

TREINAMENTO JAVA SE (AULAS 1 A 61)
UNIVERSIDADE XTI (RICARDO LOPES COSTA)

<https://www.youtube.com/watch?v=NZDzuve7kho&list=PLxQNfKs8YwvGhXHbHtxtoB-tRRv6r3Rlr>

Resumo do curso feito por Roberto Pinheiro

1. INSTALAÇÃO E CONFIGURAÇÃO

Edições da Linguagem Java:

- JSE - Java Standard Edition (Edição Padrão)
- JEE - Java Enterprise Edition (Edição Empresarial)
- JME - Java Micro Edition (aplicativos para dispositivos mobiles)
- Java Card
- Java TV

Baixar em:

<http://java.sun.com>

<http://www.oracle.com/technetwork/java/index.html>

Baixar edição JSE e a sua documentação.

Windows x86 → 32 bits

JDK (Java Development Kit)

jdk-7u5-windows-i586.exe

Diretório padrão de instalação:

C:\Arquivos de programas\Java\jdk1.7.0_05

Documentação:

jdk-7u4-apidocs

Colocar a pasta "docs" dentro de:

C:\Arquivos de programas\Java\jdk1.7.0_05

Documentação online:

<http://docs.oracle.com/javase/7/docs/api/index.html>

java -version

javac

Meu Computador → Propriedades → Avançadas → Variáveis de ambiente

Path=c:\foxbase;c:\windows;c:\windows\system32;c:\Arquivos de programas\Java\jdk1.7.0_05;c:\Arquivos de programas\Java\jdk1.7.0_05\bin;
JAVA_HOME=c:\Arquivos de programas\Java\jdk1.7.0_05;
CLASSPATH=;

Baixar editor de textos:

Notepad ++

<http://notepad-plus-plus.org/>

2. HELLO WORLD

O nome do arquivo deve ser igual ao nome da classe.

Arquivo: HelloWorld.java

```
public class HelloWorld {  
    public static void main (String[] args) {  
        System.out.print("Oi Mundo!");  
    }  
}
```

Compilar:

```
javac HelloWorld.java
```

gera o byte code:

```
HelloWorld.class
```

Para executar o programa na JVM (Java Virtual Machine):

```
java HelloWorld
```

OBS.: Não é necessário colocar a extensão ".class"



3. FUNDAMENTOS JAVA

- Classe
- Métodos
- Instruções

```
public class HelloWorld {  
    public static void main (String[] args) {  
        System.out.print("Oi Mundo!");  
    }  
}
```

OBS.: Cada instrução é finalizada com ponto-e-vírgula (;).

3.1. Comentários

Existem 3 tipos de comentários:

- 1) // texto → comentário de uma única linha
- 2) /* texto */ → comentário de múltiplas linhas
- 3) /** texto */ → comentário para criar documentação do código

Exemplo:

```
/**  
 * Programa que imprime texto na tela.  
 * @author Roberto Pinheiro  
 */  
  
public class HelloWorld {  
    /* Método principal da classe */  
    public static void main (String[] args) {  
        System.out.print("Oi Mundo!");  
    } // fim do método  
} // fim da classe
```

3.2. Impressão em linhas diferentes

Arquivo: HelloWorld2.java

```
public class HelloWorld2 {  
    public static void main (String[] args) {  
        System.out.println("Oi Mundo!");  
        System.out.println("Java SE");  
    }  
}
```

OBS.: `System.out.println()` → imprime texto e insere uma quebra de linha.

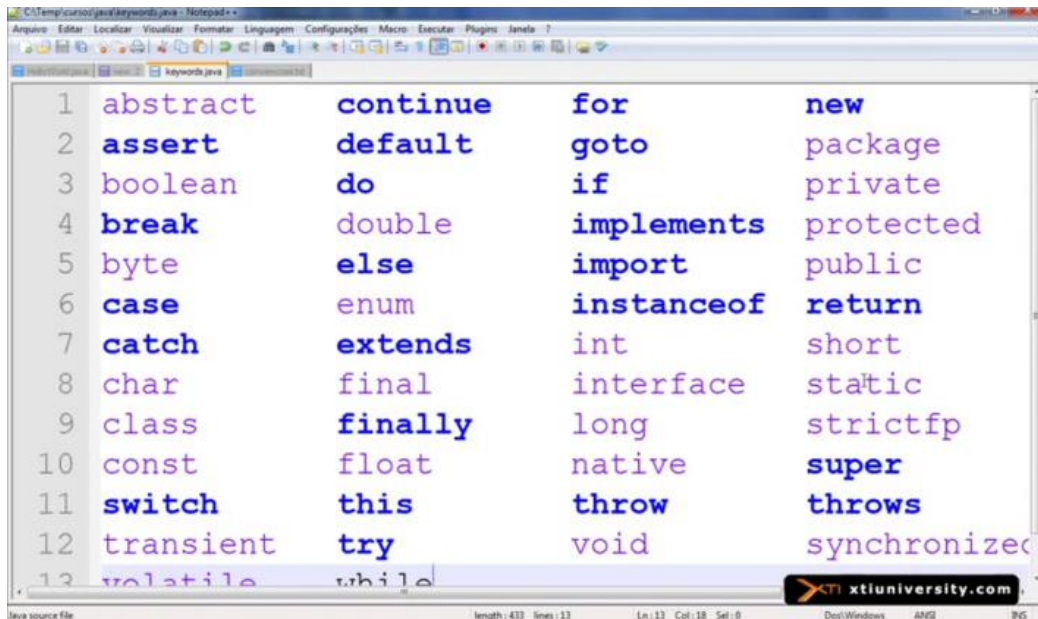


3.3. Convenções de código Java

Convenções de código Java → Padrões de boas práticas dos bons programadores → <http://www.oracle.com/technetwork/java/codeconv-138413.html>

Regras:

- 1) O nome do arquivo (fonte java) deve ser o mesmo da classe java que está dentro dele.
- 2) O nome das classes devem sempre começar com letra maiúscula (as demais devem ser letras minúsculas). Se o nome for composto, a primeira letra de cada nome deve ser maiúscula (e as demais letras minúsculas). Ex: HelloWorld. Não colocar espaço (não é válido) ou underline (_) entre os nomes (ex: Hello World ou Hello_World).
- 3) Não iniciar nome de classes com números (não é válido), underline (_) ou cifrão (\$).
- 4) O nome dos métodos devem sempre começar com letras minúsculas (ex: main). Se o nome do método for composto, usar a inicial do primeiro nome em letra minúscula e a inicial dos demais nomes em letras maiúsculas (ex: nomeComposto).
- 5) Com relação ao nome de classes e métodos não é permitido usar palavras reservadas da linguagem Java.



- 6) O nome de uma classe deve ser um substantivo.
- 7) Os métodos são ações. Usar como nome um verbo.

3.4. Sequências de escape

\n = quebra de linha
\t = tabulação
\" = aspas
\\ = barra invertida

Arquivo: TesteEscape.java

```
public class TesteEscape {  
    public static void main (String[] args) {  
        System.out.println("\tOi Mundo!");  
        System.out.println("Java \"SE\"");  
    }  
}
```

4. VARIÁVEIS E SÍNTAXE

Variáveis são referências a dados.

Arquivo: Variavel.java

```
public class Variavel {  
    public static void main (String[] args) {  
        String nome, sobrenome;  
        nome = "Roberto";  
        sobrenome = "Pinheiro";  
        int idade = 51;  
        boolean casado = false;  
        System.out.println(nome + " " + sobrenome);  
    }  
}
```

OBS.: Os nomes de variáveis devem sempre ser iniciados com letras minúsculas.



```
Saída - CursoJavaUniversidadeXTI (run) X  
run:  
Roberto Pinheiro  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

5. TIPOS PRIMITIVOS

- byte (inteiro - 8 bits)
- short (inteiro - 16 bits)
- int (inteiro - 32 bits)
- long (inteiro - 64 bits)
- float (fracionário - 32 bits)
- double (fracionário - 64 bits)
- char (16 bits)
- void (representa a inexistência de um valor)
- Boolean (1 bit)

Arquivo: TiposPrimitivos.java

```
public class TiposPrimitivos {
    public static void main (String[] args) {

        int idade = 51;
        double preco = 12.45;
        char sexo = 'M'; // padrao Unicode
        boolean casado = true;

        byte b = 127; // ate aproximadamente 100
        short s = 32767; // ate aproximadamente 32 mil
        int i = 2147483647; // ate aproximadamente 2 bilhões
        long l = 9000000000000000000L; // até 9 quintilhões
        double d = 1.7976931348623157E+308D; // padrão IEEE 754
        float f = 123F;
        long l2 = 8534;

        i = s;
        i = (int) l2;

        System.out.println(i);

        // números na forma binária

        // preceder o número com 0b.

        byte bb = 0b01010101; // 8 bits - 1 byte
        short ss = 0b0101010101010101; // 16 bits - 2 bytes
        int ii = 0b01010101010101010101010101010101; // 32 bits

        System.out.println(ii);
    }
}
```

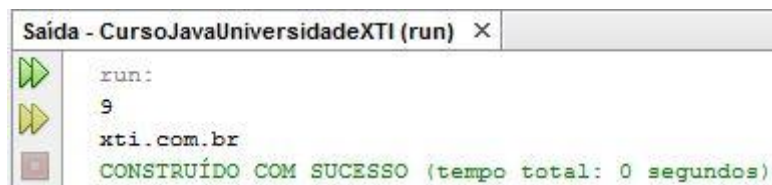


6. GARBAGE COLLECTOR

Variável de referência é toda variável que aponta para um objeto.

Arquivo: VarRef.java

```
public class VarRef {  
    public static void main (String[] args) {  
  
        // Variavel primitiva  
        int x = 7;  
        x = 9;  
  
        // Variavel de referencia  
        String y = "XTI";  
        y = "xti.com.br";  
  
        System.out.println(x);  
        System.out.println(y);  
    }  
}
```



O Garbage Collector gerencia os objetos que não tem mais referência com nenhuma variável no programa.

O Garbage Collector atua nos dados das variáveis de referência. Ele limpa (libera espaço) da memória os dados de referência antigos que não estão mais linkados com a variável de referência. No exemplo acima, ele limpa da memória o dado "XTI".

Para limpar totalmente a referência da variável y, faça o seguinte:

```
y = null;
```

7. CONSTANTES E MODIFICADOR FINAL

Arquivo: ModFinal.java

```
public class ModFinal {  
  
    public static void main (String[] args) {  
  
        final double PI = 3.14159265358979323846;  
        final char SEXO_MASCULINO = 'M';  
        final char SEXO_FEMININO = 'F';  
  
        System.out.println(3 * PI);  
  
    }  
}
```



OBS.: Um modificador final não pode ser modificado. Se isto for tentado, ao se compilar o programa, será apresentada uma mensagem de erro.

A regra é sempre escrever o nome das constantes em letras maiúsculas e se for um nome composto utilizar o underline (_).

8. WRAPPER CLASSES

Essas classes são empacotadores do tipo primitivo. São invólucros de tipos primitivos de dados. Para cada tipo primitivo da linguagem Java nós temos o seu espelho Wrapper.

Tipo primitivo	Wrappers primitivas
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Character
void	Void
boolean	Boolean

Wrappers são classes. Essas classes oferecem alguns recursos de conversão para serem utilizados na linguagem Java.

Arquivo: Wrapper.java

```
public class Wrapper {  
  
    public static void main (String[] args) {  
  
        // int idade = 51;  
        Integer idade = new Integer (51);  
  
        // double preco = 12.45;  
        Double preco = new Double ("12.45");  
        double d = preco.doubleValue();  
        int i = preco.intValue();  
        byte b = preco.byteValue();  
  
        // boolean casado = true;  
        Boolean casado = new Boolean ("true");  
  
        // Conversao estatica  
        double d1 = Double.parseDouble ("123.45");  
        int i1 = Integer.parseInt ("123");  
        float f1 = Float.parseFloat ("3.14F");  
  
        int i2 = Integer.valueOf ("101011",2);  
  
        System.out.println(i2);  
  
    }  
  
}
```



```
Saída - CursoJavaUniversidadeXTI (run) X  
run:  
43  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

OBS.: Use a documentação da linguagem Java (Java SE API em [index.html](#)) para saber os métodos que cada uma dessas classes Wrapper oferece.

9. ENTRADA DE DADOS COM SCANNER

Arquivo: Entrada.java

```
/**
 * Testar entrada de dados com Scanner.
 * @author Roberto Pinheiro
 */

import java.util.Scanner;

public class Entrada {

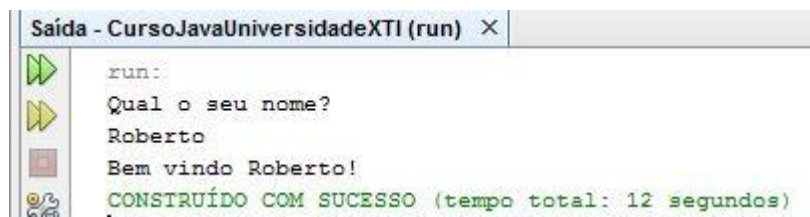
    public static void main (String[] args) {

        // Chamada do programa
        // System.out.println(args[0]);

        // Interagindo com o usuário
        Scanner s = new Scanner (System.in);
        System.out.println("Qual o seu nome?");
        String nome = s.nextLine();
        System.out.println("Bem vindo " + nome + "!");

    }

}
```



OBS.: java.lang → pacote padrão

10. ENTRADA GRÁFICA DE DADOS COM JOPTIONPANE

Arquivo: Entrada2.java

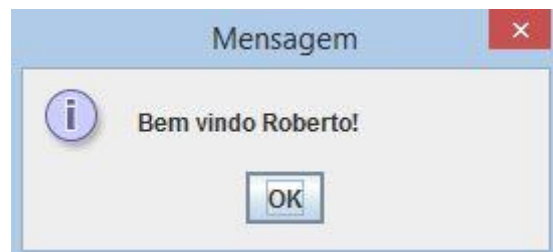
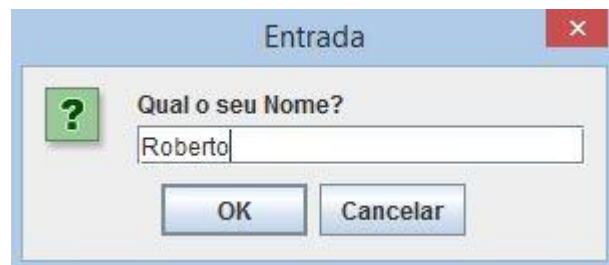
```
/**
 * Testar entrada gráfica de dados com JOptionPane.
 * @author Roberto Pinheiro
 */

import javax.swing.JOptionPane;

public class Entrada2 {

    public static void main (String[] args) {
        String nome = JOptionPane.showInputDialog("Qual o seu Nome?");
        // System.out.println(nome);
        JOptionPane.showMessageDialog(null, "Bem vindo " + nome + "!");
    }

}
```



11. OPERADORES

Os operadores são utilizados na construção de expressões.

11.1. Operadores e Operandos

$2 + 3$

- $+$ é um operador
- 2 e 3 são operandos
- $2 + 3$ é uma expressão

Tipos de operadores:

- unários (é um operador que trabalha com apenas um operando)
- binários (é um operador que trabalha com dois operandos)
- ternários (é um operador que trabalha com três operandos)

$=$ \rightarrow operador de atribuição

Operadores Aritméticos: Os habituais

Símbolo	Operação
$+$	soma
$-$	subtração
$*$	multiplicação
$/$	divisão
$\%$	resto da divisão

Operadores de Atribuição: O principal é $=$ mas existem mais operadores de atribuição com diferentes funções que explicamos brevemente agora.

Símbolo	Operação	Equivale a:
$+=$	$op1 += op2$	$op1 = op1 + op2$
$-=$	$op1 -= op2$	$op1 = op1 - op2$
$*=$	$op1 *= op2$	$op1 = op1 * op2$
$/=$	$op1 /= op2$	$op1 = op1 / op2$
$\% =$	$op1 \% = op2$	$op1 = op1 \% op2$

Operadores Unários: O mais ($+$) e o menos ($-$). Para mudar o sinal do operando.

Operador Instanceof: Verifica o tipo de um objeto. Permite-nos saber se um objeto pertence a uma classe ou não.

NomeObjeto instanceof NomeClasse

Operadores de Incremento e Decremento: São os operadores que nos permitem incrementar as variáveis em uma unidade. Podem ser usados diante ou atrás da variável dependendo do que quisermos, ou seja, se quisermos que se incremente ou vice-versa antes de utilizar ou o contrário.

Símbolo	Operação
$++$	incrementa a variável em uma unidade
$--$	decrementa a variável em uma unidade

Operadores de Comparação: Permitem comparar variáveis segundo a relação de igualdade/desigualdade ou relação maior/menor. Devolvem sempre um valor booleano.

Símbolo	Operação
>	maior que
<	menor que
==	igual
!=	diferente
>=	maior ou igual
<=	menor ou igual

Operadores Lógicos: Permite-nos construir expressões lógicas.

Símbolo	Operação
&&	devolve true se ambos operandos forem true.
	devolve true se algum dos operandos for true.
!	Nega o operando que se passa.
&	devolve true se ambos operandos forem true, avaliando ambos.
	devolve true se um dos operandos for true, avaliando ambos.

Operador de concatenação com cadeia de caracteres: '+':

Exemplo: `System.out.println("O total é"+ result +"unidades");`

Operadores que atuam a nível de bits: São muito menos utilizados.

Símbolo	Operação
>>	deslocamento à direita dos bits do operando.
<<	deslocamento à esquerda dos bits do operando.
&	operador and a nível de bit.
	operador or a nível de bit.

Outros operadores:

Símbolo	Operação
.	acessar as propriedades de um objeto.
[]	indexar arrays.
()	chamar métodos.
~	complemento unário sobre bits.
?	operador ternário.
new	operador para criar um novo objeto.
(tipo)	coerção unária.
,	separação de expressões

11.2. Precedência dos Operadores

Divisão e multiplicação tem precedência sobre soma e subtração.

`7 - 4 + 3 * 2 = 9`

Tudo que é realizado entre parênteses é calculado com antecedência.

`(7 - 4 + 3) * 2 = 12`

Arquivo: Operador.java

```
/**
 * Testar alguns tipos de operadores.
 * @author Roberto Pinheiro
 */

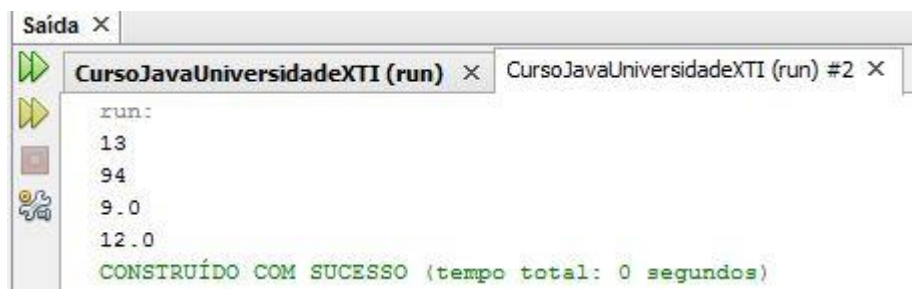
public class Operador {
    public static void main (String[] args) {

        // 2 + 3 --> operador binario
        // -2 --> operador unario
        // true ? "sim" : "não" --> operador ternario

        int x = 9 + 4;
        String y = "9" + "4";
        double z = 7 - 4 + 3 * 2;
        double w = (7 - 4 + 3) * 2;

        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        System.out.println(w);

    }
}
```



The screenshot shows a Java IDE's output window titled "Saída X". It contains two tabs: "CursoJavaUniversidadeXTI (run)" and "CursoJavaUniversidadeXTI (run) #2 X". The first tab is active and displays the output of the program. The output consists of four lines of numbers: 13, 94, 9.0, and 12.0, which correspond to the values of x, y, z, and w respectively. At the bottom of the output, it says "CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)".

```
run:
13
94
9.0
12.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


12. OPERADORES MATEMÁTICOS

Utilizando o operador % é possível determinar se um número é par ou ímpar. Sempre que o resultado da divisão de um número por 2 for:

- 0 → o número é par
- 1 → o número é ímpar.

O operador "-" inverte o sinal de um número.

12.1. Operador de incremento e Operador de decremento

Arquivo: OpMat.java

```
/**
 * Testar operadores matemáticos.
 * @author Roberto Pinheiro
 */

public class OpMat {
    public static void main (String[] args) {

        int x1 = 7 + 3;
        int x2 = 7 % 2;
        String y = "Oi " + "Programador Java";

        System.out.println(x1);
        System.out.println(x2);
        System.out.println(y);

        int a = 6; // a = 6
        int b = 4; // b = 4
        int c = ++a; // pre-incremento: a = a + 1 e b = a
        // (primeiro incrementa "a" e depois atribui o resultado a "c")
        int d = b++; // pos-incremento: d = b e b = b + 1
        // (atribui a "d" o valor de "b" e depois incrementa "b")

        System.out.println("\na = " + a); // a = 7
        System.out.println("c = " + c); // c = 7
        System.out.println("b = " + b); // b = 5
        System.out.println("d = " + d); // d = 4

        int e = --a; // pre-decremento: a = 6 ; e = 6
        int f = b--; // pos-decremento: f = 5 ; b = 4

        System.out.println("\na = " + a); // a = 6
        System.out.println("e = " + e); // e = 6
        System.out.println("b = " + b); // b = 4
        System.out.println("f = " + f); // d = 5

    }
}
```



run:

10

1

Oi Programador Java

a = 7

c = 7

b = 5

d = 4

a = 6

e = 6

b = 4

f = 5

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

12. OPERADORES DE COMPARAÇÃO

O operador instanceof só é válido para objetos.

Arquivo: OpComp.java

```
/**
 * Testar operadores de comparação.
 * @author Roberto Pinheiro
 */

public class OpComp {

    public static void main (String[] args) {

        int x = 6;

        System.out.println("\nx igual a 6:");
        System.out.println(x == 6);

        System.out.println("\nx igual a 7:");
        System.out.println(x == 7);

        System.out.println("\nx diferente de 6:");
        System.out.println(x != 6);

        System.out.println("\nx diferente de 7:");
        System.out.println(x != 7);

        System.out.println("\nx maior que 7:");
        System.out.println(x > 7);

        System.out.println("\nx menor que 7:");
        System.out.println(x < 7);

        System.out.println("\nx maior ou igual a 6:");
        System.out.println(x >= 6);

        Integer y = 6;

        System.out.println("\ny é uma instancia da classe Integer?");
        System.out.println(y instanceof Integer);

        String nome = "Roberto";
        System.out.println("\nA variavel nome contem uma String?");
        System.out.println(nome instanceof String);

    }
}
```



run:

x igual a 6:
true

x igual a 7:
false

x diferente de 6:
false

x diferente de 7:
true

x maior que 7:
false

x menor que 7:
true

x maior ou igual a 6:
true

y é uma instancia da classe Integer?
true

A variavel nome contem uma String?
true

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

13. OPERADORES LÓGICOS

Arquivo: OpLog.java

```
/**
 * Testar operadores lógicos.
 * @author Roberto Pinheiro
 */

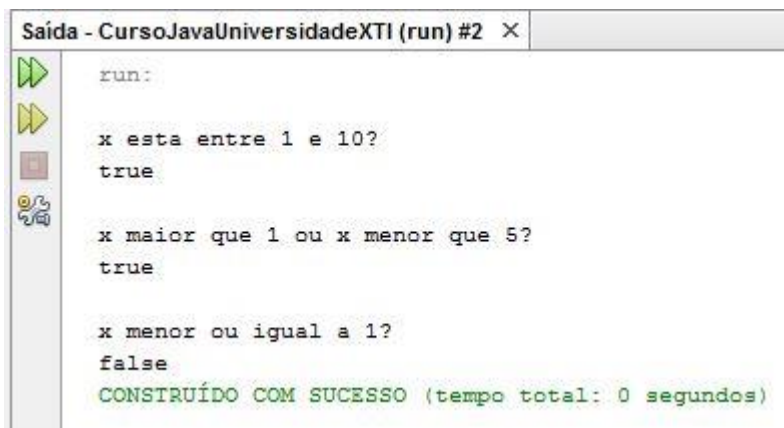
public class OpLog {
    public static void main (String[] args) {

        int x = 6;

        System.out.println("\nx esta entre 1 e 10?");
        System.out.println((x >=1) && (x<=10));

        System.out.println("\nx maior que 1 ou x menor que 5?");
        System.out.println((x >=1) || (x<=5));

        System.out.println("\nx menor ou igual a 1?");
        System.out.println(!(x >=1));
    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #2 X
run:
x esta entre 1 e 10?
true
x maior que 1 ou x menor que 5?
true
x menor ou igual a 1?
false
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

14. OPERADORES DE ATRIBUIÇÃO

Arquivo: OpAtrib.java

```
/**
 * Testar operadores de atribuição.
 * @author Roberto Pinheiro
 */

public class OpAtrib {
    public static void main (String[] args) {
        int x = 6;
        x += 3; // equivale a x = x + 3

        int y = 6;
        y -= 3; // equivale a y = y - 3

        int z = 6;
        z *= 3; // equivale a z = z * 3

        int w = 6;
        w /= 3; // equivale a w = w / 3

        int a = 6;
        a %= 3; // equivale a a = a % 3

        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        System.out.println(w);
        System.out.println(a);
    }
}
```



Saída - CursoJavaUniversidadeXTI (run) #2 X

```
run:
9
3
18
2
0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

15. OPERADORES ESPECIAIS

Arquivo: OpEsp.java

```
/**
 * Testar operadores de atribuição.
 * @author Roberto Pinheiro
 */

import java.util.Scanner;

public class OpEsp {

    public static void main (String[] args) {

        int idade = 6;
        String x = (idade >= 18) ? "Maior de idade" : "Menor de Idade";
        System.out.println("\n" + x);

        String sexo = "M", pais = "Brasil";
        System.out.println("\n" + sexo);
        System.out.println(pais);

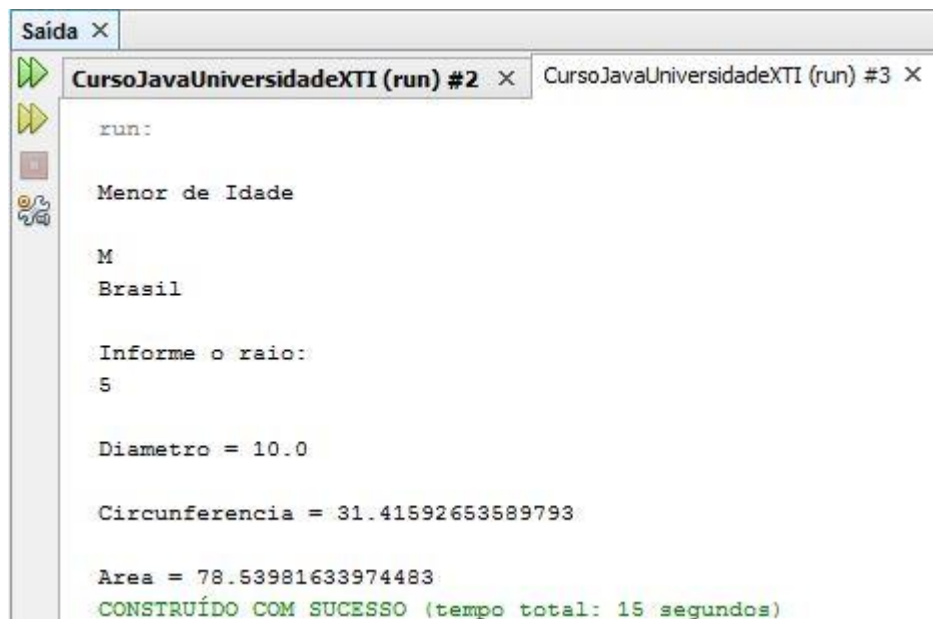
        Scanner s = new Scanner(System.in);
        System.out.println("\nInforme o raio:");
        double raio = s.nextDouble();

        // Diametro = 2r
        // double raio = 10;
        double diametro = 2 * raio;
        System.out.println("\nDiametro = " + diametro);

        // Circunferencia = 2 PI r
        double pi = Math.PI;
        double circunferencia = 2 * pi * raio;
        System.out.println("\nCircunferencia = " + circunferencia);

        // Area = PI r2
        double area = pi * (raio * raio);
        System.out.println("\nArea = " + area);

    }
}
```



```
Saída X
CursoJavaUniversidadeXTI (run) #2 X  CursoJavaUniversidadeXTI (run) #3 X

run:
Menor de Idade
M
Brasil
Informe o raio:
5
Diametro = 10.0
Circunferencia = 31.41592653589793
Area = 78.53981633974483
CONSTRUÍDO COM SUCESSO (tempo total: 15 segundos)
```

16. DESAFIO DO CÁLCULO DO IMC

Arquivo: IMC.java

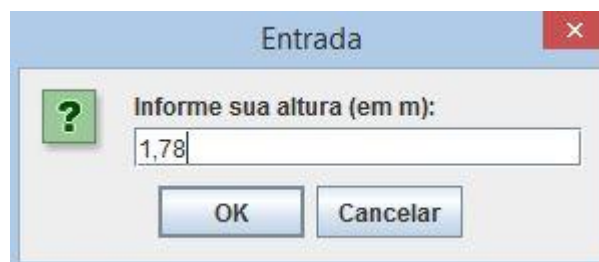
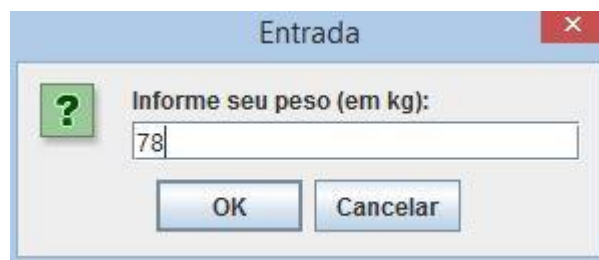
```
/**
 * Calcular o IMC (índice de massa corporal)
 *  $IMC = \text{pesoEmQuilogramas} / (\text{alturaEmMetros} * \text{alturaEmMetros})$ 
 * @author Roberto Pinheiro
 */

import javax.swing.JOptionPane;

public class IMC {
    public static void main (String[] args) {

        String peso = JOptionPane.showInputDialog("Informe seu peso (em kg):");
        String altura = JOptionPane.showInputDialog("Informe sua altura (em m):");
        double pesoEmQuilogramas = Double.parseDouble(peso);
        double alturaEmMetros = Double.parseDouble(altura);
        double imc = pesoEmQuilogramas / (alturaEmMetros * alturaEmMetros);
        String msg = (imc >= 20 && imc <= 25) ? "Peso ideal" : "Fora do peso ideal";
        msg = "IMC = " + imc + "\n" + msg;
        JOptionPane.showMessageDialog(null, msg);

    }
}
```



17. ARRAY

Arquivo: ArraySimples.java

```
/**
 * Apresentar conceitos de Array.
 * @author Roberto Pinheiro
 */

import java.util.Arrays;

public class ArraySimples {
    public static void main (String[] args) {

        int[] impares = new int[5];
        impares[0] = 1;
        impares[1] = 3;
        impares[2] = 5;
        impares[3] = 7;
        impares[4] = 9;

        String [] paises = {"Brasil", "Russia", "India", "China"};
        paises[0] = "Argentina";

        System.out.println("\n" + paises[0]);
        System.out.println("\n" + paises.length);
        System.out.println("\n" + Arrays.toString(paises));

        int posicao = Arrays.binarySearch(paises, "Russia");
        System.out.println("\n" + posicao);
        Arrays.sort(paises, 0, paises.length);
        System.out.println("\n" + Arrays.toString(paises));

        Double [] valores = {12.35, 3456.3456};
        System.out.println("\n" + valores[0].doubleValue());
    }
}
```



Saida - CursoJavaUniversidadeXTI (run) #3 X

```
run:
Argentina
4
[Argentina, Russia, India, China]
1
[Argentina, China, India, Russia]
12.35
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

18. ARRAY MULTIDIMENSIONAL

Arquivo: ArrayMult.java

```
/**
 * Apresentar conceitos de Array Multidimensional.
 * @author Roberto Pinheiro
 */

import java.util.Random;

public class ArrayMult {

    public static void main (String[] args) {

        String [][] duas = {"Ricardo", "M", "DF"}, {"Sandra","F", "MG"};

        System.out.println("\n" + duas [1][0]);
        System.out.println(duas.length);
        System.out.println(duas[0].length);

        duas[1][0] = "Denise";
        System.out.println("\n" + duas[1][0]);

        String [] faces =
        {"As","2","3","4","5","6","7","8","9","10","Valete","Dama","Rei"};

        String [] nipes = {"Espadas", "Paus", "Copas", "Ouros"};

        Random r = new Random();

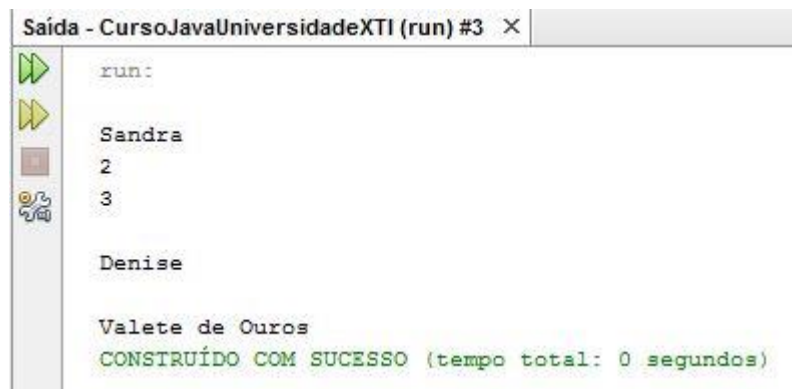
        int indiceFace = r.nextInt(faces.length);
        String face = faces[indiceFace];

        int indiceNipe = r.nextInt(nipes.length);
        String nipe = nipes[indiceNipe];

        String carta = face + " de " + nipe;
        System.out.println("\n" + carta);

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
Sandra
2
3
Denise
Valete de Ouros
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

19. ARRAYLIST

As estruturas de array citadas anteriormente possuem algumas limitações no que se refere à inclusão ou remoção de elementos dentro dessa lista. Para resolver esse problema foi criado o framework chamado Collection Framework. O ArrayList permite adicionar, remover, pesquisar e recuperar elementos dessa lista.

Arquivo: UsoArrayList.java

```
/**
 * Usar a classe ArrayList para criacao de listas.
 * @author Roberto Pinheiro
 */

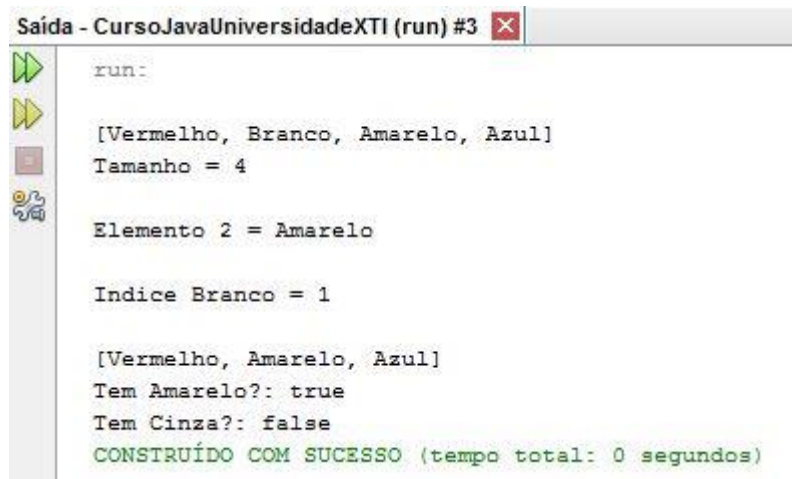
import java.util.ArrayList;

public class UsoArrayList {

    public static void main (String[] args) {
        ArrayList<String> cores = new ArrayList<String>();
        cores.add("Branco");
        // cores.add("Vermelho");
        cores.add(0, "Vermelho");
        cores.add("Amarelo");
        cores.add("Azul");
        System.out.println("\n" + cores.toString());
        System.out.println("Tamanho = " + cores.size());
        System.out.println("\nElemento 2 = " + cores.get(2));
        System.out.println("\nIndice Branco = " + cores.indexOf("Branco"));

        cores.remove("Branco");

        System.out.println("\n" + cores.toString());
        System.out.println("Tem Amarelo?: " + cores.contains("Amarelo"));
        System.out.println("Tem Cinza?: " + cores.contains("Cinza"));
    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:

[Vermelho, Branco, Amarelo, Azul]
Tamanho = 4

Elemento 2 = Amarelo

Indice Branco = 1

[Vermelho, Amarelo, Azul]
Tem Amarelo?: true
Tem Cinza?: false
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

20. IF ELSE

Arquivo: Fluxo.java

```
public class Fluxo {
    public static void main(String[] args){

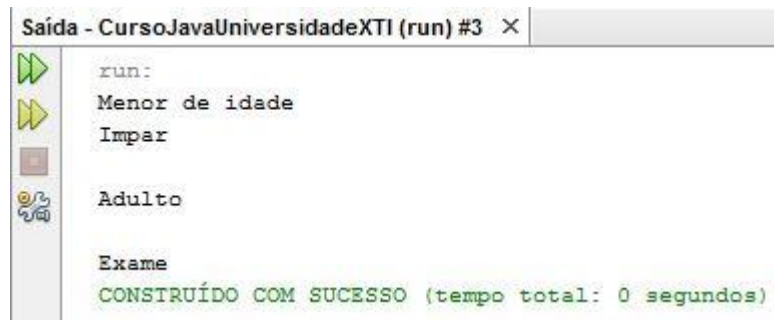
        int idade = 10;
        if (idade < 18) {
            System.out.println("Menor de idade");
        }

        int numero = 11;
        if((numero % 2) == 0){
            System.out.println("Par");
        } else {
            System.out.println("Impar");
        }

        idade = 51;
        if(idade <= 10){
            System.out.println("\nCrianca");
        } else if (idade > 10 && idade <=18){
            System.out.println("\nAdolescente");
        } else if (idade > 18 && idade <=60){
            System.out.println("\nAdulto");
        } else {
            System.out.println("\nIdoso");
        }

        double nota = 5.5;
        if(nota >= 7){
            System.out.println("\nAprovado");
        } else {
            if (nota >= 5 && nota < 7){
                System.out.println("\nExame");
            } else {
                System.out.println("\nReprovado");
            }
        }

    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
Menor de idade
Impar
Adulto
Exame
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

21. SWITCH

Arquivo: Fluxo2.java

```
public class Fluxo2 {  
  
    public static void main(String[] args){  
  
        char sexo = 'M';  
        switch(sexo) {  
            case 'M':  
                System.out.println("\nMasculino");  
                break;  
            case 'F':  
                System.out.println("\nFeminino");  
                break;  
            default:  
                System.out.println("\nOutro");  
        }  
  
        String tecnologia = "sqlserver";  
        switch(tecnologia) {  
            case "java":  
            case "c++":  
            case "cobol":  
                System.out.println("\nLinguagem de Programacao");  
                break;  
            case "oracle":  
            case "sqlserver":  
            case "postgresql":  
                System.out.println("\nBanco de dados");  
                break;  
            default:  
                System.out.println("\nTecnologia desconhecida");  
        }  
    }  
}
```



22. DESAFIO DOS DADOS

Arquivo: JogoDados.java

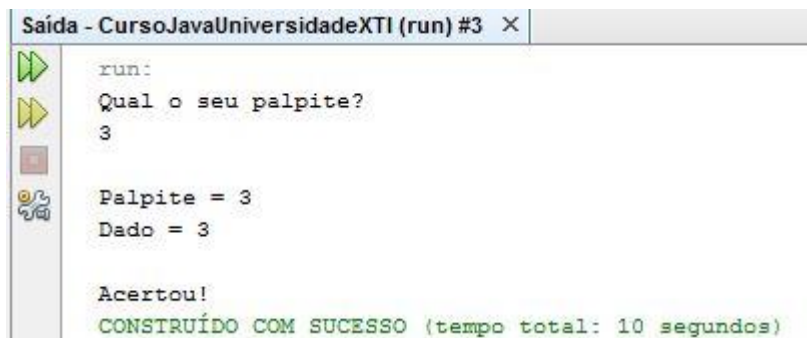
```
import java.util.Scanner;
import java.util.Random;

public class JogoDados {

    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.println("Qual o seu palpite?");
        int palpite = s.nextInt();
        Random n = new Random();
        int dado = n.nextInt(6) + 1;

        System.out.println("\nPalpite = " + palpite);
        System.out.println("Dado = " + dado);

        if(palpite == dado) {
            System.out.println("\nAcertou!");
        } else {
            System.out.println("\nErrou!");
        }
    }
}
```

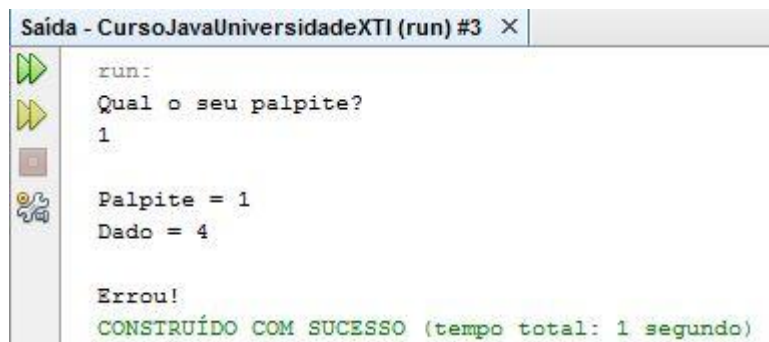


Saída - CursoJavaUniversidadeXTI (run) #3 X

```
run:
Qual o seu palpite?
3

Palpite = 3
Dado = 3

Acertou!
CONSTRUÍDO COM SUCESSO (tempo total: 10 segundos)
```



Saída - CursoJavaUniversidadeXTI (run) #3 X

```
run:
Qual o seu palpite?
1

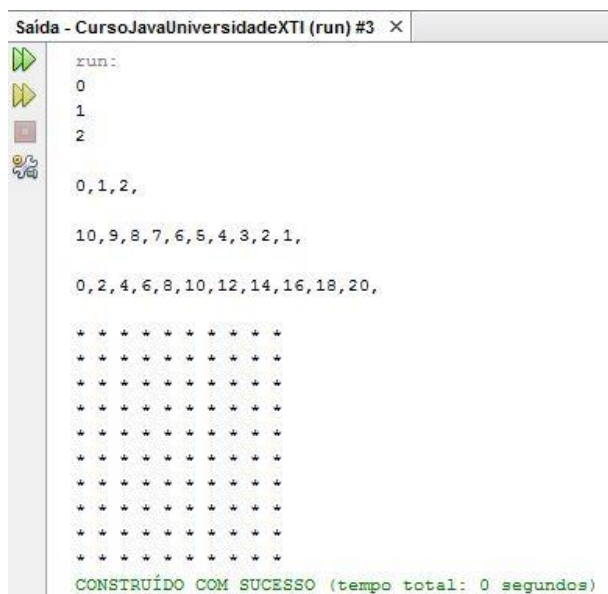
Palpite = 1
Dado = 4

Errou!
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

23. FOR

Arquivo: FluxoRepet.java

```
public class FluxoRepet {  
  
    public static void main(String[] args) {  
  
        for(int i=0; i<3; i++) {  
            System.out.println(i);  
        }  
  
        String texto = "";  
        for(int i=0; i<3; i++) {  
            texto += i + ",";  
        }  
        System.out.println("\n" + texto);  
  
        String texto2 = "";  
        for(int i=10; i>0; i--) {  
            texto2 += i + ",";  
        }  
        System.out.println("\n" + texto2);  
  
        String texto3 = "";  
        for(int i=0; i<=20; i++) {  
            if(i % 2 == 0){  
                texto3 += i + ",";  
            }  
        }  
        System.out.println("\n" + texto3);  
  
        System.out.println();  
        int tamanho = 10;  
        for(int x=0; x<tamanho; x++) {  
            for(int i=0; i<tamanho; i++){  
                System.out.print("* ");  
            }  
            System.out.println();  
        }  
    }  
}
```



```
run:  
0  
1  
2  
  
0,1,2,  
  
10,9,8,7,6,5,4,3,2,1,  
  
0,2,4,6,8,10,12,14,16,18,20,  
  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

23. FOREACH

Arquivo: VarrerArray.java

```
import java.util.ArrayList;

public class VarrerArray {

    public static void main(String[] args){

        /* foreach */
        int[] pares = {2,4,6,8};
        for(int i=0; i<pares.length; i++){
            int par = pares[i];
            System.out.println(par);
        }

        System.out.println();

        // for aprimorado
        /* so percorre os elementos para a frente. */
        for(int par : pares){
            System.out.println(par);
        }

        System.out.println();

        /* foreach */
        ArrayList<Integer> list = new ArrayList<Integer>();
        for(int i=0; i<10; i++){
            list.add(i);
        }
        for(Integer numero : list){
            System.out.println(numero);
        }

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
0
1
2
3
4
5
6
7
8
9
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


24. WHILE, DO WHILE

Arquivo: FluxoRepet2.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class FluxoRepet2{

    public static void main(String[] args){

        int i = 0;
        while(i<=10){
            System.out.println(i);
            i++;
        };

        System.out.println();

        i = 11;
        do {
            System.out.println(i);
            i++;
        } while (i<=10);

        System.out.println();

        ArrayList<String> produtos = new ArrayList<String>();
        Scanner s = new Scanner(System.in);
        System.out.println("Liste seus produtos - Para sair digite FIM");

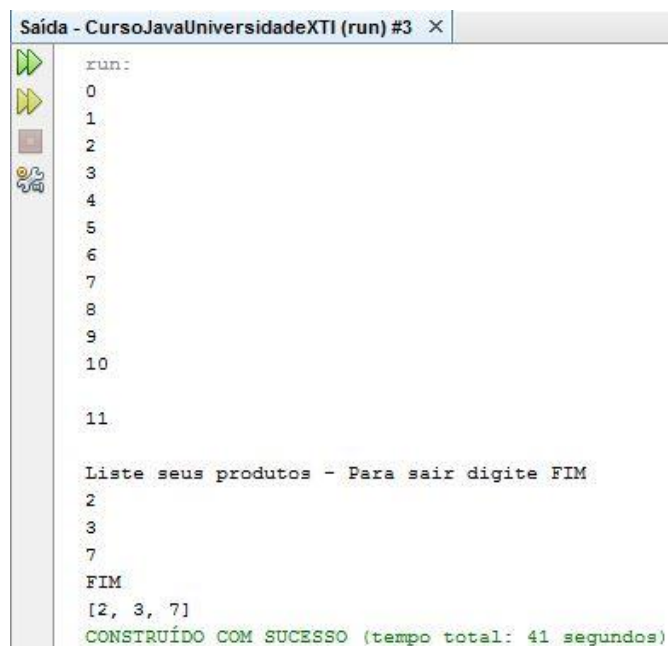
        String produto;

        while (!"FIM".equals(produto = s.nextLine())) {
            produtos.add(produto);
        }

        System.out.println(produtos.toString());

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 x
run:
0
1
2
3
4
5
6
7
8
9
10

11

Liste seus produtos - Para sair digite FIM
2
3
7
FIM
[2, 3, 7]
CONSTRUÍDO COM SUCESSO (tempo total: 41 segundos)
```

25. FIBONACCI

Arquivo: Fibonacci.java

```
/* DESAFIO: Fibonacci
 * Começa-se a serie com 0 (zero) e 1 (um)
 * Obtem-se o proximo numero de Fibonacci
 * somando-se os dois anteriores e, assim
 * sucessiva e infinitamente.
 *
 * Ex: 1+2[3] 2+3[5] 3+5[8] 5+8[13] ...
 * 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,
 * 144, 233, 377, 610, 987, 1597, 2584, ...
 */

public class Fibonacci{

    public static void main(String[] args){

        int anterior = 0;
        int proximo = 1;

        System.out.println(anterior);

        while(proximo < 3000){
            System.out.println(proximo);
            proximo = anterior + proximo; // proximo numero Fibonacci
            anterior = proximo - anterior; // atualizando o numero anterior
        }

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

26. LABEL, BREAK E CONTINUE

As palavras reservadas Break e Continue são utilizadas para encerrar loops. O break encerra um bloco inteiro e o Continue encerra apenas a interação atual.

Os rótulos servem para indicar em qual looping se quer continuar ou parar.

Arquivo: EstrutAux.java

```
public class EstrutAux {

    public static void main(String[] args){

        while(true) {
            System.out.println("Entrou");
            break;
        }

        System.out.println();

        for(int i=0; i<10; i++){
            if(i==5){
                break;
            }
            System.out.println(i);
        }

        System.out.println();

        for(int i=0; i<10; i++){
            if(i==5){
                continue;
            }
            System.out.println(i);
        }

        System.out.println();

        boolean[][] matrix =
        {
            {false, true, false, false, false},
            {false, false, false, false, false}
        };

        busca:
        for(int a=0; a < matrix.length; a++){
            System.out.print("A ");
            for(int b=0; b < matrix[a].length; b++){
                if(matrix[a][b]){
                    System.out.print("TRUE ");
                    break busca;
                }
                System.out.print("B ");
            }
        }

    }

}
```

```
Saida - CursoJavaUniversidadeXTI (run) #3 X
run:
Entrou
0
1
2
3
4

0
1
2
3
4
6
7
8
9

A B TRUE CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

26. AUTOBOXING

Arquivo: AutoBoxing.java

```
public class AutoBoxing{

    public static void main(String[] args){

        Integer x = new Integer(555); // empacotado
        int a = x.intValue(); // desempacotar
        a++; // incrementa
        x = new Integer (a); // re-empacotar
        System.out.println(x.intValue());

        System.out.println();

        // recursos inseridos a partir da versao 5 do Java SE

        Integer y = 555;
        y++; // desempacota, incrementa e reempacota
        System.out.println(y);

        System.out.println();

        Boolean v = new Boolean("true");
        if(v){
            System.out.println(v);
        }
    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
556
556
true
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

27. IDE ECLIPSE - INSTALAÇÃO

IDE (Integrated Development Environment) = ambiente de desenvolvimento integrado.

Usar a IDE permite aumentar a produtividade e a qualidade do código desenvolvido.

Principais IDE's:

- Eclipse (IDE mais utilizada do mercado) → baixar em: www.eclipse.org
- NetBeans → baixar em www.netbeans.org

Outras:

- BlueG
- jGrasp

Baixar Eclipse Classic 3.7 (Windows 32 bits).

Descompactar em um diretório de livre (C:) escolha e a instalação estará feita.
(C:\Eclipse)

Criar um atalho no desktop.

Configurações do Eclipse → C:\Users\Z\workspace (usar o padrão).

28. IDE ECLIPSE, ATALHOS

Atalhos do Eclipse:

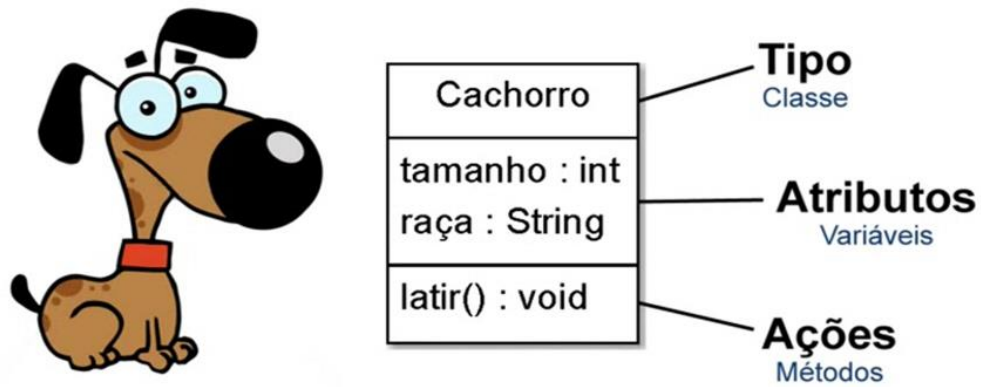
Ctrl + 1	Correção de erros
Ctrl + Espaço	Completa códigos
Ctrl + Shift + F	Formata o código
Ctrl + O	Navegação rápida
Ctrl + Shift + O	Importação automática

sysout + Ctrl espaço → System.out.println();

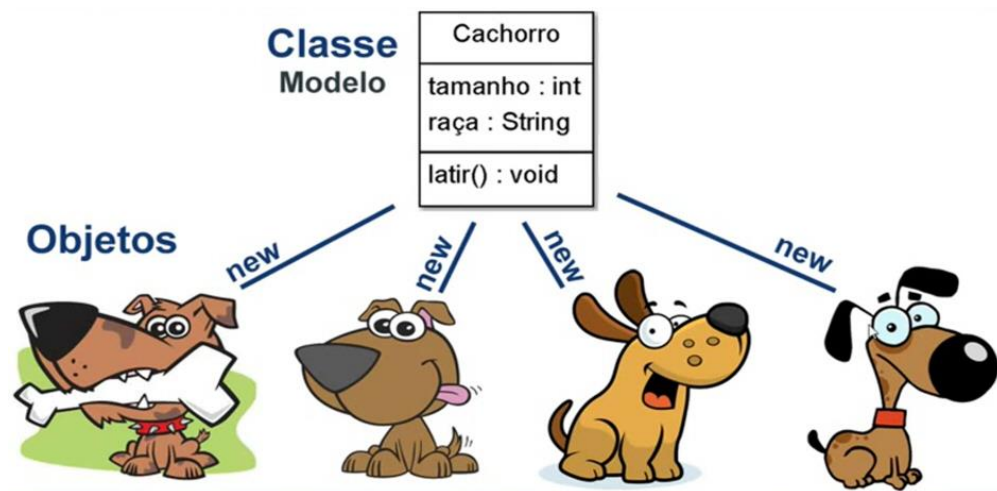
Ctrl + Shift + F → formata o código de acordo com as normas de formatação da Sun.

Ctrl + O → leva rapidamente ao código da classe ou a um dos métodos do código.

29. ORIENTAÇÃO A OBJETOS (OO)



- Atributos são variáveis da classe.
- A classe não é um objeto. A classe é um projeto de um objeto.
- Os objetos criados a partir da classe também são chamados de instâncias de objeto.
- Um programa orientado a objetos pressupõe a construção de objetos que conversam entre si (que trocam mensagens entre si).



30. PACOTES

No Eclipse:

Quando as nossas classes não pertencem a nenhum pacote, então elas estão dentro do pacote default (default package). E dentro de JRE há a biblioteca java.

rt.jar → onde encontramos todos os pacotes java.

java.lang (linguagem java) → é o pacote da linguagem de programação java. Dentro dele vamos encontrar todas as classes que podemos utilizar dentro da nossa classe java sem precisar importar um pacote.

Um pacote é uma estrutura de diretórios aonde se coloca todas as classes por um determinado assunto. Um pacote é um agrupamento de classes organizado por uma determinada categoria. É a forma que se tem de modularizar o seu programa.

Os pacotes ajudam a evitar conflitos de nomes.

Principais motivos para se utilizar pacotes;

- organizar o programa em módulos.
- evitar conflitos de nome com classes de outros programas .

Ex:

empresa: xti
domínio na Internet: www.xti.com.br

Todas as classes criadas para essa empresa deverão iniciar com o domínio dela invertido (ao contrário), ou seja, os pacotes deverão ser criados com o nome:

br.com.xti... (continua de acordo com seus projetos, em módulos).

30.1. Criação de um pacote dentro do Eclipse

Clicar com o botão direito dentro do projeto e selecionar:

New → Package

Source folder: xti
Name: br.com.xti.poo

Mover as duas classes (Cachorro.java e Cachorro.Test) para dentro do pacote criado, arrastando-as.

Todas as classes que estão dentro de um pacote iniciam com a palavra-chave "package" e o endereço daquele pacote a qual ela pertence. Uma classe só pode pertencer a um único pacote e a declaração do pacote deve vir no início da classe, antes de qualquer outra coisa.

Para importar todas as classes de um pacote, usar "*":

```
import br.com.xti.poo.*;
```

Por default sempre que se cria uma classe, o compilador já coloca nos byte codes da nossa classe a seguinte declaração:

```
import java.lang.*;
```

significando que todas as classes do pacote java.lang devem ser importadas para a classe. Isso é feito de forma automática (não é necessário compilar). É uma ação padrão do compilador.

Todas as classes devem pertencer a um pacote. Não é uma boa organização ter as classes fora de pacotes.

31. MÉTODOS COM PARÂMETROS

Arquivo: Conta.java

```
package br.com.xti.poo;

public class Conta {

    String cliente;
    double saldo;

    void exibeSaldo(){
        System.out.println(cliente + " seu saldo é " + saldo);
    }

    void saca(double valor){
        saldo -= valor; // saldo = saldo - valor;
    }

    void deposita(double valor){
        saldo += valor; // saldo = saldo + valor;
    }

    void transferePara(Conta destino, double valor){
        this.saca(valor);
        destino.deposita(valor);
    }

}
```

Arquivo: ContaTest.java

```
package br.com.xti.poo;

public class ContaTest {

    /**
     * @param args
     */
    public static void main(String[] args) {

        Conta conta = new Conta();
        conta.cliente = "Ricardo";
        conta.saldo = 10000.00;
        conta.exibeSaldo();

        Conta destino = new Conta();
        destino.cliente = "Lawrence";
        destino.saldo = 80000.00;
        destino.exibeSaldo();

        conta.saca(1000);
        conta.exibeSaldo();
        conta.deposita(6000);
        conta.exibeSaldo();

        conta.transferePara(destino, 1550.00);
        conta.exibeSaldo();
        destino.exibeSaldo();

    }

}
```

Saída - CursoJavaUniversidadeXTI (run) #3 X



```
run:
Ricardo seu saldo é 10000.0
Lawrence seu saldo é 80000.0
Ricardo seu saldo é 9000.0
Ricardo seu saldo é 15000.0
Ricardo seu saldo é 13450.0
Lawrence seu saldo é 81550.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

32. MÉTODOS COM RETORNO

Arquivo: Matematica.java

```
package br.com.xti.poo;

public class Matematica {

    double soma(double n1, double n2){
        return n1 + n2;
    }

}
```

Arquivo: MatematicaTest.java

```
package br.com.xti.poo;

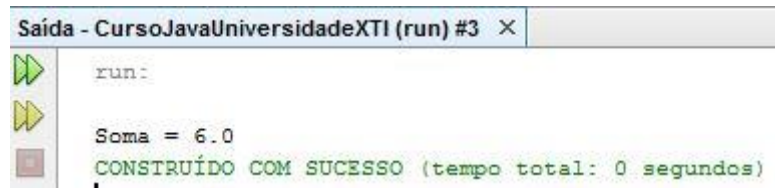
public class MatematicaTest {

    public static void main(String[] args) {

        Matematica m = new Matematica();

        double so = m.soma (2,4);
        System.out.println("\nSoma = " + so);
    }

}
```



Saida - CursoJavaUniversidadeXTI (run) #3 X

run:

Soma = 6.0

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

33. DESAFIO RAIZ QUADRADA

Equação de Pell's → É um método para obter a raiz quadrada simplesmente subtraindo números ímpares.

Exemplo: Para se obter a raiz quadrada de 27 fazemos:

1. $27 - 1 = 26$
2. $26 - 3 = 23$
3. $23 - 5 = 18$
4. $18 - 7 = 11$
5. $11 - 9 = 2$

5 passos foram tomados, o que nos permite deduzir que 5 é a parte inteira da raiz quadrada de 27.

Arquivo: Matematica2.java

```
package br.com.xti.poo;

public class Matematica2 {

    /** retorna a raiz quadrada do numero segundo a equacao de Pell */
    int raiz(int numero){
        int raiz = 0, impar = 1;
        while (numero >= impar){
            numero -= impar;
            impar += 2; // proximo numero impar
            ++raiz;
        }
        return raiz;
    }
}
```

Arquivo: Matematica2Test.java

```
package br.com.xti.poo;

public class Matematica2Test {

    public static void main(String[] args) {

        Matematica2 m = new Matematica2();

        int raiz = m.raiz(27);
        System.out.println("\n" + raiz);

        raiz = m.raiz(276);
        System.out.println("\n" + raiz);

        System.out.println(Math.sqrt(27));
        System.out.println(Math.sqrt(276));

    }
}
```

Saída - CursoJavaUniversidadeXTI (run) #3 X



run:



5



16

5.196152422706632

16.61324772583615

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

34. MODIFICADORES DE ACESSO

Principais modificadores de acesso:

- **private** → restringe o acesso a variável, classe e método apenas a membros dentro da própria classe. A variável só fica acessível dentro da classe em que foi criada.
- **package** <nenhum> → o modificador package é simbolizado pela ausência de modificadores. Este modificador é acessível a todos os membros dentro do mesmo pacote dessa classe, bem como dentro da própria classe.

OBS.: Independente do modificador, todos os elementos de uma mesma classe conseguem se enxergar.

- **protected** → é utilizado para restringir o acesso aos filhos de uma classe, ou seja, suas subclasses.
- **public** → ele indica que aquela variável, aquele método ou aquela classe são acessíveis em qualquer lugar por qualquer membro.

As variáveis, métodos ou classes com modificador private são representadas por um ícone quadrado e vermelho dentro do Eclipse para indicar essa restrição.

Variáveis, métodos e classes sem um modificador são acessíveis por todos os membros que estão dentro do mesmo pacote. A ausência de um modificador de acesso vai deixar visível essa variável para todos os membros dentro do mesmo pacote, mas continua inacessível para uma classe fora daquele pacote.

A ausência de um modificador de acesso é apresentada por um triângulo azul no Eclipse.

O modificador de acesso protected é usado para dar acesso dentro da classe ou para os filhos dessa classe.

O modificador public deixa a variável acessível a todos os membros. Nesse caso a variável é representada por um círculo verde no Eclipse.

O modificador private não costuma ser usado em classes.

35. ENCAPSULAMENTO

O encapsulamento protege as variáveis, de forma que não se possa configurá-las com um valor inapropriado.

Métodos & Variáveis

- GET / IS
Para captura

- SET
Para configuração

- Para variáveis booleanas o método de captura começa com "is".
- É uma boa prática proteger as variáveis da sua classe.

Arquivo: Funcionario.java

```
package br.com.xti.poo;

/* Teste dentro da classe Funcionario */

public class Funcionario {

    private String nome;
    private boolean ativo;

    public String getNome() {
        return nome;
    }
    public void setNome(String n) {
        this.nome = n;
    }
    public boolean isAtivo() {
        return ativo;
    }
    public void setAtivo(boolean ativo) {
        this.ativo = ativo;
    }

}
```

Arquivo: FuncionarioPacote.java

```
package br.com.xti.poo;

/* Teste dentro do mesmo pacote da classe Funcionario */

public class FuncionarioPacote {

    /**
     * @param args
     */
    public static void main(String[] args) {

        Funcionario f = new Funcionario();
        f.setNome("Wellington");
        System.out.println(f.getNome());

    }

}
```


36. VARIÁVEIS E MÉTODOS ESTÁTICOS

Arquivo: Galinha.java

```
package br.com.xti.poo;

public class Galinha {

    public static int ovosDaGranja; // variavel global
    public int ovos; // total de ovos do objeto galinha

    public Galinha botar() {
        this.ovos++;
        Galinha.ovosDaGranja++;
        return this;
    }

    public static double mediaDeOvos(int galinhas) {
        return (double) Galinha.ovosDaGranja / galinhas;
    }

}
```

Arquivo: GalinhaTest.java

```
package br.com.xti.poo;

public class GalinhaTest {

    public static void main(String[] args) {

        Galinha g1 = new Galinha();
        g1.botar().botar().botar();

        Galinha g2 = new Galinha();
        g2.botar().botar();

        System.out.println(g1.ovos);
        System.out.println(g2.ovos);
        System.out.println(Galinha.ovosDaGranja);
        System.out.println(Galinha.mediaDeOvos(2));
    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
3
2
5
2.5
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

ovosDaGranja não pertence a um objeto, pertence a uma classe (classe Galinha).

O método estático não pode acessar métodos e variáveis locais. Dentro de um método estático não se pode usar a palavra-chave "this", pois "this" representa uma instância de objeto e o método estático não conhece instâncias de objeto.

37. MÉTODOS COM ARGUMENTOS VARIÁVEIS

Arquivo: Matematica3.java

```
package br.com.xti.poo;

public class Matematica3 {

    double soma(String titulo, double ... numeros){
        System.out.println(titulo);
        double total = 0;
        for(double n : numeros){
            total += n;
        }
        return total;
    }
}
```

Arquivo: Matematica3Test.java

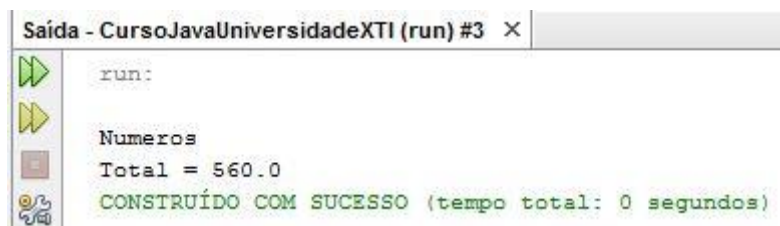
```
package br.com.xti.poo;

public class Matematica3Test {

    public static void main(String[] args) {

        Matematica3 m = new Matematica3();

        double total = m.soma("\nNumeros",2,3,5,6,7,123,345,69);
        System.out.println("Total = " + total);
    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
Numeros
Total = 560.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

38. MÉTODOS SOBRECARREGADOS

Métodos sobrecarregados tem o mesmo nome dentro de uma classe, porém com assinaturas diferentes.

Arquivo: Matematica4.java

```
package br.com.xti.poo;

public class Matematica4 {

    double soma(double ... numeros){
        // System.out.println(titulo);
        double total = 0;
        for(double n : numeros){
            total += n;
        }
        return total;
    }

    double media(double x, double y) {
        System.out.print("\nmedia(int x, int y): ");
        return (x + y) / 2;
    }

    double media(String x, String y) {
        System.out.print("media(String x, String y): ");
        int ix = Integer.parseInt(x);
        int iy = Integer.parseInt(y);
        return (ix + iy) / 2;
    }

    double media(double x, double y, double z) {
        System.out.print("media(int x, int y, int z): ");
        return (x + y + z) / 3;
    }

    double media(double ... numeros) {
        System.out.print("media(double ... numeros): ");
        return this.soma(numeros)/numeros.length;
    }

}
```

Arquivo: Matematica4Test.java

```
package br.com.xti.poo;

public class Matematica4Test {

    public static void main(String[] args) {

        Matematica4 m = new Matematica4();

        System.out.println(m.media(5,3));
        System.out.println(m.media("50","30"));
        System.out.println(m.media(56,34,29));
        System.out.println(m.media(56,34,76,44));

    }

}
```

Saida - CursoJavaUniversidadeXTI (run) #3 X



run:

```
media(int x, int y): 4.0  
media(String x, String y): 40.0  
media(int x, int y, int z): 39.666666666666664  
media(double ... numeros): 52.5  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

39. CONSTRUTORES

Os construtores são blocos de comandos que podem ser usados para inicializar objetos. Toda a classe tem construtores mesmo que não seja declarado explicitamente um construtor para ela (nesse caso o compilador cria um construtor para ela).

O construtor se parece muito com um método, mas não é porque ele não retorna um valor (não tem um tipo de retorno).

Todo construtor tem o mesmo nome da classe.

Arquivo: Carro.java

```
package br.com.xti.poo;

public class Carro {
    String modelo;
    int velMax;
    double seg0A100;

    // construtor vazio

    public Carro() {

    }

    public Carro(String modelo, int velMax, double seg0A100) {
        this.modelo = modelo;
        this.velMax = velMax;
        this.seg0A100 = seg0A100;
    }
}
```

Arquivo: CarroTest.java

```
package br.com.xti.poo;

public class CarroTest {

    public static void main(String[] args) {

        Carro ferrari = new Carro(); // Carro() --> construtor
        ferrari.modelo = "Ferrari Enzo";
        ferrari.velMax = 349;
        ferrari.seg0A100 = 3.2;

        Carro koenigsegg = new Carro("Koenigsegg CCXR", 430, 3.1);
        System.out.println(koenigsegg.modelo);
    }
}
```



Saída - CursoJavaUniversidadeXTI (run) #3 X

```
run:
Koenigsegg CCXR
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

40. COMPOSIÇÃO (RELACIONAMENTO TEM UM)

Arquivo: Motor.java

```
package br.com.xti.poo;

public class Motor {

    String tipo;
    int potencia;

    public Motor() {

    }

    public Motor(String tipo, int potencia) {
        this.tipo = tipo;
        this.potencia = potencia;
    }

}
```

Arquivo: Carro2.java

```
package br.com.xti.poo;

public class Carro2 {

    String modelo;
    int velMax;
    double seg0A100;
    Motor motor;

    // construtor vazio

    public Carro2() {

    }

    public Carro2(String modelo, int velMax, double seg0A100) {
        this.modelo, velMax, seg0A100, null);
    }

    public Carro2(String modelo, int velMax, double seg0A100, Motor motor) {
        this.modelo = modelo;
        this.velMax = velMax;
        this.seg0A100 = seg0A100;
        this.motor = motor;
    }

}
```

Arquivo: Carro2Test.java

```
package br.com.xti.poo;

public class Carro2Test {

    public static void main(String[] args) {

        Carro2 ferrari = new Carro2(); // Carro() --> construtor
        ferrari.modelo = "Ferrari Enzo";
        ferrari.velMax = 349;
        ferrari.seg0A100 = 3.2;

        Motor v12 = new Motor();
        v12.tipo = "V12";
        v12.potencia = 660;
        ferrari.motor = v12; // adiciona o motor ao carro

        Carro2 k = new Carro2("Koenigsegg CCXR", 430, 3.1);
        Motor v8 = new Motor("V8", 1018);
        k.motor = v8;

        Motor w16 = new Motor();
        w16.tipo = "W16";
        w16.potencia = 1200;

        Carro2 bugatti = new Carro2("Bugatti Veyron", 430, 2.2, w16);

        System.out.println(k.modelo);
        System.out.println(k.motor.potencia);
        System.out.println(bugatti.modelo);
        System.out.println(bugatti.motor.potencia);

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
Koenigsegg CCXR
1018
Bugatti Veyron
1200
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

41. ENUMERAÇÃO

Constantes enum → São as constantes profissionais da linguagem java.

As enumerações foram construídas para criação de constantes static final utilizadas dentro da aplicação. É uma forma profissional de criar conjunto de constantes.

Arquivo: PecasXadrez.java

```
package br.com.xti.poo;

public enum PecasXadrez {

    PEÃO, TORRE, BISPO, CAVALO, REI, RAINHA;

}
```

Arquivo: Medida.java

```
package br.com.xti.poo;

public enum Medida {

    MM("Milímetro"), CM("Centímetro"), M("Metro");

    public String titulo;

    Medida(String titulo) {
        this.titulo = titulo;
    }

}
```

Arquivo: EnumTest.java

```
package br.com.xti.poo;

public class EnumTest {

    public static final double PI = 3.14;

    public static void andar(Medida medida, int total){
        if(medida == Medida.M){
            //... codigo
        }
    }

    public static void main(String[] args) {

        System.out.println(PecasXadrez.BISPO);
        System.out.println(Medida.CM.titulo);

        for(Medida m : Medida.values()) {
            System.out.println(m + " : " + m.titulo);
        }

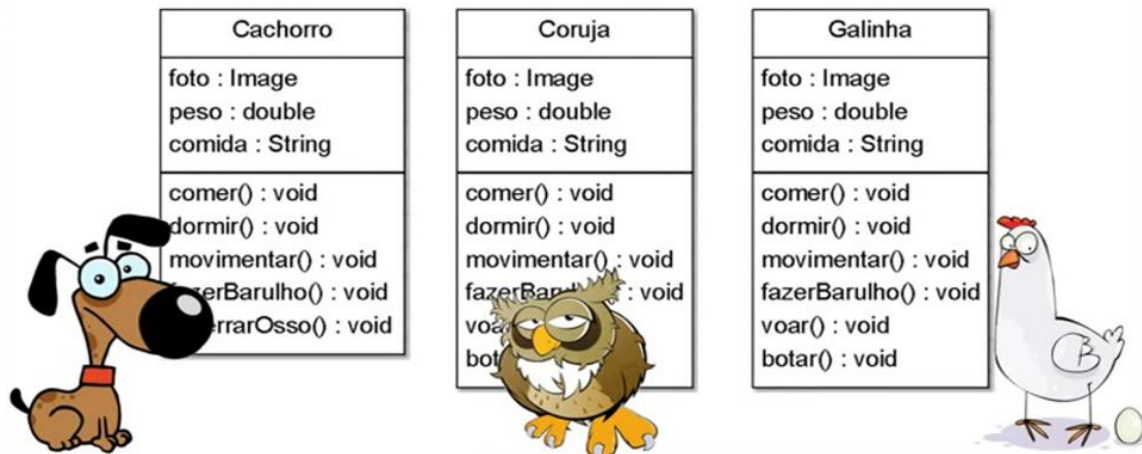
        andar(Medida.M, 100);

    }

}
```

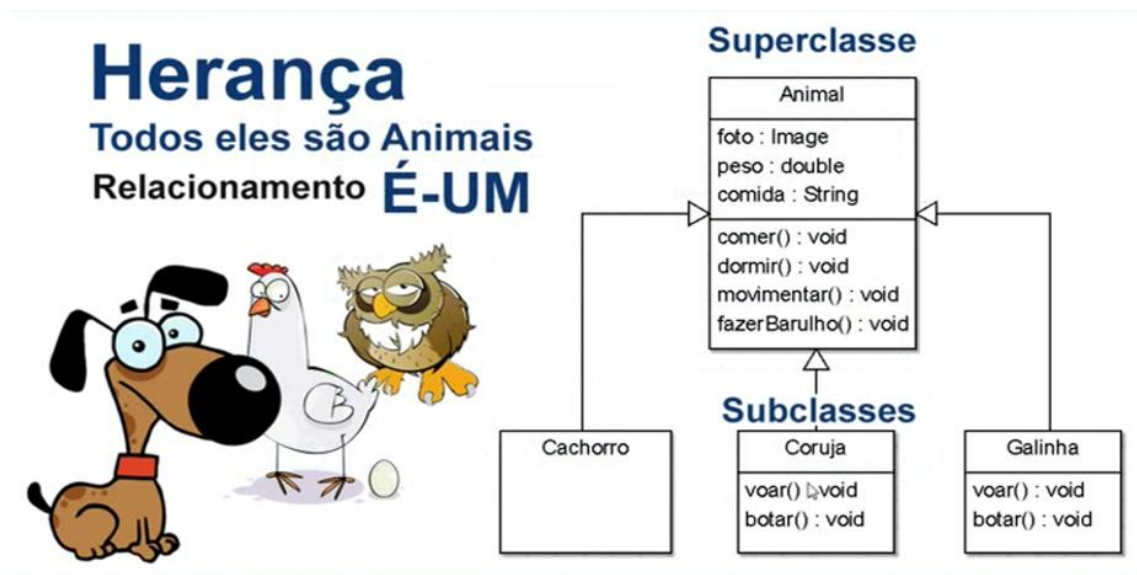

42. HERANÇA

Herança



A herança é um relacionamento "é um". A herança permite reaproveitar códigos. A herança dá a possibilidade de criar superclasses que recebam atributos e métodos semelhantes.

Com a palavra-reservada "extends" transforma-se uma classe em subclasse de uma outra classe. Quando uma classe herda de outra significa dizer que ela passa a ter todos os seus métodos e atributos.



Executamos sempre o método da classe mais próxima daquela que estamos utilizando.

Para identificar se uma classe deveria herdar de outra classe utiliza-se o teste do "é um".

Arquivo: Animal.java

```
package br.com.xti.heranca;
public class Animal {
    double peso;
    String comida;
    void dormir(){System.out.println("Dormiu");}
    void fazerBarulho(){System.out.println("Fazer Barulho");}
}
```

Arquivo: Cachorro.java

```
package br.com.xti.heranca;
public class Cachorro extends Animal {
}
```

Arquivo: Galinha.java

```
package br.com.xti.heranca;
public class Galinha extends Animal {
}
```

Arquivo: AnimalTest.java

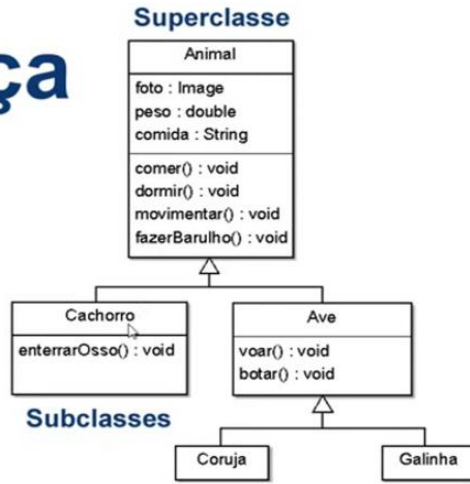
```
package br.com.xti.heranca;
public class AnimalTest {
    public static void main(String[] args) {
        Cachorro toto = new Cachorro();
        toto.comida = "Carne";
        toto.dormir();

        Galinha carijo = new Galinha();
        carijo.dormir();
    }
}
```



```
Saída - CursoJavaUniversidadeXTI (run) #3 X
run:
Dormiu
run:
Dormiu
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Herança



43. HERANÇA E CONSTRUTOR (SUPER)

A palavra chave "super" faz referência à superclasse, inclusive em questões de atributos e métodos.

Para que os atributos sejam vistos apenas pelas subclasses, usar o modificador protected.

Todas as classes são instâncias (extendem) da classe Object, mesmo que não se faça isso explicitamente. A classe Object tem alguns métodos que todos os outros objetos criados têm acesso.

Mesmo que não se utilize: extends Object, ainda assim o compilador irá colocar essa extensão no momento da compilação.

- obj1.equals(obj2); → compara dois objetos e verifica se os valores são iguais.

ex: toto.equals(carijo);

- obj.hashCode(); → gera uma espécie de número serial. É mais utilizado no armazenamento desse objeto dentro de coleções (como por exemplo um array list).

ex: toto.hashCode

- obj.getClass(); → retorna o tipo da classe desse objeto.

ex: toto.getClass(); // retorna a classe Cachorro.

- obj.toString(); → retorna a representação em string (texto) desse objeto.

ex: toto.toString();

Arquivo: Animal.java

```
package br.com.xti.heranca;

public class Animal {

    private int serial;
    double peso;
    String comida;

    public Animal(double peso, String comida){
        this.peso = peso;
        this.comida = comida;
    }

    void dormir(){System.out.println("Dormiu");}
    void fazerBarulho(){System.out.println("Fazer Barulho");}

}
```

Arquivo: Cachorro.java

```
package br.com.xti.heranca;

    public class Cachorro extends Animal {

        public Cachorro(){
            super(30, "Carne");
        }

    }
```

Arquivo: Galinha.java

```
package br.com.xti.heranca;

public class Galinha extends Animal {

    public Galinha() {
        super(2, "Milho");
    }

}
```

Arquivo: AnimalTest.java

```
package br.com.xti.heranca;

public class AnimalTest {

    public static void main(String[] args) {

        Cachorro toto = new Cachorro();
        toto.comida = "Carne";
        toto.dormir();

        Galinha cariyo = new Galinha();
        cariyo.dormir();

        System.out.println(toto instanceof Cachorro);
        System.out.println(toto instanceof Animal);

    }

}
```

Saída - CursoJavaUniversidadeXTI (run) #3 ×



run:



Dormiu



Dormiu



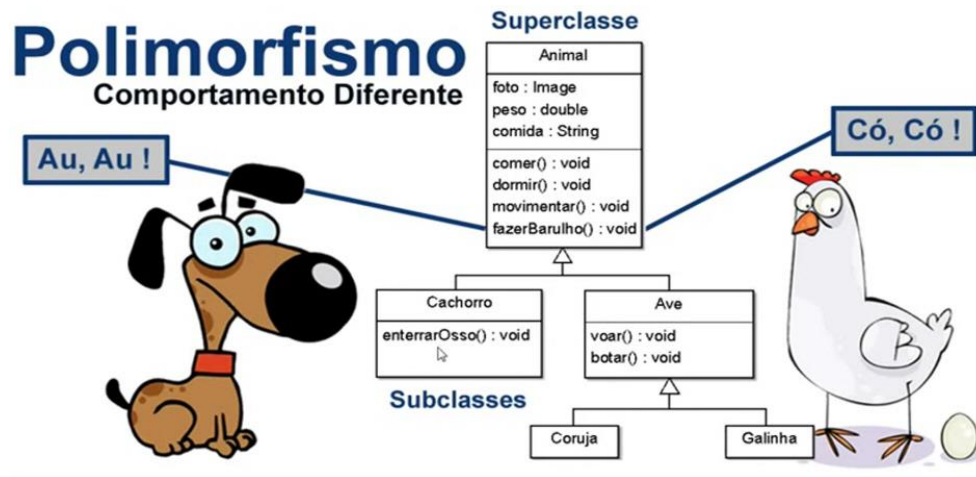
true



true

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

44. POLIMORFISMO - SOBRESCRITA DE MÉTODOS



A herança dá acesso ao comportamento padrão da sua superclasse. O polimorfismo dá a oportunidade de modificar o comportamento natural da herança. O polimorfismo permite sobrescrever métodos.

Arquivo: Animal.java

```
package br.com.xti.heranca;

public class Animal {
    private int serial;
    double peso;
    String comida;

    public Animal(double peso, String comida){
        this.peso = peso;
        this.comida = comida;
    }

    final void dormir(){
        System.out.println("Dormiu");
    }

    void fazerBarulho(){
        System.out.println("Fazer Barulho");
    }
}
```

Arquivo: Galinha.java

```
package br.com.xti.heranca;

public class Galinha extends Animal implements AreaCalculavel {
    public Galinha(){
        super(2, "Milho");
    }

    void fazerBarulho(){
        System.out.println("Có, Có !");
    }

    @Override
    public double calculaArea() {
        return 0;
    }
}
```

Arquivo: Cachorro.java

```
package br.com.xti.heranca;

public class Cachorro extends Animal {

    public Cachorro(){
        super(30, "Carne");
    }

    void fazerBarulho(){
        System.out.println("Au, Au !");
    }

}
```

Arquivo: AnimalTest.java

```
package br.com.xti.heranca;

public class AnimalTest {

    // uso de polimorfismo

    public static void barulho(Animal animal){
        animal.fazerBarulho();
    }

    public static void barulhoSemPolimorfismo(String animal){
        if(animal.equals("Cachorro")){
            System.out.println("Au Au !");
        } else if (animal.equals("Galinha")){
            System.out.println("CÓ, Có !");
        }
    }

    public static void main(String[] args) {

        Cachorro toto = new Cachorro();
        toto.comida = "Carne";
        toto.dormir();

        Galinha carijo = new Galinha();
        carijo.dormir();

        Animal generico = new Animal(0,null);

        System.out.println(toto instanceof Cachorro);
        System.out.println(toto instanceof Animal);

        toto.fazerBarulho();
        carijo.fazerBarulho();
        generico.fazerBarulho();

        System.out.println();

        barulho(toto);
        barulho(carijo);

        System.out.println();

        barulhoSemPolimorfismo("Cachorro");
        barulhoSemPolimorfismo("Galinha");

    }
}
```

Saída - CursoJavaUniversidadeXTI (run) X



```
run:
Dormiu
Dormiu
true
true
Au, Au !
CÓ, Có !
Fazer Barulho

Au, Au !
CÓ, Có !

Au Au !
CÓ, Có !
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


Arquivo: OperacaoMatematica.java

```
package br.com.xti.heranca;
public class OperacaoMatematica {
    public double calcular(double x, double y){
        return 0;
    }
}
```

Arquivo: Soma.java

```
package br.com.xti.heranca;

public class Soma extends OperacaoMatematica {
    public double calcular(double x, double y){
        return x + y;
    }
}
```

Arquivo: Multiplicacao.java

```
package br.com.xti.heranca;

public class Multiplicacao extends OperacaoMatematica{
    public double calcular(double x, double y){
        return x * y;
    }
}
```

Arquivo: OperacaoTest.java

```
package br.com.xti.heranca;

public class OperacaoTest {

    public static void calcule(OperacaoMatematica o, double x, double y){
        System.out.println(o.calcular(x,y));
    }

    public static void calculeSemPolimorfismo(String o, double x, double y){
        if(o.equals("Soma")){
            System.out.println(x + y);
        } else if (o.equals("Multiplicacao")) {
            System.out.println(x * y);
        }
    }

    public static void main(String[] args) {

        calcule(new Soma(), 5, 5);
        calcule(new Multiplicacao(), 5, 5);

        System.out.println();

        calculeSemPolimorfismo("Multiplicacao",4,8);

    }
}
```

Saída - CursoJavaUniversidadeXTI (run) X

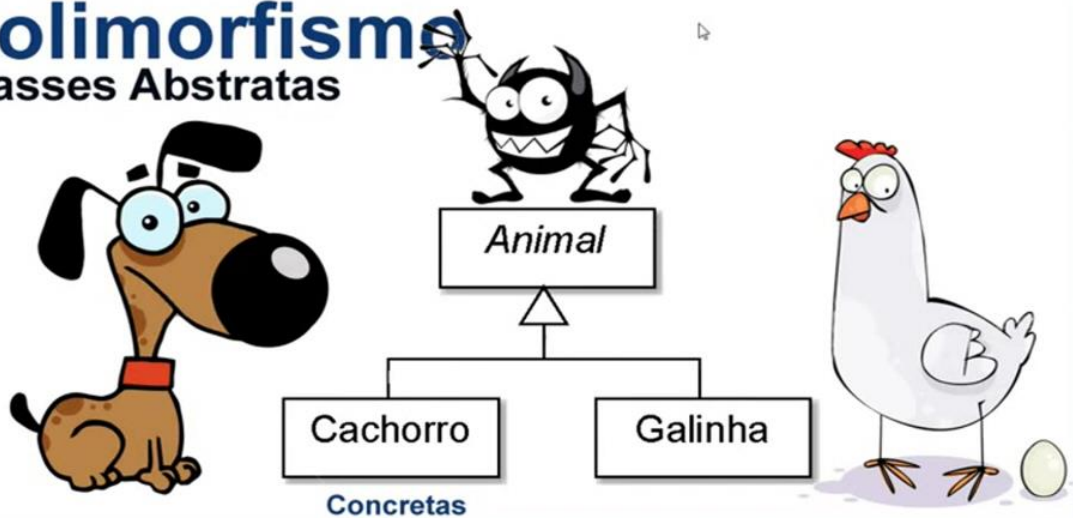


```
run:
10.0
25.0

32.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
.
```

45. POLIMORFISMO - CLASSES ABSTRACT

Polimorfismo Classes Abstratas



As classes abstratas não podem ser instanciadas. As classes abstratas têm métodos, comportamentos indefinidos para que você possa implementar nas suas subclasses.

Um método sem corpo não tem as chaves, ele termina com ponto-e-vírgula.

Os métodos abstratos só podem existir em uma classe que não pode ser instanciada. Só podem existir em uma classe abstrata. As classes abstratas também podem ter métodos implementados, assim como métodos por implementar.

Você só não precisará implementar um método abstrato da sua superclasse se você também for uma classe abstrata. Aí não há a necessidade de se implementar esses métodos. Eles deverão ser implementados pela primeira classe concreta que implementar de alguma forma aquela classe. As classes abstratas deverão, em algum momento, ser implementadas pelas classes chamadas de concretas.

Arquivo: Animal.java

```
package br.com.xti.heranca;
```

```
public abstract class Animal {
```

```
    private int serial;
```

```
    double peso;
```

```
    String comida;
```

```
    public Animal(double peso, String comida){
```

```
        this.peso = peso;
```

```
        this.comida = comida;
```

```
    }
```

```
    void dormir(){System.out.println("Dormiu");}
```

```
    // void fazerBarulho(){System.out.println("Fazer Barulho");}
```

```
    abstract void fazerBarulho();
```

```
}
```

Arquivo: Cachorro.java

```
package br.com.xti.heranca;

public class Cachorro extends Animal {
    public Cachorro() {
        super(30, "Carne");
    }

    @Override
    void fazerBarulho() {
        System.out.println("Au, Au !");
    }
}

ou

package br.com.xti.heranca;

public abstract class Cachorro extends Animal {
    public Cachorro() {
        super(30, "Carne");
    }
}
```

Arquivo Galinha.java

```
package br.com.xti.heranca;

public class Galinha extends Animal {

    public Galinha() {
        super(2, "Milho");
    }

    @Override
    void fazerBarulho() {
        System.out.println("Có, Có !");
    }

    public double calculaArea() {
        return 0;
    }
}
```

```

package br.com.xti.heranca;

public class AnimalTest {

    // uso de polimorfismo
    public static void barulho(Animal animal) {
        animal.fazerBarulho();
    }

    public static void barulhoSemPolimorfismo(String animal) {
        if (animal.equals("Cachorro")) {
            System.out.println("Au Au !");
        } else if (animal.equals("Galinha")) {
            System.out.println("Có, Có !");
        }
    }

    public static void main(String[] args) {

        Cachorro toto = new Cachorro();
        toto.comida = "Carne";
        toto.dormir();

        Galinha carijo = new Galinha();
        carijo.dormir();

        //      Animal generico = new Animal(0, null);

        System.out.println(toto instanceof Cachorro);
        System.out.println(toto instanceof Animal);

        toto.fazerBarulho();
        carijo.fazerBarulho();
        //      generico.fazerBarulho();

        System.out.println();

        barulho(toto);
        barulho(carijo);

        System.out.println();

        barulhoSemPolimorfismo("Cachorro");
        barulhoSemPolimorfismo("Galinha");

    }
}

```

Saída - CursoJavaUniversidadeXTI (run) X

```

run:
Dormiu
Dormiu
true
true
Au, Au !
Có, Có !

Au, Au !
Có, Có !

Au Au !
Có, Có !
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

46. POLIMORFISMO - CLASSES FINAL

Quando o modificador final é utilizado em uma variável, esta não pode mais ser alterada, transformando-se então em uma constante.

O modificador final também pode ser utilizado em classes e métodos. Ao se aplicar o modificador final em uma classe, ela não pode mais ser herdada. Nenhuma outra classe vai poder estender aquela classe.

Quando implementamos o modificador final em um método isto implica que o método não pode ser estendido, não pode ser modificado por outra classe.

O modificador final, em variáveis impede a alteração do valor daquela variável. Em classes impede que ela seja estendida. Em métodos impede que eles sejam sobrescritos.

47. POLIMORFISMO - INTERFACES

Dentro de uma interface você só pode declarar métodos abstratos. Todos os métodos de uma interface são abstratos, ou seja, você não precisa declarar o corpo dele e nem dos modificadores "public abstract".

As classes abstratas podem ter métodos abstratos, mas também podem implementar alguns métodos. As interfaces são 100% abstratas.

Para utilizar uma interface implemente o nome da interface que você vai utilizar. (implements areaCalculavel) e aí implemente todos os métodos daquela interface.

As interfaces são utilizadas para padronizar as interações das suas aplicações. As interfaces definem e padronizam como coisas, pessoas e sistemas interagem entre si. Dentro de uma interface todos os métodos serão implementados pela classe que implementar aquela interface.

Uma característica importante da interface é a herança múltipla. Com as interfaces você pode implementar várias interfaces, herdando de todas as interfaces que você desejar os seus atributos e métodos.

Outra característica é que você pode relacionar tipos díspares. Uma interface é geralmente utilizada quando classes díspares (que não estejam relacionadas entre si) precisam compartilhar métodos e constantes comuns. Isto permite que objetos de classes que não estejam relacionadas sejam processadas polimorficamente. Os objetos de classes que implementam a mesma interface podem responder as mesmas chamadas de métodos.

Com as interfaces você tem muito mais flexibilidade que uma classe abstrata aonde todos os objetos que estendem daquela classe estão seguindo uma herança simples. Eles não tem a possibilidade de adicionar comportamentos semelhantes a tipos muito diferentes.

A unidade fundamental de programação na linguagem java é a classe. É na classe que você vai colocar os seus algoritmos, os seus códigos. A unidade fundamental de projeto orientado a objeto é o tipo.

Enquanto classes definem tipos, é bastante útil que você possa definir um tipo sem necessariamente definir e implementar uma classe. Uma interface ela é uma expressão de projeto puro, enquanto que uma classe é uma mistura de projeto e implementação. Uma classe pode implementar os métodos de uma interface de qualquer maneira que o projetista da classe escolher.

Todos os atributos das interfaces são constantes, ou seja, se você declarar dentro da interface um atributo, qualquer que seja ele, esse atributo sempre será uma constante. Ele sempre terá as características de uma variável que você coloque os modificadores "public static final". Você pode colocar esses modificadores, mas eles são redundantes. Dentro de uma interface todas as propriedades são "public static final". Elas são automaticamente constantes.

Arquivo: AreaCalculavel.java

```
package br.com.xti.heranca;

public interface AreaCalculavel {
    double calculaArea();
}
```

Arquivo: VolumeCalculavel.java

```
package br.com.xti.heranca;

public interface VolumeCalculavel {
    int x = 1; // public static final
    double calculaVolume();
}
```

Arquivo: Quadrado.java

```
package br.com.xti.heranca;

public class Quadrado implements AreaCalculavel {

    double lado;
    public Quadrado(double lado) {
        this.lado = lado;
    }

    @Override
    public double calculaArea() {
        return lado * lado;
    }

}
```

Arquivo: Cubo.java

```
package br.com.xti.heranca;
public class Cubo implements VolumeCalculavel, AreaCalculavel {
    double lado;

    public Cubo(double lado){
        this.lado = lado;
    }

    @Override
    public double calculaArea() {
        return 6 * lado * lado;
    }

    @Override
    public double calculaVolume() {
        return lado * lado * lado;
    }

}
```

Arquivo: InterfaceTest.java

```
package br.com.xti.heranca;
public class InterfaceTest {
    public static void area(AreaCalculavel o) {
        System.out.println(o.calculaArea());
    }

    public static void volume(VolumeCalculavel o) {
        System.out.println(o.calculaVolume());
    }

    public static void main(String[] args) {

        // Quadrado q = new Quadrado(2);
        // AreaCalculavel a = q;
        area(new Quadrado(2));
        volume(new Cubo(2));

    }

}
```




```
Saída - CursoJavaUniversidadeXTI (run) X
run:
4.0
8.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Arquivo: Galinha.java

```
package br.com.xti.heranca;
public class Galinha extends Animal implements AreaCalculavel {
    public Galinha() {
        super(2, "Milho");
    }

    void fazerBarulho() {
        System.out.println("Có, Có !");
    }

    @Override
    public double calculaArea() {
        return 0;
    }
}
```

48. TRATAMENTO DE EXCESSÕES

Esse é o recurso utilizado para construir programas robustos e tolerantes a falhas na linguagem Java.

Uma exceção (exception) é uma indicação de um problema que ocorre durante a execução de um programa. O nome "exceção" significa que o problema não ocorre frequentemente, ou seja, é uma exceção à regra. Com o tratamento de exceções o programa pode continuar executando, em vez de encerrar, ou seja, ao lidar com os problemas o programa continua a funcionar, isso ajuda a assegurar a robustez dos aplicativos que você está construindo colaborando pelo que chamamos de computação crítica, de negócios críticos.

O finally é um bloco que é executado independente se teve ou não erro dentro do seu método. O finally é sempre executado. Esse bloco é utilizado para limpar variáveis, para fechar conexões, fechar strings de dados.

System.err.println(); → é utilizado para imprimir uma mensagem de erro do sistema. Ele simplesmente vai imprimir as mensagens de erro com uma cor diferente (em vermelho).

Arquivo: DividePorZero.java

```
package br.com.xti.erros;

import java.util.InputMismatchException;
import java.util.Scanner;

public class DividePorZero {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);
        boolean continua = true;

        do{

            try {
                System.out.print("\nNumero: ");
                int a = s.nextInt();
                System.out.print("Divisor: ");
                int b = s.nextInt();

                System.out.println(a / b);
                continua = false;
            }

            catch(InputMismatchException e1){
                System.out.println("Numeros devem ser inteiros");
                s.nextLine();
                // descarta a entrada errada e libera novamente para o usuario
            }
            catch(ArithmeticException e2){
                System.err.println("Divisor deve ser diferente de Zero");
            }
            finally {
                System.out.println("Finally executado...");
            }

        } while (continua);

    }
}
```

```
Saída - CursoJavaUniversidadeXTI (run) X
run:
Numero: 39
Divisor: 3
13
Finally executado...
CONSTRUÍDO COM SUCESSO (tempo total: 17 segundos)
```

```
Saída - CursoJavaUniversidadeXTI (run) X
run:
Numero: 3
Divisor: 0
Divisor deve ser diferente de Zero
Finally executado...

Numero: 3
Divisor: 1
3
Finally executado...
CONSTRUÍDO COM SUCESSO (tempo total: 34 segundos)
```

O Mult Catch é a possibilidade de tratar vários tipos de exceções no mesmo bloco (foi introduzido na linguagem Java à partir da versão 7).

O stackTrace é a sua pilha de erros. Um erro pode ser começado em uma determinada classe e pode ser propagado em suas classes.

Arquivo: ExcecoesComuns.java

```
package br.com.xti.erros;

import br.com.xti.heranca.Animal;
import br.com.xti.heranca.Cachorro;
import br.com.xti.heranca.Galinha;

public class ExcessoesComuns {
    // static int[] arrayNull; // objeto nulo --> nao existe

    static int[] arrayNull = new int[0];

    public static void main(String[] args) {

        // NullPointerException
        // System.out.println(arrayNull.length);

        // ArithmeticException
        // int x = 5 / 0;

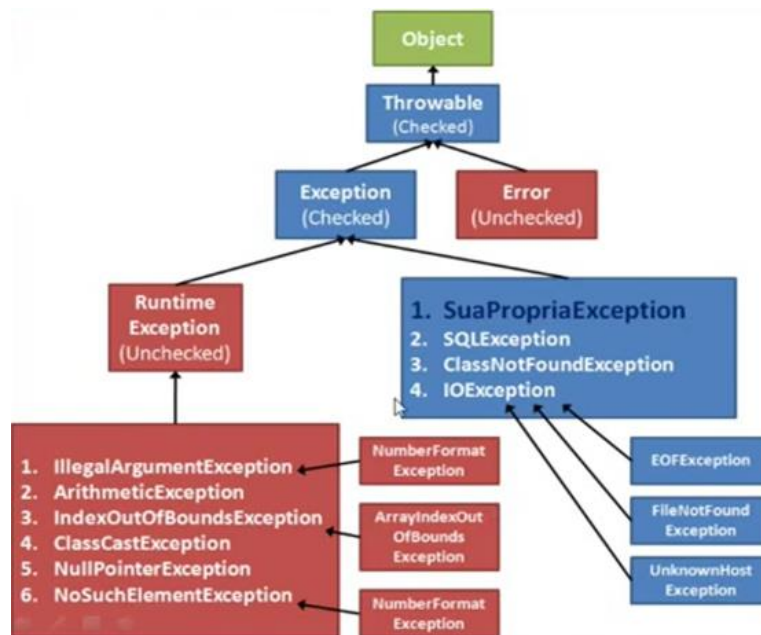
        // ArrayIndexOutOfBoundsException
        // System.out.println(arrayNull[1]);

        // ClassCastException
        // Animal a = new Galinha();
        // Cachorro c = (Cachorro) a;

        // NumberFormatException
        // double d = Double.parseDouble("XTI");

    }
}
```

49. EXCEPTION - HIERARQUIA E CRIAÇÃO



Quando você torna verificada uma exceção você está documentando a existência da exceção e assegurando que o invocador do seu método, ou seja, que quando o seu programa chamar aquele método ele vai de alguma forma tratar aquela exceção. As exceções não verificadas (unchecked) durante a execução representam condições que refletem erros na lógica do seu programa e não podem ser razoavelmente recuperadas durante a execução, isso em se tratando da "Runtime Exception". Esses são erros que devem ser corrigidos no código do seu programa.

Todas as exceções que você criar no seu programa devem estender da classe `Exception` tornando então assim um exceção verificada. Vamos criar a nossa própria exceção estendendo de `Exception`.

Arquivo: SenhaInvalidaException.java

```
package br.com.xti.erros;

public class SenhaInvalidaException extends Exception {

    public SenhaInvalidaException(String mensagem) {
        super(mensagem);
    }

}
```

Arquivo: SenhaTest.java

```
package br.com.xti.erros;

public class SenhaTest {

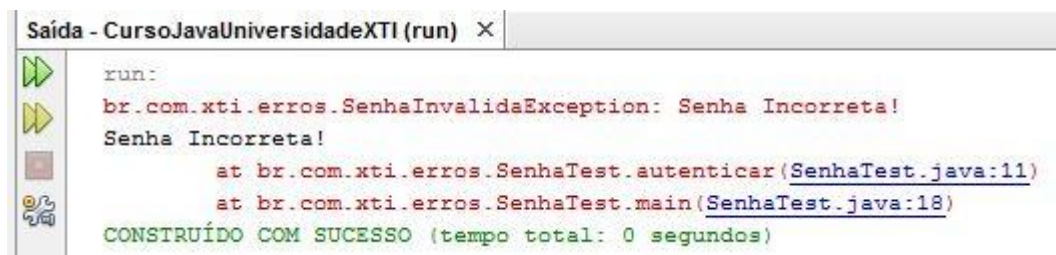
    static void autenticar(String senha) throws SenhaInvalidaException {
        if("123".equals(senha)) {
            // autenticado
            System.out.println("Autenticado");
        } else {
            // senha incorreta
            throw new SenhaInvalidaException("Senha Incorreta!");
        }
    }

    public static void main(String[] args) {

        try {
            autenticar("12345678910");
        } catch (SenhaInvalidaException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) X
run:
br.com.xti.erros.SenhaInvalidaException: Senha Incorreta!
Senha Incorreta!
    at br.com.xti.erros.SenhaTest.autenticar(SenhaTest.java:11)
    at br.com.xti.erros.SenhaTest.main(SenhaTest.java:18)
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Nosso programa recuperou através do método getMessage a mensagem informada no método construtor.

printStackTrace() → traz a pilha de execução desse nosso método aonde ele encontrou aquele erro (ele está propagando aonde o erro passou desde o início).

50. ASSERTION

As assertions são verificações feitas no seu sistema em tempo de desenvolvimento e normalmente são utilizadas para assegurar que coisas que não podem acontecer sejam notificadas quando elas acontecerem. Não se deve utilizar assertion para testar coisas que você sabe que às vezes podem acontecer. Para isso existe as exceções. As assertions são feitas para testar coisas que nunca deveriam acontecer. Em um programa funcionando de forma adequada nenhuma assertion vai falhar.

Assertion é um recurso de teste usado em tempo de desenvolvimento, ou seja, enquanto você está criando e programando o seu aplicativo. Depois que ele estiver pronto as assertions não deverão ser utilizadas.

Por default as assertions estão desligadas, porque quando você distribuir o seu programa, quando você colocar ele em produção, as assertions devem estar desligadas, porque o propósito delas é de auxiliar o programador enquanto ele está construindo o programa e não depois que o programa está pronto. Depois que o programa está pronto, para você fazer as verificações de possíveis erros que podem acontecer no sistema, você utiliza as exceções.

Para habilitar uma assertion entrar no prompt de comando do DOS e digitar:

```
java -ea br.com.xti.erros.Assertions
```

As assertions errors não devem ser tratadas dentro do seu programa.

Para habilitar dentro do Eclipse, entre em "Run configurations" e na aba "argumentos", no campo "VM arguments" digite -ea. Depois é só rodar o programa.

Você pode habilitar suas assertions para um pacote específico e desabilitar as assertions. Para desabilitar as assertions usar: -da. Para habilitar as assertions num pacote específico entre com:

```
java -ea:br.com.xti... br.com.xti.erros.Assertions
```

Para desabilitar as assertions para um pacote específico:

```
java -ea:br.com.xti... br.com.xti.erros.Assertions
```

```
java -da:java.lang... br.com.xti.erros.Assertions
```

Arquivo: Assertions.java

```
package br.com.xti.erros;

import java.util.Scanner;

public class Assertions {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);
        System.out.println("Entre com um numero de 0 a 10: ");
        int numero = s.nextInt();

        assert (numero >= 0 && numero <= 10) : "Numero invalido " + numero;

        System.out.println("Você entrou com o numero " + numero);

    }

}
```

Saída - CursoJavaUniversidadeXTI (run) X



```
run:
Entre com um numero de 0 a 10:
5
Você entrou com o numero 5
CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
```

51. DEBUG NO ECLIPSE

Debug é um processo de localizar erros e eliminá-los do seu sistema.

Breakpoints são pontos de partida, é por onde começamos a analisar o código da nossa aplicação. Para incluir um breakpoint na classe basta clicar duas vezes na linha que você deseja inicializar o seu debug (a sua depuração).

Para visualizar a execução de uma classe no modo debug acionar o botão com a imagem de um besouro. O Eclipse irá solicitar a sua autorização para alternar para a perspectiva de debug. Clique em "yes" para ver essa nova perspectiva onde temos algumas abas muito importantes, a primeira delas é a aba "Variables"; ela apresenta a relação de variáveis dentro do método que você está analisando. Dentro dessa aba você ainda pode aumentar o leque de opções para serem analisados e na opção "Java" você pode habilitar a visualização de constantes, de variáveis estáticas e outras preferências.

Na aba "Breakpoints" você pode visualizar todos os breakpoints incluídos dentro da sua aplicação. Você pode marcar e desmarcar os breakpoints que serão acessados nessa perspectiva de debug.

A aba "Expressions" é utilizada para avaliação de expressões dentro da sua aplicação sem a necessidade de incluir código direto na sua classe java. Em tempo de execução você pode adicionar novas expressões (Add Watch Expression) para avaliar se um determinado valor atende a um determinado critério e assim por diante.

Na aba "Debug" navegamos nas linhas que estão sendo debugadas. Há várias formas de se navegar. As duas principais formas são através dos botões:

Step Over (F6) → percorre linha a linha, independente da instrução que você tenha dentro do seu código. Clicando em "Step Over" ele vai para a próxima linha de execução.

Step Into (F5) → se a próxima linha for um método, ao clicar nesse botão ele entra na primeira linha do método (bloco).

Resume (F8) → Para passar por todo o processo sem precisar percorrer linha a linha do bloco.

Watchpoint → é utilizado para monitorar variáveis de instância. Variável de instância é uma variável do seu objeto. Clicando duas vezes em frente a variável a ser monitorada será adicionado um watchpoint naquela variável de instância e esse watchpoint pode ser configurado na opção "Breakpoint Properties". Você pode configurar para ser informado quando a variável for acessada e/ou modificada, além de poder monitorar quantas vezes esses acessos aconteceram.

Exception Breakpoint → é a possibilidade que você tem de monitorar uma exceção específica.

Class Breakpoint → permite monitorar uma classe específica (quando uma classe foi carregada dentro de uma aplicação).

52. STRING - PRINCIPAIS OPERAÇÕES

As strings representam textos e os textos nada mais são do que conjunto de caracteres.

src.zip (em c:\Arquivos de programas\Java\jdk1.7.0_05 → contém todos os fontes da classe java.

Arquivo: StringOperacoes.java

```
package br.com.xti.java;

public class StringOperacoes {

    public static void main(String[] args) {

        String s1 = "Write Once";
        String s2 = s1 + " Run Anywhere";
        String s3 = new String("Java Virtual Machine");
        char[] array = {'J', 'a', 'v', 'a'};
        String s4 = new String(array);

        // OPERACOES

        int tamanho = s1.length();
        // retorna o numero de caracteres dentro da string
        System.out.println(tamanho);

        char letra = s1.charAt(0); // de 0 a length() - 1
        System.out.println(letra);

        String texto = s1.toString(); // retorna a propria string
        System.out.println(texto);

        // LOCALIZACAO

        int posicao = s3.indexOf('J');
        System.out.println(posicao);

        int firstpos = s3.indexOf("Virtual");
        System.out.println(firstpos);

        int lastpos = s3.lastIndexOf('a');
        System.out.println(lastpos);

        boolean vazia = s3.isEmpty();
        System.out.println(vazia);

        // COMPARACAO

        String xti = "XTI";
        boolean x = xti.equals("xti");
        System.out.println(x);

        boolean x2 = xti.equalsIgnoreCase("xti");
        System.out.println(x2);

        boolean x3 = xti.startsWith("XT");
        System.out.println(x3);

        boolean x4 = xti.startsWith("TI");
        System.out.println(x4);

        boolean x5 = xti.endsWith("TI");
        System.out.println(x5);

        int c = "amor".compareTo("bola"); // -1
        System.out.println(c);
    }
}
```

```

        int c1 = "bola".compareTo("amor"); // 1
        System.out.println(c1);

        int c2 = "amor".compareTo("amor"); // 0
        System.out.println(c2);

        int c3 = "123".compareTo("234"); // -1
        System.out.println(c3);

        String so = "Olhe, olhe!";
        // boolean o = so.regionMatches(6, "Olhe", 0, 4);
        boolean o = so.regionMatches(true, 6, "Olhe", 0, 4);
        System.out.println(o);

        // EXTRACAO

        String l = "O Brasil é Lindo";
        String sl = l.substring(11);
        System.out.println(sl);

        String sl2 = l.substring(2, 8);
        System.out.println(sl2);

        sl = l.concat(" que Maravilha!");
        System.out.println(sl);

        // TROCA

        sl = l.replace('s', 'z');
        System.out.println(sl);

        sl = l.replaceFirst(" ", "X");
        System.out.println(sl);

        sl = l.replaceAll(" ", "X");
        System.out.println(sl);

        // MODIFICACAO

        sl = l.toUpperCase();
        System.out.println(sl);

        sl = l.toLowerCase();
        System.out.println(sl);

        System.out.println("    espaços    ".trim());
        // remove espacos em branco antes e depois da string
    }
}

```

Saída - CursoJavaUniversidadeXTI (run) X

```

-1
true
Lindo
Brasil
O Brasil é Lindo que Maravilha!
O Brazil é Lindo
OXBrasil é Lindo
OXBrasilXéXLindo
O BRASIL É LINDO
o brasil é lindo
espaços
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

53. TOKENIZAÇÃO DE STRINGS

A tokenização é o processo de pegar grandes pedaços de dados e dividi-los em pedaços pequenos. Os tokens são os pedaços propriamente ditos dos dados. Os delimitadores são as expressões que separam os tokens uns dos outros.

Na classe String temos o método Split() que realiza a quebra desses dados.

Arquivo: StringToken.java

```
package br.com.xti.java;

public class StringToken {

    public static void main(String[] args) {

        String s = "XHTML;CSS;JavaScript;jQuery;Java";
        // String[] tokens = s.split(";");
        s = "Venha trabalhar na XTI";
        String[] tokens = s.split(" ");

        System.out.println(tokens.length + " tokens");

        for (String token : tokens) {
            System.out.println(token);
        }

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) X
run:
4 tokens
Venha
trabalhar
na
XTI
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

54. STRINGBUFFER E STRINGBUILDER

Um texto de um objeto String nunca pode ser modificado, para isso existe a classe StringBuffer que permite a modificação do seu conteúdo. As classes StringBuffer e StringBuilder também representam textos e são utilizadas para modificações de textos. Essas duas classes têm o mesmo conjunto de métodos, a única diferença é que os métodos da StringBuffer são sincronizados.

O método capacity() não existe num objeto String. Ele informa a capacidade de armazenamento de novos caracteres dentro desses objetos sem alocar mais memória.

O método reverse() inverte a ordem dos caracteres.

O método append() permite incluir ou concatenar novos conteúdos à String.

Arquivo: StringMutavel.java

```
package br.com.xti.java;

public class StringMutavel {

    public static void main(String[] args) {

        StringBuffer s0 = new StringBuffer();
        StringBuilder s1 = new StringBuilder("Java");

        s1.toString();
        s1.length();

        System.out.println(s1.length());

        s1.capacity();
        System.out.println(s1.capacity());

        // System.out.println(s1);
        // s1.reverse();

        s1.append(" Trabalhando ");
        char[] c = {'c', 'o', 'm'};
        s1.append(c).append(" Textos.");
        System.out.println(s1);

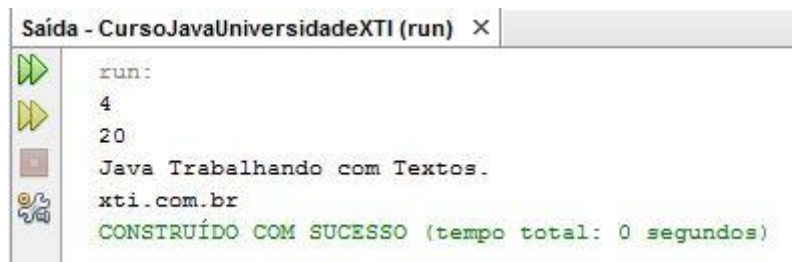
        String s = "oi" + " como " + "vai você?";
        String sb = new StringBuilder("Oi").append(" como ").
            append("vai você?").toString();

        // REMOCAO

        String url = new StringBuilder("www.xti.com.br").
            delete(0, 4).toString();
        System.out.println(url);

    }

}
```



```
Saída - CursoJavaUniversidadeXTI (run) X
run:
4
20
Java Trabalhando com Textos.
xti.com.br
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

55. EXPRESSÕES REGULARES

Regular expression é um padrão de pesquisa e substituição de textos. Elas são muito úteis para validar dados do programa e assegurar que esses dados estejam em um determinado formato.

As expressões regulares são um padrão de pesquisa no formato String e você utiliza esse padrão para validar a ocorrência dele em um texto. As expressões regulares são sensíveis à letras maiúsculas e minúsculas.

Metacaracteres são caracteres especiais que indicam a ocorrência de número, letra no seu texto.

Arquivo: ExpressaoRegular.java

```
package br.com.xti.java;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ExpressaoRegular {

    public static void main(String[] args) {

        // String padrao = "Java";
        // String texto = "Java";
        // boolean b = texto.matches(padrao);

        boolean b = "Java".matches("Java");
        System.out.println(b);

        boolean b1 = "Java".matches("java");
        System.out.println(b1);

        /* MODIFICADORES
        (?I), Ignora maiuscula e minuscula
        (?x), Comentarios
        (?m), Multilinhas
        (?s), Dotal */

        boolean b2 = "Java".matches("(?i)java");
        System.out.println(b2);

        /* METACARACTERES
        .   qualquer caracter
        \d   dígitos           [0-9]
        \D   não é dígito      [^0-9]
        \s   espaços          [ \t\n\x0B\f\r]
        \S   não é espaço      [^\s]
        \w   letra             [a-zA-Z_0-9]
        \W   não é letra */

        b = "@".matches(".");
        b = "2".matches("\\d");
        b = "a".matches("\\d");
        b = "a".matches("\\w");
        b = "#".matches("\\d");
        b = " ".matches("\\s");
        b = "R".matches("\\s");
        b = "Pi".matches(".");
        b = "Pi".matches("..");
        b = "Pie".matches("...");
        b = "21".matches("\\d\\d");

        String cep = "\\d\\d\\d\\d\\d-\\d\\d\\d\\d";
        b = "70294-070".matches(cep);
        System.out.println(b);
```

```

/* QUANTIFICADORES
X{n}      X, exatamente n vezes
X{n,}     X, pelo menos n vezes
X{n,m}    X, pelo menos n mas não mais do que m vezes
X?        X, 0 ou 1 vez
X*        X, 0 ou + vezes
X+        X, 1 ou + vezes */

b = "21".matches("\\d{2}");
b = "123".matches("\\d{2,}");
b = "12345".matches("\\d{2,5}");
b = "123456".matches("\\d{2,5}");
b = "".matches(".?"); // 0 ou 1 vez
b = "ab".matches(".*"); // 0 ou + vezes
b = "".matches(".+"); // 1 ou + vezes
b = "123".matches(".+");

b = "70294-070".matches("\\d{5}-\\d{3}");
b = "12/02/1980".matches("\\d{2}/\\d{2}/\\d{4}");
System.out.println(b);

/* METACARACTER DE FRONTEIRA
^ inicia
$ finaliza
| ou */

b = "Pier21".matches("^Pier.*");
b = "Pier21".matches(".*21$");
b = "tem java aqui".matches(".*java.*");
b = "tem java aqui".matches("^tem.*aqui$");
b = "sim".matches("sim|nao");
System.out.println(b);

/* AGRUPADORES
[...]      Agrupamento
[a-z]      Alcance
[a-e]{i-u} União
[a-z&&[aeiou]] Intersecção
[abc]      Exceção
[a-z&&[^m-p]] Subtração
\\x        Fuga literal */

b = "x".matches("[a-z]");
b = "2".matches("[a-z]");
b = "A".matches("[a-z]");
b = "3".matches("[0-9]");

b = "True".matches("[tT]rue"); // true True
b = "yes".matches("[tT]rue|[yY]es"); // true True yes Yes
b = "Beatriz".matches("[A-Z][a-zA-Z]*"); // Primeiro nome
b = "Beatriz".matches("[A-Z][a-z]*");
b = "olho".matches("[^abc]lho");
b = "alho".matches("[^abc]lho");
b = "crau".matches("cr[ae]u"); // crau creu

b = "rh@xtiuniversity.com".matches("\\w+@\\w+\\.\\w{2,3}");
System.out.println(b);

String doce = "Qual é o Doce mais doCe que o doce?";
Matcher matcher = Pattern.compile("(?i)doce").matcher(doce);
while(matcher.find()){
    System.out.println(matcher.group());
}

```

```
// SUBSTITUIÇÕES

String r = doce.replaceAll("(?i)doce", "DOCINHO");

String rato = "O rato roeu a roupa do rei de roma";
r = rato.replaceAll("r[aeiou]", "XX");
r = "Tabular este texto".replaceAll("\\s", "\\t");
System.out.println(r);


String url = "www.xti.com.br/clientes-2011.html";
// http://www.xti.com.br/2011/clientes.jsp --> Objetivo
String re = "www.xti.com.br/\\w{2,}-\\d{4}.html";
b = url.matches(re);
System.out.println(b);


re = "(www.xti.com.br)/(\\w{2,})-(\\d{4}).html";
r = url.replaceAll(re, "http://$1/$3/$2.jsp");
System.out.println(r);

}

}
```

```
Saida - CursoJavaUniversidadeXTI (run) X
run:
true
false
true
true
true
true
true
true
Doce
doCe
doce
Tabular este      texto
true
http://www.xti.com.br/2011/clientes.jsp
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```