

SERVIÇO NACIONAL DE APRENDIZAGEM COMERCIAL
SENAC - RIO GRANDE DO SUL

LINGUAGEM JAVA



Porto Alegre, 2007

LINGUAGEM JAVA

1ª Edição

Elaboração e Edição

SENAC - Rio Grande do Sul

Documento produzido para uso interno do

SENAC - Rio Grande do Sul

SENAC.RS - Departamento Regional no Rio Grande do Sul

Linguagem Java

1ª edição

Porto Alegre: SENAC – Informática / RS, 2007.

69 pp.

Sumário

Introdução ao Java	7
J2SE	7
J2ME	7
J2EE	7
Principais Características	7
A Plataforma Java	7
Java Runtime Environment (JRE)	7
Java Virtual Machine (JVM)	7
Java Application Program Interface	8
Java Developer Kit (Java SDK)	8
Palavras Reservadas	8
Unicode	8
Estrutura de um programa em Java	9
Convenções	9
Pacotes	10
Tipos Primitivos de Dados	10
Tipos Inteiros	10
byte	11
short	11
int	11
long	11
Tipos Ponto Flutuante (Real)	11
float	11
double	11
char	11
Tipo Lógico	12
boolean	12
Variáveis	12
Operador de Atribuição	12
Operadores Aritméticos	13
Operadores de atribuição compostos	13
Operadores de Incremento e decremento	13
Operadores Relacionais	14
Operadores Lógicos	15
Tabela Verdade do Operador Lógico (&&)	15
Tabela Verdade do Operador Lógico ()	15
Tabela Verdade do Operador Lógico (^)	15
Tabela Verdade do Operador Lógico (!)	16
Estruturas de Controle	16
Estrutura condicional if	16
Estrutura de seleção switch	17
Estruturas de Repetição	18
Estrutura while	18
Estrutura do	18
Estrutura de Repetição for	18
Arrays	19
Programação Orientada a Objeto	20
Classes	20
Métodos Construtores	21
Objetos	21

Modificadores	21
Métodos	21
Herança	22
Classes abstratas	23
Interface	23
Polimorfismo	24
Tratamento de Exceções (Exceptions)	24
Exceções Verificadas	25
Entrada e saída	25
Arquivos Binários	25
InputStream	25
Principais Métodos	26
FileInputStream	26
Principais Construtores	26
Principais Métodos	26
ObjectInputStream	26
Principais Construtores	26
Principais Métodos	27
BufferedInputStream	27
Principais Construtores	27
Principais Métodos	27
DataInputStream	27
Principais Construtores	27
Principais Métodos	27
OutputStream	28
Principais Métodos	28
FileOutputStream	28
Principais Construtores	28
Principais Métodos	28
ObjectOutputStream	29
Principais Construtores	29
Principais Métodos	29
BufferedOutputStream	29
Principais Construtores	29
Principais Métodos	29
DataOutputStream	29
Principais Construtores	29
Principais Métodos	30
Arquivos Texto	30
Reader	30
Principais Métodos	30
BufferedReader	30
Principais Construtores	30
Principais Métodos	31
LineNumberReader	31
Principais Construtores	31
Principais Métodos	31
InputStreamReader	31
Principais Construtores	31
Principais Métodos	32
Principais Construtores	32
Principais Métodos	32
Writer	32
Principais Métodos	32
BufferedWriter	33
Principais Construtores	33

Principais Métodos	33
OutputStreamWriter	33
Principais Construtores	33
Principais Métodos	33
FileWriter	34
Principais Construtores	34
PrintWriter	34
Principais Construtores	34
Principais Métodos	34
Classe de Entrada e Saída	34
RandomAccessFile	34
Principais Construtores	34
Principais Métodos	35
Interface Gráfica	36
Swing	36
Tabela 1.1 Classes AWT e Swing	36
Componentes Visuais	37
JComponent	37
Principais Métodos	37
JFrame	37
Principais Construtores	37
Principais Métodos	37
JPanel	38
Principais Construtores	38
Principais Métodos	38
JLabel	39
Principais Construtores	39
Principais Métodos	39
JButton	40
Principais Construtores	40
Principais Métodos	40
JToggleButton	41
Principais Construtores	41
Principais Métodos	42
JCheckBox	42
Principais Construtores	42
Principais Métodos	43
JRadioButton	43
Principais Construtores	43
Principais Métodos	44
JComboBox	44
Principais assinaturas:	45
Principais Métodos	45
JList	45
Principais Construtores	46
Principais Métodos	46
JTextField	46
Principais Construtores	47
Principais Métodos	47
JOptionPane	47
Principais Construtores	47
JTextArea	49
Principais Construtores	49
Principais Métodos	49
JTabbedPane	50
Principais Construtores	50
Principais Métodos	50

MenuBar, Jmenu e JMenuitem	51
Principais Construtores	51
Principais Métodos	51
JTable	52
Principais Construtores	52
JFileChooser	53
Principais Construtores	53
Principais Métodos	53
JInternalFrame	54
Principais Construtores	54
Principais Métodos	54
Layout	55
Gerenciadores de Layout	55
FlowLayout	55
Principais Construtores	55
GridLayout	56
Principais Construtores	56
BorderLayout	57
Eventos	58
ActionEvent	58
ItemEvent	59
TextEvent	59
FocusEvent	59
KeyEvent	60
MouseEvent	60
WindowEvent	61
Classes Adapter	61
Programação Multitarefa em Java	62
Threads	62
A classe Thread	62
Principais Métodos	62
Criando Threads em Java	62
Criando Theads implementando Runnable	63
Estados de uma Thread	64
Novo	64
Executável	64
Execução	64
Espera/bloqueio/suspensão	64
Inativo	64
Impedindo a execução de uma thread	64
Método sleep()	64
Método yield()	65
Método setPriority()	65
O método join()	65
Sincronização	65
Interação com as Threads	66
Método Wait()	66
Método Notify()	66
Método NotifyAll()	66
Coleções	67
List	67
Principais Métodos	67
Set	68
Map	68

INTRODUÇÃO AO JAVA

Java é uma linguagem poderosa, moderna, utilizada para desenvolvimento de aplicações para computadores desktops, mainframes, dispositivos móveis como celulares ou PDA's, aplicações para a web de pequeno, médio e grande porte, robôs, entre outras.

Ela foi desenvolvida para ser independente de plataforma, ou seja, podemos utilizar o Java com o Windows, Linux, Unix, Solaris, bastando tão somente termos instalado a Máquina Virtual Java. (JVM) para o sistema operacional correspondente.

O Java é dividido em três partes básicas:

J2SE

Java 2 Standard Edition é a parte principal. Nela estão contidas todas as API's necessário para o desenvolvimento de software para computadores Desktop.

J2ME

No Java 2 Micro Edition estão contidas as classes, rotinas necessário para o desenvolvimento de aplicativos móveis, tais como celulares e PDA's.

J2EE

O Java 2 Enterprise Edition é um conjunto de especificações para o desenvolvimento desde pequenas aplicações web até grande sites de acesso concorrido.

PRINCIPAIS CARACTERÍSTICAS

- Orientada a objetos
- Código fonte interpretado e compilado
- Distribuída
- Dinâmica
- Alta portabilidade
- Alta performance
- Robusta
- Multitarefa
- Segura
- Arquitetura neutra
- Portável

A PLATAFORMA JAVA

A plataforma Java é dividida em duas partes: a Java Virtual Machine (JVM), e a Application Program Interface (Java API)

JAVA RUNTIME ENVIRONMENT (JRE)

O JRE é um aplicativo que instala o ambiente básico, ou seja, a API e a JVM no computador que queremos executar aplicativos desenvolvidos em Java.

JAVA VIRTUAL MACHINE (JVM)

A Java Virtual Machine é uma camada de software, intermediária, entre o sistema Operacional e o aplicativo Java. Esta é responsável pelo ambiente de execução dos programas criados em Java.

Seu principal componente é o comando java que nos possibilita a execução de um aplicativo desenvolvido nesta linguagem.

Exemplo:

C:\Pasta> java NomeDaClassePrincipal

JAVA APPLICATION PROGRAM INTERFACE

A API do Java é uma grande coleção de componentes de software reutilizáveis, prontos para uso, organizados em pacotes.

Exemplos:

► javax.swing

Pacote para desenvolver interfaces gráficas de usuário (gui's).

► java.io

Pacote para desenvolver aplicativos com entrada e saída

JAVA DEVELOPER KIT (JAVA SDK)

O Java SDK é um suíte de ferramentas completa para desenvolvimento de softwares em Java. Seu principal comando é o javac, que permite compilarmos o nosso código fonte java.

Exemplo:

C:\Pasta> javac NomeDoArquivo.java

PALAVRAS RESERVADAS

As palavras-chave em Java são as seguintes:

abstract	continue	for	new
assert	default	goto	package
boolean	do	if	private
break	double	implements	protected
byte	else	import	public
case	enum	instanceof	return
catch	extends	int	short
char	final	interface	static
class	finally	long	strictfp
const	float	native	super

- **Observação:** true, false e null, são palavras reservadas. Não podemos nomear componentes de software com palavras reservadas.

UNICODE

Para suportar os diversos idiomas que a plataforma Java pode ser executada, esta utiliza o padrão unicode em vez de ASCII, pois este suporta praticamente todos os idiomas atuais.

ESTRUTURA DE UM PROGRAMA EM JAVA

A linguagem Java trabalha de forma case sensitive, ou seja, diferencia letras maiúsculas de minúsculas.

Blocos de instruções são inseridos entre “{”

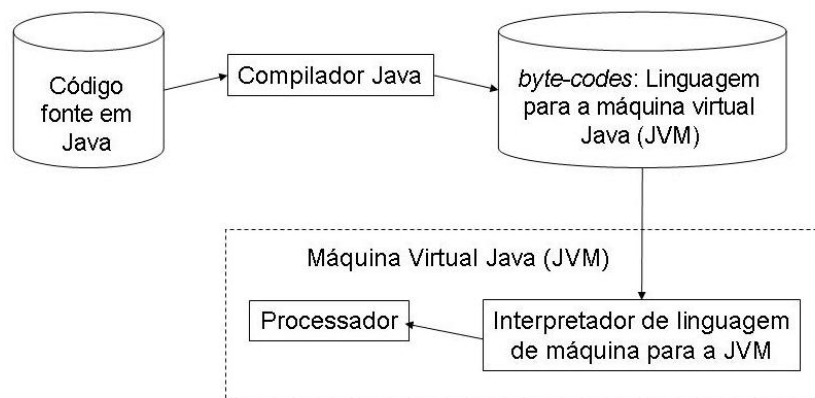
Para finalizar cada instrução utilizamos “;”

Na maioria das vezes criamos uma classe por arquivo;

Um programa funcional em Java deve ter no mínimo uma classe pública, de mesmo nome do arquivo .java, onde nesta conterà um método chamado public static void main(String[] args)

```
// Arquivo PrimeiroPrograma.java
public class PrimeiroPrograma() {
    public static void main(String[] args) {
        // o código fica aqui!
    }
}
```

PLATAFORMA JAVA



CONVENÇÕES

Para melhor legibilidade, apodemos adotar algumas convenções:

Nomes de classes iniciam com letra maiúscula

Exemplos:

- a) **MinhaClasse**
- b) **Pessoa**

Métodos e atributos iniciam com letra minúscula. Se for palavra composta, a primeira palavra será minúscula e as demais iniciam com maiúscula.

Exemplos:

- a) insere;
- b) calcula**P**roduto

Constantes devem ser todas as letras em maiúsculas.

Mais de uma palavra, utilizamos o caractere “_”.

Exemplos:

- a) VALOR
- b) TOTAL_VALOR

Devemos utilizar endentações com 2 ou 4 espaços

Exemplo:

```
public class Pessoa {  
    private int cpf;  
    public int getCpf() {  
        return this.cpf;  
    }  
}
```

PACOTES

Pacote é uma entidade que organiza um conjunto de classes e interfaces relacionadas.

Para evitar confusão com nome de classes idênticas nos programas em java, utilizamos o comando package, que define um pacote. Cada pacote corresponde a uma pasta (Windows), ou diretório (Linux) no sistema de arquivos.

O package deve, obrigatoriamente, ser o primeiro comando do arquivo.

Para que possamos utilizá-lo a partir de outro pacote, utilizamos a instrução import.

A Sun, criadora do Java, recomenda nomes de domínio Internet invertidos para prefixar a definição de pacotes.

```
package meuopacote; // define o pacote meuopacote //  
import java.io.*; // importa todas as classes necessárias do pacote java.io  
import java.lang.StringBuffer; // só importa a classe StringBuffer  
public class PrimeiroPrograma() {  
    public static void main(String[] args) {  
        // o código fica aqui!  
    }  
}
```

TIPOS PRIMITIVOS DE DADOS

Os tipos primitivos de dados em Java são os seguintes

TIPOS INTEIROS

Representam números sem casas decimais.

Os tipos inteiros em Java podem ser:

BYTE

7 7

Inteiro de tamanho 8 bits que varia de - 2 até 2 - 1. (-128 à 127)

SHORT

15 15

Inteiro de tamanho 16 bits que varia de -2 até 2 - 1. (-32.768 à 32.767)

INT

31 31

Inteiro de tamanho 32 bits que varia de -2 até 2 - 1. (-2.147.483.648 à 2.147.483.647)

LONG

63 63

Inteiro de tamanho 64 bits que varia de -2 até 2 - 1. (-9.223.372.036.854.775.808 à 9.223.372.036.854.775.809).

TIPOS PONTO FLUTUANTE (REAL)

Representa os números com casas decimais.

Os tipos Ponto flutuante obedecem ao padrão IEEE 754.

FLOAT

Ponto flutuante simples de tamanho 32 bits

- a) 7.9
- b) 456.87

DOUBLE

Ponto flutuante de precisão dupla de tamanho 64 bits.

Exemplos:

- a) 354335.65678765
- b) 65.9998764367676

CHAR

Representa um único caractere unicode ou um número.

Devemos utilizá-lo entre apóstrofes. Podemos também armazenar valores de 0 à 65535.

Exemplos:

- a) 'a'
- b) 'd'
- c) 654

TIPO LÓGICO

BOOLEAN

Informação que pode assumir somente dois estados: true ou false.

Exemplos:

- a) true
- b) false

Tipo	Tamanho/ Formato	Valores literais	Domínio
<i>(números inteiros)</i>			
byte	8 bits	10, ...	[-128, 127]
short	16 bits	234, ...	[-32768, 32767]
int	32 bits	176, ...	[-2147483648, 2147483647]
long	64 bits	8374L, ...	[-9223372036854775808, ... 07]
<i>(números decimais)</i>			
float	32-bit	3.14f, 200.482F, ...	[+/-1.4E-45, +/-~3.40E38]
double	64-bit	18.0, 1.8e1, 18.0d, ...	[+/-4.9E-324, +/-~1.78E308]
<i>(outros tipos)</i>			
char	16 bits/Unicode	'A', ' ', '€', ...	[... '!', ... 'ÿ', ...] ou [\u0000, \uffff]
boolean	(não definido)	false e true	{false, true}

VARIÁVEIS

Para que possamos armazenar dados na memória do computador utilizamos as variáveis.

As variáveis devem possuir um nome, um tipo de dado e um conteúdo.

Em Java podemos, junto com declaração, inicializarmos as variáveis.

Exemplo de declaração de variáveis em Java

```
{
  int Codigo;
  String nome = "Jorge M.";
  float altura = 1.89;
  char c = 'H';
  boolean v = true;
}
```

OPERADOR DE ATRIBUIÇÃO

Na linguagem Java, utilizamos o operador "=" para atribuição.

Exemplo:

- a) int x; // declara x do tipo inteiro
- b) x = 9; // atribui o valor 9 a variável x

OPERADORES ARITMÉTICOS

Operador	Significado	Exemplo	Resultado
+	Soma dois números	5.8 + 7.3	13.1
-	Subtrai o segundo número do primeiro	9 - 2	7
*	Multiplica dois números	4 * - 8	-32
/	Divide o primeiro número pelo segundo	7 / 2	3,5
%	Obtém o resto da divisão de dois números	13 % 4	1

OPERADORES DE ATRIBUIÇÃO COMPOSTOS

Podemos utilizar estes operadores para realizar tarefas comuns de atribuição

Operador	Significado	Exemplo	Equivalente
+=	Soma e atribui	x += 5	x = x+ 5
-=	Subtrai e atribui	x -= 2	x = x- 2
*=	Multiplica e atribui	x *= 3	x = x* 3
/=	Divide e atribui	x /= 9	x = x/ 9
%=	Resto e atribui	x %= 7	x = x% 7

OPERADORES DE INCREMENTO E DECREMENTO

Operador	Significado	Exemplo	Exemplo
++	Incrementa	x++	++x
--	Decrementa	x--	--x

Os operadores incrementam ou decrementam uma variável numérica.

Exemplos:

a)

```
int x=1;
x++;
println(x); // escreve 2.
```

b)

```
int a=7;
println(a--); // primeiro escreve 7 depois decrementa
println(a); // escreve 6.
```

OPERADORES RELACIONAIS

Operadores Relacionais são utilizados para compararmos dois operandos com o objetivo de saber se uma sentença é verdadeira ou falsa

Operador	Resultado verdadeiro quando
==	Ambos operandos são iguais
>	O primeiro é maior que o segundo
<	O primeiro é menor que o segundo
>=	O primeiro é maior ou igual ao segundo
<=	O primeiro é menor ou igual ao segundo
!=	Os operandos são diferentes

Exemplos:

- a) 5 == 7 // Falso
- b) 4 < 4 // Falso
- c) 15 > 23 // Falso
- d) -2 >= -2 // Verdadeiro
- e) 3 <= 6 // Verdadeiro
- f) 7 != -7 // Verdadeiro

Exemplos:

a)

```
public class OperadoresAritmeticos {  
    public static void main (String args[]) {  
        int a = 5;  
        int b = 2;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("-b = " + (-b));  
        System.out.println("(float) a / b = " + ((float)a / b));  
        System.out.println("a + b = " + (a + b));  
        System.out.println("a - b = " + (a - b));  
        System.out.println("a * b = " + (a * b));  
        System.out.println("a / b = " + (a / b));  
        System.out.println("a % b = " + (a % b));  
        System.out.println("a++ = " + (a++));  
        System.out.println("--b = " + (--b));  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
    }  
}
```

b)

```
public class OperadoresRelacionais {
    public static void main (String args[]) {
        int a = 25;
        int b = 11;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("a < b -> " + (a < b));
        System.out.println("a > b -> " + (a > b));
        System.out.println("a <= b -> " + (a <= b));
        System.out.println("a >= b -> " + (a >= b));
        System.out.println("a == b -> " + (a == b));
        System.out.println("a != b -> " + (a != b));
    }
}
```

OPERADORES LÓGICOS

Operadores Lógicos são utilizados, freqüentemente, com operadores relacionais ou valores lógicos para definirmos o fluxo de um programa.

Operador	Resultado verdadeiro quando	Operandos
&&	Ambos operandos são verdadeiros.	2
	Qualquer um dos operandos é verdadeiro.	2
^	Um, e apenas um, dos operandos é verdadeiro.	2
!	Inverte o valor lógico do operando.	1

TABELA VERDADE DO OPERADOR LÓGICO (&&)

Primeiro Operando	Operador	Segundo Operando	Resultado
Verdadeiro	&&	Verdadeiro	Verdadeiro
Verdadeiro	&&	Falso	Falso
Falso	&&	Falso	Falso
Falso	&&	Verdadeiro	Falso

TABELA VERDADE DO OPERADOR LÓGICO (||)

Primeiro Operando	Operador	Segundo Operando	Resultado
Verdadeiro		Verdadeiro	Verdadeiro
Verdadeiro		Falso	Verdadeiro
Falso		Falso	Falso
Falso		Verdadeiro	Verdadeiro

TABELA VERDADE DO OPERADOR LÓGICO (^)

Primeiro Operando	Operador	Segundo Operando	Resultado
Verdadeiro	^	Verdadeiro	Falso
Verdadeiro	^	Falso	Verdadeiro
Falso	^	Falso	Falso
Falso	^	Verdadeiro	Verdadeiro

TABELA VERDADE DO OPERADOR LÓGICO (!)

Operador	Operando	Resultado
!	Verdadeiro	Falso
!	Falso	Verdadeiro

Exemplos:

- a) Falso && Verdadeiro // Falso
- b) Verdadeiro || Falso // Verdadeiro
- c) Verdadeiro ^ Verdadeiro // Falso
- d) ! Falso // Verdadeiro
- e) $2 < 9 \ \&\& \ 7 < 3$ //Falso
- f) $7 = 3 \ || \ 8 - 9 = -1$ // Verdadeiro
- g) $6 != 7 \wedge 9 \leq 8$ // Falso
- h) $!(5 > 3)$ // Falso

ESTRUTURAS DE CONTROLE

As estruturas de controle são utilizadas para decidir o fluxo do algoritmo.

ESTRUTURA CONDICIONAL IF

Utilizada para decidir se executa um ou mais comandos a partir de uma condição

// Se a condição for verdadeira executa ComandoÚnico

if (condição for verdadeira)

ComandoÚnico;

// Se a condição for verdadeira executa Comando1, depois Comando2, e por último, comando3.

if (condição for verdadeira) {

Comando1;

Comando2;

Comando3;

}

// Se a condição for verdadeira executa Comando1, Senão executará Comando2;

if (condição for verdadeira)

Comando1;

else

Comando2;

// Se a condição for verdadeira executa Comando1, depois Comando2, Senão Executa Comandos 3,4,5.

if (condição for verdadeira) {

Comando1;

Comando2;

}

else

{

Comando3;

Comando4;

Comando5;

}

Exemplos:

```
if (7 > 6) {  
    System.out.println("Escreva "7 é maior do que 6");  
}  
  
if (1 == 8) {  
    System.out.println("Escreva "Isto não será exibido");  
    System.out.println("Escreva "Porque 1 é diferente de 8");  
}  
else  
{  
    System.out.println("Escreva "1 não é igual a 8");  
}
```

ESTRUTURA DE SELEÇÃO SWITCH

A estrutura switch seleciona através de uma variável a opção exata correspondente, como se fosse um menu.

Só podemos usar o switch com variáveis do tipo "short", "int", ou "char".

A instrução "break" é necessária para que saia imediatamente do switch.

// De acordo com "v" seleciona um comando para executar.

```
int v;  
switch(v) {  
    case 1: comando1; break;  
    case 2: comando2; break;  
    case 4: comando3; break;  
    default: comando5;  
}
```

a)

```
public class TestaSwitch {  
    public static void main(String[] args) {  
        int v;  
        v = 2;  
        switch(v) {  
            case 1: System.out.println(" um" ); break;  
            case 2: System.out.println(" dois" ); break;  
            case 3: System.out.println(" três" ); break;  
            default: System.out.println(" nenhum " );  
        }  
    }  
}
```

ESTRUTURAS DE REPETIÇÃO

ESTRUTURA WHILE

Enquanto a condição for verdadeira repete o laço.

O teste é executado no início.

```
while (condição for verdadeira)  
    Comando;
```

```
while (condição for verdadeira) {  
    Comando1;  
    Comando2;  
    Comando3  
}
```

ESTRUTURA DO

Enquanto a condição for verdadeira repete o laço. O teste é executado no final.

```
do {  
    Comando1;  
    Comando2  
    Comando3  
    Comando4;  
while (condição ser verdadeira); // Repete enquanto a condição for verdadeira.
```

ESTRUTURA DE REPETIÇÃO FOR

A estrutura for repete uma quantidade pré-determinada de vezes especificada pela condição:

Exemplos:

a)

```
for (inicialização;condição;incremento)  
    ComandoÚnico;
```

b)

```
for (inicialização;condição;incremento) {  
    Comando1;  
    Comando2;  
    Comando3;  
}
```

c)

```
int x;  
for (x=1;x<=10;x++)  
    System.out.println(x);
```

d)

```
int x;  
for (x=0;x<=60;x++) {  
    System.out.println("Valor de x:");  
    System.out.println(x);  
}
```

ARRAYS

Arrays são objetos que armazenam diversas variáveis de um mesmo tipo. Os arrays são referenciados por um, ou mais, índices.

Estes índices, entre colchetes, indicam a posição de cada elemento.

O primeiro índice de um array sempre começa com zero;

Devemos instanciá-lo com o seu tamanho antes da utilização

Exemplos de utilização de arrays:

a)

```
public class b {  
    public static void main(String[] args) {  
        int x;  
        int v[]; // declara o vetor  
        v = new int[3]; // cria o objeto  
        v[0] = 18; // atribui valores  
        v[1] = 03;  
        v[2] = 80;  
        for (x=0;x<3;x++)  
            System.out.println(v[x]);  
    }  
}
```

b)

```
public class b {  
    public static void main(String[] args) {  
        int x,y;  
        int m[][];  
        m = new int[3][2];  
        m[0][0] = 7;  
        m[0][1] = 8;  
        m[1][0] = 3;  
        m[1][1] = 6;  
        m[2][0] = 10;  
        m[2][1] = 12;  
        for (x=0;x<3;x++) {  
            for (y=0;y<2;y++) {  
                System.out.println(m[x][y]);  
            }  
        }  
    }  
}
```

PROGRAMAÇÃO ORIENTADA A OBJETO

CLASSES

Classes em Java são criadas da seguinte forma:

```
class NomeDaClasse {  
    // aqui ficam os atributos  
    // e os métodos  
}
```

Exemplo:

```
class NomeDaClasse {  
    public class Teste {  
        // atributos  
        private int codigo;  
        private String nome;  
  
        // métodos  
        public void setCodigo(int codigo) {  
            this.codigo = codigo;  
        }  
        public int getCodigo() {  
            return codigo;  
        }  
  
        public void setNome(String nome) {  
            this.nome = nome;  
        }  
        public String getNome() {  
            return this.nome;  
        }  
        public NomeDaClasse() { // método construtor da classe  
            this.codigo = 0;  
            this.nome = "";  
        }  
        public NomeDaClasse(int v, String p) { // método construtor da classe  
            this.codigo = v; // inicializa valores  
            this.nome = p;  
        }  
  
        public static void main(String[] args) {  
            Teste n = new Teste(1,"Jorge");  
            System.out.println(n.codigo);  
            System.out.println(n.nome);je  
            n.setCodigo(12);  
            n.setNome("RF");  
            System.out.println(n.codigo);  
            System.out.println(n.nome);  
        }  
    }  
}
```

MÉTODOS CONSTRUTORES

Os métodos construtores de uma classe servem para que possamos inicializarmos um objeto.

Os métodos construtores de uma classe têm a mesma estrutura de um método comum, mas não possuindo tipo de retorno.

Exemplos:

```
public class Pessoa {  
    Pessoa { // método construtor da classe Pessoa  
        x = 1;           // rotinas de inicialização  
        nome = "teste";  
    }  
    int x;  
    String nome;  
}
```

OBJETOS

Objetos são instâncias de uma classe.

Para que possamos trabalhar com estes, faz-se necessário utilizarmos o comando "new".

Exemplos:

- a) Pessoa p; // declara o objeto p do tipo classe Pessoa
p = new Pessoa();
- b) Pessoa p = new Pessoa(); // Declaração e instanciação na mesma linha.

MODIFICADORES

Na linguagem Java existem diversos modificadores.

Modificadores de acesso:

Modificadores	Restrições
public	sem restrições
private	restringe o uso a somente dentro da classe
protected	dentro da classe e classes filhas
padrão	dentro do pacote

Outros Modificadores:

Modificadores	Significado
final	não poderá ser modificado
static	pertencente a classe
abstract	abstrato, não pode ser instanciado

MÉTODOS

Definem as operações (mensagens) que podem ser executadas em um objeto.

São declarados dentro da classe que os contém.

Deve sempre conter um tipo de retorno.

Para retornar um valor, usamos o comando "return".

Quando não queremos retornar um valor, utilizamos void como tipo de retorno.

Exemplo:

```
class TestaMetodo {
    int calculaDobro(int n) { // Retorna um inteiro contendo o dobro
        return n+n;
    }
    void exibeMensagem(String n) { // Não há nada a retornar, usamos void
        System.out.println("Ola, " + n);
    }
    public static void main(String[] args) {
        TestaMetodo n = new TestaMetodo();
        System.out.println(n.calculaDobro(10));
        n.exibeMensagem("Teste" );
    }
}
```

HERANÇA

Herança é um mecanismo da qual a classe filha herda os estados e comportamentos da classe pai.

Para utilizarmos herança em Java faz-se necessário a utilização da instrução extends seguida do nome da classe ancestral.

Exemplo:

```
public class Inicio {
    public static void main(String args) {
    }
}
class Pessoa {
    int codigo;
    char sexo;
    String nome;
    void cadastrar() {
        // rotina para cadastrar uma pessoa
    }
}
class PessoaFisica extends Pessoa { // herda Atributos e métodos de Pessoa
    String cpf;
    String rg;
    String ufRg;
}
class PessoaJuridica extends Pessoa {
    String cnpj;
    String razaoSocial;
}
```

CLASSES ABSTRATAS

Uma classe abstrata não pode ser instanciada, pois esta contém métodos abstratos, ou seja, sem implementação. Não podemos utilizar classes abstratas diretamente. Devemos utilizar uma classe filha herdando desta e implementando todos os seus métodos abstratos.

Exemplo:

```
abstract class Pessoa {
    String nome;
    abstract void Cadastrar();
    public int excluir(){
        // rotina p/ excluir
    }
}

class PessoaFisica extends Pessoa {
    String cpf;
    void cadastrar() {
        // implementando o método
        // abstrato da classe pai
    }
}
```

INTERFACE

Interface é uma estrutura que representa uma classe totalmente abstrata em Java.

Não têm atributos de dados, somente constantes estáticas.

Todos seus métodos são abstratos.

Para utilizarmos as interfaces devemos usar a cláusula implements, não esquecendo de implementar todos os seus métodos abstratos.

```
interface Veiculo {
    public static final int x = 5;
    public abstract void andar();
    public abstract void parar();
}

public class Carro implements Veiculo {
    String modelo;
    public void andar() {
        // implementando o método andar ()
    }
}

public void parar() {
    // implementando o método parar ()
}
}
```

POLIMORFISMO

No polimorfismo, um objeto pode se comportar de várias formas dependendo de sua instanciação. Podemos reescrever os métodos herdados da classe pai de tal forma que dependendo do objeto criado, o método referente a classe deste objeto será executado.

```
public class Inicio {
    public static void main(String[] args) {
        Pessoa p = new Pessoa();
        p.cadastrar();
        p.cadastrar(6);
        PessoaFisica pf = new PessoaFisica();
        pf.cadastrar();
        pf.cadastrar(8);
    }
}

class Pessoa {
    int codigo;
    String Nome;
    String Endereco;
    void cadastrar() {
        System.out.println("cadastrar() da classe Pessoa! ");
    }
    void cadastrar(int c) {
        System.out.println("Cadastrar() codigo "+ c + " da classe Pessoa!");
    }
}

class PessoaFisica extends Pessoa {
    void cadastrar() {
        System.out.println("Método cadastrar() da classe PessoaFisica!");
    }
}
```

TRATAMENTO DE EXCEÇÕES (EXCEPTIONS)

O tratamento de erros é necessário para podermos garantir um mínimo de segurança na execução de um sistema de software. O Java usa exceções para manipular erros e outros eventos que podem corromper a execução normal do fluxo de um aplicativo.

Exceções podem ocorrer enquanto tentamos abrir um arquivo que já não está mais disponível, uma divisão por zero, ou uma conversão inválida. Para nos auxiliar, o Java conta com as estruturas try-catch e finally.

```
try {
    // código que pode
    // ocorrer um erro
}
catch(ClasseErro obj) {
    // se ocorrer um erro do tipo ClasseErro
    // o código que está aqui deve ser executado
}
catch(ClasseOutroErro obj){
    // Se ocorrer um erro do tipo ClasseOutroErro
    // o código que está aqui deve ser executado
}
finally{
    // Código que sempre será executado.
}
```


Principais exceções	Significado
Trowable	Superclasse de todas as exceções
Exception	Exceções em geral
Error	Erro grave
RuntimeException	Erros em tempo de exceção
NullPointerException	Exceção de ponteiro nulo
ArithmeticException	Exceções aritméticas
IndexOutOfBoundsException	Índice da String ou array inválido
ArrayStoreException	Atribuição incompatível com o array

Exemplo

```
try {  
    x = x / y;  
} catch()
```

EXCEÇÕES VERIFICADAS

Podemos utilizar a cláusula `throws` para indicar que um método pode lançar uma exceção do tipo especificado. Cabe ao programador tratar corretamente a exceção ou propagá-la.

Exemplo:

a)

```
void metodoTeste() throws IOException {  
    // este código pode gerar um exceção do tipo IOException  
}
```

b)

```
void maisTeste() throws IOException {  
  
    if (ErroArquivo()) {  
        throw new IOException(); // Lança a exceção IOException  
    }  
}
```

ENTRADA E SAÍDA

O principal pacote responsável pela manipulação de entrada e saída do Java é o `java.io`. Lá estão contidas diversas classes para que possamos manipular arquivos binários e arquivos de texto (caractere)

As classes que são chamadas de streams operam em nível de byte (binário). E as classes chamadas de Readers/Writers operam em nível de caractere.

Existe a possibilidade de utilizarmos filtros de formatação para facilitar a manipulação ou conversão de dados.

Grande parte dos métodos destas classes geram exceções verificadas do tipo `IOException`, significando que devemos tratá-las ou propagá-las.

ARQUIVOS BINÁRIOS

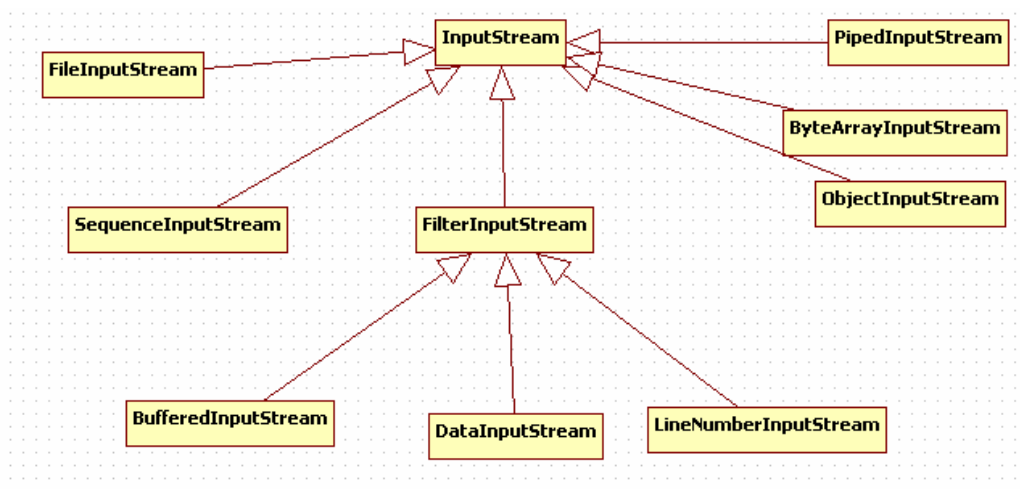
Arquivos binários são manipulados como seqüências de bytes. Para entrada de dados utilizamos a classe `InputStream` e para a saída a `OutputStream` e suas descendentes.

INPUTSTREAM

`InputStream` é a superclasse de todos os streams de entrada

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
close()	Fecha a stream de entrada e libera recursos	void close()
available()	Retorna o número de bytes que pode ser lido	int available()
skip()	Pula e descarta x bytes do stream de entrada	long skip(long x)
read()	Lê o próximo byte da stream de entrada.	int read()
read()	Lê alguns bytes e armazena em um buffer definido por b	int read(byte[] b)
read()	Lê até n bytes e armazena em um array de bytes	int read(byte[] b, int indice, int tam)



As principais subclasses de InputStream:

FILEINPUTSTREAM

Classes para leitura de arquivos em bytes.

PRINCIPAIS CONSTRUTORES

- ▶ `FileInputStream(File arq)` throws `FileNotFoundException`
- ▶ `FileInputStream(String arq)` throws `FileNotFoundException`

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
skip()	Pula e descarta x bytes do stream de entrada	long skip(int x)
read()	Lê o próximo byte da stream de entrada.	int read()
read()	Lê até n bytes e e armazena em um array de bytes	int read(byte[] b, int inicio, int tam)

OBJECTINPUTSTREAM

Classe para leitura de objetos serializados.

PRINCIPAIS CONSTRUTORES

- ▶ `ObjectInputStream(InputStream entrada)` throws `IOException`, `StreamCorruptedException`

Cria um `ObjectInputStream`

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
skipBytes()	Pula e descarta x bytes do stream de entrada	long skip(int x)
read()	Lê o próximo byte da stream de entrada.	int read()
read()	Lê até n bytes e e armazena em um array de bytes	int read(byte[] b, int inicio, int tam)
readUTF()	Lê uma String em UTF.	String readUTF()

BUFFEREDINPUTSTREAM

Classe que implementa uma InputStream bufferizada

PRINCIPAIS CONSTRUTORES

► BufferedInputStream(InputStream imp)

Cria um BufferedInputStream

► BufferedInputStream(inputStream imp, int tam)

Cria um BufferedInputStream especificando o tamanho na variável int “tam”.

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
skip()	Pula e descarta x bytes do stream de entrada	long skip(long x)
read()	Lê o próximo byte da stream de entrada.	int read()
read()	Lê até n bytes e armazena em um array de bytes	int read(byte[] b, int inicio, int tam)

DATAINPUTSTREAM

Implementa a leitura de todos os tipos primitivos do Java.

PRINCIPAIS CONSTRUTORES

► DataInputStream(InputStream imp)

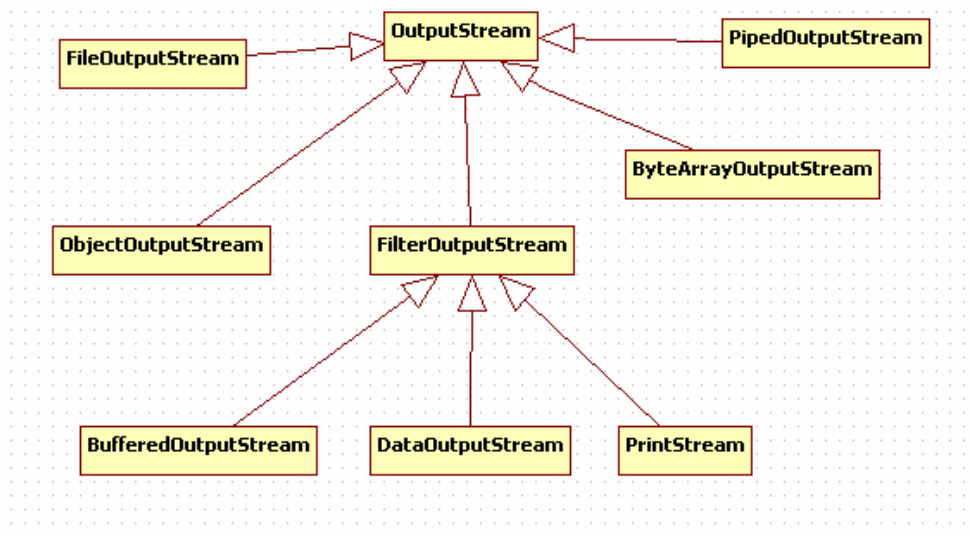
Cria um objeto DataInputStream

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
skipBytes()	Pula e descarta x bytes do stream de entrada	int skip(int x)
read()	Lê alguns bytes da stream de entrada e armazena em um array x	int read(byte[] x)
readShort()	Lê e retorna um short do InputStream	short readShort()
readInt()	Lê e retorna um int do InputStream	int readInt()
readBoolean()	Lê e retorna um boolean do InputStream	boolean readBoolean()
readChar()	Lê e retorna um char do InputStream	char readChar()
readDouble()	Lê e retorna um double do InputStream	double readDouble()
readByte()	Lê e retorna um byte do InputStream	byte readByte()
readLong()	Lê e retorna um long do InputStream	long readLong()

OUTPUTSTREAM

OutputStream é a superclasse de todos os streams de saída



PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
close()	Fecha a stream de saída e libera recursos	void close()
flush()	Descarrega a saída esvaziando o buffer	void flush()
write()	Grava um byte na stream de saída.	void write(byte[] b)
write()	Grava n bytes na stream de saída.	int write(byte[] b, int inicio, int quant)

FILEOUTPUTSTREAM

Classes para gravação de arquivos em bytes.

PRINCIPAIS CONSTRUTORES

► `FileOutputStream(File arq)` throws `IOException`

Cria um `FileOutputStream` a partir de uma variável `File`

► `FileOutputStream(String arq)` throws `IOException`

Cria um `FileOutputStream` a partir de uma variável `String`

► `FileOutputStream(String arq, s boolean adiciona)` throws `IOException`

Cria um `FileOutputStream` a partir de uma variável `String` com a opção de adicionar ao final de uma arquivo existente.

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
flush()	Descarrega a saída esvaziando o buffer	void flush()
write()	Grava um int na stream de saída.	void write(int b)
write()	Grava um byte na stream de saída.	void write(byte[] b)
write()	Grava n bytes na stream de saída.	int write(byte[] b, int inicio, int tam)

OBJECTOUTPUTSTREAM

Classe para gravação objetos serializados.

PRINCIPAIS CONSTRUTORES

► `ObjectOutputStream(OutputStream saída)` throws `IOException`, `StreamCorruptedException`

Cria `ObjectOutputStream` a partir de um `OutputStream`

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>write()</code>	Grava um int na stream de saída.	<code>void write(int b)</code>
<code>write()</code>	Grava um byte na stream de saída.	<code>void write(byte[] b)</code>
<code>write()</code>	Grava n bytes na stream de saída.	<code>int write(byte[] b, int inicio, int tam)</code>

BUFFEREDOUTPUTSTREAM

Classe que implementa uma `OutputStream` bufferizada

PRINCIPAIS CONSTRUTORES

► `BufferedOutputStream(OutputStream saída)`

Cria um `BufferedOutputStream` a partir de um `OutputStream`

► `BufferedOutputStream(OutputStream saída, int Tamanho)`

Cria um `BufferedOutputStream` a partir de um `OutputStream`, definindo o tamanho do buffer

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>write()</code>	Grava um int na stream de saída.	<code>void write(int b)</code>
<code>write()</code>	Grava n bytes na stream de saída.	<code>void write(byte[] b, int inicio, int tam)</code>

DATAOUTPUTSTREAM

Implementa a gravação de todos os tipos primitivos do Java.

PRINCIPAIS CONSTRUTORES

► `DataOutputStream(OutputStream saída)`

Cria um `DataOutputStream` a partir de um `OutputStream`

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
writeShort()	Grava um short no OutputStream	void writeShort(short x)
writeInt()	Grava um int no OutputStream	void writeInt(int x)
writeBoolean()	Grava um boolean no OutputStream	void writeBoolean(Boolean x)
writeChar()	Grava um char no OutputStream	void writeChar(int x)
writeDouble()	Grava um doublé no OutputStream	void writeDouble(double x)
writeByte()	Grava um byte no OutputStream	void writeByte(byte x)
writeLong()	Grava um long no OutputStream	void writeLong(long x)

ARQUIVOS TEXTO

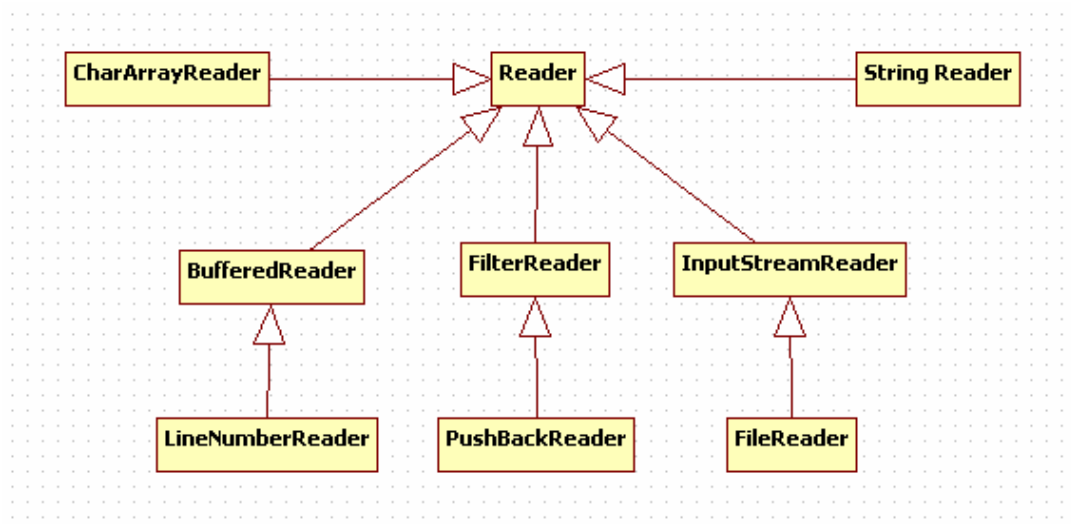
Arquivos texto são manipulados como seqüência de caracteres. Para a entrada utilizamos a classe Reader e suas descendentes. Para a saída usamos a Writer.

READER

Subclasse de todas as classes de entrada de caracteres (readers).

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
read()	Lê um caractere de entrada	Int read()
read()	Lê uma seqüência de caracteres colocando no array a partir de ind	Int read(char[] c, int ind, int quant)
read()	Lê um array de caracteres de entrada	Int read(char[] c)
close()	Fecha o reader	void close()



BUFFEREDREADER

Classe Reader com buffer, otimizando o acesso aos dados.

PRINCIPAIS CONSTRUTORES

► `BufferedReader (Reader r)`

Cria um `BufferedReader` a partir de um `Reader`

► **BufferedReader (Reader r, int tamanho)**

Cria um **BufferedReader** a partir de um **Reader** especificando o tamanho

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
read()	Lê um caractere de entrada	Int read()
read()	Lê uma seqüência de caracteres colocando no array a partir de ind	Int read(char[] c, int ind, int tam)
readLine()	Lê uma linha de caracteres	String readLine()
close()	Fecha o reader	void close()

LINENUMBERREADER

Classe **Reader** para controle de linhas

PRINCIPAIS CONSTRUTORES

► **LineNumberReader (Reader r)**

Cria um **LineNumberReader** a partir de um **Reader**

► **LineNumberReader (Reader r, int tamanho)**

Cria um **LineNumberReader** a partir de um **Reader**, especificando o tamanho

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
read()	Lê um caractere de entrada	Int read()
read()	Lê uma seqüência de caracteres colocando no array a partir de ind	Int read(char[] c, int ind, int tam)
readLine()	Lê uma linha de caracteres	String readLine()
setLineNumber()	Posiciona a leitura na linha x	void setLineNumber(int x)
getLineNumber()	Retorna a linha atual	Int getLineNumber()
close()	Fecha o reader	void close()
Método	Descrição	Assinatura
read()	Lê um caractere de entrada	Int read()
read()	Lê uma array de caracteres	Int read(char[] c)
read()	Lê uma seqüência de caracteres colocando no array a partir de ind	Int read(char[] c, int ind, int quant)
close()	Fecha o reader	void close()

INPUTSTREAMREADER

Classe **Reader** que age como conversora entre stream e caracteres.

PRINCIPAIS CONSTRUTORES

► **InputStreamReader (InputStream imp)**

Cria um **InputStreamReader** a partir de um **InputStream**

► **InputStreamReader (InputStream imp, int tamanho)**

Cria um `InputStreamReader` a partir de um `InputStream` especificando o tamanho do buffer

PRINCIPAIS MÉTODOS

► `FileReader`

Classe `Reader` para leitura de arquivos de caracteres.

PRINCIPAIS CONSTRUTORES

► `FileReader (File arq)`

Cria um `FileReader` a partir de um objeto `File`

► `FileReader (String arq)`

Cria um `FileReader` a partir de uma `String`

PRINCIPAIS MÉTODOS

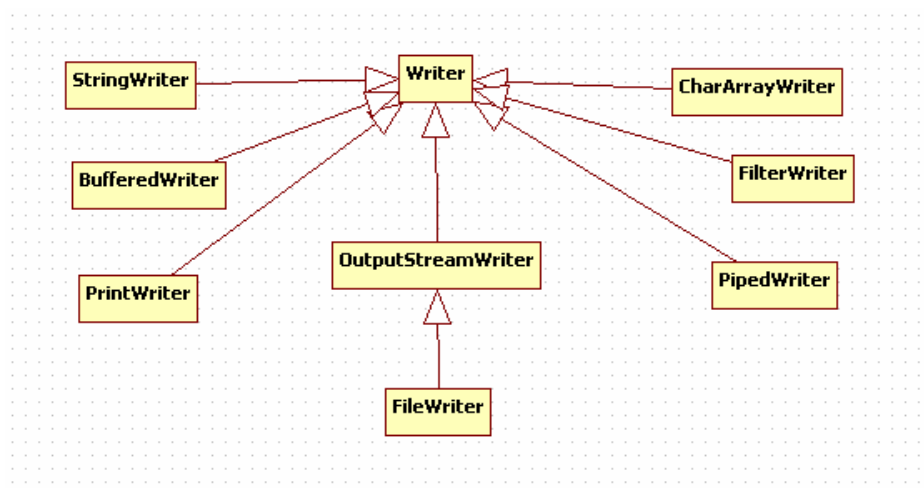
Método	Descrição	Assinatura
<code>read()</code>	Lê um caractere de entrada	<code>int read()</code>
<code>read()</code>	Lê uma array de caracteres	<code>int read(char[] c)</code>
<code>read()</code>	Lê uma sequência de caracteres colocando no array a partir de <code>ind</code>	<code>int read(char[] c, int ind, int tam)</code>
<code>close()</code>	Fecha o reader	<code>void close()</code>

WRITER

Subclasse de todas as classes de saída de caracteres (writers).

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>
<code>write()</code>	Grava uma sequência de caracteres no array a partir de <code>ind</code>	<code>void write(char[] c, int ind, int tam)</code>
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>close()</code>	Fecha o writer	<code>void close()</code>



BUFFEREDWRITER

Classe Reader com buffer, otimizando o acesso aos dados.

PRINCIPAIS CONSTRUTORES

► `BufferedWriter(Writer w)`

Cria um objeto `BufferedWriter` a partir de um `Writer`

► `BufferedWriter(Writer w, int tam)`

Cria um objeto `BufferedWriter` a partir de um `Writer`, especificando o tamanho do buffer

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>
<code>write()</code>	Grava uma sequência de caracteres no array a partir de ind	<code>void write(char[] c, int ind, int tam)</code>
<code>newLine()</code>	Adiciona uma nova linha	<code>void newLine()</code>
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>close()</code>	Fecha o writer	<code>void close()</code>

Método	Descrição	Assinatura
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>
<code>write()</code>	Grava uma String no array a partir de ind	<code>void write(String c, int ind, int tam)</code>
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>close()</code>	Fecha o writer	<code>void close()</code>

OUTPUTSTREAMWRITER

Classe `Writer` que age como conversora entre stream e caracteres.

PRINCIPAIS CONSTRUTORES

► `OutputStreamWriter (OutputStream saída)`

Cria um `OutputStreamWriter` a partir de um `OutputStream`

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>
<code>write()</code>	Grava uma String no array a partir de ind	<code>void write(String c, int ind, int tam)</code>
<code>flush()</code>	Descarrega a saída esvaziando o buffer	<code>void flush()</code>
<code>close()</code>	Fecha o writer	<code>void close()</code>

FILEWRITER

Classe específica para a manipulação de arquivos de caracteres

PRINCIPAIS CONSTRUTORES

► `FileWriter(File arq)`

Cria um `FileWriter` a partir de um objeto `File`

► `FileWriter(String arq)`

Cria um `FileWriter` a partir de uma `String`

PRINTWRITER

Classe `Writer` específica para gravar caracteres formatados de acordo com a plataforma em execução.

PRINCIPAIS CONSTRUTORES

► `PrintWriter(OutputStream saída, boolean af)`

Cria um `PrintWriter` a partir de um `OutputStream`, especificando se deseja o `autoflush`.

► `PrintWriter(Writer saída, boolean af)`

Cria um `PrintWriter` a partir de um `Writer`, especificando se deseja o `autoflush`.

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>print()</code>	Grava um boolean	<code>void println(boolean s)</code>
<code>print()</code>	Grava um int	<code>void println(int s)</code>
<code>println()</code>	Grava um double	<code>void println(double s)</code>
<code>println()</code>	Grava uma <code>String</code> e um final de linha	<code>void println(String s)</code>
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>
<code>write()</code>	Grava uma <code>String</code> no array a partir de <code>ind</code>	<code>void write(String c, int ind, int tam)</code>
<code>write()</code>	Grava um caractere na saída	<code>void write(int c)</code>
<code>write()</code>	Grava um array de caracteres na saída	<code>void write(char[] c)</code>

CLASSE DE ENTRADA E SAÍDA

RANDOMACCESSFILE

PRINCIPAIS CONSTRUTORES

► `RandomAccessFile(String arq, String modo)` throws `IOException`

Cria um objeto do tipo `RandomAccessFile` associando-o a `arq`, sendo o modo definido como `"r"` somente leitura e `"rw"` leitura e gravação.

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
read()	Lê um byte do arquivo	int read()
read()	Lê um array de bytes de um arquivo	int read(byte[] x)
read()	Lê x bytes colocando em um array a partir de ind	int read(byte[] x, int ind, int quant)
readLine()	Lê uma linha de caracteres do arquivo	String readLine()
skipBytes()	Descarta x bytes da entrada	void skipBytes(long x)
seek()	Posiciona o cursor no arquivo na posição x	seek(long x)
length()	Retorna o tamanho do arquivo	long length()
write()	Escreve um array de bytes no arquivo	void write(byte[] dados)
close()	Fecha o arquivo	void close
readBoolean()	Lê um boolean do arquivo	boolean readBoolean()
readInt()	Lê um int do arquivo	int readInt()
ReadLong()	Lê um long do arquivo	long readLong()
readDouble()	Lê um double do arquivo	double readDouble()
ReadChar()	Lê um char do arquivo	char readChar()
readByte()	Lê um byte do arquivo	byte readByte()
ReadFloat()	Lê um float do arquivo	float readFloat()
WriteBoolean()	Grava um boolean no arquivo	void writeBoolean(boolean x)
writeInt()	Grava um int no arquivo	void writeInt(int x)
WriteLong()	Grava um long no arquivo	void writeLong(long x)
writeDouble()	Grava um double no arquivo	void writeDouble(double x)
WriteChar()	Grava um char no arquivo	void writeChar(char x)
writeByte()	Grava um byte no arquivo	void writeByte(byte x)
WriteFloat()	Grava um float no arquivo	void writeFloat(byte x)

Exemplo 1 – Utilizando FileInputStream e FileOutputStream para copiar arquivos.

```

import java.io.*;

public class TesteStream {
    public static void main(String[] args) throws IOException {
        FileInputStream entrada = null;
        FileOutputStream saida = null;
        try {
            entrada = new FileInputStream("dados.txt");
            saida = new FileOutputStream("resultado.txt");
            int c;
            while ((c = entrada.read()) != -1) {

                saida.write(c);
            }
        } finally {
            if (entrada != null) {
                entrada.close();
            }
            if (saida != null) {
                saida.close();
            }
        }
    }
}

```

Exemplo 2 – Utilizando FileReader e FileWriter para copiar arquivos

```
import java.io.*;
public class TesteReaderWriter {
    public static void main(String[] args) throws IOException {
        FileReader entrada = null;
        FileWriter saida = null;
        try {
            entrada = new FileReader("Dados.txt");
            saida = new FileWriter("Resultado.txt");
            int c;
            while ((c = entrada.read()) != -1) {
                saida.write(c);
            }
        } finally {
            if (entrada != null) {
                entrada.close();
            }
            if (saida != null) {
                saida.close();
            }
        }
    }
}
```

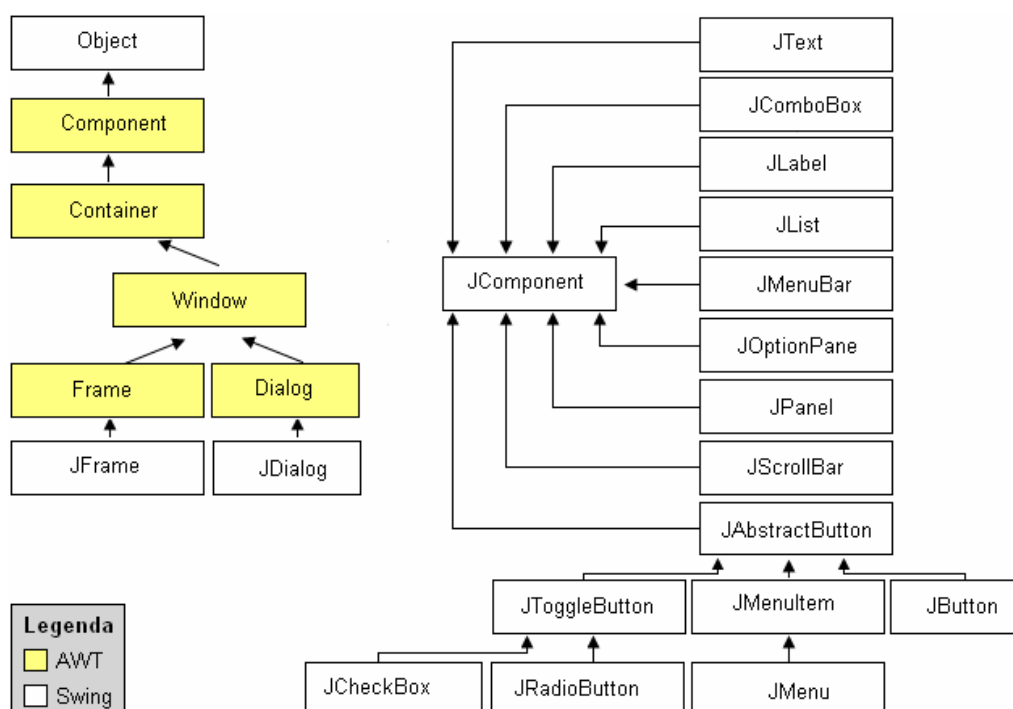
INTERFACE GRÁFICA

SWING

O JFC/Swing é o pacote padrão de construção de interfaces gráfica em Java. Ele é a extensão do antigo pacote awt, utilizado nas versões anteriores.

Para que possamos utilizar o swing é necessário importarmos as classes dos pacotes javax.swing.

TABELA 1.1 CLASSES AWT E SWING



COMPONENTES VISUAIS

Veremos agora os principais componentes Swing:

JComponent

JComponent é a superclasse de todos os componentes visuais.

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
setBackground()	Muda a cor de fundo do componente	void setBackground(Color cor)
setEnabled()	habilita ou desabilita um componente	void setEnabled(boolean habilitado)
setFont()	Muda a fonte do componente	void setFont(Font fonte)
setVisible()	Deixa visível ou invisível um componente	void setVisible(boolean visível)
setToolTipText()	Adiciona dica a um componente	void setToolTipText(String dica)

JFrame

JFrame é o componente que implementa uma janela. Ele é composto por uma barra de menus (opcional), e um contentPane (área útil da janela)



PRINCIPAIS CONSTRUTORES

- ▶ JFrame();
- ▶ JFrame(String Título);

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
setDefaultCloseOperation()	Configura a opção de fechar	setDefaultCloseOperation(int modo)
getContentPane()	Retorna o painel de conteúdo	ContentPane getContentPane()
setLayout()	Muda o Layout do componente	void Layout(LayoutManager lm)
setVisible()	Configura a visibilidade do componente	void setVisible(v boolean)
setSize()	Altera o tamanho do componente	void setSize(int x, int y)
setTitle()	Muda o título do componente	void setTitle(String Título)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJFrame {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame();
        tela.setTitle("Janela");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.getContentPane().setBackground(Color.RED);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```

JPANEL

A classe JPanel implementa um container, ou seja, componente gráfico que fornece o espaço para colocarmos outros componentes em seu interior. Uma vez criado ele deve ser adicionado a um objeto JFrame ou similar.



PRINCIPAIS CONSTRUTORES

- ▶ JPanel()
- ▶ JPanel(LayoutManager lm)

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
setVisible()	Configura a visibilidade do componente	void setVisible(v boolean)
setSize()	Altera o tamanho do componente	void setSize(int x, int y)
setBounds()	Altera posição e a altura e largura do componente	Void setBounds(int x, int y, int a int l)
setLocation()	Altera a localização do componente	void setLocation(int x, int y)
setBorder()	Configura a borda do componente	

Exemplo de Código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJPanel {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JPanel");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JPanel painel = new JPanel();
        painel.setBorder(BorderFactory.createEtchedBorder());
        tela.getContentPane().add(painel);
        painel.setBounds(250,160,240,240);
    }
}
```

JLABEL

É o componente Swing que serve para exibirmos um texto, uma imagem ou ambos.

O texto pode conter códigos HTML e a imagem suporta os formatos jpeg, gif ou png.

PRINCIPAIS CONSTRUTORES

- ▶ JLabel()
- ▶ JLabel(String Texto)
- ▶ JLabel(Icon Imagem)

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
getText()	Retorna o texto do rótulo	String getText()
setText()	Modifica o texto do rótulo	void setText(String Texto)
setIcon()	Modifica o ícone do rótulo	void setIcon(icon icone)
setDisabledIcon()	Modifica o ícone do rótulo quando este estiver desabilitado	void setDisabledIcon(icon icone)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJLabel {
    public static void main(String[] args) {
        JFrame tela = new JFrame("Teste JLabel");
        JLabel rotulo = new JLabel();
        JLabel rotuloAux = new JLabel("SENAC");
        tela.setLayout(null);
        tela.getContentPane().setBackground(Color.WHITE);
        tela.add(rotulo);
        tela.add(rotuloAux);
        tela.setSize(800,600);
        tela.setVisible(true);
    }
}
```

```

tela.add(rotulo);
tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
rotulo.setSize(200,150);
rotulo.setBackground(Color.RED);
rotulo.setForeground(Color.WHITE);
rotuloAux.setSize(150,50);
Icon icone = new ImageIcon("l.jpg");
rotulo.setIcon(icone);
rotuloAux.setText("Teste");
rotulo.setLocation(250,180);
rotuloAux.setLocation(10,100);
}
}

```

JBUTTON

O JButton assim como o JLabel comporta a exibição de texto e imagem. O diferencial é que este é utilizado quando devemos tomar alguma ação através de um comando.



PRINCIPAIS CONSTRUTORES

- ▶ JButton()
- ▶ JButton(String Texto)
- ▶ JButton(Icon Imagem)

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
getText	Retorna o texto do botão	String getText()
setText	Modifica o texto do botão	Void setText(String Texto)
setIcon	Modifica o ícone do botão	void setIcon(icon icone)
setDisabledIcon	Modifica o ícone do botão quando este estiver desabilitado	void setDisabledIcon(icon icone)

Exemplo de código

```
import javax.swing.*;
import java.awt.*;

public class TesteJButton {
    public static void main(String[] args) {
        JFrame tela;
        JButton botao;
        JButton botaoAux;
        tela = new JFrame();
        botao = new JButton();
        botaoAux = new JButton();
        tela.setLayout(null);
        tela.setTitle("Este \u00E9 o T\u00EDtulo da Janela");
        tela.getContentPane().setBackground(Color.GREEN);
        tela.add(botao);
        tela.add(botaoAux);
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.getContentPane().add(botao);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        botao.setSize(200,150);
        botao.setBackground(Color.RED);
        botao.setForeground(Color.WHITE);
        botaoAux.setSize(150,50);
        botaoAux.setToolTipText("Este \u00E9 um bot\u00E3o criado com o
Java!!");
        botao.setToolTipText("Este \u00E9 um bot\u00E3o colorado!!");
        botao.setText("Bot\u00e3o Vermelho");
        Icon icone = new ImageIcon("1.png");
        botao.setIcon(icone);
        botaoAux.setText("Bot\u00e3o de Teste");
        botao.setLocation(250,180);
        botaoAux.setLocation(10,100);
    }
}
```

JTOGGLEBUTTON

Classe que representa um botão de dois estados: pressionado ou não pressionado.



PRINCIPAIS CONSTRUTORES

- ▶ JToggleButton()
- ▶ JToggleButton(String Texto)
- ▶ JToggleButton(Icon Imagem)

► JToggleButton(String Texto, Icon Imagem)

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
getText()	Retorna o texto do botão	String getText()
setText()	Modifica o texto do botão	void setText(String Texto)
setIcon()	Modifica o ícone do botão	void setIcon(icon icone)
isSelected()	Retorna se o botão está pressionado	void isSelected()

Exemplo de código:

```
import javax.swing.*.*;
import java.awt.*.*;
public class TesteJToggleButton {
    public static void main(String[] args) {
        JFrame tela = new JFrame("Teste JToggleButton");
        JToggleButton botao = new JToggleButton();
        tela.setLayout(null);
        tela.getContentPane().setBackground(Color.WHITE);
        tela.add(botao);
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.add(botao);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        botao.setSize(200,150);
        botao.setBackground(Color.RED);
        botao.setForeground(Color.WHITE);
        botao.setLocation(250,180);
    }
}
```

JCheckBox

É o componente que exibe uma caixa e um rótulo que utilizamos para selecionar uma opção entre dois estados possíveis.



PRINCIPAIS CONSTRUTORES

- JCheckBox()
- JCheckBox (String Texto)
- JCheckBox (Icon Imagem)
- JCheckBox (Icon Imagem, boolean isSelected)

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
setSelected()	Configura o botão como selecionado ou não selecionado	Boolean setSelected()
setText()	Modifica o texto do botão	void setText(String Texto)
setIcon()	Modifica o ícone do botão	void setIcon(icon icone)
isSelected()	Retorna se o botão está pressionado	void isSelected()

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJCheckBox {
    public static void main(String[] args) {
        JFrame tela = new JFrame("Teste JCheckBox");
        JCheckBox botao = new JCheckBox("Teclado");
        JCheckBox botao2 = new JCheckBox("Mouse");
        tela.setLayout(null);
        tela.getContentPane().setBackground(Color.WHITE);
        tela.getContentPane().add(botao);
        tela.getContentPane().add(botao2);
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        botao.setBounds(20,40,100,30);
        botao2.setBounds(20,140,100,30);
        botao2.setSelected(true);
        botao.setBackground(Color.RED);
        botao.setForeground(Color.WHITE);
        botao2.setBackground(Color.BLUE);
        botao2.setForeground(Color.YELLOW);
    }
}
```

JRADIOBUTTON

Esta é a subclasse de JToggleButton que também implementa um botão de dois estados.

Este componente deve ser associado a um componente ButtonGroup.

Componente que associa um conjunto de JRadioButtons para somente um botão ser selecionado por vez.

PRINCIPAIS CONSTRUTORES

- ▶ JRadioButton()
- ▶ JRadioButton (String Texto)



PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
setSelected()	Configura o botão como selecionado ou não selecionado	boolean setSelected()
setText()	Modifica o texto do botão	void setText(String Texto)
setSelectedIcon()	Modifica o ícone do botão selecionado	void setSelectedIcon(icon icone)
isSelected()	Retorna se o botão está pressionado	void isSelected()

Exemplo de Código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJRadioButton {
    public static void main(String[] args) {
        JFrame tela = new JFrame("Teste JRadioButton");
        JRadioButton botao = new JRadioButton("Teclado");
        JRadioButton botao2 = new JRadioButton("Mouse");
        ButtonGroup grupo = new ButtonGroup();
        JPanel p = new JPanel();
        tela.setLayout(null);
        tela.getContentPane().setBackground(Color.WHITE);
        grupo.add(botao);
        grupo.add(botao2);
        p.setBorder(BorderFactory.createEtchedBorder());
        p.setBounds(250, 160, 240, 240);
        p.add(botao);
        p.add(botao2);
        tela.getContentPane().add(p);
        tela.setSize(800, 600);
        tela.setVisible(true);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        botao.setBackground(Color.RED);
        botao.setForeground(Color.GREEN);
        botao2.setBackground(Color.BLUE);
        botao2.setForeground(Color.YELLOW);
    }
}
```

JComboBox



Componente editável ou não que permite selecionar itens em uma lista suspensa

PRINCIPAIS ASSINATURAS:

- ▶ JComboBox()
- ▶ JComboBox(Object[] Itens)

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
clearSelection	Limpa a seleção da lista	void clearSelection()
getSelectedIndex	Retorna o índice do item selecionado	int getSelectedIndex()
Add	Adiciona um texto à lista	void add(String Texto)
getSelectedItem	Retorna o texto selecionado da lista	String getItem()
getItem	Retorna o texto do índice especificado	String getItem(int indice)
getSelectedIndexes	Retorna os índices dos itens selecionados	int getSelectedIndexes()
getSelectedItems	Retorna os textos selecionados da lista	String getSelectedItems()

Exemplo prático:

```
import javax.swing.*;
import java.awt.*;

public class TesteJComboBox {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JComboBox");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JComboBox cb = new JComboBox();
        tela.getContentPane().add(cb);
        cb.setBounds(10,10,70,20);
        cb.addItem("RS");
        cb.addItem("SC");
        cb.addItem("PR");
        cb.setEditable(true);
    }
}
```

JLIST

Componente que permite selecionarmos um item ou mais de uma lista



PRINCIPAIS CONSTRUTORES

- ▶ JList()
- ▶ JList(Object[] itens)

PRINCIPAIS MÉTODOS

Comando	Descrição	Assinatura
clearSelection()	Limpa a seleção da lista	void clearSelection()
getSelectedIndex()	Retorna o índice do item selecionado	int getSelectedIndex()
setSelectionMode()	Define se o modo de seleção da lista é simples ou múltipla	void setSelectionMode(int modo)
getSelectedItem()	Retorna o texto selecionado da lista	String getItem()
getItem()	Retorna o texto do índice especificado	String getItem(int indice)
getSelectedIndexes()	Retorna os índices dos itens selecionados	int getSelectedIndexes()
getSelectedItems()	Retorna os textos selecionados da lista	String getSelectedItems()

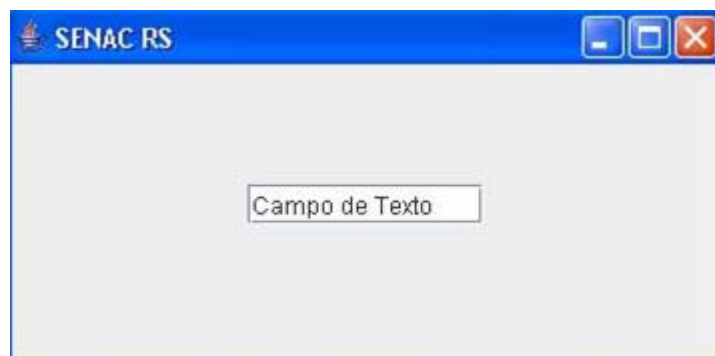
Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJList {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JList");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        String[] lista = {"RS", "SC", "PR", "SP", "RJ"};
        JList ls = new JList(lista);
        tela.getContentPane().add(ls);
        ls.setBounds(10,10,30,70);
        ls.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
    }
}
```

JTEXTFIELD

CampoTexto de somente uma linha



PRINCIPAIS CONSTRUTORES

- ▶ `TextField()`
- ▶ `TextField(String Texto)`

PRINCIPAIS MÉTODOS

Métodos	Descrição	Assinatura
<code>GetText</code>	Retorna o texto do campo	<code>String getText()</code>
<code>SetText</code>	Modifica o texto do campo	<code>void setText(String Texto)</code>

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJTextField {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JTextField");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JTextField tf = new JTextField("Isto é um JTextField");
        tela.getContentPane().add(tf);
        tf.setBounds(10,10,150,20);
    }
}
```

JOPTIONPANE







Esta classe nos permite interagir com o usuário através de mensagens. Estas mensagens pode ser do tipo Confirm, "Input" ou "Message"

PRINCIPAIS CONSTRUTORES

- ▶ `JOptionPane(Object Texto)`
- ▶ `JOptionPane(Object Texto, int TipoDeMensagem)`
- ▶ `JOptionPane(Object Texto, int TipoDeMensagem, int TipoDeOpcao)`
- ▶ `JOptionPane(Object Texto, int TipoDeMensagem, int TipoDeOpcao, Icon icone)`

Métodos	Descrição	Assinatura
showConfirmDialog()	Exibe uma mensagem de questionamento	static int showConfirmDialog(Component pai, Object msg, String titulo, int TipodeOpcao)
showInputDialog()	Pergunta por uma entrada do usuário	static int showInputDialog(Object msg)
showMessageDialog()	Exibe uma mensagem de informação ao usuário	static int showConfirmDialog(Component pai, Object msg, String titulo, int TipodeMsg)
ShowOptionDialog()	Mensagem unificando as opções acima	static int showConfirmDialog(Component pai, Object msg, String titulo, int TipodeOpcao, int TipodeMsg)

Tipos de Mensagens	Descrição	Símbolo
ERROR_MESSAGE	Mensagens de Erro	
INFORMATION_MESSAGE	Mensagens informativas	
WARNING_MESSAGE	Mensagens de aviso	
QUESTION_MESSAGE	Mensagens Interrogativas	
PLAIN_MESSAGE	Mensagens sem o ícone	

Tipos de Opções	Descrição
YES_NO_OPTION	Botões Sim Não
YES_NO_CANCEL_OPTION	Botões Sim, Não e Cancelar
OK_CANCEL_OPTION	Botões OK, Cancelar
DEFAULT_OPTION	Botão OK

Tipos de Retorno	Descrição
YES_OPTION	O usuário retornou Sim
NO_OPTION	O usuário retornou Não
CANCEL_OPTION	O usuário retornou Cancelar
OK_OPTION	O usuário retornou OK

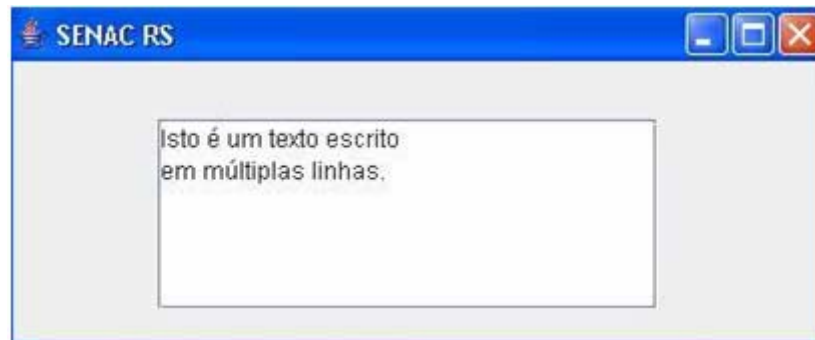
Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJOptionPane {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JOptionPane");
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JOptionPane j = new JOptionPane();
        j.showMessageDialog(tela,"Java Vive!!!","SENAC RS",
        JOptionPane.WARNING_MESSAGE);
    }
}
```


JTEXTAREA

Este componente nos permite exibirmos texto em múltiplas linhas.



PRINCIPAIS CONSTRUTORES

- ▶ `JTextArea()`
- ▶ `JTextArea(String texto)`
- ▶ `JTextArea(String texto, int linhas, int colunas)`

PRINCIPAIS MÉTODOS

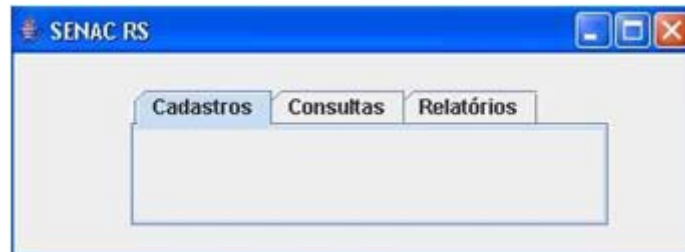
Métodos	Descrição	Assinatura
<code>getText()</code>	Retorna o texto do campo	<code>String getText()</code>
<code>setText()</code>	Modifica o texto do campo	<code>void setText(String Texto)</code>
<code>append()</code>	Insere um texto no final	<code>void append (String texto)</code>

Exemplo de Código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJTextArea {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JTextArea");
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JTextArea ta = new JTextArea("texto um");
        ta.setBounds(100,100,200,200);
        tela.getContentPane().add(ta);
        tela.setSize(800,600);
        tela.setVisible(true);
        ta.append(" teste");
    }
}
```

JTABBEDPANE

Componente que serve para exibir guias. Onde cada guia devemos adicionar um JPanel.



PRINCIPAIS CONSTRUTORES

- ▶ JTabbedPane()
- ▶ JTabbedPane(int direcao)

PRINCIPAIS MÉTODOS

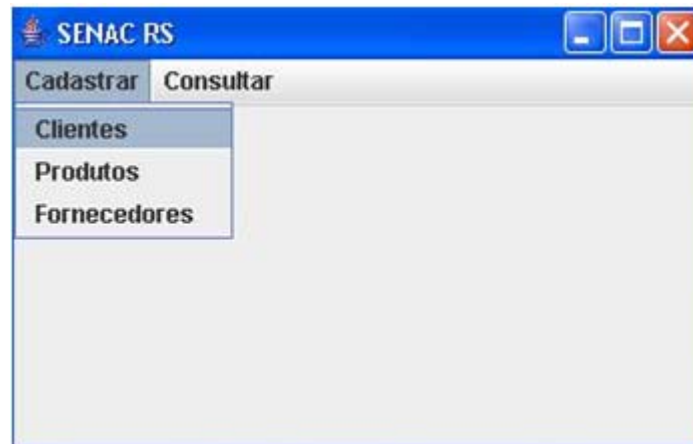
Métodos	Descrição	Assinatura
add()	Adiciona um componente com um texto	Component add(String texto, Component componente)
remove()	Remove um componente especificado pelo indice	int remove(int indice)
setIconAt()	Altera um ícone	void setIconAt(int indice, Icon icone)
setSelectedIndex()	Configura a guia selecionada	void setSelectedIndex(int indice)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJTabbedPane {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame("Teste JTabbedPane");
        tela.setLayout(null);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JTabbedPane tp = new JTabbedPane(JTabbedPane.LEFT);
        tp.setBounds(100,100,200,200);
        tela.getContentPane().add(tp);
        tp.add("A",new JPanel());
        tp.add("B",new JPanel());
        tp.add("C",new JPanel());
        tela.setSize(800,600);
        tela.setVisible(true);
    }
}
```

MENUBAR, JMENU E JMENUITEM



A classe JMenuBar é utilizada, juntamente com JMenu e JMenuItem, para gerar menus em Java.

PRINCIPAIS CONSTRUTORES

- ▶ JMenuBar()
- ▶ JMenuBar(String texto)

PRINCIPAIS MÉTODOS

Métodos	Descrição	Assinatura
add()	Adiciona um componente ao menu	Component add(Component c)
remove()	Remove um componente JMenuItem	void remove(JMenuItem item)
Insert()	Insere um JMenuItem em uma posição	JMenuItem insert(JMenuItem m, int pos)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteJMenuBar {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame();
        tela.setTitle("Teste JMenuBar");
        JMenuBar barra = new JMenuBar();
        JMenu menu1 = new JMenu("Cadastrar");
        JMenu menu2 = new JMenu("Consultar");
        JMenuItem item1 = new JMenuItem("Clientes");
        JMenuItem item2 = new JMenuItem("Fornecedores");
        JMenuItem item3 = new JMenuItem("Produtos");
        JMenuItem item4 = new JMenuItem("Vendas");
        menu1.add(item1);
        menu1.add(item2);
        menu1.add(item3);
        menu2.add(item4);
        barra.add(menu1);
        barra.add(menu2);
        tela.setJMenuBar(barra);
    }
}
```

```

tela.setSize(800,600);
tela.setVisible(true);
tela.getContentPane().setBackground(Color.RED);
tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
}

```

JTABLE



Produto	Preço	Quantidade	Medida
Queijo	R\$ 150,00	78	Kg
Refrigerante	R\$ 2,00	200	Unidade
Presunto	R\$ 15,00	89	Kg
Salame	R\$ 18,00	74	Kg

PRINCIPAIS CONSTRUTORES

- ▶ `JTable(int li, int co)`
- ▶ `JTable(Object dados, Object colunas)`

Métodos	Descrição	Assinatura
<code>clearSelection()</code>	Limpa a seleção de todas colunas	<code>void clearSelection()</code>
<code>getSelectedColumn()</code>	Retorna a última coluna selecionada	<code>Int getSelectedColumn()</code>
<code>getSelectedRow()</code>	Retorna a última linha selecionada	<code>Int getSelectedRow()</code>
<code>getSelectedRows()</code>	Retorna as linhas selecionadas	<code>Int[] getSelectedRows()</code>
<code>getValueAt()</code>	Retorna o valor da célula indicada pela linha e coluna	<code>Object getValueAt(int linha, in coluna)</code>
<code>setValueAt()</code>	Troca o valor da célula indicada pela linha e coluna	<code>Object setValueAt(Object valor, int linha, int coluna)</code>
<code>setSelectionMode()</code>	Configura o modo de seleção da tabela	<code>void setSelectionMode(int modo)</code>

Exemplo de código:

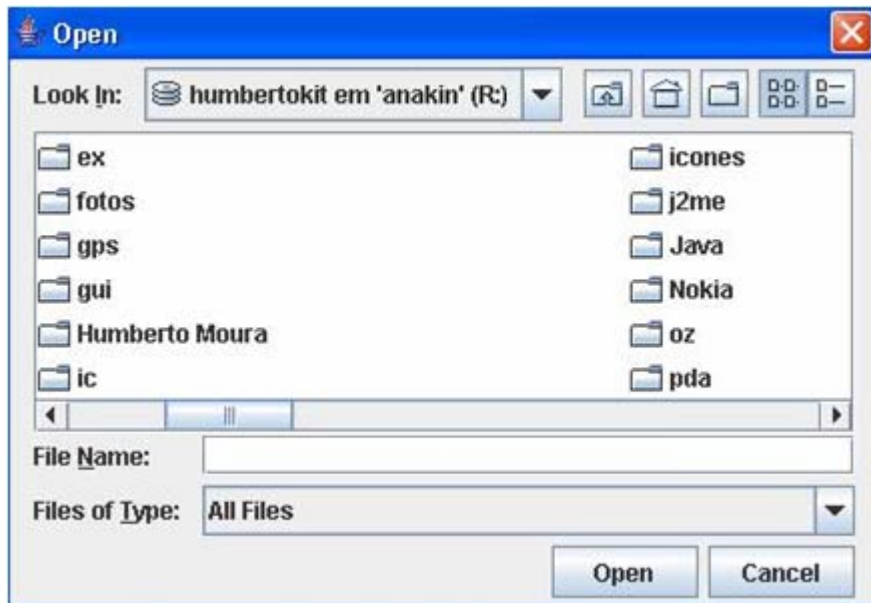
```

import javax.swing.*;
import java.awt.*;

public class TesteJTable {
    public static void main(String[] args) {
        JFrame tela = new JFrame("Teste JTable");
        tela.getContentPane().setBackground(Color.WHITE);
        String[] colunas = new String []{"Nome", "Idade"};
        String[][] dados = new String [][] {{"Humberto Moura", "26"}, {"", ""}};
        JTable tabela = new JTable(dados, colunas);
        JScrollPane sp = new JScrollPane(tabela);
        tela.getContentPane().add(sp);
        tela.pack();
        tela.setVisible(true);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

JFILECHOOSER



JFileChooser implementa uma caixa de diálogo para que o usuário possa selecionar um arquivo local.

PRINCIPAIS CONSTRUTORES

► JFileChooser()

PRINCIPAIS MÉTODOS

Métodos	Descrição	Assinatura
showOpenDialog()	Exibe a caixa de seleção de arquivo	Component int showOpenDialog(Component pai)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
public class TesteJFileChooser {
    public static void main(String[] args) {
        JFrame tela;
        tela = new JFrame();
        tela.setTitle("Teste JFileChooser");
        tela.setSize(800,600);
        tela.setVisible(true);
        int ok;
        JFileChooser sa = new JFileChooser();
        TesteJFileChooser a = new TesteJFileChooser();
        ok = sa.showOpenDialog(null);
        if (ok == JFileChooser.APPROVE_OPTION) {
            System.out.println("Arquivo: " + sa.getSelectedFile().getName());
        }
        tela.getContentPane().setBackground(Color.RED);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}
```

JINTERNALFRAME

Utilizado em aplicações de múltiplas janelas (mdi)

PRINCIPAIS CONSTRUTORES

▶ `JInternalFrame(String titulo, boolean redm, boolean fecha, boolean max, boolean min)`

PRINCIPAIS MÉTODOS

Métodos	Descrição	Assinatura
<code>show()</code>	Exibe o <code>JInternalFrame</code>	<code>void show()</code>
<code>dispose()</code>	fecha o <code>jinternalframe</code>	<code>void dispose()</code>

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
public class Mdi {
    public static void main(String[] args) {
        new Teste();
    }
}

class Teste extends JFrame{
    JDesktopPane painel= new JDesktopPane();
    JInternalFrame i1 = new JInternalFrame("Java",true,true,true,true);
    JInternalFrame i2 = new JInternalFrame("Desktop",true,true,true,true);

    Teste() {
        setTitle("SENAC RS");
        getContentPane().add(painel);
        painel.add(i1);
        painel.add(i2);
        painel.setVisible(true);
        i1.setSize(270,270);
        i2.setSize(270,270);
        i1.show();
        i2.show();
        setSize(800,600);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Codigo:	Nome:	Endereco:	Cidade:	Estado:	Cep:	Telefone:	Cel:	Dt. Nascimento:	E-mail:	Site:	CPF:	RG:	Estado Civil:
000001	Humberto Moura	Av. Venâncio Aires, 99	Cidade Baixa	RS	96000-000	(0xx)51-30293633	(0xx)51-92527855	18/03/1980	humberto@humbertomm	www.senacrs.com.br	345345345-34	87454337837584	Solteiro

Codigo:	Codigo Forcedor:	Codigo Produto:	Preco Custo:	Disponibilidade:
00001	834823	3437	R\$ 78897.993	Imediata

LAYOUT

GERENCIADORES DE LAYOUT

O Gerenciador de layout, é um objeto que implementa a interface `LayoutManager` determinando o modo de exibição dos componentes dentro de um container.

Para configurarmos o layout de um container utilizamos o método `setLayout(new TipoLayout())`. Sendo `TipoLayout` substituído pela classe de layout escolhida. Se não quisermos utilizar nenhum passamos `null` como parâmetro a este método.

FLOWLAYOUT

O `FlowLayout` coloca os componentes em um fluxo seqüencial na janela, da esquerda para direita e de cima para baixo, como se os componentes fossem palavras sendo escritas em um caderno.



PRINCIPAIS CONSTRUTORES

- ▶ `FlowLayout()`
- ▶ `FlowLayout(int alinhamento, x, y)`

Exemplo de código:

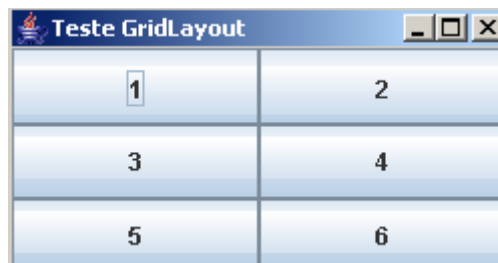
```
import javax.swing.*;
import java.awt.*;

public class Inicio {
    public static void main(String[] args) {
        new Teste();
    }
}
```

```
class Teste extends JFrame{
    Teste() {
        setTitle("SENAC RS");
        JButton t1 = new JButton("Teste");
        JButton t2 = new JButton("do");
        JButton t3 = new JButton("layout");
        JButton t4 = new JButton("Flow");
        JButton t5 = new JButton("Layout");
        getContentPane().add(t1);
        getContentPane().add(t2);
        getContentPane().add(t3);
        getContentPane().add(t4);
        getContentPane().add(t5);
        setLayout(new FlowLayout());
        setSize(800,600);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

GRIDLAYOUT

O GridLayout nos permite distribuir os componentes em linhas e colunas, como se fosse uma grade no container. A distribuição é feita da esquerda para direita e de cima para baixo.



PRINCIPAIS CONSTRUTORES

- ▶ GridLayout()
- ▶ GridLayout(int linhas, int colunas)
- ▶ GridLayout(int linhas, int colunas, int horizontal, int vertical)

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;

public class TesteGridLayout {
    public static void main(String[] args) {
        JFrame tela;
        JButton botao1 = new JButton("1");
        JButton botao2 = new JButton("2");
        JButton botao3 = new JButton("3");
        JButton botao4 = new JButton("4");
        JButton botao5 = new JButton("5");
        JButton botao6 = new JButton("6");
```



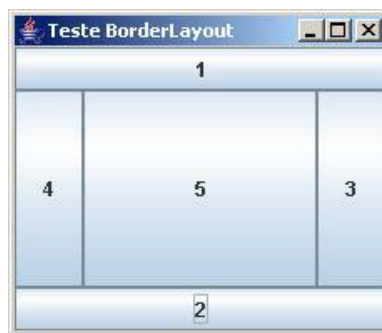
```

tela = new JFrame();
tela.setTitle("Teste GridLayout");
tela.setLayout(new GridLayout(3,3));
tela.getContentPane().add(botao1);
tela.getContentPane().add(botao2);
tela.getContentPane().add(botao3);
tela.getContentPane().add(botao4);
tela.getContentPane().add(botao5);
tela.getContentPane().add(botao6);
tela.pack();
tela.setVisible(true);
tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}
}

```

BORDERLAYOUT

Divide o container em cinco regiões imaginárias chamadas de “PAGE_START” (1), “PAGE_END” (2) “LINE_END” (3) “LINE_START” (4) e “CENTER” (5), sendo que apenas um componente poderá ocupar cada região.



Exemplo de código:

```

import javax.swing.*;
import java.awt.*;
public class TesteBorderLayout {
    public static void main(String[] args) {
        JFrame tela;
        JButton botao1 = new JButton("1");
        JButton botao2 = new JButton("2");
        JButton botao3 = new JButton("3");
        JButton botao4 = new JButton("4");
        JButton botao5 = new JButton("5");
        tela = new JFrame();
        tela.setTitle("Teste BorderLayout");
        tela.setLayout(new BorderLayout());
        tela.getContentPane().add(botao1, BorderLayout.PAGE_START);
        tela.getContentPane().add(botao2, BorderLayout.PAGE_END);
        tela.getContentPane().add(botao3, BorderLayout.LINE_END);
        tela.getContentPane().add(botao4, BorderLayout.LINE_START);
        tela.getContentPane().add(botao5, BorderLayout.CENTER);
        tela.setVisible(true);
        tela.pack();
        tela.getContentPane().setBackground(Color.RED);
        tela.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
}

```

EVENTOS

O tratamento de eventos é feito através de delegação de tratamento.

Para que possamos trabalhar com devemos importar o pacote `java.awt.event`

Para cada tipo de evento existe uma interface "listener" responsável por este. Devemos, então, selecionar a interface correta para cada componente e evento, implementando todos os seus métodos ou deixando-os vazios se não necessários.

Quanto ao componente que queremos manipular os eventos, devemos registrar esta interface através do método "add" apropriado.

ACTIONEVENT

São manipuladores de evento de ação

Interface	Método
ActionListener	ActionPerformed(ActionEvent)

Componentes	Método Add
JButton	AddActionListener()
TextField (quando pressionado ENTER)	
JList (clique duplo)	
JMenuItem	

Principais Métodos	Descrição	Assinatura
getActionCommand()	Comando relacionado a ação	String getActionCommand()

Exemplo de código:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TesteActionEvent implements ActionListener {
    public TesteActionEvent() {
        JFrame tela;
        JButton botao;
        JButton botaoAux;
        tela = new JFrame();
        botao = new JButton();
        tela.setLayout(null);
        tela.setTitle("Teste ActionEvent");
        tela.getContentPane().setBackground(Color.GREEN);
        tela.add(botao);
        tela.setSize(800,600);
        tela.setVisible(true);
        tela.getContentPane().add(botao);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        botao.setSize(200,150);
        botao.setText("OK");
        botao.setLocation(250,180);
        botao.setActionCommand("ok");
        botao.addActionListener(this);
    }
}
```

```

    }
    public static void main(String[] args) {
        new TesteActionEvent();
    }
    public void actionPerformed(ActionEvent ae) {
        if ("ok".equals(ae.getActionCommand())) {
            System.out.println("OK Pressionado!!!");
        }
    }
}

```

ITEMEVENT

São manipuladores de eventos de seleção de itens

Interface	Método
ItemListener	ItemStateChanged(ItemEvent)

Componentes	Método Add
JcheckBox	AddItemListener()
List	

Principais Métodos	Descrição	Assinatura
getItem()	Retorna o Item	Object getItem()
getItemSelectable()	Retorna o controle	ItemSelectable getItemSelectable()

TEXTEVENT

São manipuladores de eventos de texto

Interface	Método
TextListener	textValueChanged(TextEvent)

Componentes	Método Add
JTextArea	AddTextListener()
JTextField	

Principais Métodos	Descrição	Assinatura

FOCUSEVENT

São manipuladores de eventos de foco

Interface	Método
FocusListener	focusGained(FocusEvent)
	FocusLost(FocusEvent)

Componentes	Método Add
Todos os componentes	AddFocusListener()
List	

KEYEVENT

São manipuladores de eventos do teclado

Interface	Métodos
KeyListener	keyPressed(KeyEvent)
	KeyReleased(KeyEvent)
	KeyTyped(KeyEvent)

Componentes	Método Add
Todos os componentes	AddKeyListener()

Principais Métodos	Descrição	Assinatura
getKeyChar()	Retorna um caracter	char getKeyChar()
getKeyCode()	Retorna o código do caracter	int getKeyCode()

MOUSEEVENT

São manipuladores de evento do mouse

Interface	Método
MouseListener	mouseClicked(MouseEvent)
	mouseEntered(MouseEvent)
	mouseExited(MouseEvent)
	mousePressed(MouseEvent)
	mouseReleased(MouseEvent)

Componentes	Método Add
Todos os componentes	addMouseListener()
List	

Principais Métodos	Descrição	Assinatura
getX()	Retorna a coordenada x do clique	int getX()
getY()	Retorna a coordenada y do clique	int getY()

WINDOWEVENT

São manipuladores de eventos de janela

Interface	Método
WindowListener	WindowActivated(WindowsEvent)
	WindowClosed(WindowsEvent)
	WindowClosing(WindowsEvent)
	WindowDeactivated(WindowsEvent)
	WindowDeiconfied(WindowsEvent)
	WindowIconfied (WindowsEvent)
	WindowOpened (WindowsEvent)

Componentes	Método Add
JFrame	AddWindowListener()

Principais Métodos	Descrição	Assinatura
getWindow()	Retorna a janela	Window getWindow

CLASSES ADAPTER

São classes que utilizamos para o tratamento de eventos. A vantagem de se utilizar uma classe Adapter está no fato de que só precisamos escrever os métodos que iremos realmente utilizar, nos poupando de escrever todos os métodos obrigatórios da interface Listener.

Classe Adapter	Interface Listener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
WindowAdapter	WindowListener

PROGRAMAÇÃO MULTITAREFA EM JAVA

THREADS

Threads são linhas em execução de um programa. Tecnicamente, um thread é um único fluxo seqüencial de execução. A linguagem Java permite que seu aplicativo execute mais de uma linha simultaneamente em sistemas operacionais que permitam multitarefa.

Podemos criar uma thread para cada operação que necessitamos, desta forma teríamos estas operações sendo virtualmente executadas ao mesmo tempo.

Se em nosso aplicativo precisássemos fazer diferentes cálculos com informações de banco de dados remotas de diversas localidades do mundo, sendo que queremos o resultado destas o mais rápido possível, como poderíamos proceder de tal forma que uma quantidade maior de cálculos seja concluída em menos tempo possível?

Uma possível solução poderia ser alocar uma thread a cada rotina de cálculo.

A CLASSE THREAD

A classe `java.lang.Thread` é responsável pela programação multitarefa em java.

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
<code>start()</code>	Inicia execução da thread	<code>void start()</code>
<code>yeld()</code>	Pausa a thread passando a vez para a próxima de mesma prioridade que a atual	<code>static void yeld()</code>
<code>sleep()</code>	Pausa a tread passando a vez para a próxima	<code>static void sleep(long ms)</code>
<code>run()</code>	Método conter o código da thread	<code>void run()</code>
<code>join()</code>	Adiciona a thread em execução ao final da thead que executou o método	<code>void join()</code>
<code>wait()</code>	Pausa a thread em execução até ser notificada	<code>void wait()</code>
<code>notify()</code>	Notifica a thread que foi pausada por <code>wait()</code>	<code>void notify()</code>
<code>notifyAll()</code>	Notifica todas as threads que foram pausadas com <code>wait()</code>	<code>void notifyAll()</code>
<code>setPriority()</code>	Configura a prioridade	<code>void setPriority(int p)</code>
<code>setName()</code>	Configura um nome a uma thread	<code>Void setName(String nome)</code>

CRIANDO THREADS EM JAVA

Existem duas formas de cruar threads em java

Estender a classe `Thread`

Implementar a interface `Runnable`

Criando Theads estendendo de `java.lang.Thead`

Esta é a maneira mais simples de criarmos uma thread:

Devemos estender da classe `Thead` e depois subscrever o método `run()`

Exemplo:

```
public class TesteThreads {
    public void executa() {

        Linha l1 = new Linha();
        Linha2 l2 = new Linha2();
        l1.start();
        l2.start();
    }
}
```

```
}  
public static void main(String[] args) {  
    new TesteThreads().executa();  
}  
}  
class Linha extends Thread {  
    public void run() {  
        for(int x=0;x<200;x++) {  
            System.out.println("1");  
        }  
    }  
}  
class Linha2 extends Thread {  
    public void run() {  
        for(int x=0;x<200;x++) {  
            System.out.println("2");  
        }  
    }  
} }
```

CRIANDO THREADS IMPLEMENTANDO RUNNABLE

Esta abordagem é mais trabalhosa, mas nos dá um maior controle sobre as threads

Implementando Runnable, podemos estender de qualquer classe, além de definir o comportamento que será executado por um thread separado.

Exemplo

```
public class TesteThreads2 {  
    public void executa() {  
        Linha l = new Linha();  
        Linha2 l2 = new Linha2();  
        Thread t = new Thread(l);  
        Thread t2 = new Thread(l2);  
        t.setName("um");  
        t2.setName("dois");  
        t.start();  
        t2.start();  
    }  
    public static void main(String[] args) {  
        new TesteThreads2().executa();  
    }  
}  
class Linha implements Runnable {  
    public void run() {  
        for(int x=0;x<2;x++) {  
            System.out.println("1 " + Thread.currentThread().getName());  
        }  
    }  
}  
class Linha2 implements Runnable {  
    public void run() {  
        for(int x=0;x<2;x++) {  
            System.out.println("2 " + Thread.currentThread().getName());  
        }  
    }  
}
```

ESTADOS DE UMA THREAD

O thread pode estar em cinco estados:

Novo

Significa que o thread foi instanciado, mas ainda não foi executado o método `start()`. Ele é considerado inativo

EXECUTÁVEL

Este estado é quando a thread está esperando para ser executada pela JVM. A primeira vez que entra neste estado é na chamada ao método `start()`.

EXECUÇÃO

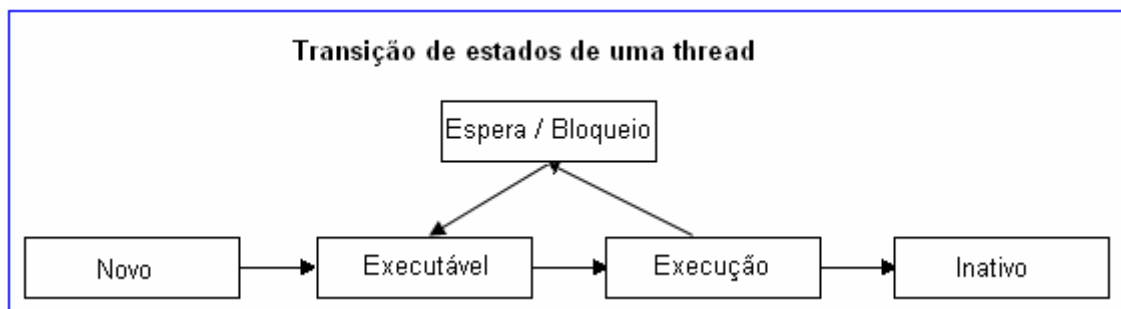
Este estado é quando o thread está sendo executado. Somente o agendador pode efetivamente colocá-lo neste estado.

ESPERA/BLOQUEIO/SUSPENSÃO

Ainda não está no estado executável, mas podera ocorrer em breve.

INATIVO

Um thread está inativo quando o seu método `run()` foi concluído.



IMPEDINDO A EXECUÇÃO DE UMA THREAD

MÉTODO SLEEP()

Podemos utilizar o método estático `sleep(long n)` para pausar a execução de uma thread por `n` milissegundos. Utilizamos este método quando queremos desacelerar a execução de uma thread.

▶ `Thread.sleep(1000)`

Exemplo:

```
public class TesteThreads3 {
    public void executa() {
        Linha l1 = new Linha();
        Linha2 l2 = new Linha2();
        Thread t1 = new Thread(l1);
        Thread t2 = new Thread(l2);
        t1.setName("um");
        t2.setName("dois");
        t1.start();
        t2.start();
    }
    public static void main(String[] args) {
```



```
        new TesteThreads3().executa();
    }
}

class Linha implements Runnable {
    public void run() {
        for(int x=0;x<200;x++) {
            System.out.println("1 " + Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            } catch( InterruptedException e ) { }
        }
    }
}

class Linha2 implements Runnable {
    public void run() {
        for(int x=0;x<200;x++) {
            System.out.println("2 " + Thread.currentThread().getName());
            try {
                Thread.sleep(1000);
            } catch( InterruptedException e ) { }
        }
    }
}
```

MÉTODO YELD()

O método yield é utilizado para dar a vez na execução para uma thread com a mesma prioridade. Para configurarmos a prioridade de uma thread utilizamos o método setPriority()

MÉTODO setPriority()

Ajusta a propriedade de execução de uma thread()

As threads podem ter prioridades de execução entre Thread.MIN_PRIORITY até MAX_PRIORITY. Geralmente estes números variam de 1 a 10, de acordo com a JVM em uso.

Exemplo:

```
t = new Thread();
t.setPriority(8);
```

O MÉTODO JOIN()

O método join() permite adicionarmos uma thread ao final de outra. Seu uso está associado a possibilidade de termos que esperar o final de uma thread X para só então iniciarmos uma thread Y.

Exemplo:

```
Thread t = new Thread();
t.start();
t.join();
```

O código acima pega a thread que está em execução e adiciona ao final da thread t.

SINCRONIZAÇÃO

O uso de threads nos permite a execução de tarefas simultaneamente.

Mas, muitas vezes, existem tarefas que não queremos que sejam executadas simultaneamente.

Para definir que dois ou mais métodos ou blocos de código não devem ser executados ao mesmo tempo pelo mesmo objeto, utilizando a palavra chave synchronized.

Exemplo:

```
synchronized void beber() {
    int cerveja;
    int boca;
    cerveja = 24;
    boca = cerveja;
}
synchronized void dirigir() {
    ligarVeiculo();
    engatarMarcha();
    largarEmbreagem();
    acelerar();
    try{
        ultrapassar();
    }
    catch(Acidente e) {}
}
```

INTERAÇÃO COM AS THREADS

Os métodos `wait()`, `notify()` e `notifyAll()` permite-nos ter controle sobre a suspensão das threads que estão em execução. Devem somente ser utilizados dentro de métodos ou blocos de código "synchronized".

MÉTODO WAIT()

Indica que o thread ativo deve esperar até que o mesmo seja notificado.

MÉTODO NOTIFY()

Notifica o thread que foi suspenso após `wait()` colocando-o em estado executável.

MÉTODO NOTIFYALL()

Notifica todos os threads que foram suspensos colocando-os em estado executável.

Exemplo:

```
public class A {
    public static void main(String [] args) {
        B b = new B();
        b.start();
        synchronized(b) {
            try {
                System.out.println("Esperando b terminar");
                b.wait();
            } catch(InterruptedException e) {}
            System.out.println("Total: " + b.total);
        }
    }
}
class B extends Thread {
    int total;
    public void run() {
        synchronized(this) {
            for(int i=0;i<100;i++){
```

```
total += i;
}
notify();
}
}
}
```

Tipo Enumerados

COLEÇÕES

As principais interfaces que implementam coleções são:

LIST

É um conjunto que admite elementos repetidos

PRINCIPAIS MÉTODOS

Método	Descrição	Assinatura
add()	Adiciona um objeto a lista	void add()
get()	Retorna o objeto da lista pelo índice	object get(int indice)
remove()	Remove um objeto da lista pelo índice	void remove(int indice)
indexOf()	procura um objeto na lista	int indexOv(Object o)

Exemplo:

```
import java.util.*;
import java.util.Collections;
public class TesteList{
    public static void main(String[] args){
        List<String> lista;// repeticoes permitidas e ordenadas
        lista = new ArrayList<String>();
        lista.add("1");
        lista.add("4");
        lista.add("3");
        lista.add("2");
        System.out.println(lista.get(1));
        System.out.println(lista);
        Collections.sort(lista);
        System.out.println(lista);
    }
}
```

SET

Conjunto que não admite elementos repetidos

Método	Descrição	Assinatura
add()	Adiciona um objeto no conjunto	void add()
contains()	Procura no conjunto	boolean contains(Object e)
remove()	Remove um objeto do conjunto pelo índice	void remove(int indice)
clear()	Limpa o conjunto	void clear()

Exemplo:

```
import java.util.*;
import java.util.Collections;
public class TesteSet{
    public static void main(String[] args){

        TreeSet<String> conj;
        conj = new TreeSet<String>();
        conj.add("1");
        conj.add("4");
        conj.add("1"); // nao adiciona, pois ja existe
        conj.add("3");
        conj.add("2");
        System.out.println(conj.contains("3"));
        System.out.println(conj.headSet("3"));
        System.out.println(conj.tailSet("3"));
        System.out.println(conj.contains("9"));
        System.out.println(conj);

    }
}
```

MAP

Map é um conjunto de pares chave/valor.

Método	Descrição	Assinatura
put()	Insere um par chave-valor	void put(K chave,V Valor)
get()	Retorna o valor da chave associada	V get(K chave)
containsKey()	Retorna true se contém a chave	boolean containsKey(Object chave)
containsValue()	Retorna true se contém o valor	boolean containsValue(Object valor)

Exemplo:

```
import java.util.*;
import java.util.Collections;
public class TesteMap{
    public static void main(String[] args){

        Map<String,Integer> mapa;
        mapa = new TreeMap<String,Integer>();
        mapa.put("um",1);
        mapa.put("dois",2);
        mapa.put("tres",3);
```

```
mapa.put("quatro", 4);  
System.out.println(mapa.get("tres"));  
System.out.println(mapa.get("um"));  
System.out.println(mapa);  
  
}  
}
```