Métodos Prof. Nelson Bellincanta Filho

Métodos

- São sub-rotinas associadas aos objetos;
- Trechos de código que permitem realizar ações ou transformações sobre os valores dos atributos desse código, modificando seu estado e proporcionando o comportamento desejado;
- ▶ Os métodos podem ser declarados dentro de suas classes com esta sintaxe:

```
<especAcesso> <tipoRetorno> nomeMétodo ( [listaParâmetros] ) {
    // corpo do método
}
```

Métodos

- O especificador de acesso determina se o método tem uso livre (public), restrito às suas subclasses (protected) ou de uso interno (private);
- O tipo de retorno indica qual é a espécie de valor devolvida como resultado do acionamento do método;
- O nome identifica o método dentro da classe;
- Os parênteses são obrigatórios após o nome do método e contêm uma lista opcional de parâmetros;
- ▶ O corpo do método é um bloco de código que define suas ações.

Métodos

► Cada parâmetro da lista deve ser declarado como uma variável distinta, com um par tipo-nome, separada das demais por uma vírgula, mesmo que possuam tipo idêntico a outros parâmetros, como no código esquematizado a seguir:

```
// um parâmetro de tipo int
boolean par(int n) { ... }
// dois parâmetros do mesmo tipo
int maximo(int a, int b) { ... }
// vários parâmetros void coord(int n, double a, double b, String s) { ... }
```

Assinatura de método



Assinatura de método

- ▶ O nome do método e a lista dos tipos de seus parâmetros são denominados assinatura do método;
- ▶ O tipo de retorno do método não faz parte de sua assinatura.

```
Classe exemplo utilização métodos: ExMetodoHora.java
    IFPR - Campus Cascavel
    Disciplina: Programação Orientada à Objetos
    Professor: Nelson Bellincanta
 6
    Data da criação: 17/05/2023
    */
 8
 9
10
    // declaração da classe ExMetodoHora
    public class ExMetodoHora {
11
        public int hor, min, seg; // três atributos do tipo int
12
13
        public void setHorario(int h, int m, int s) {
14
            hor = h;
15
            min = m;
16
            seq = s;
17
        public String toString() {
18
19
            return hor + ":" + min + ":" + seg;
20
```

Tipos de métodos



Tipos de métodos

- Métodos de alteração (ou mutação) modificam o valor dos campos de um objeto, alterando seu estado;
- Métodos de observação apenas retornam um valor contido num campo do objeto, ou o resultado de algum cálculo, sem que o estado do objeto seja alterado;



Tipos de métodos

- Métodos de produção são capazes de retornar um outro objeto, do mesmo tipo, contendo uma cópia ou uma parte dos dados que contém;
- Métodos de criação permitem a criação de novos objetos;
- Métodos de destruição são responsáveis pela remoção de um objeto da memória.



▶ O operador . (seletor) permite acessar os campos públicos de uma classe, assim como acionar seus métodos públicos por meio de suas variáveis de instância.

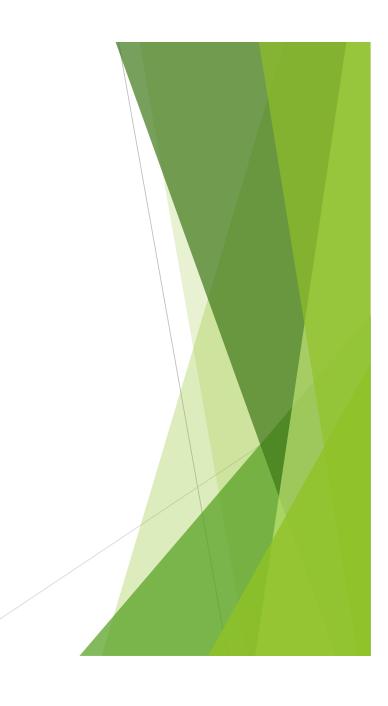
```
// instanciação de objeto tipo ExMetodoHora
ExMetodoHora inicio = new ExMetodoHora();
// ajuste dos campos hor, min e seg usando método setHorario
inicio.setHorario(12, 45, 38);
// exibição simplificada do horário
System.out.println("Inicio: " + inicio.toString() );
```

Os métodos podem e devem ser construídos para facilitar a utilização dos objetos de uma classe e também empregados para garantir a consistência de seus valores internos:

```
// instanciação de objeto tipo ExMetodoHora
ExMetodoHora horario = new ExMetodoHora ();
// ajuste dos campos hor, min e seg com valores inválidos
horario.hor = 44;
horario.min = 1023;
horario.seg = -23;
// uso do método horário com valores inválidos
horario.setHorario(234, -90, 75);
```

- ► Embora tais valores sejam inaceitáveis na definição de um horário, são perfeitamente compatíveis com o tipo int indicado para seus campos, caracterizando um erro lógico sério e de difícil solução na programação convencional;
- A restrição do acesso direto a esses campos, ou seja, de sua visibilidade, é uma forma de resolver elegantemente essa questão;
- Declarar os campos hor, min e seg como private evita seu uso indevido pelas instâncias, resolvendo o problema inicial, mas impedindo a atribuição de valores.

Parâmetros e argumentos



Parâmetros e argumentos

- Um parâmetro formal é uma variável declarada pelo método em sua lista de parâmetros, o que indica sua obrigatoriedade como um valor de entrada para esse método;
- Cada parâmetro deve possuir um nome distinto e a definição de seu tipo de dados. Um método não precisa exigir parâmetros, mas pode requerer um, dois ou vários parâmetros;
- No corpo do método, os parâmetros funcionam como variáveis locais, que permitem alterar a execução do código ali presente.

```
<especAcesso> <tipoRetorno> nomeMétodo ( [listaParâmetros] ) {
    /* corpo do método */
}
```

Parâmetros e argumentos

- Quando um método é acionado, é necessário fornecer, dentro dos parênteses obrigatórios, um valor para cada parâmetro especificado na sua lista de parâmetros, na ordem e com o tipo de dados indicado;
- Os valores efetivamente fornecidos para um método são seus argumentos.



Escopo de classe e escopo de instância

- Existem para
 - Atributos;
 - Métodos.

```
public class Celular{
public String numero;
public static String empresa;
}
```

Escopo de Classe	Escopo de Instância
Uso do modificador static.	Não usa static.
O conteúdo da variável pertence à classe.	O conteúdo da variável pertence aos objetos.
Somente um valor comum poderá ser armazenado na variável.	Cada objeto pode manter um valor diferente para sua variável.



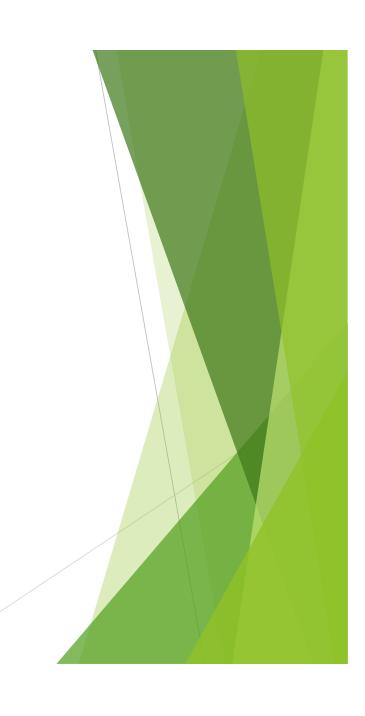
Encapsulamento

- Mecanismo que possibilita restringir o acesso a variáveis e métodos da classe (até a própria classe);
- Define o que está acessível na classe;



Encapsulamento - Vantagens

- Tornar o código mais legível;
- Minimizar os erros de programação;
- Restringir o conteúdo das variáveis;
- Facilitar a ampliação do código em função de novas atualizações.



Encapsulamento - nível de acesso

- public: Nível sem restrições, equivalente a não encapsular, ou seja, se uma variável for definida como pública, não será possível realizar o encapsulamento;
- private: Nível de maior restrição em que apenas a própria classe pode ter acesso a variáveis e/ou métodos;
- protected: Nível intermediário de encapsulamento em que as variáveis e métodos podem ser acessados pela própria classe ou por suas subclasses;
- package: nível em que a classe pode ser acessada apenas por outras classes pertencentes ao mesmo pacote.

Encapsulamento - nível de acesso

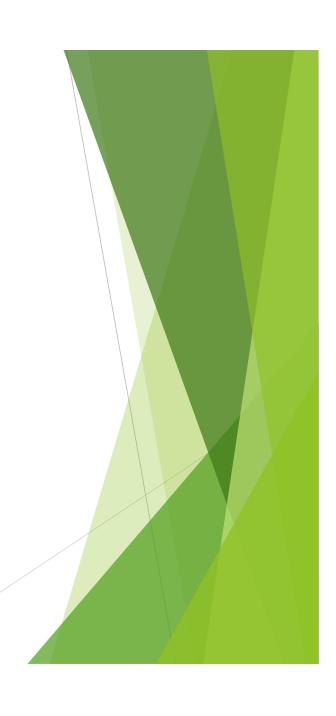
Televisor

-volune : int
-canal : int

+aumentarVolume() : void +reduzirVolume() : void

+trocarCanal(int canal): void

+mostrar(): String



Encapsulamento - nível de acesso

- Conteúdo das variáveis é acessado por métodos públicos get e set;
- O método set recebe como parâmetro o mesmo tipo de dado do atributo da classe e retorna void

```
public void setNomeDoAtributo (tipo_do_atributo nomeDoAtributo) {
    this.nomeDoAtributo = nomeDoAtributo;
}
```

 O método get não recebe nenhum parâmetro e sempre retorna o mesmo tipo de dado do atributo da classe

```
public tipo_do_atributo getNomeDoAtributo () {
    return nomeDoAtributo;
}
```



Uso da palavra reservada this

A palavra reservada this faz referência ao objeto corrente, isto é, ao objeto que chamou o método.

```
public void setCanal (int c){
canal = c;
}
```

- ▶ O parâmetro "c" é local, enquanto a variável "canal" é global à classe e se refere ao atributo da classe;
- Nesse caso, o valor a ser atribuído ao atributo "canal" é recebido por meio do parâmetro "c".

Uso da palavra reservada this

Na maioria das aplicações, é desejável manter o mesmo nome da variável tanto para o parâmetro quanto para o atributo da classe;

```
public void setCanal (int canal){
canal = canal;
}
```

- Esse código traria problemas, porque o parâmetro local "canal" (que está dentro dos parênteses) tem o mesmo nome do atributo da classe;
- lsso faria com que as duas variáveis fossem consideradas locais, isto é, do próprio método "setCanal".

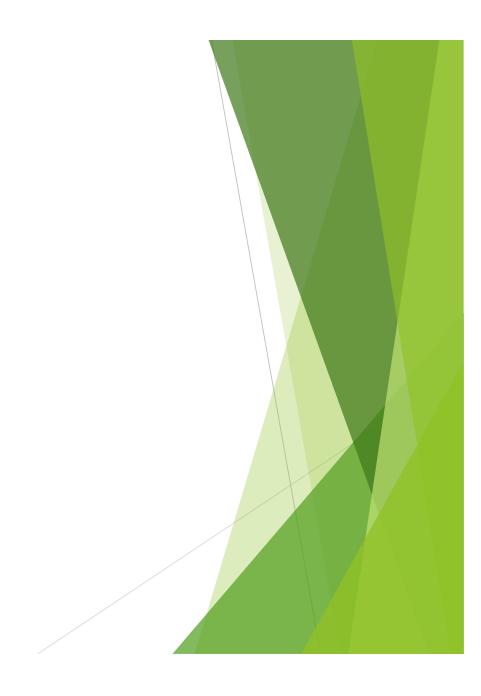
Uso da palavra reservada this

Para resolver esse problema, basta inserir a palavra reservada this ao lado do nome do atributo;

```
public void setCanal (int canal){
    this.canal = canal;
}
```

- O uso da palavra reservada this permite diferenciar as duas variáveis, a local (do parâmetro do método) e a global (referente à variável do atributo);
- O uso da palavra reservada this faz referência ao objeto corrente, então this.canal é o mesmo que tv.canal, pois usamos o objeto "tv" para invocar o método "setCanal".

Construtores



Construtores

- São métodos especiais destinados à inicialização e ao preparo de novos objetos durante sua criação, ou seja, durante sua instanciação;
- Assim como os métodos comuns, os construtores (constructors) podem receber parâmetros, o que permite caracterizar um objeto durante sua criação, no entanto só podem ser acionados por meio do operador new, responsável pela criação de novos objetos;
- Obrigatoriamente, os construtores devem ter o mesmo nome que suas classes, além de não possuírem tipo de retorno, pois o resultado de sua chamada é sempre uma nova instância;
- Dependendo das necessidades, uma classe pode conter de 0 a N construtores declarados.

Finalizadores e coleta de lixo

Finalizadores e coleta de lixo

- A criação de objetos em Java é realizada explicitamente por meio do uso do operador new combinado com os construtores da classe usada;
- Contudo a destruição dos objetos (sua remoção da memória), é automática e fica a cargo do coletor de lixo automático (automatic garbage collector), uma tarefa realizada pela JVM, enquanto os programas Java são executados;
- ▶ O coletor de lixo é um processo automático da JVM que procura continuamente por objetos não mais referenciados, que não podem mais ser usados, marcando os para futura remoção quando a memória que ocupam for necessária para novos objetos.

Finalizadores e coleta de lixo

- A coleta automática de lixo resolve a maioria das situações envolvendo a criação e destruição de objetos, mas existem casos particulares em que o objeto precisa devolver recursos alocados do sistema ou encerrar conexões com outros programas (por exemplo, bancos de dados);
- Esses casos pedem a implementação dos finalizadores, métodos cuja assinatura obrigatória é:

```
protected void finalize () {
    /* devolução de recursos ou encerramento de conexões */
}
```

Referências

- ▶ DEITEL, Harvey M.; DEITEL, Paul J. **Java: como programar.** 10ª ed. Editora Pearson Education do Brasil, 2017.
- ► FURGERI, Sérgio. Java 8: Ensino Didático. 1ª ed., São Paulo: Érica, 2015.
- ▶ JUNIOR, Peter Jandl. **Java Guia do Programador.** 4ª Edição: Atualizado para Java 16. Novatec Editora, 2021.