
    }

    public static void limpatela() {
        for (int i = 1; i <= 25; i++) {
            System.out.println();
        }
    }

    public static void main(String[] args) {
        String frase = "O gato roeu a roupa do rei de Roma ";
        CriacaoMetodos01.limpatela();
        tela(frase);
    }
}
```

Obs. Note que os métodos tela e limpatela são estáticos, mas é opcional a colocação da classe na sua chamada, pois ambos métodos pertencem a mesma classe que o método main (CriacaoMetodos01)

Recursividade

- Assim como vimos em C, podemos utilizar a recursividade em Java.

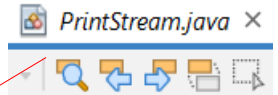
- A recursividade consiste na chamada de um método por si mesmo, direta ou indiretamente por meio de outro método.

Exemplo - Recursividade

```
package criacaometodos02;

public class CriacaoMetodos02 {

    public static long fatorial(long num) {
        if (num <= 1) {
            return 1;
        } else {
            return (num * fatorial(num - 1));
        }
    }

    public static void main(String[] args) {
        for (long i = 0; i <= 10; i++) {
            System.out.println(i + " ! = " + fatorial(i));
        }
    }
}
```

Sobrecarga de métodos

- A linguagem Java permite que vários métodos sejam definidos como o **mesmo nome**, desde que eles tenham **assinatura diferente**, ou seja, essas diferenças podem ser com base no número, tipo ou ordem dos argumentos recebidos ou ainda com base no tipo de retorno.



- O método **System.out.println** é um bom exemplo de sobrecarga de métodos, é possível passar um int, um float, um double, uma String ou até mesmo não passar nenhum argumento que uma função será chamada, isso se deve a sobrecarga de métodos.

```
public void println() {...3 lines }
public void println(boolean x) {...6 lines }
public void println(char x) {...6 lines }
public void println(int x) {...6 lines }
public void println(float x) {...6 lines }
public void println(double x) {...6 lines }
public void println(char x[]) {...6 lines }
public void println(String x) {...6 lines }
```

Exemplo - Sobrecarga

```
package criacaometodos03;

public class CriacaoMetodos03 {

    public static double area(double x) {
        return (x * x);
    }

    public static double area(double x, double y) {
        return (x * y);
    }

    public static void main(String[] args) {
        System.out.println("Area de um  quadrado " + area(3));
        System.out.println("Area de um  retangulo " + area(3, 2));
    }
}
```

Obs. Nesse exemplo a sobrecarga é determinada pela quantidade de argumentos.

Classes sem o método main

- Essas classes funcionam como bibliotecas de funcionalidades para os programas implementados.

Exemplo de classe sem o método main

```
package criacaometodos04;

public class Calc {

    public static int somar(int n1, int n2) {
        return (n1 + n2);
    }

    public static int subtrair(int n1, int n2) {
        return (n1 - n2);
    }

    public static int multiplicar(int n1, int n2) {
        return (n1 * n2);
    }

    public static int dividir(int n1, int n2) {
        return (n1 / n2);
    }
}
```

Acesso a método de outras classes

- Como todos os métodos que estamos tratando são estáticos, para acessar um método de outra classe basta fazer:

nome_da_classe.nome_do_método(argumentos)

```
package criacaometodos04;

public class CriacaoMetodos04 {

    public static void main(String[] args) {
        System.out.println("Somando " + Calc.somar(3, 4));
        System.out.println("Subtraindo " + Calc.subtrair(3, 4));
        System.out.println("Multiplicando " + Calc.multiplicar(3, 4));
        System.out.println("Dividindo " + Calc.dividir(3, 4));
    }
}
```

Dúvidas?

Exercício

- Crie uma classe Conversora para converter valores conforme a imagem abaixo:

```
package usandoconvertedora;

public class UsandoConvertedora {
    public static void main(String[] args) {
        int i1 = Convertedora.paraInteiro("123");
        int i2 = Convertedora.paraInteiro(23.5);
        System.out.println("O valor de i1+i2 é :"+ (i1+i2));

        String s1 = Convertedora.paraString(123);
        String s2 = Convertedora.paraString(23.5);
        System.out.println("O valor de s1+s2 é :"+ (s1+s2));
    }
}
```

Reflexões

- Você reparou que fez uso de sobrecarga?
- Notou que ao encapsular as classes de conversão, usadas na linguagem Java, você facilitou manutenções futuras (caso a linguagem, por exemplo, torne uma dessas classes obsoleta)?
- Acrescente as funcionalidades e use-as em UsandoConvertedora:

```
public static double paraDouble(String s){
    return Double.parseDouble(s);
}
public static double paraDouble(int i){
    return (double)i;
}
```