



#Reskilling4Employment  
Software Developer

# Paradigmas de Programação

Aula 03 - Métodos

Vitor Santos



# Conteúdos

1. Palavras Reservadas
2. Métodos
3. Sobrecarga de Métodos
4. Encapsulamento
5. Métodos de Acesso
6. Palavras Reservadas Usadas
7. Links Úteis

# Palavras Reservadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

# Métodos

- Um **método** numa linguagem orientada a objectos é o equivalente a um procedimento/função numa linguagem não orientada a objectos.



- Um **método** é a construção de um mecanismo (método) para realizar algum ato.

### Definição Sintáctica de um método

```
void <nome_metodo>(<argumentos>...) {  
    <bloco de instruções>...  
}
```

```
<tipo_retorno> <nome_metodo>(<argumentos>...) {  
    <bloco de instruções>...  
}
```

- Para a classe `Cao` conseguimos pensar rapidamente num método que faz todo o sentido:

```
public class Cao{  
    String nome;  
    String raca;  
    String latido = "woof!";  
    int idade= 6;  
    Dog(String nomeTemp, raca racaTemp, int idadeTemp) {  
        nome = nomeTemp;  
        raca = racaTemp;  
        idade = idadeTemp;  
    }  
}
```

- Esse seria o método **`ladrar()`**!

```
public class Cao{  
    String nome;  
    String raca;  
    String latido = "woof!";  
    int idade= 6;  
  
    Dog(String nomeTemp, raca racaTemp, int idadeTemp) {  
        nome = nomeTemp;  
        raca = racaTemp;  
        idade = idadeTemp;  
    }  
  
    void ladrar( ){  
        System.out.println(latido);  
    }  
}
```

Criamos o método ladrar( ).

- Dada a instância de uma entidade podemos invocar um comportamento com um "." que faz a associação de uma instância com um método.
- Sintaxe de invocação de um método:

```
<instancia>.<comportamento>()
```

```
<variável> = <instancia>.<comportamento>(<argumentos>...)
```

- Para invocarmos o método do exemplo anterior temos de fazer o seguinte:

```
Cao fido = new Cao("Fido", "Bulldog", 5);  
fido.ladrar();
```



- Em Java podemos definir múltiplos métodos com o mesmo nome desde que tenham assinaturas diferentes.
- As assinaturas de um método são a combinação do nome do método, do tipo de retorno e da lista de argumentos.
- A linguagem Java tem a restrição do tipo de retorno que não contribui para a assinatura do método.
- Em Java não podemos ter dois métodos com o mesmo nome e lista de argumentos mas tipos de retorno diferentes.

# Sobrecarga de métodos

- Nem todos os cães soam da mesma forma.
- Para implementarmos uma alteração ao ladrar do cão podemos definir um novo método alternativo ao ladrar( ) que aceite uma String como parâmetro.



```
public class Dog {  
    (...)  
    public void ladrar() {  
        System.out.println(latido);  
    }  
    public void ladrar(String latidoNovo) {  
        System.out.println(latidoNovo);  
    }  
}
```

- Esta versão de Cao é permitida porque apesar de existirem dois métodos ladrar( ), as diferenças das assinaturas permitem ao interpretador de Java escolher a invocação do método apropriado.

- A sintaxe da definição do método `ladrar(String latido)` indica que aceita um argumento do tipo `String`.
- Como exemplo de sobrecarga de métodos considere o programa `VidaDeCao` onde são criados dois cães cada um com um comportamento diferente para o método `ladrar`.

```
public class VidaDeCao {  
    public static void main(String[] args) {  
        Cao fido = new Cao("Fido", "Bulldog", 5);  
        Cao spot = new Cao("Spot", "Golden", 2);  
        fido.ladrar( );  
        spot.ladrar("Arf Arf");  
        fido.ladrar("Ruff Ruff");  
    }  
}
```

### OUTPUT CONSOLA

woof!  
Arf Arf  
Ruff Ruff

- Como Cao suporta dois tipos de comportamento diferentes para o método ladrar foi invocado no método anterior um ladrar diferente para os cães fido e spot.
- Note-se que as alterações do ladrar do fido surgiram depois do ladrar do spot.

# Encapsulamento



- Um objecto deve ser visto como uma cápsula de forma a conseguirmos obter uma certa independência do contexto.
- Podemos com isto ter uma maior facilidade na reutilização, detecção de erros e modularidade.

# Objeto

Dados Privados

Método Público 1

Método Público 2

Método Público 3

Método Privado 1

- Dentro de um destes módulos (cápsula), os dados, os procedimentos ou ambos podem ser privados ou públicos.
- Os dados e os procedimentos privados são conhecidos e acessíveis apenas pelo próprio objeto e não por qualquer programa externo ao objeto.



- Quando os dados ou os procedimentos de um módulo são públicos, podem ser acedidos por um qualquer programa.
- Tipicamente os procedimentos públicos de um módulo são usados para fornecer um interface controlado para os elementos privados do módulo.

```
public class Cao{  
    private String nome;  
    private String raca;  
    private String latido = “woof!”;  
    private int idade= 6;  
  
    public Dog(String nomeTemp, String racaTemp, int idadeTemp) {  
        nome= nomeTemp;  
        raca = racaTemp;  
        idade= idadeTemp;  
    }  
  
    public void ladrar( ){  
        System.out.println(latido);  
    }  
  
    public void ladrar(String latidoNovo){  
        System.out.println(latidoNovo);  
    }  
}
```

# Métodos de Acesso



- Para podermos alterar o valor de uma variável de instância ao longo do tempo temos que ter um método para alterar o seu valor.
- Esse método é tipicamente referido como método de acesso.

- Tipicamente se uma classe tem variáveis de instância que são suportadas por operações "set" (de atribuição) também têm operações "get" (obter).
- Por convenção um método que afete ou altere o valor de uma variável de instância deve começar com a palavra "set".


```
public void setLatido(String latido) {  
    this.latido = latido;  
}
```

- Este método é interessante porque usa variáveis com o mesmo nome, latido.

- O **latido** definido como um parâmetro é o novo ladrar.
- Temos também mais um **latido** que é uma variável de instância de **Cao**.
- Com a linguagem Java podemos nos referir a esta variável de instância com o "**this**".

```
this.latido = latido;
```

- Para cada método "set" devemos ter o correspondente "get".



```
public String getLatido() {  
    return this.latido;  
}
```

- No caso das variáveis de instância booleanas alguns programadores gostam de usar o "is" em vez do "get".

- Na versão anterior de VidaDeCao crie um objeto fido e altere as características do ladrar de Woof para Ruff e depois invoque o método ladrar.

```
public class DogChorus {  
    public static void main(String[] args) {  
        Cao fido = new Cao();  
        fido.setLatido("Ruff.");  
        fido.ladrar();  
    }  
}
```

```
public class Cao{  
    private String nome;  
    private String raca;  
    private String latido = "woof!";  
    private int idade= 6;  
  
    public Dog(String nomeTemp, String racaTemp, int idadeTemp) {  
        this.nome= nomeTemp;  
        this.raca = racaTemp;  
        this.idade= idadeTemp;  
    }  
  
    public void setLatido(String latido){  
        this.latido = latido;  
    }  
  
    public String getLatido(){  
        return this.latido;  
    }  
    (...)  
}
```

Fazer o mesmo para as outras  
variáveis de instância



# Métodos de Instância



- Os métodos criados até agora são denominados de métodos de instância porque são invocados relativamente a uma instância de uma classe.

- É por esta razão que um método de instância pode referenciar uma variável directamente sem o qualificador `this` desde que não haja conflito com mais nenhuma variável.

```
public void ladrar() {  
    System.out.println(latido);  
}
```

# Palavras Reservadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0



#Reskilling4Employment  
Software Developer

# Paradigmas de Programação

Aula 03 - Métodos

Vitor Santos

