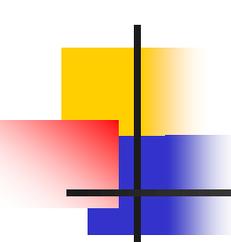


# **Classes Essenciais da API Java**

---

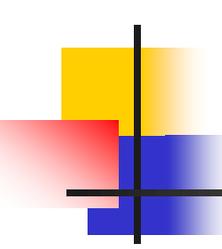
Alberto Costa Neto  
DComp - UFS



# Roteiro

---

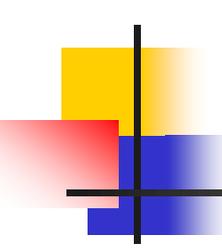
- Java API
- Pacote java.lang
- Pacote java.math
- Pacote java.util



# Java API

---

- O que é Java API?
  - *Java Application Programming Interface*
  - Centenas de classes predefinidas e compiladas (bibliotecas)
  - Lembram-se de:
    - `System.out.println...`
    - `Scanner dado = new Scanner(System.in);`



# Java API

---

- Para usar uma classe da API...
  - É preciso descobrir em qual pacote ela está...
    - Que classes existem na biblioteca?
    - Como descobrir o que cada classe faz?

**Java™ Platform  
Standard Ed. 6**

[All Classes](#)

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)

**All Classes**

- [AbstractAction](#)
- [AbstractAnnotationValueVisitor6](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChooserPanel](#)
- [AbstractDocument](#)
- [AbstractDocument.AttributeContext](#)
- [AbstractDocument.Content](#)
- [AbstractDocument.ElementEdit](#)
- [AbstractElementVisitor6](#)
- [AbstractExecutorService](#)
- [AbstractInterruptibleChannel](#)
- [AbstractLayoutCache](#)
- [AbstractLayoutCache.NodeDimensions](#)
- [AbstractList](#)
- [AbstractListModel](#)
- [AbstractMap](#)
- [AbstractMap.SimpleEntry](#)
- [AbstractMap.SimpleImmutableEntry](#)
- [AbstractMarshallerImpl](#)
- [AbstractMethodError](#)
- [AbstractOwnableSynchronizer](#)
- [AbstractPreferences](#)
- [AbstractProcessor](#)

**Overview** Package Class Use **Tree** [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#) [FRAMES](#) [NO FRAMES](#)

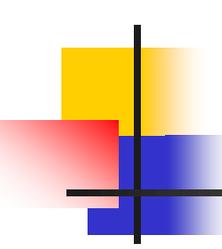
**Java™ Platform, Standard Edition  
API Specification**

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See: [Description](#)

**Packages**

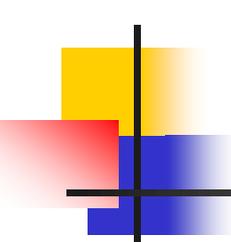
<a href="#">java.applet</a>	Provides the classes necessary to create an applet and context.
<a href="#">java.awt</a>	Contains all of the classes for creating user interfaces a
<a href="#">java.awt.color</a>	Provides classes for color spaces.
<a href="#">java.awt.datatransfer</a>	Provides interfaces and classes for transferring data be
<a href="#">java.awt.dnd</a>	Drag and Drop is a direct manipulation gesture found i mechanism to transfer information between two entities
<a href="#">java.awt.event</a>	Provides interfaces and classes for dealing with differer
<a href="#">java.awt.font</a>	Provides classes and interface relating to fonts.
<a href="#">java.awt.geom</a>	Provides the Java 2D classes for defining and performi geometry.
<a href="#">java.awt.im</a>	Provides classes and interfaces for the input method fra
<a href="#">java.awt.im.spi</a>	Provides interfaces that enable the development of inp environment.
<a href="#">java.awt.image</a>	Provides classes for creating and modifying images



# Java API

---

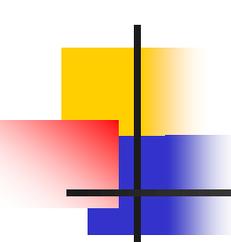
- A documentação da API não vem incorporada ao download do jdk
- <http://download.oracle.com/javase/>



# Java API

---

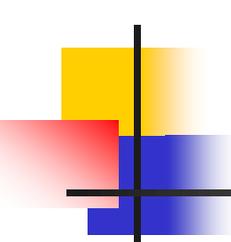
- Quais APIs existem?
  - Java SE (padrão)
  - Java EE
  - Java ME
  - Java DB
  - Java FX
  - ...



# Roteiro

---

- Java API
- Pacote java.lang
- Pacote java.math
- Pacote java.util

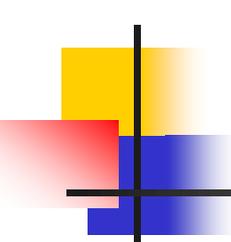


# Pacote java.lang

---

- Provê classes fundamentais para a programação Java
  - Object – raiz da hierarquia de classes
  - Class – permite identificar instâncias em tempo de execução, obter informações sobre classes e até carregar classes dinamicamente
  - Package – permite a implementação de pacotes
- Único pacote que não precisa ser importado

```
import java.lang.*; //desnecessário
```



# Pacote java.lang

---

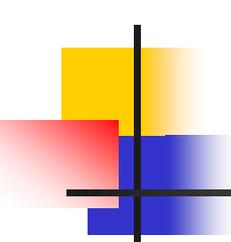
- Entendendo melhor classes que já estamos manipulando...
  - **System**: rotinas do sistema
  - **String**: manipulação de cadeias de caracteres.

# Pacote java.lang

- Classe System

- Não pode ser instanciada
- Provê variáveis estáticas que representam a entrada padrão, saída padrão, saída de erros e outros

System
«final» <u>InputStream in</u>
«final» <u>PrintStream out</u>
«final» <u>PrintStream err</u>
...



# Pacote java.lang

---

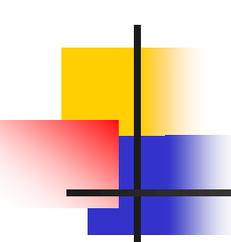
- Classe System

- Métodos

- **currentTimeMillis**: retorna o tempo corrente em milisegundos

```
long horaSistema = System.currentTimeMillis();
```

- **exit**: encerra a JVM
    - **gc**: executa o coletor de lixo



# Pacote java.lang

---

- Classe System
  - Métodos (continuação)
    - **getProperties:** Determina as propriedades atuais do sistema
    - **setIn:** Altera a Stream de entrada padrão
    - **setOut:** Altera a Stream de saída padrão
    - **setErr:** Altera a Stream de saída de erro padrão

# Pacote java.lang

- Anatomia do `System.out.println( ... )`

out é uma  
variável estática  
de System

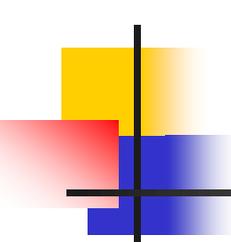


System	
«final»	<u>InputStream in</u>
«final»	<u><b>PrintStream out</b></u>
«final»	<u>PrintStream err</u>
...	



out referencia a  
um objeto que  
representa a  
saída padrão

println é um método de PrintStream



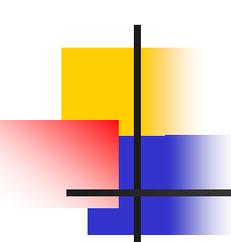
# Pacote java.lang

---

- String
  - Não é um tipo primitivo. **É um objeto!**
    - Strings em Java são instâncias da classe java.lang.String

```
String dog = "Sandy";
```

- Armazena uma cadeia de caracteres



# Pacote java.lang

---

- Como saber se duas strings são iguais?

```
String dog1 = "Sandy";  
String dog2 = "Sandy";  
  
if ( dog1 == dog2 )  
    System.out.println("São iguais");  
else  
    System.out.println("São diferentes");
```

# Pacote java.lang

- Comparação através do método **equals**

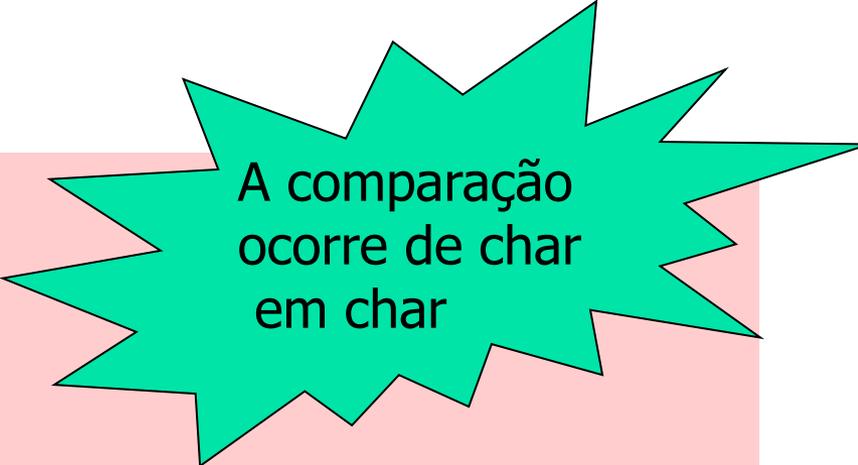
```
String dog1 = "Sandy";  
String dog2 = "Sandy";
```

```
if ( dog1.equals(dog2) )
```

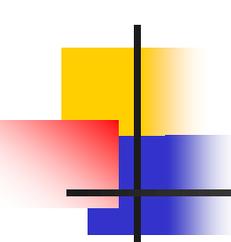
```
    System.out.println("São iguais");
```

```
else
```

```
    System.out.println("São diferentes");
```



A comparação  
ocorre de char  
em char

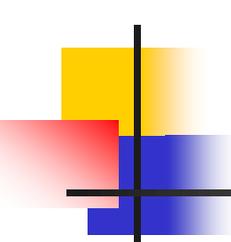


# Pacote java.lang

---

- String - principais métodos:
  - **equals**
    - Retorna true quando a String passada como argumento é igual à String cujo método foi chamado.
  - **length**
    - Retorna o comprimento da String.

```
String dog = "Sandy";  
int tam = dog.length();
```



# Pacote java.lang

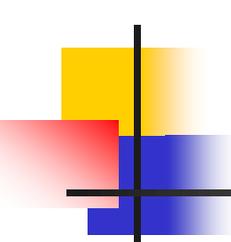
---

- String - principais métodos:

- **substring**

- Retorna uma nova String que representa uma parte da String cujo método foi chamado.

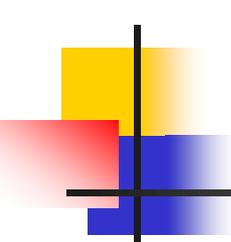
```
String texto = "Sandy e Flock são meus animais de estimação";  
String dog2 = texto.substring(8,12);
```



# Pacote java.lang

---

- String - principais métodos:
    - **charAt**
      - Retorna o caractere localizado na posição especificada (índices variam de 0 a `length() - 1`).
    - **indexOf**
      - Retorna a posição do primeiro caractere que coincide com o passado como argumento.
-

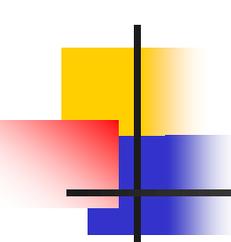


# Pacote java.lang

---

- String - principais métodos:
  - **toLowerCase**
    - Retorna uma String com os caracteres minúsculos.
  - **toUpperCase**
    - Retorna uma String com os caracteres maiúsculos.

```
String nome = "pretinha";  
String nomeUpper = nome.toUpperCase();
```



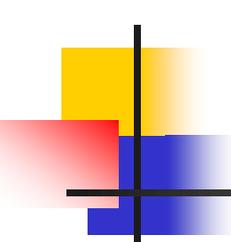
# Pacote java.lang

---

- String
  - Uma String **é imutável**

```
String nome = "pretinha";  
nome.toUpperCase();  
System.out.println(nome);
```

- Como será impresso?
-



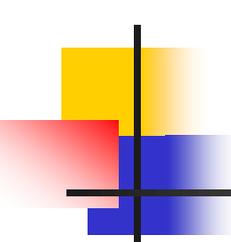
# Pacote java.lang

---

- E agora, como será impresso?

```
String nome = "pretinha";  
nome= nome.toUpperCase();  
System.out.println(nome);
```

- Criação de Strings temporárias
    - Perda de desempenho
-



# Pacote java.lang

---

- Classes importantes ainda não usadas ...
  - StringBuilder
  - StringBuffer
  - Math
  - Classes Wrappers

# Pacote java.lang

## ■ StringBuilder e StringBuffer

### StringBuilder

StringBuilder( )  
StringBuilder(int)  
StringBuilder(String)  
StringBuilder append(*primitive*)  
StringBuilder append(Object)  
StringBuilder insert(int,*primitive*)  
StringBuilder insert(int,Object)  
char charAt(int)  
StringBuilder delete(int,int)  
String toString()

....

**Strings** que podem ser alteradas dinamicamente

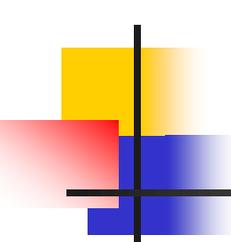
**StringBuffer** tem a mesma funcionalidade que **StringBuilder**, porém serve para contextos onde há concorrência

# Pacote java.lang

- Exemplo para inverter uma string
  - Usando String → implementação ineficiente

```
public static String reverseStr(String source) {  
    String resp = "";  
    for (int i = 0; i < source.length() ; i++)  
        resp = source.charAt(i) + resp;  
    return resp;  
}
```

Cada concatenação cria uma String nova  
O “custo” da operação é alto



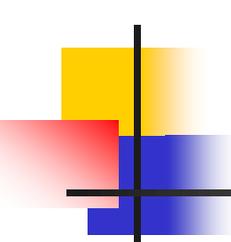
# Pacote java.lang

---

- Exemplo para inverter uma string
  - Usando StringBuilder

```
public static String reverseStr(String source) {  
    int len = source.length();  
    StringBuilder dest = new StringBuilder(len);  
  
    for (int i = (len - 1); i >= 0; i--)  
        dest.append(source.charAt(i));  
    return dest.toString();  
}
```

Obs. **StringBuilder** já tem um método **reverse**

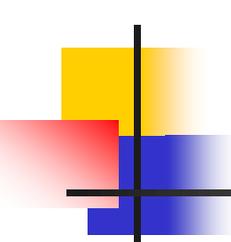


# Pacote java.lang

---

- Math

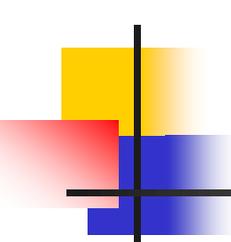
- Possibilita a execução de operações matemáticas
- Constantes
  - **Math.PI** = 3,14...
  - **Math.E** = base de logaritmos
- Métodos Estáticos
  - **Math.min(x,y)**: menor entre 2 valores
  - **Math.max(x,y)**: maior entre 2 valores
  - **Math.random()**: gera número aleatório  $\geq 0$  e  $< 1$



# Pacote java.lang

---

- Math (outros métodos)
  - **Math.pow(x,y)**: x elevado a y
  - **Math.round(x)**: arredonda um número real
  - **Math.sqrt(x)**: raiz quadrada
  - **Math.tan(x)**: Calcula a tangente
  - **Math.sin(x)**: Calcula o seno
  - **Math.cos(x)**: Calcula o coseno
  - **Math.exp(x)**: E elevado a X
  - **Math.log(x)**: Logaritmo na base E



# Pacote java.lang

---

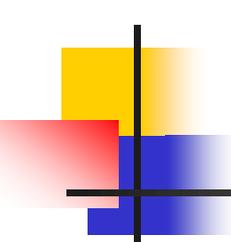
- Exemplos

```
double areaCirculo = Math.PI * Math.pow(r, 2);
```

```
double d = 4.6;  
long i = Math.round(d);
```

```
int x = -4;  
int y = Math.abs(x);
```

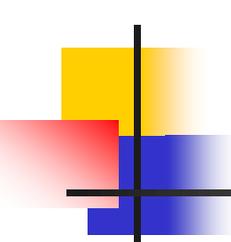
---



# Pacote java.lang

---

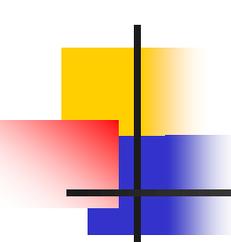
- Classes Wrappers
  - Empacotam valores contidos em variáveis de tipos primitivos em objetos
  - Objetivo
    - Oferecer funcionalidades e facilidades para a manipulação desses tipos



# Pacote java.lang

---

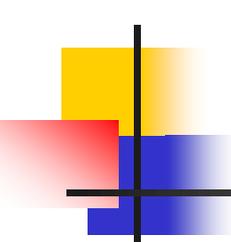
- Classe Boolean (wrapper)
  - Empacota o tipo primitivo boolean
  - Provê métodos de conversão
  - Exemplos
    - **toString**(boolean b): retorna um objeto String representando o valor
    - **valueOf**(String s): retorna um objeto Boolean a partir do conteúdo da string



# Pacote java.lang

---

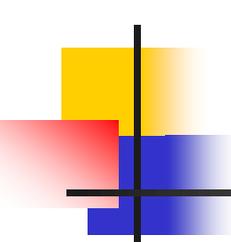
- Classe Character (wrapper)
  - Empacota o tipo primitivo char
  - Provê métodos de categorização e conversão
  - Exemplos
    - **isDigit(char c)**: Retorna true quando o caractere é um dígito
    - **isLetter(char c)**: Retorna true quando o caractere é uma letra
    - **isLetterOrDigit(char c)**: Retorna true quando o caractere é um dígito ou uma letra



# Pacote java.lang

---

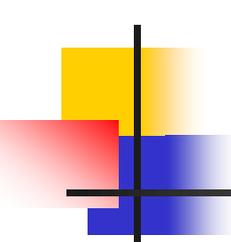
- Classe Character (wrapper)
  - Exemplos
    - **isLowerCase(char c)**: Retorna true se o caractere for minúsculo
    - **isUpperCase(char c)**: Retorna true se o caractere for maiúsculo
    - **toLowerCase(char c)**: Retorna o caractere minúsculo correspondente
    - **toUpperCase(char c)**: Retorna o caractere maiúsculo correspondente



# Pacote java.lang

---

- Classes Wrappers Numéricas
  - Short, Byte, Integer, Long, Float e Double
  - Empacotam o tipo primitivo respectivo
  - Provê métodos de conversão e manipulação
  - Exemplos
    - **valueOf (String s)**: cria uma instância da classe wrapper a partir do valor contido na String passada
    - **parseTipo (String s)**: retorna o valor contido na String em uma variável do tipo primitivo correspondente
    - **toString**(tipoPrimitivo i): retorna um objeto String representando o tipo primitivo especificado



# Pacote java.lang

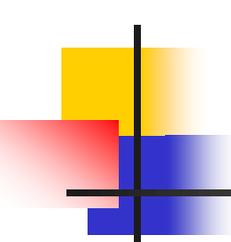
---

- Classes Wrappers Numéricas
  - Exemplos

```
String s1 = "14.5";  
Float f1 = Float.valueOf(s1);
```

```
String s2 = "101";  
int i1 = Integer.parseInt(s2);
```

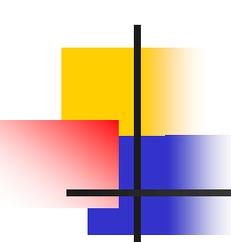
```
Float f2 = new Float(9.5f);  
Int i2 = f2.intValue();
```



# Dever de Sala

---

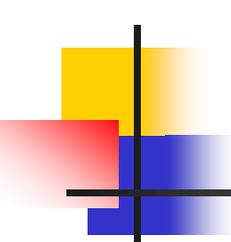
- 1) Escreva um programa em Java com um método que recebe o nome do funcionário e imprime o nome do funcionário em maiúsculo e minúsculo.
  
  - 2) Escreva um programa em Java que leia dois números e em seguida imprime:
    - a) O primeiro número elevado ao segundo.
    - b) Raiz quadrada de cada um dos números
-



# Dever de Sala

---

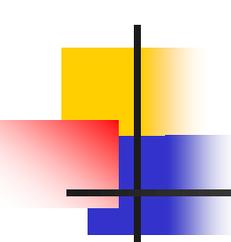
- 3) Escreva um programa em Java que leia o valor do raio, calcule e mostre:
- a) O comprimento do círculo;  $C = 2 * \text{PI} * \text{raio}$
  - b) A área do círculo;  $A = 2 * \text{PI} * R^2$
  - c) O volume da esfera;  $V = \frac{3}{4} * \text{PI} * R^3$
-



# Roteiro

---

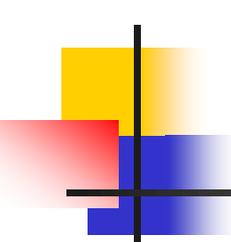
- Java API
- Pacote java.lang
- Pacote java.math
- Pacote java.util



# Pacote java.math

---

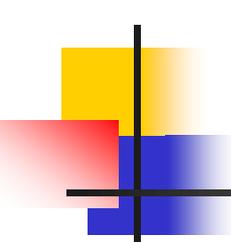
- Pacote que provê classes numéricas com grande capacidade
  - **BigInteger**: Inteiro
  - **BigDecimal**: Decimal
- Seus objetos
  - São imutáveis
  - Têm tamanho indeterminado
- Operações que podem ser realizadas
  - adição, subtração, multiplicação, divisão...
  - Conversão para tipos primitivos



# Roteiro

---

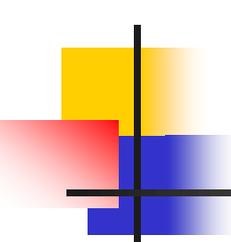
- Java API
- Pacote java.lang
- Pacote java.math
- Pacote java.util



# Pacote java.util

---

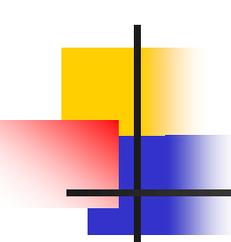
- Provê uma miscelânea de classes utilitárias
  - Tratamento de datas e tempo
  - Impressão e entrada de dados
  - Coleções
  - Internacionalização
  - ...



# Pacote java.util

---

- Tratamento de datas e tempo
  - Classe Date
  - Classe Calendar



# Pacote java.util

---

- Classe Date

- Representa um instante específico no tempo, com precisão de milisegundos

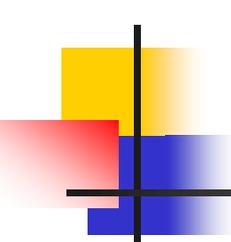
- Construtores

- Date()

- Cria o objeto contendo a data/hora atual

- Date (long l)

- Recebe um número long que representa o número de milisegundos a partir de 1º de Janeiro de 1970, 0h GMT.



# Pacote java.util

---

- Classe Date

- Métodos

- **Comparação com outro Date**

- after, before, compareTo e equals

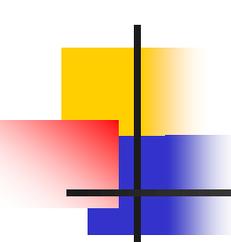
- **getTime:** retorna o número em milisegundos armazenado no objeto

- **setTime:** altera o número em milisegundos armazenado no objeto

- **clone:** clona o objeto

- **toString:** converte a data em String

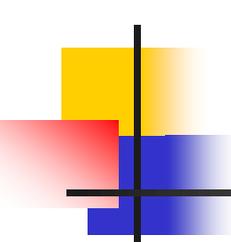
---



# Pacote java.util

---

- Classe Calendar
  - Possibilita o tratamento de diversos tipos de calendários
  - Atributos
    - Identificam componentes de uma data
    - YEAR
    - MONTH
    - DAY\_OF\_MONTH
    - DAY\_OF\_WEEK
    - DAY\_OF\_YEAR
    - HOUR\_OF\_DAY ...



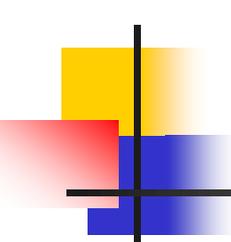
# Pacote java.util

---

- Classe Calendar

- Métodos

- **getInstance:** Retorna o calendário correspondente ao fuso horário e ao local onde está sendo executado
    - **getTime:** Retorna a data armazenada no calendário
    - **getTimeInMillis:** Retorna a data armazenada no calendário em milisegundos
    - **setTime:** Altera a data armazenada no calendário
    - **setTimeInMillis:** Altera a data armazenada no calendário passando um novo valor em milisegundos



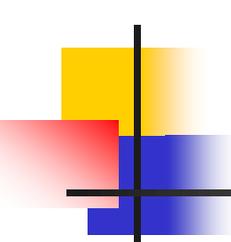
# Pacote java.util

---

- Classe Calendar

- Métodos

- **add:** Adiciona a um campo da data um valor
    - **Comparação:** métodos after, before e equals
    - **clone:** Clona o objeto
    - **get:** Recebe como parâmetro uma das constantes definidas nessa classe que identificam os componentes da data e retorna seu valor
    - **set:** Altera o valor de um ou mais campos
    - **roll:** Rola para cima ou para baixo o valor de um campo



# Referências

---

- Slides “Recursividade e Java.lang” Prof. Marcos Dósea. UFS. 2010.
- Slides “Essenciais” Prof Giovanni . Java.UFS. 2009.
- Slides “Classes Essenciais da API Java”, Prof<sup>a</sup>. Débora. UFS. 2010
- Caelum. Java e Orientação a Objetos
  - <http://www.caelum.com.br/apostilas/>
  - Capítulos 14 e 15