

# Programação Orientada a Objetos

## Semana 04

## Reflexão

“Ninguém é tão grande que não possa aprender, nem tão pequeno que não possa ensinar.”

Esopo

## Manipulação de datas

As principais classes para manipulação de datas pertencem aos pacotes `java.util` e `java.text`. São elas:

`java.util.Date`

`java.text.DateFormat`

`java.text.SimpleDateFormat`

## Manipulação de datas

Para instanciar um objeto Date é muito simples:

```
Date hoje = new Date();  
System.out.println ("A data de hoje é: "+ hoje);
```

A saída de execução deste programa será:

A data de hoje é: Wed Dec 21 01:00:20 BRST 2016

# SimpleDateFormat

Para formatar a representação de um objeto Date é possível utilizar a classe SimpleDateFormat.

Esta classe fornece um conjunto de caracteres padrão para formatação do objeto Date.

Símbolo	Significado	Apresentação	Exemplo
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1~12)	(Number)	12
H	hour in day (0~23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1~24)	(Number)	24
K	hour in am/pm (0~11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
	escape for text	(Delimiter)	
	single quote	(Literal)	

## Exemplos

```
Date hoje = new Date();
```

```
System.out.println ("A data de hoje é: "+ hoje);
```

```
String formato = "dd/MM/yyyy";
```

```
SimpleDateFormat formatter = new SimpleDateFormat(formato);
```

```
System.out.println("A data formatada é: "+ formatter.format(hoje));
```

```
formatter = new SimpleDateFormat("EEEE, dd 'de' MMMM 'de' yyyy");
```

```
System.out.println("Ou, hoje é: "+ formatter.format(hoje));
```

A data de hoje é: Wed Dec 21 01:08:48 BRST 2016

A data formatada é: 21/12/2016

Ou, hoje é: Quarta-feira, 21 de Dezembro de 2016

## Exemplos

```
Locale brasil = new Locale ("pt", "BR");
```

```
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG,  
brasil);
```

```
System.out.println("Hoje no Brasil: "+ df.format(hoje));
```

Hoje no Brasil: 21 de Dezembro de 2016

```
df = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);  
System.out.println("Aujourd'hui au France: "+ df.format(hoje));
```

```
df = DateFormat.getDateInstance(DateFormat.LONG, Locale.US);  
System.out.println("Today in USA: "+ df.format(hoje));
```

Aujourd'hui au France: 21 décembre 2016  
Today in USA: December 21, 2016

# Métodos

Termos importantes:

- ✓ Mecanismo de comunicação
- ✓ Assinatura
- ✓ Sem retorno
- ✓ Com retorno
- ✓ Parâmetros
- ✓ Visibilidade



## Mecanismo de comunicação

Aumentam a facilidade!

- Reduzem a complexidade
  - ✓ Abstração
  - ✓ Encapsulam a informação
  - ✓ Minimizam o tamanho do código

- Aumentam a manutenibilidade e a correção
  - ✓ Evitam duplicação do código
  - ✓ Limitam o efeito das mudanças
  - ✓ Promovem a reutilização do código

Diminuem o custo!

## Assinatura de Métodos

A assinatura de um método é o que permite sua identificação.

A assinatura consiste do nome do método e da lista de parâmetros

Exemplos:

```
public static void main(String args[]);
```

**Assinatura**

```
public static double sqrt(double pVal);
```

## Assinatura de Métodos

É possível sobrecarregar um método, colocando duas definições diferentes para ele.

Isto significa que, dependendo dos parâmetros que ele receber, ele vai ter um comportamento diferente.

Exemplo:

```
public static double sqrt(double pVal);
```

```
public static float sqrt(float pVal);
```

## Assinatura de Métodos

Logo, se dois métodos têm o mesmo nome, o método correto será chamado com base nos argumentos que lhe são passados.

Por exemplo, o código abaixo chama os diferentes métodos definidos anteriormente.

```
float hypotenuse = 10.0f;  
double longLeg = 5.0;
```

```
MyClass.sqrt(hypotenuse);  
MyClass.sqrt(longLeg);
```

## Definindo métodos

Para declarar métodos, usamos a seguinte sintaxe:

**Fixo, por enquanto**

```
public static <tipo_retorno> <nome> (<tipo> arg1, <tipo> arg2, ...)
```

**Tipo válido do Java, incluindo objetos**

**Identificador válido do Java**

```
{  
    ...código do método...  
    return <valor de retorno>  
}
```

**Valor do mesmo tipo da definição do método**

## Métodos sem retorno

Não retornam valores. Os métodos que não retornam valores devem ser definidos como **void**.

### Sintaxe:

modificador-de-acesso **void** nome-do-método ([lista-de-argumentos]) { código do corpo }

### Exemplo:

O exemplo mostra a chamada de um método (imprime) que imprime na tela uma frase qualquer.

```
3 public class TesteMetodo {  
4     public static void main(String args[]) {  
5         semRetorno(); // invocação do método  
6     }  
7     private static void semRetorno() { // declaração do método  
8         System.out.println("Aprendendo a Linguagem Java");  
9     }  
10 }
```

## Métodos com retorno

São métodos que retornam valores, os quais podem ser comparados às funções de outras linguagens.

### Sintaxe:

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

### Exemplo:

O exemplo mostra a chamada de um método (retornaNome) que retorna uma String.

```
public String retornaNome() { return nome; }
```

## Métodos com retorno – Exemplo

```
3 public class Metodos {
4     String nome = "Sérgio Furgeri";
5
6     public String retornaNome() {
7         return nome;
8     }
9     public static void main(String[] args) {
10         Metodos retorno = new Metodos();
11         System.out.println(retorno.retornaNome());
12     }
13 }
```



# Métodos com passagem de parâmetros

São métodos que recebem parâmetros como valores de entrada.

## Sintaxe:

```
modificador-de-acesso [tipo do método] nome-do-  
método ([parâmetros]) { código do corpo }
```

O exemplo seguinte mostra a chamada de um método (soma) que recebe dos valores como passagem de parâmetro.

## Exemplo

```
3 import javax.swing.*;
4 public class PassagemParametro {
5     public static void main(String args[]) {
6         int n1 = Integer.parseInt(JOptionPane.showInputDialog
7             (null, "forneça o 1º número inteiro"));
8         int n2 = Integer.parseInt(JOptionPane.showInputDialog
9             (null, "forneça o 2º número inteiro"));
10        int res = soma(n1, n2);
11        JOptionPane.showMessageDialog(null, "Numeros fornecidos : "
12            + n1 + ", " + n2 + "\nResultado = " + res);
13    }
14
15    public static int soma(int numero1, int numero2) { // declaração do método
16        return (numero1 + numero2); // retorna a soma dos números passados }
17    }
18 }
```

## Exercícios

1. Uma operação necessária entre datas é a verificar se uma data ocorre antes de outra. O algoritmo para comparação é muito simples, e seus passos estão abaixo. Nesse algoritmo, consideramos que dia1, mês1 e ano1 são os dados da primeira data, e que dia2, mês2 e ano2 são os dados da segunda data.

Se ano1 < ano2 a primeira data vem antes da segunda.

Se ano1 > ano2 a primeira data vem depois da segunda.

Se ano1 == ano2 e mês1 < mês2 a primeira data vem antes da segunda.

Se ano1 == ano2 e mês1 > mês2 a primeira data vem depois da segunda.

Se ano1 == ano2 e mês1 == mês2 e dia1 < dia2 a primeira data vem antes da segunda.

Se ano1 == ano2 e mês1 == mês2 e dia1 > dia2 a primeira data vem depois da segunda.

Se nenhum desses casos ocorrer, as datas são exatamente iguais.

Escreva um método chamado `compararDatas` que recebe duas datas e retorna um valor inteiro conforme abaixo:

0	As datas são iguais
1	A data 1 ocorre antes da data 2
2	A data 2 ocorre antes da data 1
-1	Erro, houve erro no processamento das datas

## Exercícios

2. Crie uma aplicação em Java que contenha o nome e a data de aniversário de uma pessoa e implemente um método que retorne uma mensagem de feliz aniversário, precedido do nome da pessoa, caso seja o dia de seu aniversário (baseado na data atual do sistema).

3. Execute o código abaixo e explique seu funcionamento.

```
import java.util.Date;
import javax.swing.JOptionPane;
public class Data1 {
    public static void main(String[] args) {
        final long MILI_SEGUNDOS_POR_DIA = 1000 * 60 * 60 * 24;
        Date agora = new Date();
        long tempo = agora.getTime();
        Date novaData = new Date(agora.getTime() + (MILI_SEGUNDOS_POR_DIA * 10));
        JOptionPane.showMessageDialog(null,
            "Milisegundos desde 1970: " + tempo +
            "\nDias: " + tempo / MILI_SEGUNDOS_POR_DIA +
            "\nMeses: " + tempo / MILI_SEGUNDOS_POR_DIA / 30 +
            "\nAnos: " + tempo / MILI_SEGUNDOS_POR_DIA / 365 +
            "\nData: " + novaData);
    }
}
```

## Exercícios

4. Um estudante de geometria deseja estudar mais sobre triângulos. Para ajudá-lo nesta tarefa, crie uma aplicação que armazena as medidas dos lados de um triângulo e contém os seguintes métodos:
- a) **setLados:** recebe os três lados de um triângulo. Este método deve validar se os lados podem formar um triângulo. Para isto, a soma de dois lados quaisquer deve ser maior que o terceiro lado. Caso não seja possível formar um triângulo, mostre uma mensagem de aviso.
  - b) **tipoTriangulo:** este método deve retornar uma String indicando se o triângulo é equilátero (todos os lados com medidas iguais), isósceles (apenas dois lados com medidas iguais) ou escaleno (todos os lados com medidas diferentes).
  - c) **ehTrianguloRetangulo:** este é um método lógico que avalia se os lados formam um triângulo retângulo. lembre-se que para ser retângulo, a soma do quadrado dos dois lados menores (catetos) tem que ser igual ao quadrado do lado maior (hipotenusa). Deve retornar verdadeiro ou falso.

## Exercícios

5. Crie uma aplicação que solicite ao usuário duas datas, e apresente em tela, as duas datas fornecidas, a diferença de dias entre elas, e qual a nova data 30 dias após a maior data.
6. Crie uma aplicação que contenha dois métodos lógicos: um denominado letraVogal que retornará true se a letra recebida for uma vogal e false caso contrário e outro denominado letraConsoante que faz o mesmo para uma consoante.
7. Um professor aplicou 3 provas durante um semestre mas só vai levar em conta as duas notas mais altas para calcular a média. Crie uma aplicação em Java que peça o valor das 3 notas e execute os métodos seguintes:
  - a) calcularMedia: apresenta em tela a média das duas notas mais altas. Esse método deve receber a nota das três provas em tipo float.
  - b) maiorNota: apresenta em tela a maior nota entre as 3 provas. Esse método deve receber a nota das três provas em tipo float.

## Exercícios

8. Crie uma aplicação que simule uma "Calculadora" com os atributos `numero1` e `numero2` e os seguintes métodos:
  - a) **definirValores** – este método deve receber como parâmetros, dois valores inteiros que devem ser colocados em cada um dos atributos (obs.: este método não deve retornar valor algum).
  - b) **retornarSoma** – este método não deve receber parâmetros, mas deve retornar a soma dos valores atuais dos seus atributos (`numero1` e `numero2`).
  - c) **retornarMultiplicacao** – este método não deve receber parâmetros, mas deve retornar a multiplicação dos valores atuais dos seus atributos.
  - d) Crie o método ***main*** para executar os métodos acima.

## Exercícios

9. Crie uma aplicação que contenha um método sem retorno, para mostrar na tela 10 vezes a frase “Esse é um método sem retorno”.
10. Crie uma aplicação contendo um método com passagem de parâmetro que realize a conversão entre as temperaturas Celsius e Fahrenheit. Sendo C a temperatura em Célsius e F em fahrenheit, as fórmulas de conversão são:

$$C = 5.(F-32)/9$$

$$F = (9.C/5) + 32$$

Solicite o tipo de temperatura a ser convertido (C ou F), o valor da temperatura e mostre a conversão.