Métodos

Todas as ações das aplicações são consideradas métodos.

Uma classe é definida por atributos e métodos. Já vimos que atributos são, em sua grande maioria, variáveis de diferentes tipos e valores. Os métodos, por sua vez, correspondem a **funções** ou **sub-rotinas** disponíveis dentro de nossas classes.

Critério de nomeação de Métodos

Esses critérios não são obrigatórios, mas é recomendável que sejam seguidos, pois essas convenções facilitam a vida dos programadores ao trabalharem em códigos de forma colaborativa. Ao seguir estas convenções, tornamos o código mais legível para nós e também para outras pessoas. Para métodos, os critérios são:

- Deve ser nomeado como verbo;
- Seguir o padrão camelCase (Todas as letras minúsculas com a exceção da primeira letra da segunda palavra).

Exemplos sugeridos para nomenclatura de métodos:

```
somar(int n1, int n2){}

abrirConexao(){}

concluirProcessamento() {}

findById(int id){} // não se assuste, você verá muito método em inglês em sua jornada

calcularImprimir(){} // há algo de errado neste método, ele deveria ter uma única finalid
```

ATENÇÃO! Não existe em **Java** o conceito de **métodos** globais. Todos os **métodos** devem SEMPRE ser definidos dentro de uma classe.

Critério de definição de métodos

Mas, como sabemos a melhor forma, de definir os métodos das nossas classes? Para chegar à essa conclusão, somos auxiliados por uma convenção estrutural para todos os métodos. Essa convenção é determinada pelos aspectos abaixo:

1. **Qual a proposta principal do método?** Você deve se perguntar constantemente até compreender a real finalidade do mesmo.
2. **Qual o tipo de retorno esperado após executar o método?** Você deve analisar se o método será responsável por retornar algum valor ou não.

Caso o método não retorne nenhum valor, ele será representado pela palavra-chave `void`.

1. **Quais os parâmetros serão necessários para execução do método?** Os métodos às vezes precisarão de argumentos como critérios para a execução.
2. **O método possui o risco de apresentar alguma exceção?** Exceções são comuns na execução de métodos, as vezes é necessário prever e tratar a possível existência de uma exceção.
3. **Qual a visibilidade do método?** Avaliar se será necessário que o método seja visível a toda aplicação, somente em pacotes, através de herança ou somente a nível a própria classe.

Abaixo, temos um exemplo de uma classe com dois métodos e suas respectivas considerações:

```
public class MyClass {

    public double somar(int num1, int num2){
        //LOGICA - FINALIDADE DO MÉTODO
        return ... ;
    }

    public void imprimir(String texto){
        //LOGICA - FINALIDADE DO MÉTODO
        //AQUI NÃO PRECISA DO RETURN
        //POIS NÃO SERÁ RETORNADO NENHUM RESULTADO
    }
    // throws Exception : indica que o método ao ser utilizado
    // poderá gerar uma exceção
    public double dividir(int dividendo, int divisor) throws Exception{}

    // este método não pode ser visto por outras classes no projeto
    private void metodoPrivado(){}

    //alguns equívocos estruturais
    public void validar(){
        //este método deveria retornar algum valor
        //no caso boolean (true ou false)
    }
    public void calcularEnviar(){
        //um método deve representar uma única responsabilidade
    }
    public void gravarCliente(String nome, String cpf, Integer telefone, ....){
        //este método tem a finalidade de gravar
        //informações de um cliente, por que não criar
        //um objeto cliente e passar como parâmetro ?
        //veja abaixo
    }
    public void gravarCliente(Cliente cliente){}
    //ou
    public void gravar(Cliente cliente){}
}
```

Exercitando

Vamos criar um exemplo de uma classe para representar uma SmartTV onde:

1. Ela tenha as características: ligada (boolean), canal (int) e volume (int);
2. Nossa TV poderá ligar e desligar e assim mudar o estado ligada;
3. Nossa TV aumentará e diminuirá o volume sempre em +1 ou -1;
4. Nossa TV poderá mudar de canal de 1 em 1 ou definindo o número correspondente.

Previous
Operadores

Next
Escopo

Last updated 1 year ago