

JAVA

LINGUAGEM DE PROGRAMAÇÃO ORIENTADA
OBJETOS – JAVA

JOSE PICOVSKY

JPICOVSKY@GMAIL.COM

Paradigmas de Programação

Programação Procedural

- Baseada no conceito de chamadas a procedimento (linguagens: C, C++, Fortran, Pascal, MATLAB).

Paradigmas de Programação

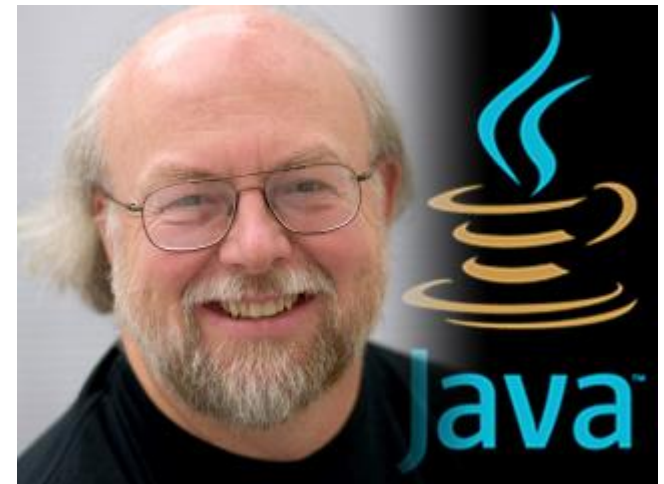
Programação Orientada por Objetos

- Baseado na composição e interação entre diversas unidades de software chamadas de objetos;
- Permite re-uso de código e flexibilidade no desenvolvimento.

História do Java

Em 1992, A Sun criou um time (conhecido como Green Team) para desenvolver inovações tecnológicas;

Esse time foi liderado por James Gosling, considerado o Pai do Java;



História do Java

Idéia de criar um interpretador para pequenos dispositivos, facilitando a reescrita de software para aparelhos eletrônicos;

O projeto resultou em uma linguagem baseada em C e C++ chamada Oak;

História do Java

A inspiração do nome Java surgiu em uma cafeteria local, cujo café vinha de uma ilha da Indonésia chamada Java;

1993 e a Sun aposta no imediato potencial de utilizar Java para criar páginas da Web com o chamado conteúdo dinâmico;

História do Java

Atualmente, Java é utilizado :

- páginas da Web com conteúdo interativo e dinâmico;
- Aplicativos corporativos de grande porte;
- Fornecer aplicativos para dispositivos destinados ao consumidor final.

Principais Características

Orientada a objetos

- Java é uma linguagem puramente orientada a objetos;
- Tudo em Java são classes ou instâncias de classes, com exceção de seus tipos primitivos de dados.

Sem Ponteiros

- Java não possui ponteiros, isto é, Java não permite a manipulação direta de endereços de memória.

Principais Características

Coletor de lixo (Garbage Collector)

- Possui um mecanismo automático de gerenciamento de memória.

Permite Multithreading

- Recursos que permite o desenvolvimento de aplicações capazes de executar múltiplas rotinas concorrentemente.

Principais Características

Independente de plataforma

- Programas Java são compilados para uma forma intermediária (bytecodes).

Tratamento de exceções

- Permite o tratamento de situações excepcionais.
- Possui exceções embutidas e permite a criação de novas exceções.

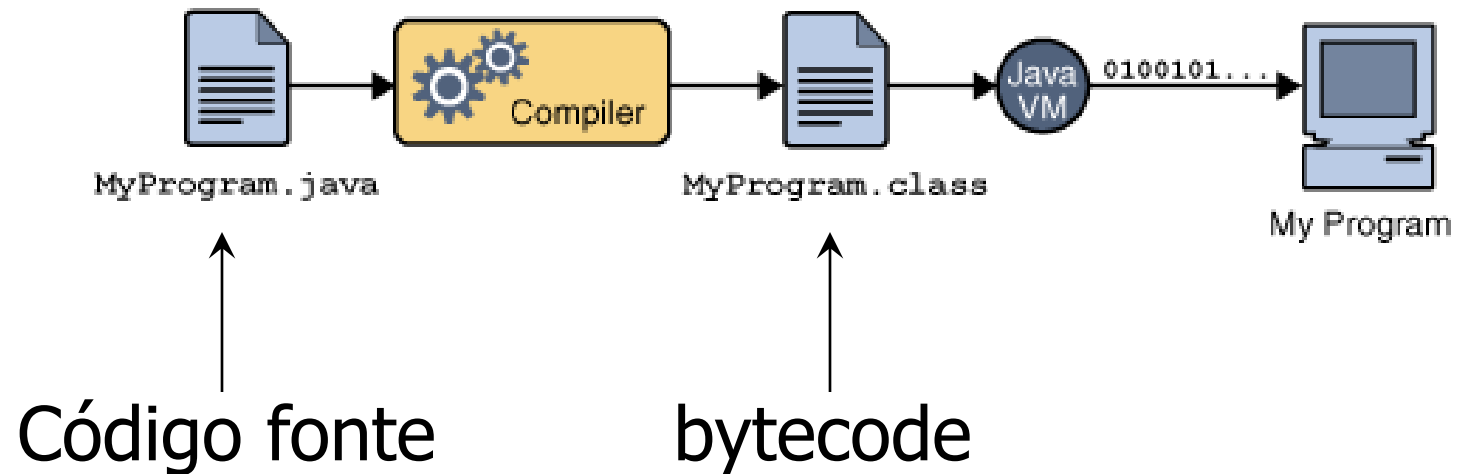
Características e Diferenciais

Processo de desenvolvimento de software em Java

- Todo código fonte escrito em arquivo texto possui extensão .java
- Este arquivo é compilado com o javac gerando o arquivo .class
- O arquivo .class não contém código de máquina nativo, e sim o bytecodes
 - Bytecode: linguagem de máquina da JVM

Características e Diferenciais

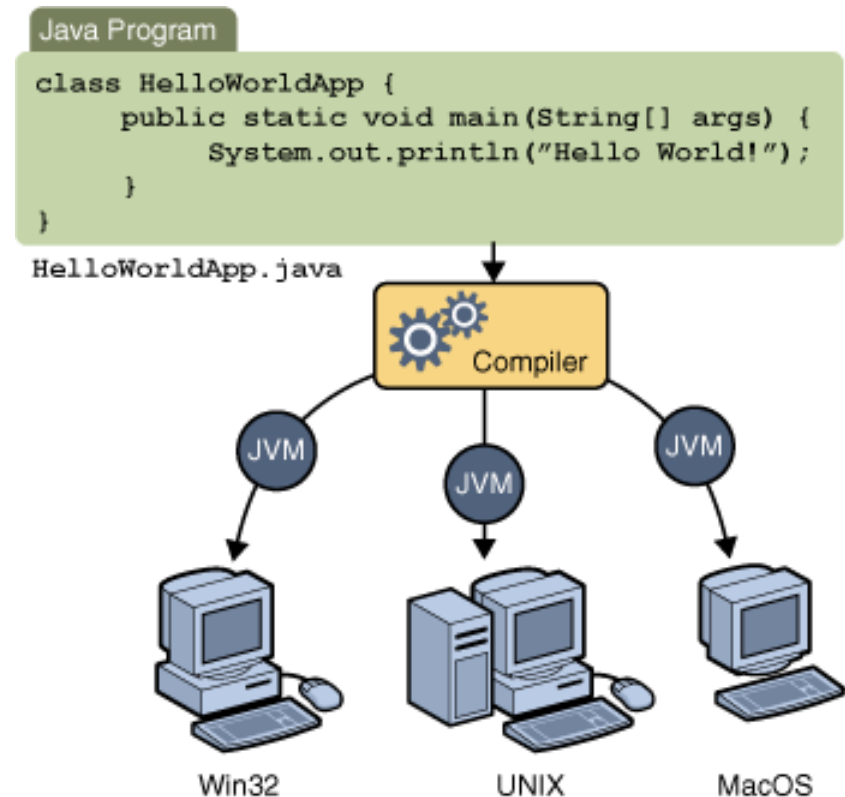
Processo de desenvolvimento de software



Características e Diferenciais

JVM: Disponível em diferentes sistemas operacionais

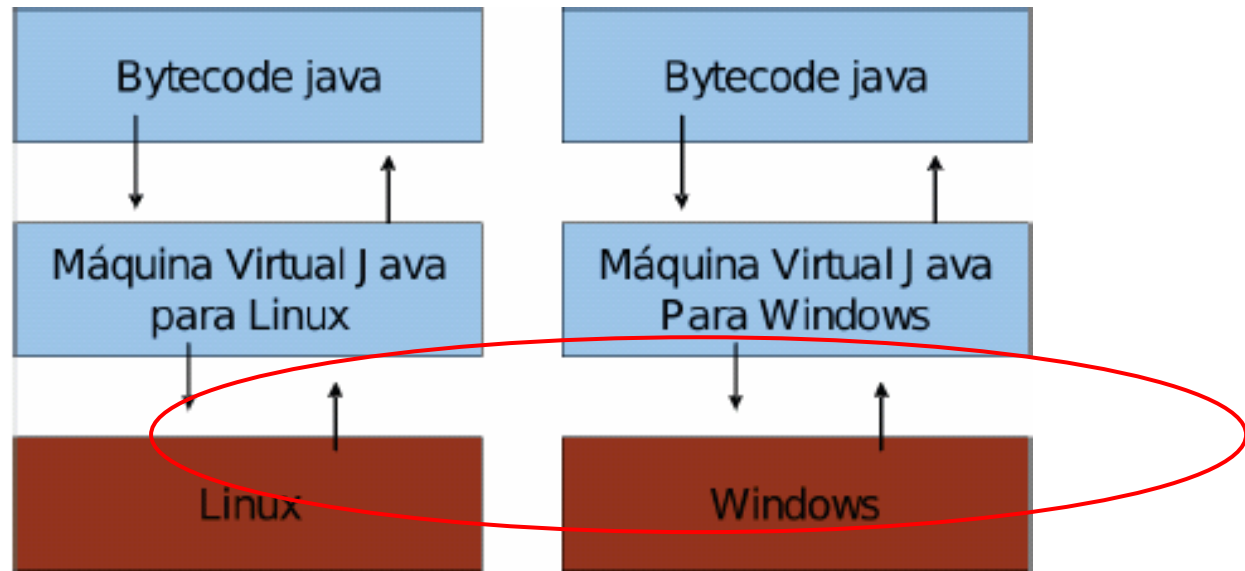
Arquivo class gerado em um sistema operacional pode ser executado em qualquer outro



Principais Características

Máquina Virtual Java

- Utiliza o conceito de máquina virtual;
- Camada responsável por interpretar os bytecodes.



Interpretada, Neutra, Portável

Bytecodes executam em qualquer máquina que possua uma JVM, permitindo que o código em Java possa ser escrito *independente da plataforma*.

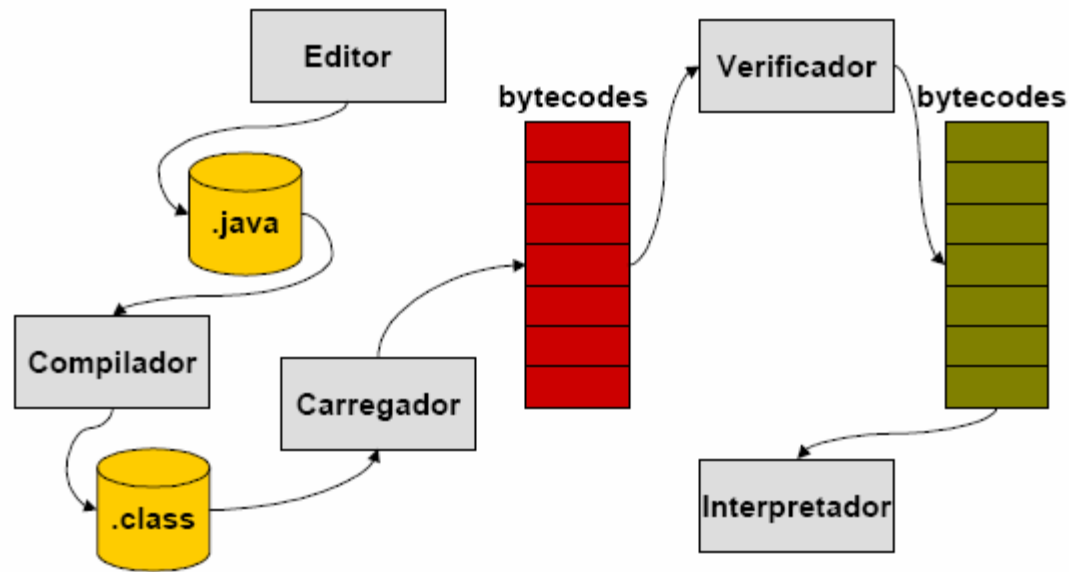
A característica de ser *neutra em relação à arquitetura* permite uma grande *portabilidade*.

Principais Características

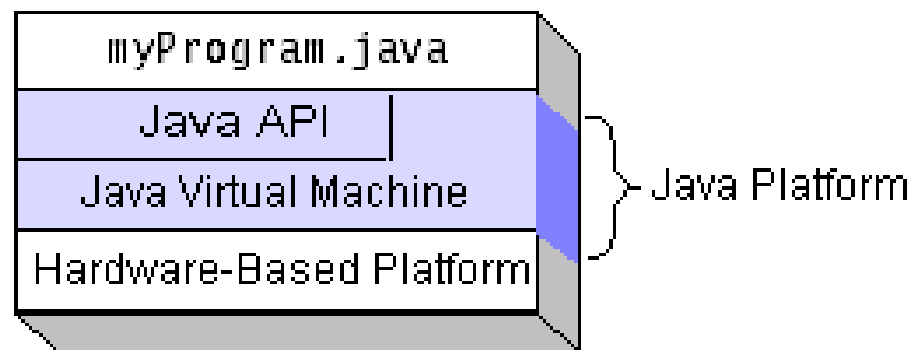
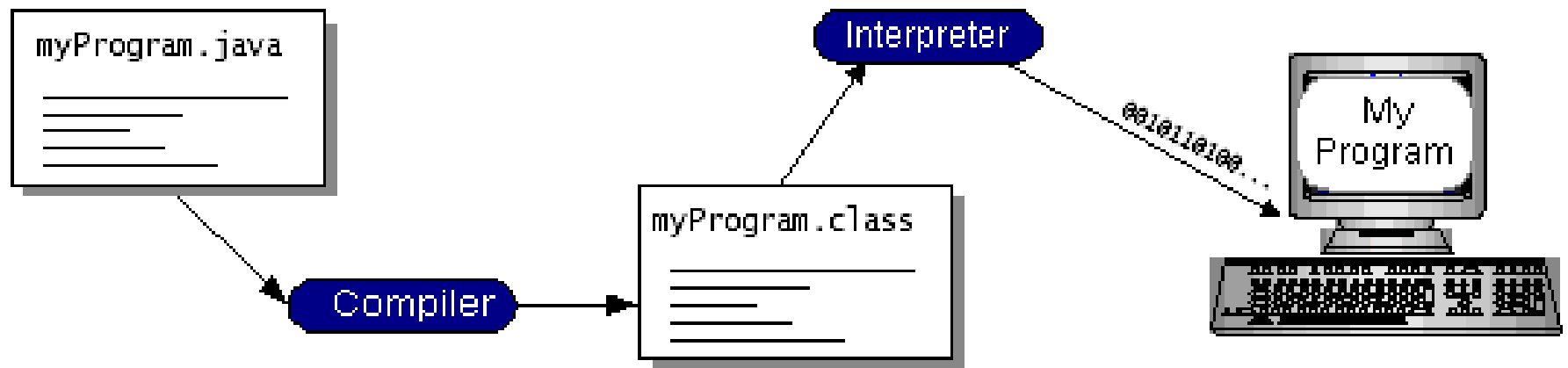
Segurança

- Pode executar programas via rede com restrições de execução, além de itens específicos da linguagem, como ser fortemente tipada, ter assinaturas digitais em suas classes compiladas, etc;

Fases de um programa Java



Interpretada, Neutra, Portável



Fases de um programa Java

Os programas Java normalmente passam por cinco fases para serem executados:

- **Fase 1 (Edição):** Consiste em editar um arquivo com código em Java e salvá-lo com a extensão .java;
- **Fase 2 (Compilação):** Comando javac compila o programa. O compilador Java traduz (.java => .class);

Fases de um programa Java

- **Fase 3 (Carga):** Carrega o programa na memória antes de ser executado. Carregador de classe, pega o arquivo(s) .class que contém os bytecodes.
- **Fase 4 (Verificação):** O verificador assegura que os bytecodes são válidos e não violam as restrições de segurança de Java.

Fases de um programa Java

- **Fase 5 (Execução):** A JVM máquina virtual Java (Interpretador) interpreta (em tempo de execução) o programa, realizando assim a ação especificada pelo programa.

Fases de um programa Java

Tempo de Execução é o período em que um programa de computador permanece em execução;

Tempo de Compilação é uma referência ao período em que o código é compilado para gerar um programa executável.

Plataformas Java

A linguagem Java conta com três ambientes de desenvolvimento:

- **JSE (Java Platform, Standard Edition):** É a base da plataforma; inclui o ambiente de execução e as bibliotecas comuns é voltada a aplicações para PCs e servidores.
- **JEE (Java Platform, Enterprise Edition):** A edição voltada para o desenvolvimento de aplicações corporativas e para Internet.

Plataformas Java

- **JME (Java Platform, Micro Edition):** A edição para o desenvolvimento de aplicações para dispositivos móveis e embarcados.

Plataformas Java

Além disso, pode-se destacar outras duas plataformas Java mais específicas:

- **Java Card:** Voltada para dispositivos embarcados com limitações de processamento e armazenamento, como smart cards.
- **JavaFX:** Plataforma para desenvolvimento de aplicações multimídia em desktop/web (JavaFX Script) e dispositivos móveis (JavaFX Mobile).

Componentes de uma plataforma Java:

JRE (Java Runtime Environment)

- É composta de uma JVM e por um conjunto de bibliotecas que permite a execução de softwares em Java.
- Apenas permite a execução de programas, ou seja é necessário o programa Java compilado (.class).

Componentes de uma plataforma Java:

JDK (Java Development Kit) É composto basicamente por:

- Compilador (javac) + JVM;
- visualizador de applets , bibliotecas de desenvolvimento (os packages java);

Componentes de uma plataforma Java:

JDK (Java Development Kit) É composto basicamente por:

- programa para composição de documentação (javadoc);
- depurador básico de programas (jdb) e a versão runtime do ambiente de execução (**JRE**).

Instalando e configurando o Java

Para iniciar o trabalho com Java, é necessário executar os seguintes passos:

1. Fazer o download do Java SE Development Kit (JDK) e instalá-lo;
<https://www.oracle.com/technetwork/pt/java/javase/downloads/index.html>
1. Configurar as variáveis de ambiente no Windows;
2. Fazer o download de algum IDE para desenvolvedores Java.

Instalando e configurando o Java


oracle.com/technetwork/pt/java/javase/downloads/index.html

Oracle Technology Network / Java / Java SE / Transferências

Java SE
Suporte para Java SE
Java Embedded
Java EE
Java ME
Java FX
Java DB
Camada da web
Comunidade


Resumo Transferências Documentação Comunidade Tecnologia Formação

Downloads de Java SE



DOWNLOAD

Plataforma Java (JDK) 8u111 / 8u112



DOWNLOAD

NetBeans com JDK 8

Plataforma Java, Edição Standard

Java SE 8u111 / 8u112

Java SE 8u111 inclui correções de segurança importantes. A Oracle recomenda enfaticamente que todos os usuários do Java SE 8 atualizem para esta versão. Java SE 8u112 é uma atualização do conjunto de patches, incluindo todos os 8u111 mais recursos adicionais (descritos nas notas de lançamento). [Saber mais](#)

Mudança planejada importante para JARs assinados com MD5

Começando com os lançamentos da atualização do patch crítico de abril, planejados para 18 de abril de 2017, todas as versões do JRE tratarão os JARs assinados com MD5 como não assinados. [Saiba mais](#) e [veja as instruções de teste](#).
Para obter mais informações sobre o suporte de algoritmo criptográfico, verifique o [JRE e JDK Crypto Roadmap](#).

- instruções de instalação
- Notas de Lançamento
- Licença Oracle
- Produtos Java SE
- Licenças de terceiros

JDK

Baixar JDK

Servidor JRE

Baixar

Instalando e configurando o Java

Painel de Controle- Sistemas – Configuração Avançada – Variáveis de Ambiente

The screenshot displays the Windows 10 Control Panel interface. The 'Sistema' (System) category is selected on the left sidebar. The main content area shows the 'Propriedades do Sistema' (System Properties) window, which is open to the 'Avançado' (Advanced) tab. This tab contains three sections: 'Desempenho' (Performance), 'Perfis de Usuário' (User Profiles), and 'Inicialização e Recuperação' (Startup and Recovery). Each section has a 'Configurações...' (Settings...) button. At the bottom of the 'Avançado' tab, there is a 'Variáveis de Ambiente...' (Environment Variables...) button. The background shows the 'Sobre' (About) section of the Control Panel, which displays system information such as 'Edição' (Edition) as Windows 10 Pro, 'Versão' (Version) as 20H2, and 'Instalado em' (Installed on) as 23/08/2020. On the right side of the Control Panel, there are links for 'Configurações relacionadas' (Related settings), 'Configurações de BitLocker' (BitLocker settings), 'Gerenciador de Dispositivos' (Device Manager), 'Área de trabalho remota' (Remote Desktop), 'Proteção do sistema' (System Protection), 'Configurações avançadas do sistema' (Advanced system settings), and 'Renomear este computador' (Rename this computer). At the bottom right, there are links for 'Obtenha ajuda' (Get help) and 'Enviar comentários' (Send feedback).

Sobre

O computador está monitorado e protegido.

Propriedades do Sistema

Nome do Computador Hardware Avançado Proteção do Sistema Remoto

Para tirar o máximo proveito destas alterações, é preciso ter feito login como administrador.

Desempenho

Efeitos visuais, agendamento de processador, uso de memória e memória virtual

Perfis de Usuário

Configurações da área de trabalho relativas à entrada

Inicialização e Recuperação

Informações sobre inicialização do sistema, falha do sistema e depuração

Configurações relacionadas

- Configurações de BitLocker
- Gerenciador de Dispositivos
- Área de trabalho remota
- Proteção do sistema
- Configurações avançadas do sistema
- Renomear este computador

Obtenha ajuda

Enviar comentários

Especificações do Windows

Edição	Windows 10 Pro
Versão	20H2
Instalado em	23/08/2020

Instalando e configurando o Java

Painel de Controle- Sistemas – Configuração Avançada – Variáveis de Ambiente

Clique no botão “Nova” em “Variáveis do sistema”;

Nome da variável: JAVA_HOME

Valor da variável: coloque aqui o endereço de instalação (o caminho tem que ser o mesmo onde o java foi instalado

C:\Arquivos de programas\Java\jdk1.8.0_241

Clique em OK

Clique novamente no botão Nova em Variáveis do sistema;

Nome da variável: CLASSPATH

Os valores da variável encontram-se abaixo, sempre insira um do lado outro sem espaços e com o ; (ponto e vírgula) no final.

```
; %JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\dt.jar;  
%JAVA_HOME%\lib\htmlconverter.jar;%JAVA_HOME%\jre\lib;%JAVA_HOME%\jre\lib\rt.jar;
```

Selecione a variável PATH em Variáveis do sistema e clique no botão Editar.

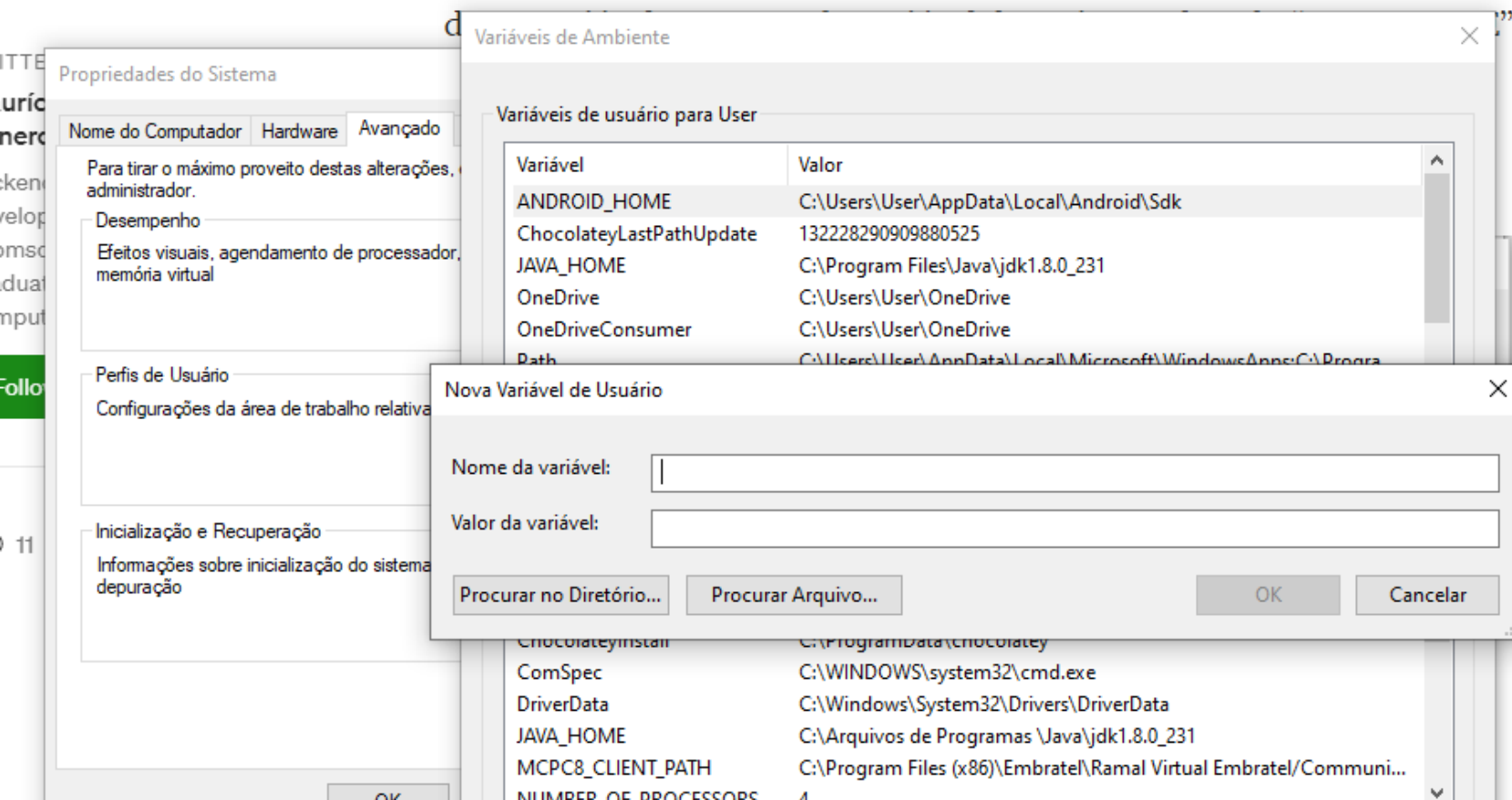
Defina o valor dessa variável com o caminho da pasta Bin. No caso, pode-se utilizar a variável JAVA_HOME previamente definida.

```
; %JAVA_HOME%\bin
```


Instalando e configurando o Java

Painel de Controle- Sistemas – Configuração Avançada – Variáveis de Ambiente

- Irá abrir outra tela para ser configurado o nome da variável e o valor



Instalando e configurando o Java

Configurando o PATH

Após configurar o JAVA_HOME, devemos configurar o PATH para que as ferramentas de desenvolvimento do JDK estejam disponíveis para uso, para isso vamos colocar a pasta “bin” do JAVA_HOME na variável PATH.

Na tela das variáveis de ambiente, procure pela variável PATH, selecione-a e clique em “editar”:

```
;%JAVA_HOME%\bin
```

Usa-se o %JAVA_HOME% neste local para referenciar uma outra variável configurada, ou seja, o windows irá entender que a nossa configuração está apontando para “C:\Arquivos de programas\Java\jdk1.8.0_241” que é o valor da variável “JAVA_HOME”, e com isso vai encontrar a pasta bin.

Instalando e configurando o Java

Configurando o CLASSPATH

CLASSPATH é o caminho onde estão as bibliotecas do Java, que você importa nos seus programas. Se você quer usar servlets, por exemplo, tem que incluir o `servlet.jar` no classpath.

A variável CLASSPATH é opcional, não precisa ser configurada, ela é uma variável usada para personalizar onde o carregador de classes do Java irá procurar as classes para compilar ou executar um programa e você deve usá-la somente se realmente for necessário, pois todos os programas Java irão percorrer as classes referenciadas nessa variável.

Para configurá-la, basta fazer assim como foi feito para criar a variável `JAVA_HOME`, porém, deve-se colocar o nome da variável como CLASSPATH e colocar o conteúdo desejado.

Um primeiro programa Java

```
1 public class MeuPrograma
2 {
3     /**
4      * Meu primeiro programa em Java
5      */
6     public static void main (String[ ] args) {
7         //Mostra a mensagem "Meu primeiro programa Java"
8         System.out.println("Meu primeiro programa Java");
9     }
10 }
```

Usando o editor de texto e o console

Passo 1: Iniciar o editor (IDE)

Passo 2: Abrir a janela de console

Passo 3: Escrever as instruções utilizando o editor de texto (IDE)

Passo 4: Salvar seu programa Java

- Nome do arquivo: MeuPrograma.java

Usando o editor de texto e a console

Passo 5: Compilar o programa

Passo 6: Executar o programa

Entendendo o primeiro programa Java

```
1 public class MeuPrograma
2 {
3     /**
4      * Meu primeiro programa em Java
5      */
```

- O nome da classe é **MeuPrograma**;

Em Java, todo e qualquer código deve pertencer a uma classe;

Entendendo o primeiro programa Java

- Esta classe usa um identificador de acesso **public**. Indica que está acessível para outras classes de diferentes pacotes (pacotes são coleções de classes).

Entendendo o primeiro programa Java

```
1 public class MeuPrograma
```

```
2 {
```

```
3    /**
```

```
4     * Meu primeiro programa em Java
```

```
5     */
```

- A próxima linha contém uma chave `{` e indica o início de um bloco de instruções.

Entendendo o primeiro programa Java

```
1 public class MeuPrograma
```

```
2 {
```

```
3  /**
```

```
4   * Meu primeiro programa em Java
```

```
5   */
```

- As próximas 3 linhas indicam um comentário em Java.

Entendendo o primeiro programa Java

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

```
7      // Mostra a mensagem “Meu primeiro
```

```
8 // programa Java”
```

```
9 // System.out.println(“Meu primeiro
```

```
10 // programa Java”);
```

As linhas seguintes são comentário de linhas.

Entendendo o primeiro programa Java

```
6 public static void main (String[ ] args) {  
7     // Mostra a mensagem “Meu primeiro  
8 // programa Java”  
9 // System.out.println(“Meu primeiro  
10 // programa Java”);
```

A instrução `System.out.println()`, mostra, na saída padrão, o texto descrito entre as aspas.

Criação de Métodos / Método main()

Método main()

- Todo aplicativo é composto por uma ou mais classes e uma delas deve conter o método especial chamado main(). Este método contém as instruções que são lidas quando a classe é executada pelo interpretador Java
- Quando uma classe é iniciada pela JVM, a mesma irá procurar por um método, dentro da classe, que tem exatamente a assinatura abaixo:

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

Criação de Métodos / Método main()

Método main()

- Características:
 - Deve ser público (public) para ser invocado externamente pela JVM
 - Deve ser estático (static), pois não há objetos quando a JVM inicia a classe
 - Tipo de retorno void, pois este método não deve ter um valor de retorno
 - Possui um parâmetro array String para passagem de parâmetros via linha de comando

Criação de Classes

Sintaxe para Criação de Classe

```
public class Pessoa{  
    }  
}
```

Toda classe possui um nome(identificador) e neste exemplo, está representado por “Pessoa”

Nome da classe é igual ao nome do arquivo (Pessoa.java), pois é uma classe pública (public)

O termo class é utilizado na declaração de novas classes

Criação de variáveis

Uma **variável** representa a unidade básica de armazenamento temporário de dados e é composta por um tipo, um identificador (nome), um tamanho e um valor

Padrões para definição de identificadores para variáveis.

- O uso de nomes curtos é indicado para reduzir o volume de código a ser escrito
- Evitar o uso de caracteres acentuados e símbolos especiais pode ser aconselhável para tornar o programa potencialmente mais portátil

Criação de variáveis

Sintaxe para criação de variáveis e atributos

```
1    public class Pessoa{  
2        String nome;  
3        byte idade;  
4        String endereco;  
5        float salario;  
6    }
```

Tipos Primitivos e de Instância

Java possui duas categorias de tipos de dados:

- Tipo Primitivo
- Tipo de Instância ou Referência

Tipo primitivo: São tipos de dados predefinidos pela linguagem e correspondem a dados mais simples ou escalares

Tipos Primitivos

- Podem ser agrupados em quatro categorias:
 - *Tipos Inteiros*: Byte, Inteiro Curto, Inteiro e Inteiro Longo.
 - *Tipos Ponto Flutuante*: Ponto Flutuante Simples, Ponto Flutuante Duplo.
 - *Tipo Caractere*: Caractere.
 - *Tipo Lógico*: Booleano.

Tipos Primitivos - Inteiros

Tipos de Dados Inteiros	Faixas
Byte	-128 a +127
Short	-32.768 a +32.767
Int	-2.147.483.648 a +2.147.483.647
Long	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

Tipos Primitivos – Ponto Flutuante

Tipos de Dados em Ponto Flutuante	Faixas
Float	$\pm 1.40282347 \times 10^{-45}$ a $\pm 3.40282347 \times 10^{+38}$
Double	$\pm 4.94065645841246544 \times 10^{-324}$ a $\pm 1.79769313486231570 \times 10^{+308}$

- Exemplos:
 - 1.44E6 é equivalente a $1.44 \times 10^6 = 1.440.000$.
 - 3.4254e-2 representa $3.4254 \times 10^{-2} = 0.034254$.

Tipos Primitivos - Caractere

- O tipo char permite a representação de caracteres individuais.

Tipos Primitivos - Booleano

- É representado pelo tipo lógico boolean.
- Assume os valores *false* (falso) ou *true* (verdadeiro).
- O valor default é *false*.
- Ocupa 1 bit.

Os tipos de dados – Tipos Primitivos

- Tipos numéricos
 - Tipos Inteiros
 - tipo **byte**
 - tipo **short**
 - tipo **int**
 - tipo **long**
 - Tipos em ponto flutuante
 - tipo **float**
 - tipo **double**
- Tipos caracter
 - tipo **char**
- Tipos booleanos
 - tipo **boolean**

O tipo **byte**

A faixa de valores correspondentes a este tipo é -128 a 127.

As variáveis que armazenam valores deste tipo tem tamanho de 8 bits.

O tipo **short**

A faixa de valores correspondentes a este tipo é -32768 a 32767.

As variáveis que armazenam valores deste tipo tem tamanho de 16 bits.

O tipo **int**

A faixa de valores correspondentes a este tipo é -2147483648 a 2147483647.

As variáveis que armazenam valores deste tipo tem tamanho de 32 bits.

O tipo **long**

A faixa de valores correspondentes a este tipo é -9223372036854775808 a 9223372036854775807.

As variáveis que armazenam valores deste tipo tem tamanho de 64 bits.

A faixa de valores dos tipos em Ponto Flutuante é a seguinte:

O tipo **float**

A faixa de valores correspondentes a este tipo é (+ ou -)3.40282347E+38

As variáveis que armazenam valores deste tipo tem tamanho de 32 bits.

O tipo **double**

A faixa de valores correspondentes a este tipo é

(+ ou -)1.79769313486231570E+308 .

As variáveis que armazenam valores deste tipo tem tamanho de 64 bits.

Tipos Primitivos - Caractere

<code>\b</code>	backspace
<code>\t</code>	tabulação horizontal
<code>\n</code>	newline
<code>\f</code>	form feed
<code>\r</code>	carriage return
<code>\"</code>	aspas
<code>\'</code>	aspas simples
<code>\\</code>	contrabarra
<code>\xxx</code>	o carácter com código de valor octal xxx, que pode assumir valores entre 000 e 377 na representação octal
<code>\uxxxx</code>	o carácter Unicode com código de valor hexadecimal xxxx, onde xxxx pode assumir valores entre 0000 e ffff na representação hexadecimal.

Tipos de Instância

Tipo de Instância ou Referência: Uma instância é um objeto do tipo definido pela classe. Qualquer classe (desde que não seja abstract) pode ser instanciada como qualquer outro tipo de dado da linguagem Java

Quando é declarada uma variável de um tipo referência, está se especificando que a variável irá referenciar, ou apontar, para um objeto em vez de guardar um valor simples como no caso de um primitivo

Tipos de Instância

Diferenças entre tipo primitivo e tipo de instância

- Tipo primitivo: A própria declaração já indica para a JVM reservar memória
- Tipo de instância: É necessário reservar memória para o objeto por meio do operador “new”

Criação de objetos Wrapper

Para criar um objeto wrapper, use a classe wrapper em vez do tipo primitivo. Para obter o valor, você pode simplesmente imprimir o objeto:

```
public class Main {  
    public static void main(String[] args) {  
        Integer myInt = 5;  
        Double myDouble = 5.99;  
        Character myChar = 'A';  
        System.out.println(myInt);  
        System.out.println(myDouble);  
        System.out.println(myChar);  
    }  
}
```

Criação de objetos Wrapper

Os seguintes métodos são usados para obter o valor associado ao objeto de wrapper correspondente: `intValue ()`, `byteValue ()`, `shortValue ()`, `longValue ()`, `floatValue ()`, `doubleValue ()`, `charValue ()`, `booleanValue ()` .

```
public class Main {  
    public static void main(String[] args) {  
        Integer i = 5;  
        Double d = 5.99;  
        Character c = 'A';  
        System.out.println(i.intValue());  
        System.out.println(d.doubleValue());  
        System.out.println(c.charValue());  
    }  
}
```

Criação de objetos Wrapper

Outro método útil é o método `toString()`, que é usado para converter objetos de invólucro em strings.

No exemplo a seguir, convertemos um `Integer` em `String` e usamos o método `length()` da classe `String` para gerar o comprimento da "string":

```
public class Main {  
    public static void main(String[] args) {  
        Integer i = 100;  
        String s = i.toString();  
        System.out.println(s.length());  
    }  
}
```

Palavras reservadas

<i>abstract</i>	<i>continue</i>	<i>finally</i>	<i>interface</i>	<i>public</i>	<i>throw</i>
<i>boolean</i>	<i>default</i>	<i>float</i>	<i>long</i>	<i>return</i>	<i>throws</i>
<i>break</i>	<i>do</i>	<i>for</i>	<i>native</i>	<i>short</i>	<i>transient</i>
<i>byte</i>	<i>double</i>	<i>if</i>	<i>new</i>	<i>static</i>	<i>true</i>
<i>case</i>	<i>else</i>	<i>implements</i>	<i>null</i>	<i>super</i>	<i>try</i>
<i>catch</i>	<i>extends</i>	<i>import</i>	<i>package</i>	<i>switch</i>	<i>void</i>
<i>char</i>	<i>false</i>	<i>instanceof</i>	<i>private</i>	<i>synchronized</i>	<i>while</i>
<i>class</i>	<i>final</i>	<i>int</i>	<i>protected</i>	<i>this</i>	

- Além dessas existem outras que embora reservadas não são usadas pela linguagem

<i>const</i>	<i>future</i>	<i>generic</i>	<i>goto</i>	<i>inner</i>	<i>operator</i>
<i>outer</i>	<i>rest</i>	<i>var</i>	<i>volatile</i>		

Declaração de Variáveis (1/2)

- Uma variável *não pode utilizar como nome uma palavra reservada* da linguagem.
- Sintaxe:
 - Tipo nome1 [, nome2 [, nome3 [..., nomeN]]];
- Exemplos:
 - int i;
 - float total, preco;
 - byte mascara;
 - double valormedio;

Comentários

- Exemplos:

// comentário de uma linha

/* comentário de
múltiplas linhas */

/** comentário de documentação

* que também pode

* possuir múltiplas linhas

*/

Operadores Aritméticos

Operador	Significado	Exemplo
+	Adição	$a + b$
-	Subtração	$a - b$
*	Multiplicação	$a * b$
/	Divisão	a / b
%	Resto da divisão inteira	$a \% b$
-	Sinal negativo (- unário)	-a
+	Sinal positivo (+ unário)	+a
++	Incremento unitário	++a ou a++
--	Decremento unitário	--a ou a--

Operadores Matemáticos

Veja o exemplo abaixo:

$$\begin{array}{r} 10 \overline{) 3} \\ 1 3 \end{array}$$



Resto da divisão

Em java a sintaxe é a seguinte:

```
int r = 10 % 3;
```

Operadores Relacionais

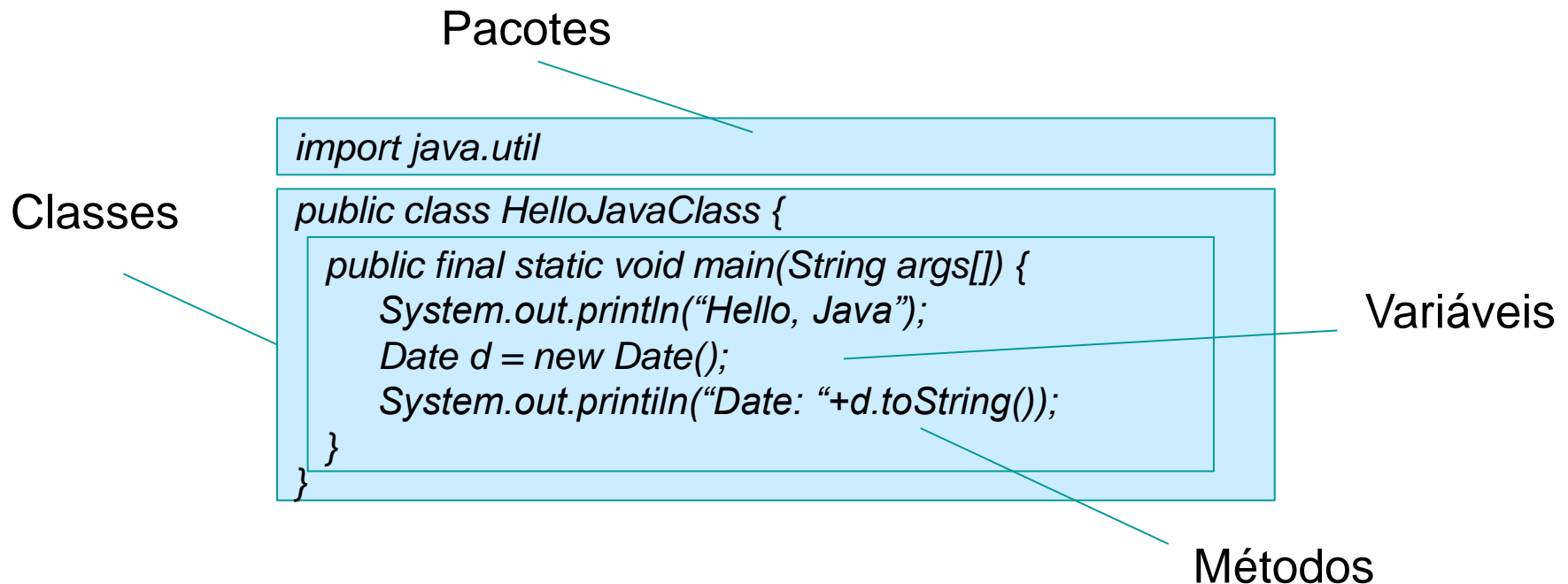
Operador	Significado	Exemplo
==	Igual	$a == b$
!=	Diferente	$a != b$
>	Maior que	$a > b$
>=	Maior ou igual a	$a >= b$
<	Menor que	$a < b$
<=	Menor ou igual a	$a <= b$

Operadores Lógicos

Operador	Significado	Exemplo
&&	E lógico (<i>and</i>)	a && b
	Ou Lógico (<i>or</i>)	a b
!	Negação (<i>not</i>)	!a

Programa Java

- Todos os programas em Java possuem quatro elementos básicos:



```
class exemplo {
```

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

```
    { int data=345;
```

```
      char tyu= 'B';
```

```
      int valor = 678;
```

```
      boolean condicao= false;
```

```
    }
```

```
}
```


O programa abaixo está errado. pois, as variáveis **a** e **b** não foram declaradas.

```
class exemplo {  
    public static void main(String[ ] args )  
    { a = a+b; }  
}
```

```
class correto {  
    public static void main(String[ ] args )  
    { int a=9, b=10;  
      a = a+b;  
    }  
}
```

```
public class Ex1 {  
    public static void main(String[] args) {  
  
        int a=100,b=50,c;  
        c=a+b;  
        System.out.println(c);  
    }  
}
```

ou

```
public class Ex1 {  
    public static void main(String[] args) {  
  
        int a=100,b=50;  
        int c=a+b;  
        System.out.println(c);  
    }  
}
```

Programa com impressão de texto e valor da soma

```
public class Ex1 {  
    public static void main(String[] args) {  
  
        int a=100,b=50;  
        int c=a+b;  
        System.out.println("A soma é igual  "+ c);  
    }  
}
```

Operadores Incremento e Decremento

O código abaixo **não** utiliza operador de atribuição

```
int a = 5;  
a = a + 1;  
System.out.println("a = " + a); // a saída gerada será 6
```

O código abaixo realiza a mesma tarefa do código acima, porém **utiliza** o operador de atribuição de adição

```
int a = 5;  
a += 1;  
System.out.println("a = " + a); // a saída gerada será 6
```

Operadores Incremento e Decremento

Veja a tabela comparativa abaixo

Com operador de atribuição	Sem operador de atribuição
<code>a += 1</code>	<code>a = a + 1</code>
<code>b -= 5</code>	<code>b = b - 5</code>
<code>c *= 3</code>	<code>c = c * 3</code>
<code>d /= 2</code>	<code>d = d / 2</code>
<code>e %= 6</code>	<code>e = e % 6</code>

Operadores Incremento e Decremento

Exemplo da utilização dos operadores pós-fixados:

Incremento

```
int a = 5;  
int b = a++;  
System.out.println("b="+b);  
System.out.println("a="+a);
```

Saída

```
b = 5  
a = 6
```

Decremento

```
int a = 5;  
int b = a--;  
System.out.println("b="+b);  
System.out.println("a="+a);
```

Saída

```
b = 5  
a = 4
```

Operadores Incremento e Decremento

Exemplo da utilização dos operadores pré-fixados:

Incremento

```
int a = 5;  
int b = ++a;  
System.out.println("b="+b);  
System.out.println("a="+a);
```

Saída

b = 6

a = 6

Decremento

```
int a = 5;  
int b = --a;  
System.out.println("b="+b);  
System.out.println("a="+a);
```

Saída

b = 4

a = 4

Operadores Incremento / Decremento

```
public class Ex2 {  
    public static void main(String[] args) {
```

```
        int a= 10 ;
```

```
        int b= 5 ;
```

```
        ++a;
```

```
        b++;
```

```
        System.out.println(a);
```



11

```
        System.out.println(b);
```

6

```
        System.out.println(++a);
```



12

```
        System.out.println(b++);
```

6

```
        System.out.println(a);
```

12

```
        System.out.println(b);
```

7

```
        a=10 ;
```

```
        b=4;
```

```
        int c= ++a + b++;
```

```
        System.out.println(c);
```



15

```
    }
```

```
}
```


Operador Condicional

Operadores de comparação

== igual

!=diferente

> maior >= maior e igual

< menor <= menor e igual

```
public class Ex3 {  
    public static void main(String[] args) {  
        int a=100 , b=50 ;  
        String c= a>b ? "Maior":"Menor";  
        System.out.println(c);  
    }  
}
```

Operador Condicional

```
package testejp;

public class TesteJp {

    public static void main(String[] args) {

        int x=100,y=200;

        int z=x+y;

        String nome=" Jose ";

        nome=nome.trim();

        System.out.println(z+" este e a soma do "+nome);

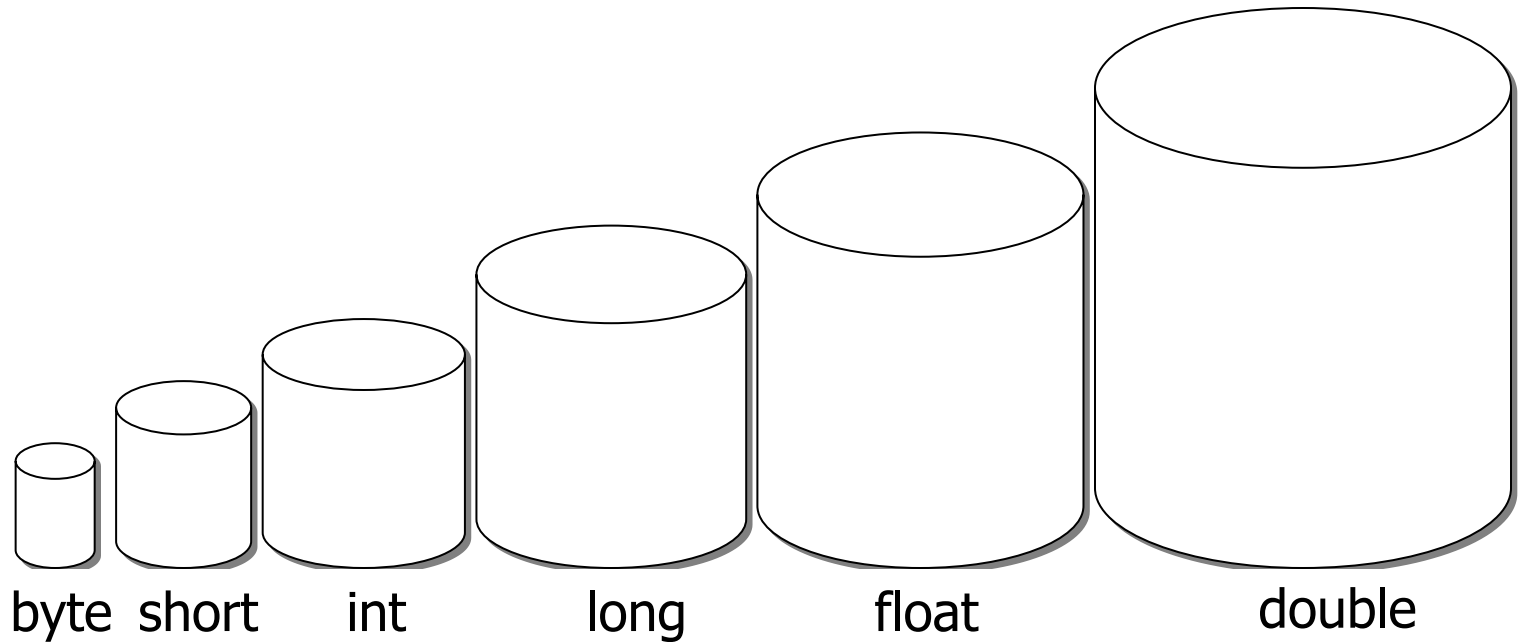
        String resp=x>y ? "x maior que y" : "x e menor que y ";

        System.out.println(resp);

    }

}
```

Tipos Primitivos - Casting



Um byte “cabe” dentro de um int

Um long “cabe” dentro de um float

Um double não “cabe” dentro de um short

Fazendo um Casting

Utilizado para alterar o tipo de dados numérico em tempo de execução de programa.

```
double x = 3.1513;
```

```
int y = x; //Não vai compilar pois y é int e x é double
```

O mesmo erro vai acontecer no próximo caso.

```
int i = 3.1415;
```

No caso abaixo, não compila do mesmo jeito.

```
double d = 10; // o double pode conter um numero inteiro
```

```
int i = d;
```

No caso abaixo , o programa aceita a compilação

```
int i = 5;
```

```
double d2 = i; // um double pode guardar um dado int mas o inverso não é verdadeiro
```

Fazendo um Casting

Para transformar um numero double para int ou um float para int podemos fazer um casting

```
double d = 3.1415;
```

```
int i = (int) d;
```

Com o casting a variável i recebe d como **int**. O valor dela agora é 3.

O mesmo vale para o exemplo, abaixo, com o long

```
long x = 10000;
```

```
int i = (int) x;
```

Uso do parênteses para agrupamento de operações

```
public class TesteString{  
    public static void main(String[] args){  
        int a= 0, b=10, c=5, d=2, e=4, f=3, g=6, h=15, i=13;  
  
        a = (b + (c + (d + e) ) ) + (f + g) + (h + i); //saída: 58  
  
        System.out.println("a = " + a);  
  
        float tt = a * (b + c) + (float)d / (e + f); //saída:870,2857142  
  
        System.out.println("tt = " + tt);  
    }  
}
```

Uso do parênteses para agrupamento de operações

```
float p=10.0f, r=20.0f, q=30.0f, w=5.0f, x=2.0f,  
y=3.0f;
```

```
float z = p * r % q + w / x - y; //saída: 19.5
```

```
System.out.println("z = " + z);
```

```
z = p * r % (q + w) / x - y; // saída: 9.5
```

```
System.out.println("z = " + z);
```

```
z = p * r % (q + w / x - y); //saída: 23
```

```
System.out.println("z = " + z);
```

```
}
```

```
}
```

Comandos de Decisão

```
package aulasgama;
import javax.swing.JOptionPane;
public class ControleRadar {
    public static void main(String[] args) {
        byte velocidade = Byte.parseByte(JOptionPane.showInputDialog("Digite a velocidade"));

        if(velocidade < 80){
            velocidade = 80;
            JOptionPane.showMessageDialog(null, "Agora voce esta a 80Km por hora");
        }
        else{
            velocidade = 80;
            JOptionPane.showMessageDialog(null, "Sua velocidade foi reduzida.\nAgora voce esta a 80Km por hora"); }}}}
```


IF

```
public class Ex5 {  
    public static void main(String[] args) {  
        int a=10, b=8;  
        if (a>b){  
            a=a+8;  
            b=a;  
        }  
        System.out.println(a);  
        System.out.println(b);  
    }  
}
```

IF e Else

```
public class Ex6 {  
    public static void main(String[] args) {  
        int a=10, b=8;  
  
        if (a <= b) {  
            a=a+8;  
            b=a;  
        }  
        else {  
            a=a+1;  
            b=b+10;  
            int z= 100+b;  
        }  
  
        System.out.println(a);  
        System.out.println(b);  
        System.out.println(z);  
    }  
}
```

=== → vai dar erro , pois z só existe no bloco que foi criado

Comandos de Decisão

Estrutura if - else if – else

- Forma para executar uma única instrução:

```
if (<condição 1>) <instrução>;  
else if (<condição 2>) <instrução>;  
else if (<condição 3>) <instrução>;  
...  
else if (<condição n>) <instrução>;  
else <instrução>;
```

If , Else e Else IF

```
public class Ex7 {  
    public static void main(String[] args) {  
        int nota1=7,nota2=5,nota3=8, media=0;  
        String conceito="";  
        media= (nota1+nota2+nota3)/3;  
        if (media>=7)  
            conceito="Aprovado";  
        else if (media>=5 && media<=6)  
            conceito="Exame";  
        else  
            conceito="Reprovado";  
  
        System.out.println("Media = " + media + " Conceito " + conceito);  
    }  
}
```

If , Else e Else IF

```
public class Ex7 {  
    public static void main(String[] args) {  
        int nota1=7,nota2=5,nota3=8, media=0;  
        String conceito="";  
        media= (nota1+nota2+nota3)/3;  
        if (media>=7){  
            conceito="Aprovado";  
        } else if (media>=5 && media<=6){  
            conceito="Exame";  
        } else{  
            conceito="Reprovado";  
        }  
        System.out.println("Media = " + media + " Conceito " + conceito);  
    }  
}
```

As chaves nos if's e Else só são obrigatórias quando existe mais de uma linha de comando nos If's e Else

Comandos de Decisão

```
public class TesteIf{  
    public static void main (String args[]){  
        boolean b1 = false;  
        if (b1)  
            System.out.println ("b1 = " + b1);  
        else if (!b1)  
            System.out.println ("b1 = " + b1);  
  
        int a = 6;  
        int b = 10;  
        if (a < 5 && b > 10)  
            System.out.println("primeiro if");  
        else if ((a > 5 && b > 20))  
            System.out.println("segundo if");  
        else if (a > 4 || b < 15)  
            System.out.println("terceiro if");  
        else  
            System.out.println("nenhum");  
    }  
}
```

Comandos de Bloco

```
public class Ex4 {  
    public static void main(String[] args) {  
        int x=100, y=0;  
        {  
            int a=10,b=5;  
            int c= a+b;  
            y= x+c;  
        }  
        System.out.println(y);  
        System.out.println(c);  
    }  
}
```

==== ➔ vai dar erro , pois c só existe no bloco que foi criado

Operadores Lógicos e Relacionais

Int a=4, b=6, c=10;

Operadores Lógicos. Saída gerada:

- (a > 3) && (b < 8): true
- ((a > 3) && (b > 8)) || (c < 20): true
- (c > 5): true
- !(c > 5): false

Operadores Lógicos e Relacionais

```
public class OperadoresLogicos {  
  
    public static void main(String[] args) {  
  
        int a = 5, b = 7, c = 10;  
  
        boolean b1 = (a > 3) && (b < 8);  
  
        System.out.println("(a > 3) && (b < 8): " + b1);  
  
        boolean b2 = ((a > 3) && (b > 8)) || (c < 20);  
  
        System.out.println("((a > 3) && (b > 8)) || (c < 20): "+b2);  
  
        boolean b3 = (c > 5);  
  
        System.out.println("(c > 5): " + b3);  
  
        boolean b4 = !(c > 5);  
  
        System.out.println("!(c > 5): " + b4);  
  
    }  
}
```

Exercícios

- ✓ Crie um programa que permita ao usuário escolher qual a figura geométrica que deseja calcular a área e o perímetro e ao final informe ao usuário o tipo de figura escolhida e seus valores (área e perímetro). Caso algum dado tenha valor negativo você deve tratar. Para tanto observe a tabela:

Figura	Área	Perímetro
Quadrado	$\text{Lado} * \text{lado}$	$4 * \text{lado}$
Triângulo	$\text{Base} * \text{altura} / 2$	Soma dos lados
Círculo	$\text{PI} * \text{raio} * \text{raio}$	$2 * \text{PI} * \text{raio}$

Entrada e Saída de Dados por JOptionPane

Caixas de diálogo:

- Dados são solicitados para o usuário por intermédio das caixas de diálogo
- Mensagens de erro, informações, alertas e avisos podem ser exibidos para o usuário

A classe `javax.swing.JOptionPane` facilita a tarefa de exibir diálogos padronizados que solicitem algum valor ao usuário ou que exibam alguma informação

Entrada e Saída de Dados por JOptionPane

Métodos da classe JOptionPane para exibir caixas de diálogo:

- showMessageDialog()
 - Caixa de diálogo que apresenta uma mensagem, possibilitando acrescentar ícones de alerta ao usuário
- showConfirmDialog()
 - Caixa de diálogo que, além de emitir uma mensagem, possibilita ao usuário responder a uma pergunta

Entrada e Saída de Dados por JOptionPane

Métodos da classe JOptionPane para exibir caixas de diálogo:

- `showInputDialog()`
 - Caixa de diálogo que, além de emitir uma mensagem, permite a entrada de um texto
- `showOptionDialog()`
 - Caixa de diálogo que abrange os três tipos anteriores

Entrada e Saída de Dados por JOptionPane

`showMessageDialog()`

- Sintaxe:

```
JOptionPane.showMessageDialog(Component, <mensagem>,  
<título da mensagem>, <tipo de mensagem>)
```

Veja no texto o significado de cada um destes argumentos

Entrada e Saída de Dados por JOptionPane

showConfirmDialog()

- Sintaxe:

```
JOptionPane.showConfirmDialog(Component, <mensagem>,  
<título da mensagem>, <tipo de mensagem>)
```

Veja no texto o significado de cada um destes argumentos

Entrada e Saída de Dados por JOptionPane

showInputDialog()

- Sintaxe:

```
JOptionPane. showInputDialog(Component, <mensagem>,  
<título da mensagem>, <tipo de mensagem>)
```

Veja no texto o significado de cada um destes argumentos

Entrada e Saída de Dados por JOptionPane

showOptionDialog()

- Sintaxe:

```
JOptionPane.showOptionDialog(Component, <mensagem>,  
<título da mensagem>, <tipo de mensagem>)
```

Veja no texto o significado de cada um destes argumentos

Exemplo

```
package aulagama;

import javax.swing.JOptionPane;

public class CaixasDialogo {

    public static void main(String[] args) //Onde tudo começa

    {
        String nome;

        int salario;

        int resp;

        nome = JOptionPane.showInputDialog(null, "Digite seu nome");
        JOptionPane.showMessageDialog(null, "Seu nome = " + nome);
        resp = JOptionPane.showConfirmDialog(null, "Deseja Continuar");
        JOptionPane.showMessageDialog(null, "Sua resposta = " + resp);
    }
}
```

Entrada Dados - Swing

```
import javax.swing.JOptionPane;

public class Ex8 {

    public static void main(String[] args) {

        String num1,num2;

        num1 = JOptionPane.showInputDialog("Digite um numero inteiro: ");

        num2 = JOptionPane.showInputDialog("Digite outro numero inteiro: ");

        int n1=Integer.parseInt(num1);

        int n2=Integer.parseInt(num2);

        // float n1=Float.parseFloat(num1);

        // double n1=Double.parseDouble(num1);

        int soma= n1+n2;

        JOptionPane.showMessageDialog(null,"Soma "+soma,"Total",JOptionPane.INFORMATION_MESSAGE);

    }

}
```

<https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

JOptionPane

Símbolos que podem ser usados no `JOptionPane.showMessageDialog`



Ícone	Comando
Pergunta	<code>JOptionPane.QUESTION_MESSAGE</code>
Informação	<code>JOptionPane.INFORMATION_MESSAGE</code>
Alerta	<code>JOptionPane.WARNING_MESSAGE</code>
Erro	<code>JOptionPane.ERROR_MESSAGE</code>
Vazio (somente mensagem)	<code>JOptionPane.PLAIN_MESSAGE</code>

Entrada usando `showInput` com varias opções

```
byte escolha = Byte.parseByte(JOptionPane.showInputDialog("Digite:\n1-Quadrado \n2-Triangulo \n3-Círculo"));
```

JOptionPane

```
import java.util.InputMismatchException;
import javax.swing.JOptionPane;

public class ExemploNextDouble {

    public static void main(String[] args) {

        try{ double num =
Double.parseDouble(JOptionPane.showInputDialog(null, "Entre com um
valor: "));

        JOptionPane.showMessageDialog(null, "num: " + num);}

        catch(InputMismatchException e){

            JOptionPane.showMessageDialog(null, "Valor Incorreto Digite apenas
numeros ");}

        catch(NumberFormatException e){

            JOptionPane.showMessageDialog(null, "Valor Incorreto Digite apenas
numeros ");

        }
    }
}
```

Exercícios

- ✓ Utilizando o JOptionPane , crie um programa que permita ao usuário escolher qual a figura geométrica que deseja calcular a área e o perímetro e ao final informe ao usuário o tipo de figura escolhida e seus valores(área e perímetro). Caso algum dado tenha valor negativo você deve tratar. Para tanto observe a tabela:

Figura	Área	Perímetro
Quadrado	$\text{Lado} * \text{lado}$	$4 * \text{lado}$
Triângulo	$\text{Base} * \text{altura} / 2$	Soma dos lados
Círculo	$\text{PI} * \text{raio} * \text{raio}$	$2 * \text{PI} * \text{raio}$

Estrutura Condicional switch

Sintaxe da estrutura switch:

```
switch (<expressão ou variável>)  
{  
    case <valor 1>:  
        <instruções>;  
        break;  
    case <valor 2>:  
        <instruções>;  
        break;  
    case <valor n>:  
        <instruções>;  
        break;  
    default:  
        <instruções default>;  
}
```

Estrutura Condicional switch

A variável ou expressão que estiver sendo avaliada no switch deverá ser de um dos seguintes tipos primitivos:

- char;
- byte;
- short;
- int

A cláusula case suporta apenas constantes de tipos primitivos compatíveis com a variável ou expressão declarada na cláusula switch


```
import javax.swing.JOptionPane;
public class Ex9 {
    public static void main(String[] args) {
        int intDia;
        String strDia = "";
        intDia = Integer.parseInt(JOptionPane.showInputDialog("Digite um numero(1 - 7):"));
        switch(intDia) {
            case 1 : strDia = "Domingo";
                break;
            case 2 : strDia = "Segunda";
                break;
            case 3 : strDia = "Terca";
                break;
            case 4 : strDia = "Quarta";
                break;
            case 5 : strDia = "Quinta";
                break;
            case 6 : strDia = "Sexta";
                break;
            case 7 : strDia = "Sabado";
                break;
            default: strDia = "Dia invalido";
        }
        System.out.println(strDia);
    }
}
```

Estrutura Condicional switch

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

Estrutura Condicional switch

Saída Gerada:

- a = 2
- a = 3

OBS:

- Quando a instrução case correspondente é encontrada, tudo que está abaixo desta instrução será executado até encontrar uma instrução break ou até o final da estrutura switch.

Estrutura Condicional switch

Recomendações:

- Prefira utilizar a instrução switch ao invés do if-else quando existem várias opções e para cada uma delas apenas uma ou duas instruções serão executadas
- O uso do if-else é recomendado quando o bloco de instruções que deve ser executado for extenso

Exercícios

- ✓ Refaça o exercício de áreas e perímetros agora usando a estrutura switch.
- ✓ Crie um programa que de acordo com a idade do usuário informe sua faixa etária.

Figura	Área	Perímetro
Quadrado	$\text{Lado} * \text{lado}$	$4 * \text{lado}$
Triângulo	$\text{Base} * \text{altura} / 2$	Soma dos lados
Círculo	$\text{PI} * \text{raio} * \text{raio}$	$2 * \text{PI} * \text{raio}$

Exercício - Solução

```
import javax.swing.JOptionPane;
public class AreaPerimetroSwitch {
    public static void main(String[] args) {
        byte escolha = Byte.parseByte(JOptionPane.showInputDialog("Digite:\n1-  
Quadrado \n2-Triangulo \n3-Círculo"));
        switch(escolha){
            case 1:
                float lado = Float.parseFloat(JOptionPane.showInputDialog("Digite o valor  
do lado"));
                JOptionPane.showMessageDialog(null, "Area do Quadrado = " + lado*lado);
                JOptionPane.showMessageDialog(null, "Perímetro do Quadrado = " +  
4*lado);
                break;
```

Estrutura de Repetição for

O laço for é uma estrutura de repetição compacta. Seus elementos de iniciação, condição e incremento são reunidos em forma de um cabeçalho e o corpo é disposto em seguida

- Sintaxe:

```
for (<iniciação>; <condição>; <incremento>)  
    <corpo para o laço for>
```

Estrutura de Repetição for

<iniciação>

- É uma expressão que inicia o loop. Esta operação será realizada uma única vez

<condição>

- A condição (de parada do laço for) deve, obrigatoriamente, ser uma expressão booleana. Quando esta expressão for falsa, o loop será encerrado

<incremento>

- A expressão especificada no incremento será executada a cada ciclo realizado

Estrutura de Repetição for

É importante mencionar que nenhum dos três elementos (iniciação, condição e incremento) é obrigatório na declaração de uma instrução for

Estrutura de Repetição for

```
public class TesteFor{  
    public static void main (String args[]){  
        int cont = 0;  
        for (; cont < 10; cont++){  
            System.out.println("cont = " + cont);  
        }  
    }  
}
```

No exemplo acima a iniciação está sendo feita fora do laço for. Note que após a abertura dos parênteses, foi colocado um ";", que indica que o primeiro elemento esta ausente.

Isto não altera o comportamento desta estrutura de repetição

Estrutura de Repetição for

```
public class TesteFor{  
    public static void main (String args[]){  
        for (int cont = 0; ; cont++){  
            System.out.println("cont = " + cont);  
        }  
    }  
}
```

A condição de parada foi retirada do laço for. Note que foram colocados dois ";" um após o outro e isto indica que o elemento <condição> foi suprimido.

Esta alteração **muda** o comportamento do laço for, pois não existe uma condição de parada, logo o laço for entrará no chamado "loop infinito" e irá imprimir o valor da variável cont por infinitas vezes

LOOP- FOR

```
public class Ex10 {  
    public static void main(String[] args) {  
        int b=0;  
        for (int a=0;a<5;a++){  
            //b=b+a; ou b+=a;  
            b=b+a;  
        }  
        System.out.println("b= "+b);  
    }  
}
```

LOOP- FOR

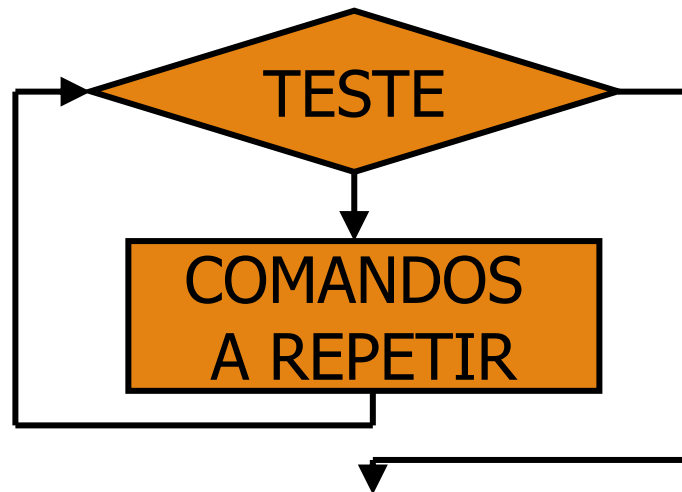
```
public class Ex11 {  
    public static void main(String[] args) {  
        int b=0;  
        for (int a=5;a>0;a--){  
            b=b+a;  
            b++;  
        }  
        System.out.println("b= "+b);  
    }  
}
```

LOOP- FOR

```
public class Ex12 {  
  
    public static void main(String[] args) {  
        for (int a=1;a<5;a=a+10){  
            System.out.println("a= "+a);  
        }  
    }  
}
```

Estrutura de Repetição while

While é a estrutura de repetição utilizada para executar repetidamente uma única instrução ou um bloco delas enquanto uma expressão booleana for verdadeira



Estrutura de Repetição while

Sintaxe:

- Para uma única instrução, não é necessário o par de chaves:

<iniciação>

while (<condição>)

 <instrução + iteração>;

- Para várias instruções é necessário o par de chaves.

<iniciação>

while (<condição>){

 <instrução1>;

 <instruçãoN>;

 <iteração>;

}

Estrutura de Repetição while

Veja o exemplo a seguir onde é utilizada uma instrução de repetição while utilizando um par de chave.

Estrutura de Repetição while

```
public class TesteWhile {  
    public static void main(String[] args) {  
        boolean continuar = true;  
        int valor = 0;  
  
        while (continuar){  
            valor += 2;  
            if (valor > 300)  
                continuar = false;  
        }  
    }  
}
```

Estrutura de Repetição while

O exemplo declara uma variável do tipo boolean "continuar" e outra do tipo int "valor"

A condição de parada do laço while é a variável "continuar" estar em false. Ou seja, enquanto ela estiver em true, o laço while será executado.

Dentro do laço while a variável "continuar" só será alterada para false quando a variável "valor" for maior que 300. E a mesma foi iniciada em zero e está sendo incrementada em duas unidades a cada iteração do laço while.

LOOP WHILE

```
public class Ex13 {  
  
    public static void main(String[] args) {  
        int a=0,b=0;  
        while (a<=5){  
            b=b+a;  
            a++;  
        }  
        System.out.println("b= "+b);  
    }  
}
```

Estrutura de Repetição do-while

A estrutura de repetição do-while é uma variação da estrutura while.

Em uma estrutura while:

- A condição é testada antes da primeira execução das instruções que compõem seu corpo. Desse modo, se a condição for falsa na primeira vez em que for avaliada, as instruções desse laço não serão executadas.

Estrutura de Repetição do-while

Em uma estrutura do-while:

- A condição somente é avaliada depois que seu bloco de instruções é executado pela primeira vez. Assim, mesmo que a condição desse laço seja falsa antes mesmo de ele iniciar, suas instruções serão executadas pelo menos uma vez

Estrutura de Repetição do-while

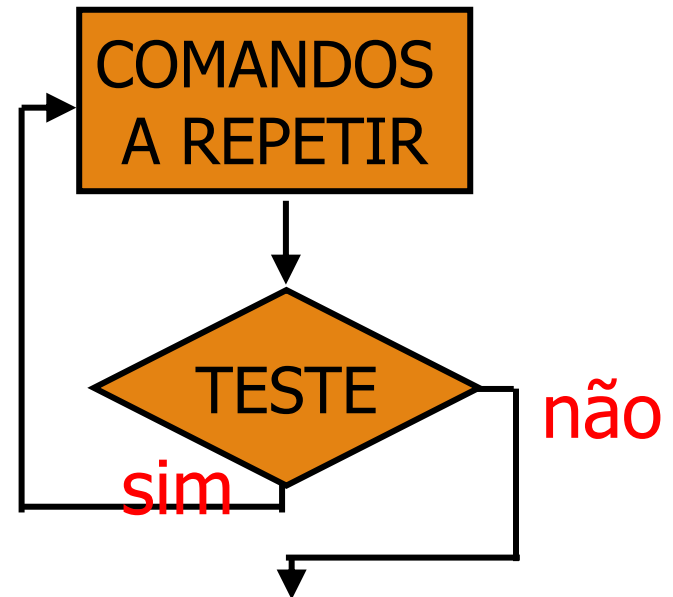
Sintaxe para uma única instrução, não é necessário o par de chaves:

<iniciação>

do

<instrução + iteração>;

while (<condição>);



Estrutura de Repetição do-while

Sintaxe para várias instruções é necessário o par de chaves

<iniciação>

do{

 <instrução1>;

 <instruçãoN>;

 <iteração>;

}while (<condição>);

LOOP WHILE

```
public class Ex14 {  
    public static void main(String[] args) {  
        int a=0,b=0;  
        do {  
            b=b+a;  
            a++;  
        }  
        while (a<=5);  
        System.out.println("b= "+b);  
    }  
}
```

Estrutura de Repetição do-while

```
public class TesteDoWhile {  
    public static void main(String[] args) {  
        boolean continuar = false;  
        int valor = 0;  
  
        do{  
            valor += 2;  
            if (valor > 300)  
                continuar = false;  
        }while (continuar);  
    }  
}
```

Break – usado num for ou while

```
public class Ex15 {  
    public static void main(String[] args) {  
        int a=0,b=0;  
        while(a<=5){  
            b=b+a;  
            a++;  
            if (a==3){  
                System.out.println("b= "+b);  
                break;  
            }  
        }  
        System.out.println("oK");  
    }  
}
```

Exercício com For ou While

Faça um programa de tabuada usando um número qualquer. Por fim pergunte ao usuário se o mesmo deseja fornecer os números novamente. Em caso afirmativo, recomece o programa.

Métodos - void

```
public class Ex16 {  
  
    public static void main(String[] args) {  
        int x=100,y=50,z=0;  
        soma (x,y);  
    }  
  
    static void soma(int a,int b){  
        int c=a+b;  
        System.out.println("Soma= "+ c);  
    }  
}
```

Métodos

```
public class Ex17 {  
    public static void main(String[] args) {  
        int x=100,y=50,z=0;  
        soma (x,y);  
        imprime() ;  
    }  
  
    static void soma(int a,int b){  
        int tot=a+b;  
    }  
  
    static void imprime(){  
        System.out.println("Soma= "+ tot);  
    }  
}
```

A variável tot é local
Portanto , o programa
esta com erro

Métodos

```
public class Ex17 {
```

static int tot; // variavel global que é enxergada em todos os métodos de uma classe

```
    public static void main(String[] args) {  
        int x=100,y=50,z=0;  
        soma (x,y);  
        imprime() ;  
    }
```

```
    static void soma(int a,int b){  
        tot=a+b;  
    }
```

```
    static void imprime(){  
        System.out.println("Soma= "+ tot);  
    }  
}
```

Métodos - return

```
public class Ex18 {  
  
    public static void main(String[] args) {  
        int x=100,y=50;  
        // int z=soma(x,y);  
        //imprime(z) ;  
        imprime(soma(x,y)) ;  
    }  
  
    static int soma(int a,int b){  
        return (a+b);  
    }  
  
    static void imprime(int tot){  
        System.out.println("Soma= "+ tot);  
    }  
}
```


Métodos e Atributos Estáticos

Os métodos **static** são funções que não dependem de fazer nenhuma instância de uma classe . Quando invocados executam um método diretamente da classe.

O programa compilado fica na memória com todos os métodos Estáticos , enquanto o procedimento por instância, só carrega na memória a classe instanciada, no momento que o programa passa a executar a linha que contém a instância.

No corpo de um método estático, **só podem ocorrer membros estáticos.**

Assim sendo, um método estático só pode invocar outro método estático e uma variável que vai ser usada num método estático, tem que ser estática.

```
public class Ex17 {
```

```
    static int tot; // variavel global que é enxergada em todos os métodos de uma classe
```

```
    public static void main(String[] args) {  
        int x=100,y=50,z=0;  
        soma (x,y);  
        imprime() ;  
    }
```

```
    static void soma(int a,int b){  
        tot=a+b;  
    }
```

```
    static void imprime(){  
        System.out.println("Soma= "+ tot);  
    }  
}
```

```
public class Ex22 {  
    public static void main(String[] args) {  
        int a=100,b=50,c=0;  
        c= Calc.soma(a,b);  
        Calc.imprime(c, "Soma");  
        c=Calc.subtrae(a,b);  
        Calc.imprime(c,"Diferença");  
    }  
}
```

A classe Ex22 invoca
métodos da classe Calc
sem fazer a instância

```
public class Calc {  
    public static int soma(int x, int y ) {  
        return (x+y);  
    }  
  
    public static int subtrae(int x, int y ) {  
        return (x-y);  
    }  
  
    public int multiplica(int x, int y ) {  
        return (x-y);  
    }  
  
    public static void imprime(int tot,String tipo){  
        System.out.println(tipo + " = " + tot);  
    }  
}
```

Conceito de classe, atributo, método e construtor

POO(Programação Orientada a Objetos)

- Na Programação Orientada a Objetos (POO) um projeto de sistema está centrado na identificação de objetos
- A partir da identificação dos elementos do mundo real, envolvidos com o problema, é que são realizadas as demais tarefas

Conceito de classe, atributo, método e construtor

POO(Programação Orientada a Objetos)

- Três das principais atividades envolvidas em um projeto orientado a objetos são as seguintes:
 - Identificação dos objetos envolvidos com o sistema a ser desenvolvido e sua representação em forma de classes
 - Identificação de suas características relevantes e sua representação em forma de atributos
 - Identificação de ações (comportamentos) realizadas por esses objetos e sua representação em forma de métodos

Conceito de classe, atributo, método e construtor

Objeto

- Objeto é qualquer entidade do mundo real que apresente algum significado, mesmo que tal entidade não se constitua em algo concreto
- Exemplos de objetos do mundo real: Uma mesa, uma cadeira, um gato, um cachorro

Conceito de classe, atributo, método e construtor

Objeto

- Todo objeto possui algumas características próprias, também chamadas de atributos
- Esses atributos são as qualidades que permitem distinguir um objeto de outros objetos semelhantes
- Por exemplo: É possível distinguir um cachorro dos outros em função de vários atributos: cor, raça, altura, comprimento, peso e etc

Conceito de classe, atributo, método e construtor

Objeto

- Pode haver dois ou mais objetos que possuam alguns atributos com os mesmos valores. Entretanto, em função de sua própria natureza, os valores de alguns outros atributos são únicos e não se repetem entre os diversos objetos semelhantes
- Exemplo: Existe uma infinidade de carros cuja cor é vermelha, mas não pode haver dois carros com a mesma placa ou chassi

Conceito de classe, atributo, método e construtor

Objeto

- Além de possuir atributos, os objetos também manifestam um comportamento
- O comportamento dos objetos diz respeito às ações que eles podem realizar
- Exemplo: O cachorro pode correr, pular, latir e etc. O computador pode ligar-se, desligar-se, processar dados e etc

Conceito de classe, atributo, método e construtor

Classe

- Na POO, o mais importante não é a identificação dos objetos existentes no mundo real
- O que realmente importa é a identificação de grupos de objetos com atributos e comportamentos comuns
- Ter atributos e comportamentos comuns não significa serem iguais

Conceito de classe, atributo, método e construtor

Classe

- Apesar de todos os cachorros possuírem uma cor, raça, altura, comprimento e peso, dois cachorros podem ser bem diferentes por possuírem diferentes valores em cada um desses atributos. Um deles pode ser preto e o outro branco, um pode ter 90cm de altura e o outro ter apenas 70cm e assim por diante

Conceito de classe, atributo, método e construtor

Classe

- Quando um conjunto de objetos possui atributos e comportamentos comuns, significa que eles pertencem a uma mesma categoria
- Exemplo: Se João e Maria são dois clientes de uma empresa, pode-se dizer que, apesar de eles serem pessoas distintas, pertencem a uma mesma categoria: a dos clientes

Conceito de classe, atributo, método e construtor

Classe

- O termo classe é utilizado para significar o mesmo que categoria. Ao invés de dizer que João e Maria pertencem à categoria dos clientes, pode-se dizer que eles pertencem à classe dos clientes

Conceito de classe, atributo, método e construtor

Classe

- Uma classe representa um grupo de objetos com características e comportamentos comuns e compõem-se, basicamente, de atributos e métodos.
 - Os atributos representam as características dos objetos
 - Os métodos representam as ações ou comportamentos que eles podem realizar

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Objeto

- **Definição**: É a instância de uma classe. Ou seja, uma variável do tipo de dado definido pela classe. Possuem características (atributos) comportamentos (métodos)
- Objetos do mundo real possuem características e comportamentos. Já objetos de software armazenam suas características em atributos (variáveis) e implementam seus comportamentos em métodos

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Objeto

- Exemplo:

```
Pessoa joao = new Pessoa();  
joao.idade = 30;  
joao.andar();
```

João é uma variável (objeto) do tipo da classe Pessoa

Idade é um atributo (característica) do objeto João

Andar é um método (comportamento) do objeto João

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Classe

- **Definição**: Representa a abstração de uma entidade (objeto) do mundo real. A Classe define as características e os comportamentos desta entidade (objeto) do mundo real. As características são os atributos e os comportamentos são os métodos

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Classe

- A classe estabelece o formato dos objetos, portanto define um tipo de dado. É a representação de um conjunto de objetos que compartilham as mesmas características e comportamentos
- Veja um exemplo no próximo slide

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

```
public class Pessoa{  
    String nome;  
    int idade;  
    String endereco;  
  
    public void andar() {  
    }  
  
    public void informarIdade() {  
    }  
}
```

João e Maria são objetos da classe Pessoa, pois compartilham os mesmos atributos (características) e métodos (comportamentos)

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Instanciação

- Instanciar uma classe significa criar um novo objeto a partir dela própria. Do mesmo modo que toda variável é de determinado tipo primitivo, todo objeto pertence a uma classe
- Assim como uma variável precisa ser declarada e iniciada, todo objeto precisa ser declarado e instanciado. A declaração de um objeto é muito parecida com a declaração de uma variável, mas o tipo do objeto será sempre uma classe ao invés de um tipo primitivo

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Instanciação

- Sintaxe para declaração de um objeto:
 - `<Classe> <nomeDoObjeto>;`
- Exemplo para declaração de objeto:
 - `String str;`

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Instanciação

- Depois de declarado, todo objeto precisa ser instanciado. A instanciação de um objeto consiste, basicamente, na alocação de um espaço na memória para que ele possa ser representado.
- Sintaxe para instanciar um objeto:
 - `<nomeDoObjeto> = new <construtor>([argumentos]);`

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- O responsável pela instanciação de um objeto sempre existirá um método construtor disponível em sua classe
- **Todo construtor possui o mesmo nome da classe a que se refere** e há um construtor padrão na maioria das classes que dispensa o uso de argumentos

CLASSES - INSTANCIA

Podemos organizar os programas em diversas classes . Cada classe vai virar um objeto que tem características (propriedades/atributos) e métodos diferentes.

Uma classe pode usar os métodos de outras classes, sendo que para isto ela deve instanciar a classe que contem o método desejado.

A instância faz com que um método seja carregado na memória para que os seus métodos possam ser utilizados por outras classes.

A Instancia se faz utilizando o operador new.

Exemplo:

```
Calcula calc = new Calcula();
```

A partir deste ponto os métodos da classe Calcula podem ser acessados através do objeto calc (uma cópia da Classe Calcula que fica na memória RAM).

Exemplo:

```
Calcula calc = new Calcula();
```

```
int x = calc .soma(100,50);
```


CLASSES - INSTANCIA

```
public class Ex19 {  
  
    public static void main(String[] args) {  
  
        int a=100,b=50,c=0;  
  
        Calcula calc=new Calcula();  
  
        c= calc.soma(a,b);  
  
        calc.imprime(c, "Soma");  
  
        c=calc.subtrae(a,b);  
  
        calc.imprime(c,"Diferença");  
    }  
}
```

```
public class Calcula {  
  
    public int soma(int x, int y ){  
        return (x+y);  
    }  
  
    public int subtrae(int x, int y ){  
        return (x-y);  
    }  
  
    public int multiplica(int x, int y ){  
        return (x-y);  
    }  
  
    public void imprime(int tot,String tipo){  
        System.out.println(tipo + " = " + tot);  
    }  
}
```

Modificadores de Acesso de uma Classe

`private` – quando os atributos e métodos de uma classe são `private`, somente estes só podem ser usados dentro da própria classe e não fora desta classe.

`public` – quando os atributos e métodos de uma classe são `public`, estes só podem ser usados dentro da própria classe e por outras classes que necessitem acessar os mesmos.

Modificadores de Acesso de Visibilidade

Os modificadores de acesso visam controlar a acessibilidade dos elementos que compõem a classe. Conseqüentemente, podemos definir que apenas os métodos podem ser chamados, restringindo o acesso direto aos atributos

Modificadores de Acesso de Visibilidade

Modificador public

- Quando utilizamos este modificador em atributos, métodos ou classes o compilador compreende que o atributo, método ou classe pode ser acessado diretamente por qualquer outra classe

Modificadores de Acesso de Visibilidade

Modificador private

- Ao utilizarmos este modificador em atributos ou métodos implica que, para o compilador, estes elementos não podem ser acessados diretamente por qualquer outra classe, exceto por ela mesma
- O modificador private não é aplicável para classes

Modificadores de Acesso de Visibilidade

Modificador protected

- As restrições impostas por esse tipo de modificador de acesso representam um nível intermediário entre o público e o privado. Com este modificador os atributos e métodos são acessíveis somente na própria classe, em classes especializadas e em classes do mesmo pacote
- O modificador protected não é aplicável para classes

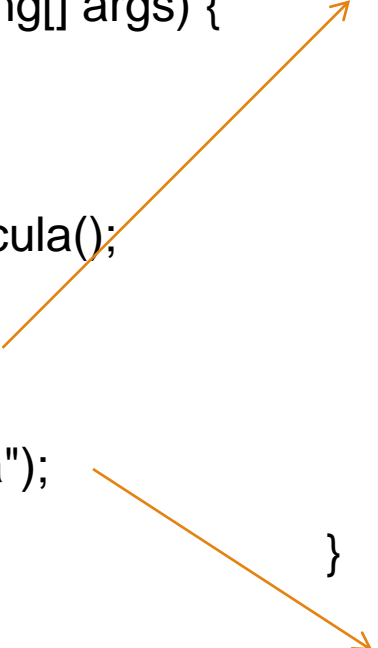
Modificadores de Acesso de Visibilidade

Modificador default

- O modificador default é utilizado quando não se especifica nenhum outro modificador, ou seja, quando não é especificado nenhum modificador, o atributo, método ou classe estão acessíveis apenas no mesmo pacote

```
public class Ex19 {  
  
    public static void main(String[] args) {  
  
        int a=100,b=50,c=0;  
  
        Calcula calc=new Calcula();  
  
        c= calc.soma(a,b);  
  
        calc.imprime(c, "Soma");  
  
        c=calc.subtrae(a,b);  
  
        calc.imprime(c,"Diferença");  
    }  
}
```

```
public class Calcula {  
  
    public int soma(int x, int y ){  
        return (x+y);  
    }  
  
    public int subtrae(int x, int y ){  
        return (x-y);  
    }  
  
    public int multiplica(int x, int y ){  
        return (x-y);  
    }  
  
    public void imprime(int tot,String tipo){  
        System.out.println(tipo + " = " + tot);  
    }  
}
```



Construtores

Numa uma classe sempre é executado um método de instância especial denominado **construtor** quando a mesma é instanciada.

Características dos métodos construtores

- tem o mesmo nome da própria classe
- construtor não retorna dados
- pode haver sobrecarga de métodos de construtores
- construtor não é herdado
- Toda classe tem um construtor e quando não houver um código para o mesmo , será criado internamente um construtor vazio.

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- O responsável pela instanciação de um objeto sempre existirá um método construtor disponível em sua classe
- **Todo construtor possui o mesmo nome da classe a que se refere** e há um construtor padrão na maioria das classes que dispensa o uso de argumentos

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Toda classe tem, pelo menos, um construtor que o compilador adiciona, quando nenhum for declarado (construtor default)
- A instanciação é feita através de uma operação de atribuição em que o objeto recebe o espaço na memória que lhe for reservado pelo construtor

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- O operador new sempre precederá o construtor utilizado nessa operação
- Caso seja utilizado o construtor padrão (default), o nome do construtor será seguido de um par de parênteses sem nada dentro, uma vez que ele não exige nenhum argumento

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Exemplo para instanciar um objeto:
 - `Str = new String();`
- Além de exercerem um papel essencial no processo de instanciação de classes, os construtores também são utilizados para iniciar os atributos com valores padrão ou com valores informados

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Os construtores são métodos especiais que são invocados em conjunto com o operador new para reservar espaço na memória do computador a ser ocupado por um novo objeto

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Algumas classes possuem um construtor padrão que é eficaz para a criação de objetos
- Entretanto, é possível substituir esse construtor padrão por outro diferente, definindo-se parâmetros e adicionando-se quaisquer tipos de instruções que deseje que sejam executadas sempre que um novo objeto for criado

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Sintaxe para a criação de um método construtor:

```
public <nome> ([<parâmetros>]) {  
  
}
```


Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Como diferenciar um construtor de um método qualquer?
- Todo construtor inicia-se com a diretiva de encapsulamento "public" ou default
- Os construtores não possuem um tipo de retorno
- Não podem ser precedidos de quaisquer outros tipos de qualificadores (static, final e etc)
- O nome do construtor deve ser **sempre** igual ao nome da classe

Diferença entre Classe, Objeto e o instanciamento a partir do construtor

Método Construtor

- Há dois procedimentos básicos a serem compreendidos para se trabalhar com construtores:
 - Primeiro: Diz respeito à substituição do construtor padrão por um construtor alternativo
 - Segundo: Faz a inclusão de um construtor alternativo sem que seja eliminado o construtor padrão

Construtor alternativo

Agora vamos criar a classe TestePessoa que irá criar um objeto da classe Pessoa

```
public class TestePessoa {  
  
    public static void main(String[] args) {  
  
        String nome = "Joao";  
        int idade = 30;  
        String endereco = "Av Paulista 900";  
        Pessoa joao = new Pessoa(nome, idade, endereco);  
        System.out.println(joao.nome);  
        System.out.println(joao.idade);  
        System.out.println(joao.endereco);  
    }  
}
```

Construtores

```
class Exemplo {
```

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

```
        biometrico K = new biometrico(20, 'm');
```

```
        K.le();
```

```
    }
```

```
}
```

```
class biometrico {  
    private int peso;  
    private char sexo;
```

```
    biometrico(int x, char y) { // aqui o construtor  
        peso = x;  
        sexo = y;  
    }
```

```
    void le()  
    {System.out.println("PESO="+peso+ " SEXO="+ sexo);  
    }  
}
```

Métodos da classe String

A classe String é uma das classes mais utilizadas em toda a plataforma Java

Um único caracter pode ser representado por um tipo primitivo char

Java não contém tipo primitivo para textos (Strings)

Representação de texto é realizada por meio da classe String

Métodos da classe String

Toda seqüência de caracteres, representada por um objeto da classe String, **é constante** e o texto contido em um objeto dessa classe **não pode ser modificado** depois que ele tiver sido criado

Objetos da classe String são **imutáveis**

Métodos da classe String

Principais métodos da classe String:

- `charAt(int index):`
 - Retorna um char que estiver localizado no índice passado como parâmetro;
- `concat(String str):`
 - Retorna uma String que será a concatenação da string com a string str indicada no parâmetro, contudo não modifica a string original;

Métodos da classe String

- `equals(String str)`:
 - Retorna true se a String de onde o método foi chamado for igual à string str, caso contrário retorna false. Este método leva em consideração maiúsculas e minúsculas;
- `equalsIgnoreCase(String str)`:
 - Este método é igual ao método "equals", porém ignora a diferença entre caracteres maiúsculos e minúsculos;

Métodos da classe String

- `indexOf(String str):`
 - Retorna o índice da primeira ocorrência da String str do parâmetro;
- `lastIndexOf(String str):`
 - Idem ao método `indexOf`, porém retorna o índice da última ocorrência;
- `length():`
 - Retorna o número de caracteres da String;

Métodos da classe String

- `replace (char oldChar, char newChar):`
 - Substitui, na string, todos os caracteres iguais a "oldChar" pelo caracter "newChar", contudo não modifica a string original;
- `startsWith(String prefix):`
 - Retorna true caso a string comece com a String prefix passada como parâmetro;
- `endsWith(String suffix):`
 - Retorna true caso a string termine com a String suffix passada como parâmetro;

Métodos da classe String

- `substring(int begin, int end):`
 - Retorna a string localizada entre o índice `begin` e o índice `end`, incluindo o primeiro caracter e excluindo o último;
- `toLowerCase():`
 - Retorna uma String com todos os seus caracteres minúsculos, mas não modifica a String original;
- `toUpperCase():`
 - Retorna uma String com todos os seus caracteres maiúsculos, mas não modifica a String original;

Métodos da classe String

- `trim()`:
 - Retira os espaços em branco à esquerda e à direita da String, mas não modifica a String;
- Veja na próxima transparência um exemplo da utilização de alguns métodos da classe String. Lembre-se: Objetos da classe String são imutáveis.

Métodos da classe String

```
public class TesteString {  
    public static void main(String[] args) {  
        String st = "Biologia é uma ciência";  
  
        st.concat(" que estuda a vida!");  
        System.out.println(st);  
  
        st.replace('o', 'a');  
        System.out.println(st);  
    }  
}
```

Métodos da classe String

Saída gerada pela execução da classe:

- Biologia é uma ciência
- Biologia é uma ciência

OBS:

- Devemos lembrar que a String é imutável, ou seja, uma vez instanciada com um valor, este valor nunca mais será alterado

Métodos da classe String

Para obter o efeito desejado no programa anterior, devemos fazer as seguintes modificações no código:

- `String st2 = st.concat(" que estuda a vida!");`
- `String st3 = st.replace('o', 'a');`

String - Métodos

Você já deve ter percebido que a maioria dos tipos de variáveis sempre são declarados com letras minúsculas (ex. int, float, double, etc.), porém as variáveis tipo String são declaradas com letra maiúscula. Isso acontece porque String não é um tipo primitivo, é uma Classe, com mais de 50 métodos, e como a maioria das classes, possui atributos, construtores e métodos. E podia ser declarado como:

```
String s = new String();
```

```
String valor = "Curso Java ";
```

```
System.out.println(valor.indexOf("J")); ➔ produz 6 (caracter J na sexta posição)
```

```
System.out.println(valor.toLowerCase()); ➔ produz curso java
```

```
System.out.println(valor.toUpperCase()); ➔ produz CURSO JAVA
```

```
System.out.println(valor.trim()); ➔ produz "Curso Java" tira os brancos a direita
```

```
int tamanho= valor.Length(); ➔ produz 11 ( 11 caracteres inclusive os brancos)
```

```
String valor = "Curso Java ";
```

```
System.out.println(valor.substring(2, 7)); ➔ produz "urso Java"
```


String - Métodos

Fornece a posição de um caractere começando em zero . Se não encontrar , o valor vai ser -1

```
String email=jpico@bol.com.br;
```

```
int i = email.indexOf("@");
```

Transforma em String um número ➔ `valueOf`

```
int a = 11 ;
```

```
long b = 222;
```

```
String S = String.valueOf(a) + " " + String.valueOf(b);
```

```
System.out.println("Conteudo de s: " + S); ➔ produz "11 222"
```

```
String a="Jose" , b="jose";
```

```
if (a.equals(b){
```

```
System.out.println("Iguais");
```

```
else
```

```
System.out.println("Diferentes");
```

String - Métodos

```
public class Exemplo {  
  
    public static void main(String[] args) {  
        String cpf="008.396.155-20";  
        int tam=cpf.lenght();  
        int pos=cpf.indexOf("-");  
        String principal=cpf.substring(0,pos);  
        String digito=cpf.substring(pos+1,tam);  
    }  
}
```

A Classe Math

As operações matemáticas básicas são suportadas pela própria linguagem Java e podem ser realizadas com o uso de operadores

Soma, subtração, multiplicação e divisão são exemplos de operações matemáticas suportadas pela linguagem através de operadores

A Classe Math

Existem diversas operações matemáticas que não encontram suporte na linguagem:

- Potência
- Raiz quadrada
- Logaritmo
- Operações trigonométricas
- Operações de arredondamento

Estas operações são exemplos que não podem ser utilizados exclusivamente com os recursos da linguagem. Para realizá-las, é preciso recorrer aos métodos da classe Math.

A Classe Math

Tipo	Descrição
Localização	<code>java.lang.Math</code>
Qualificador	<code>final</code>
Descrição	Contém métodos úteis para a realização de operações numéricas.

A Classe Math

Atributos

- Note que estes atributos são estáticos, logo o seu conteúdo pode ser recuperado a partir da própria classe:
 - `Math.E;`
 - `Math.PI;`

A Classe Math

Métodos

- A classe Math contém métodos que podem ser utilizados para a realização de operações matemáticas que não são suportadas pela linguagem
- A seguir temos a relação de alguns métodos desta classe:

A Classe Math

Métodos

- `static double abs(double a)`
 - Retorna o valor absoluto do parâmetro "a"
 - OBS: este método foi sobrecarregado com versões para float, int e long
- Exemplos:
 - `Math.abs(23.53); // retorna 23.53`
 - `Math.abs(35); // retorna 35`
 - `Math.abs(-65); // retorna 65`

A Classe Math

Métodos

- `static double ceil(double a)`
 - Retorna um valor double que é maior ou igual ao argumento e é igual a um número matemático inteiro.
- **Exemplos:**
 - `Math.ceil(12.1); // retorna 13.0`
 - `Math.ceil(35.8); // retorna 36.0`
 - `Math.ceil(-3.4); // retorna -3.0`

A Classe Math

Métodos

- `static double floor(double a)`
 - Retorna um valor `double` que é menor ou igual ao argumento e é igual a um número matemático inteiro
- Exemplos:
 - `Math.floor(12.1); // retorna 12.0`
 - `Math.floor(35.8); // retorna 35.0`
 - `Math.floor(-3.4); // retorna -4.0`

A Classe Math

Métodos

- `static double max(double a, double b)`
 - Retorna o maior valor entre os valores passados como parâmetro
 - OBS: este método foi sobrecarregado com versões para float, int e long
- Exemplos:
 - `Math.max(12.2, 35); // retorna 35.0`
 - `Math.max(-35.3, -40.8); // retorna -35.3`
 - `Math.max(100.2, 500.9); // retorna 500.9`

A Classe Math

Métodos

- `static double min(double a, double b)`
 - Retorna o menor valor entre os valores passados como parâmetro
 - OBS: este método foi sobrecarregado com versões para float, int e long
- Exemplos:
 - `Math.min(12.2, 35); // retorna 12.2`
 - `Math.min(-35.3, -40.8); // retorna -40.8`
 - `Math.min(100.2, 500.9); // retorna 100.2`

A Classe Math

Métodos

- static double pow(double a, double b)
 - Retorna o valor de "a" elevado a potência "b"
- Exemplos:
 - `Math.pow(3,2); // retorna 9.0`
 - `Math.pow(-2, 4); // retorna 16.0`
 - `Math.pow(15,2); // retorna 225.0`

A Classe Math

Métodos

- `static double random()`
 - Retorna um valor double positivo, maior ou igual a 0.0 (zero) e menor do que 1.0 (um)
- Exemplo da utilização
 - `Math.random(); // retorna um valor no intervalo $0 \leq x < 1$`

A Classe Math

Gerar valores aleatórios entre quaisquer valores

- O método `random()` gera valores aleatórios no intervalo compreendido entre 0.0 (inclusive) e 1.0
- Para gerar valores entre outros valores devemos utilizar um artifício matemático.

A Classe Math

Métodos

- `static long round(double a)`
 - Arredonda o número informado e retorna-o no formato de um long
- `static int round(float a)`
 - Arredonda o número informado e retorna-o no formato de um int
- Exemplos:
 - `Math.round(5.1); // retorna 5`
 - `Math.round(5.8); // retorna 6`
 - `Math.round(5.4); // retorna 5`
 - `Math.round(5.6); // retorna 6`
 - `Math.round(5.5); // retorna 6`

A Classe Math

Métodos

- `static double sqrt(double a)`
 - Retorna a raiz quadrada do valor passado com parâmetro
- Exemplos:
 - `Math.sqrt(9); // retorna 3`
 - `Math.sqrt(25); // retorna 5`
 - `Math.sqrt(-16); // retorna NaN`

A Classe Math

Gerar valores aleatórios entre quaisquer valores

- Veja a fórmula abaixo:

$$\text{Math.random() * (B - A) + A;}$$

OBS: Esta fórmula gera números aleatórios entre A e B.

Onde: A é o menor valor

B é o maior valor

A Classe Math

Gerar valores aleatórios entre quaisquer valores

- Exemplos:

```
//Gera um número entre 0 (zero) e 150 (cento e cinquenta).
```

- `Math.random()*150;`

```
//Gera um número entre 10 (dez) e 20 (vinte).
```

- `Math.random()*(20 - 10) + 10;`

Array de Tipo de Dados

Os arrays são estruturas de dados que podem armazenar mais que um valor de um mesmo tipo de dado. Enquanto uma variável somente consegue armazenar um único valor, um array pode armazenar um conjunto de valores.

Em Java os arrays são objetos. Isso significa que eles não se comportam como as variáveis e sim como instâncias de classes. Por isso, eles precisam ser declarados, instanciados (criados) e iniciados.

Array de Tipo de Dados

Existem dois tipos de arrays:

- Vetores:
 - Os vetores são arrays unidimensionais
- Matrizes:
 - As matrizes são arrays multidimensionais

Array de Tipo de Dados

Vetores

- Os vetores (arrays unidimensionais) funcionam como arranjos de valores de um mesmo tipo de dado. Você pode imaginar um vetor como uma linha de uma tabela, composta por diversas células. Em cada célula é possível armazenar um valor.

vt[0]	vt[1]	vt[2]	vt[3]
30	35	40	45

Array de Tipo de Dados

Vetores:

- Vamos aprender como realizar quatro operações com vetores:
 - Declarar;
 - Instanciar;
 - Iniciar;
 - Consultar

Array de Tipo de Dados

Vetores

- Declarar um vetor:
 - Podemos declarar um vetor de duas formas distintas, porém o resultado é o mesmo em ambos os casos.

```
<tipo>[] <nomeDoVetor>;
```

```
<tipo> <nomeDoVetor>[];
```

Exemplos:

```
int[] meuArray;
```

```
int meuArray[];
```


Array de Tipo de Dados

Vetores

- Instanciar um vetor:
 - Instanciar é o processo pelo qual você aloca um endereço de memória para um objeto.
 - A instanciação é a criação do vetor. Instanciar um vetor significa lhe atribuir um endereço de memória onde ele possa armazenar seus valores

- **Sintaxe:**

```
<nomeDoVetor> = new <tipo>[<posições>];
```

- **Exemplo:**

```
meuArray = new int[5];
```

Array de Tipo de Dados

Vetores

- Declarar e instanciar um vetor:
 - A declaração e a instanciação de um vetor também podem ser feitas em uma única instrução

- Sintaxe:

```
<tipo>[] <nomeDoVetor> = new <tipo>[<posição>];  
<tipo> <nomeDoVetor>[] = new <tipo>[<posição>];
```

- Exemplo:

```
int[] meuArray = new int[5];  
int meuArray[] = new int[5];
```

Array de Tipo de Dados

Vetores

- Consultar um vetor:
 - Abaixo temos a forma de como devemos consultar ou recuperar valores armazenados em um vetor.

```
int soma=meuArray[0]+meuArray[1]+meuArray[2];
```

```
int v1= meuArray[4];
```

```
System.out.println("Posição 3 do vetor = " +  
meuArray[3]);
```

Array de Tipo de Dados

Vetores

- Declarar, instanciar e iniciar um vetor:
 - Java possibilita ainda declarar, instanciar e iniciar um vetor em uma única instrução.

Sintaxe:

```
<tipo>[] <nomeDoVetor>={<valor0>,<valor1>,...,<valorN>};
```

```
<tipo> <nomeDoVetor>[]={<valor1>,<valor1>,...,<valorN>};
```

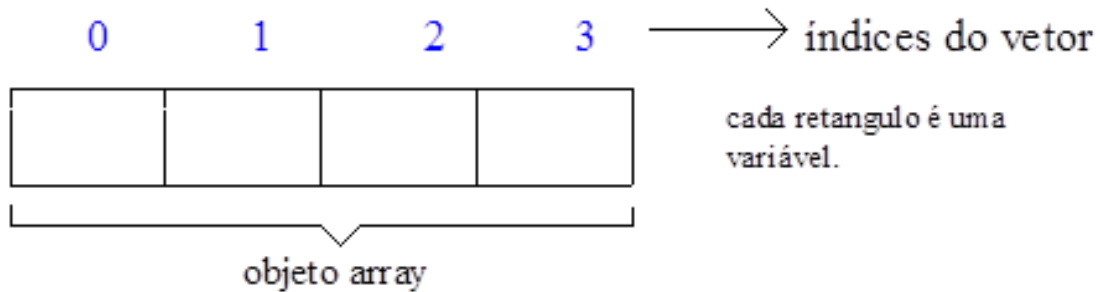
Exemplo:

```
int[] meuArray = {10, 50, 35, 45, 60};
```

```
int meuArray[] = {10, 50, 35, 45, 60};
```

Vetor

Vetor é uma sequência de variáveis em que todas são do mesmo nome e tipo de dados , porem tem um índice interno para poder diferenciar as mesmas.



```
public class Ex21 {  
    public static void main(String[] args) {  
        int m=0;  
        int a[]= new int[5];  
  
        a[0]=20;  
        a[1]=40;  
        a[2]=10;  
        a[3]=80;  
        a[4]=18;  
  
        // ou  int a[]= {20,40,10,80,18};  
        for (int i=0;i<5;i++){  
            if (a[i] > m){  
                m=a[i];  
            }  
        }  
        System.out.println(" Maior " + m);  
    }  
}
```

Matrizes

Matrizes

- As matrizes são arrays multidimensionais. Isso significa que elas podem ter duas ou mais dimensões.
- O uso mais comum de matrizes é para construção de estruturas de dados bidimensionais. Nesse caso, elas são usadas para armazenar valores em forma de linhas e colunas, tal como ocorre em uma tabela

Array de Tipo de Dados

Vetores

- Iniciar um vetor:
 - A iniciação de um vetor pode ser feita posição a posição após sua declaração e instanciação.

- Exemplo:

```
meuArray[0] = 10;
```

```
meuArray[1] = 50;
```

```
meuArray[2] = 35;
```

```
meuArray[3] = 45;
```

```
meuArray[4] = 60;
```


Matrizes

Matrizes

- Exemplo:

	vt[0]	vt[1]	vt[2]
vt[0]	10	20	35
vt[1]	30	60	55
vt[2]	50	80	75

Matrizes

Matrizes

- Assim como os vetores, as matrizes também precisam ser:
 - Declaradas;
 - Instanciadas;
 - Iniciadas;
 - Consultadas
- A sintaxe para realizar estas operações com matrizes é muito similar a sintaxe utilizada para vetores.

Matrizes

Matrizes

- Declarar uma matriz bidimensional:

```
<tipo>[] [] <nomeDaMatriz>;
```

```
<tipo> <nomeDaMatriz>[] [];
```

```
<tipo>[] <nomeDaMatriz>[];
```

- Exemplo:

```
int[] [] matriz;
```

```
int matriz[] [];
```

```
int[] matriz[];
```

Matrizes

Matrizes

- Instanciar uma matriz bidimensional:
 - Após a declaração da matriz é necessário instanciá-la, da mesma forma como é feito com vetores:
- Sintaxe:

`<nomeDaMatriz> = new <tipo>[<linhas>][<colunas>];`

- Exemplo:

```
matriz = new int[3][3];
```

Matrizes

Matrizes

- Declarar e instanciar uma matriz bidimensional:
- Sintaxe:

```
<tipo>[][] <nomeDaMatriz> = new <tipo>[<linha>][<coluna>];
```

```
<tipo> <nomeDaMatriz>[][] = new <tipo>[<linha>][<coluna>];
```

```
<tipo>[] <nomeDaMatriz>[] = new <tipo>[<linha>][<coluna>];
```

- Exemplo:

```
int[][] matriz = new int[3][3];
```

```
int matriz[][] = new int[3][3];
```

```
int[] matriz[] = new int[3][3];
```

Matrizes

Matrizes

- Iniciar uma matriz bidimensional:

```
int matriz[][] = new int[3][3];  
    matriz[0][0] = 30;  
    matriz[0][1] = 54;  
    matriz[0][2] = 62;  
    matriz[1][0] = 12;  
    matriz[1][1] = 25;  
    matriz[1][2] = 39;  
    matriz[2][0] = 50;  
    matriz[2][1] = 37;  
    matriz[2][2] = 91;
```

Matrizes

```
public class Principal {  
    public static void main(String[] args) {  
        int matriz[][] = new int[3][3];  
  
        matriz[0][0] = 30;  
        matriz[0][1] = 54;  
        matriz[0][2] = 62;  
        matriz[1][0] = 12;  
        matriz[1][1] = 25;  
        matriz[1][2] = 39;  
        matriz[2][0] = 50;  
        matriz[2][1] = 37;  
        matriz[2][2] = 91;  
  
        for (int linha = 0; linha < 3; linha++){  
            for (int coluna = 0; coluna < 3; coluna++){  
                System.out.println(matriz[linha][coluna]);  
            }  
        }  
    }  
}
```

Matrizes Multidimensionais

Matrizes multidimensionais

- Todos os conceitos vistos sobre matrizes bidimensionais são aplicáveis a matrizes multidimensionais, logo, caso haja necessidade de se criar uma matriz de mais de duas dimensões, pode-se fazê-lo por analogia às matrizes bidimensionais
- `int m[][] = new int[2][4];`

Matrizes Multidimensionais

Matrizes

- Exemplos:

```
int[][] matriz = {{1,2,3},{1,2,3},{1,2,3}};
```

```
int matriz[][] = {{1,2,3},{1,2,3},{1,2,3}};
```

```
int[] matriz[] = {{1,2,3},{1,2,3},{1,2,3}};
```

Array Multidimensional

Crie um aplicativo em Java que peça ao usuário para preencher uma matriz 3x2 (três linhas e duas colunas) com valores

```
import java.util.Scanner;
public class Principal {

    public static void main(String[] args) {
        int [][] matriz = new int [3][2];
        Scanner entrada = new Scanner(System.in);
        System.out.println("Matriz M[3][2]\n");

        for(int linha=0 ; linha < 3 ; linha++){
            for(int coluna = 0; coluna < 2 ; coluna ++){
                System.out.printf("Insira o elemento M[%d][%d]: ",linha+1,coluna+1);
                matriz[linha][coluna]=entrada.nextInt();
            } }

        System.out.println("\nA Matriz ficou: \n");
        for(int linha=0 ; linha < 3 ; linha++){
            for(int coluna = 0; coluna < 2 ; coluna ++){
                System.out.printf("\t %d \t",matriz[linha][coluna]);
            }
            System.out.println();
        } } }
```

Exercícios - Repetição

- Crie um programa que permita ao usuário criar uma Matriz quadrada na ordem por ele informada, cadastrar números dentro dela e ao final:
 - Imprima os valores da Matriz;
 - Mostre a soma dos elementos da Diagonal Principal;
 - Mostre a soma dos elementos da Diagonal Secundária;
 - Mostre o maior elemento da Diagonal Principal e
 - Mostre o menor elemento da Diagonal Secundária.

	0	1	2		Soma DP = 16
0	5	4	6		Soma DS = 21
1	9	3	14		Maior DP = 8
2	12	17	8		Menor DS = 3

```
class Sobre carga
{
    public static void main (String args[])
    {
        System.out.println("área de um quadrado..." + area(3));
        System.out.println("área de um retangulo.." + area(3,2));
        System.out.println("área de um cubo....." + area(3,2,5));
    }

    public static double area(int x)
    {
        return x * x;
    }

    public static double area(int x, int y)
    {
        return x * y;
    }

    public static double area(int x, int y, int z)
    {
        return x * y * z;
    }
}
```

Exceções

Num programa podem ocorrer três tipos de erros:

- Erro de sintaxe (comando escritos de forma que não obedecem as regras da linguagem)
- Erros de lógica
- Erros não previstos , como acesso a um servidor de banco de dados que esta fora de serviços , acesso a uma classe que não exista no sistema, divisão por zero ,..

Neste ultimo caso, podemos usar bloco com o comando `try` e caso ocorra algum problema com os comandos dentro do bloco, a execução desviará para os blocos “catch” correspondente.

A instrução `try-catch` consiste em um bloco `try` seguido por uma ou mais cláusulas `catch`, que especificam manipuladores para exceções diferentes.

Uso do try-catch-finally

O que são exceções?

- São condições anormais que podem surgir enquanto um programa estiver sendo executado
- Quando ocorrem?
 - Falhas no projeto ou implementação e erros cometidos pelo usuário durante o uso do programa

Uso do try-catch-finally

Exemplo de uma exceção:

- Um determinado programa solicita ao usuário para digitar um número inteiro, porém este usuário informa um valor do tipo "21X", "6f4" ou "t9c5d"
- Conseqüência:
 - Quando este número for convertido para inteiro, irá ocorrer uma exceção: O valor informado não é um número válido

Cabe ao programador prever este tipo de situação e trata-la!

Uso do try-catch-finally

O tratamento de exceções consiste exatamente em prever situações anormais que podem ocorrer e implementar uma solução para a mesma

Essa solução é um caminho alternativo no código para que o problema seja resolvido sem deixar inconsistência e permitir que o programa continue sendo operado

Uso do try-catch-finally

Como tratar uma exceção em Java?

Compreendendo a estrutura try-catch

- try: É utilizado para demarcar um bloco de código que pode gerar algum tipo de exceção
- catch: Oferece um caminho alternativo a ser percorrido no caso de ocorrer efetivamente uma exceção.
- Poderão existir diversos catches para o mesmo try

Uso do try-catch-finally

Sintaxe:

```
try{
    <bloco de instruções protegido>;
}
catch (<tipoDaExcecao1> <variável 1>){
    <Bloco para tratamento da exceção>;
}
catch (<tipoDaExcecao2> <variável 2>){
    <Bloco para tratamento da exceção>;
}
. . .
catch (<tipoDaExcecaoN> <variável N>){
    <Bloco para tratamento da exceção>;
}
```

Uso do try-catch-finally

OBS: O uso das chaves é OBRIGATÓRIO, mesmo que seja utilizada uma única instrução dentro do try ou dentro do catch

Exceções

```
try {  
    // código que inclui comandos/invocações de métodos  
    // que podem gerar uma situação de exceção.  
}  
  
catch (XException ex) {  
    // bloco de tratamento para a situação de exceção  
}  
  
catch (YException ey) {  
    // bloco de tratamento para a situação de exceção  
}  
  
finally {  
    // bloco de código que sempre será executado após o bloco try, independentemente de sua conclusão  
}
```

Exceções

`ArithmeticException`: indica situações de erros em processamento aritmético, tal como uma divisão inteira por 0. A divisão de um valor real por 0 não gera uma exceção (o resultado é o valor infinito);

`NumberFormatException`: indica que tentou-se a conversão de uma string para um formato numérico, mas seu conteúdo não representava adequadamente um número para aquele formato.

`IndexOutOfBoundsException`: indica a tentativa de acesso a um elemento de um vetor sendo que este índice do vetor está aquém ou além dos limites válidos.

`NullPointerException`: indica que a aplicação tentou usar uma referência a um objeto que não foi ainda definida;

`ClassNotFoundException`: indica que a máquina virtual Java tentou carregar uma classe mas não foi possível encontrá-la durante a execução da aplicação.

Uso do try–catch–finally

Erros comuns:

- `ArrayIndexOutOfBoundsException`:
 - Este tipo de exceção acontece quando há a tentativa de recuperar o conteúdo de uma posição de um array utilizando um indexador maior que a última posição (índice) do array
- Veja um exemplo:

Uso do try-catch-finally

```
public class TesteException1{  
    public static void main (String args[]) {  
        int vetor[] = {3,6,9};  
        int index = 3;  
        try{  
            System.out.println("A terceira posicao é: " +  
vetor[index]);  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("A posicao " + index + " não é  
valida");  
        }  
    }  
}
```

Uso do try-catch-finally

Erros comuns:

- `NullPointerException`:
 - Este tipo de exceção acontece quando você tenta acessar um objeto que está nulo (null), ou seja, um objeto que não foi instanciado.
- Veja um exemplo:

Uso do try – catch – finally

```
import javax.swing.JOptionPane;

public class TesteException2{

    public static void main (String args[]){

        String str;

        String mensagem = "Digite a informacao";

        int icone = JOptionPane.INFORMATION_MESSAGE;

        try{

            tr = JOptionPane.showInputDialog(null, mensagem, "msg", icone);

            char c = str.charAt(0);

        }

        catch (NullPointerException e){

            System.out.println("A variavel str está nula (null) ");

        }

    }

}
```

Uso do try-catch-finally

Erros comuns:

- `ArithmeticException`:
 - Este tipo de exceção acontece quando uma condição aritmética excepcional acontece. Por exemplo uma divisão por zero.
- Veja um exemplo:

Uso do try – catch – finally

```
public class TesteException3{  
    public static void main (String args[]){  
        int a = 25;  
        int b = 0;  
  
        try{  
            int c = a / b;  
        }  
        catch (ArithmeticException e){  
            System.out.println("Divisao por zero");  
        }  
    }  
}
```

Uso do try-catch-finally

Erros comuns:

- `NumberFormatException`:
 - Este tipo de exceção acontece quando tentamos converter uma string para um tipo primitivo numérico, porém a string não contém apenas caracteres numéricos.
- Veja um exemplo:

Use do try – catch – finally

```
public class TestException4{  
    public static void main (String args[]){  
        String str = "12x";  
  
        int num = Integer.parseInt(str);  
        System.out.println("Numero " + num);  
    }  
}
```

Uso do try – catch – finally

```
public class TesteException4{  
    public static void main (String args[]){  
        String str = "12x";  
        try{  
            int num = Integer.parseInt(str);  
        }  
        catch (NumberFormatException e){  
            System.out.println("Numero " + str + " inválido");  
        }  
    }  
}
```

Utilizando Diversos catch

```
try{

    String str1 = args[0];

    String str2 = args[1];

    int num1 = Integer.parseInt(str1);

    int num2 = Integer.parseInt(str2);

    double resp = num1 / num2;

    System.out.println("O resultado é: " + resp);

}

catch (ArrayIndexOutOfBoundsException e){

    System.out.println("Devem ser informados 2 argumentos");

}

catch (NumberFormatException e){

    System.out.println("Erro na conversão de string para inteiro");

}

catch (ArithmeticException e){

    System.out.println("Ocorreu uma divisão por zero!");

}
```

Exceções

```
class Excecao1 {
    public static void main (String args[]) {
        String n1="10",n2="0";
        int num1,num2;

        try
        {
            num1 = Integer.parseInt(n1);
            num2 = Integer.parseInt(n2);
            System.out.println("Divisao = " + (num1/num2));
        }
        catch (ArithmeticException e)
        {
            System.out.println("Erro de divisao por zero!");
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Numero de argumentos invalidos!");
        }
        catch (NumberFormatException e)
        {
            System.out.println("Digite apenas numeros inteiros!");
        }
    }
}
```


Exceções Mais Genéricas – Exception e

```
class ExGetMessage
{
    public static void main (String args[])
    {
        int x=10, y=0, z=0;
        try
        {
            z = x / y; //gera uma excecao
        }
        catch (Exception e)
        {
            System.out.println (e.getMessage()); //mostra mensagem do erro
            e.printStackTrace(); //mostra as linhas onde ocorreram erros
        }
    }
}
```

Criando Exceções Personalizadas

Existem diversas situações em que uma exceção não aconteceria pelas regras da linguagem. Mas, de acordo com as regras de negócio da aplicação, uma Exception deveria ocorrer.

Vamos imaginar que o usuário deva entrar com uma idade válida, porém o mesmo digita o número 1000. Podemos criar uma exception chamada `IdadeInvalidaException` que seria lançada sempre que um usuário fornecesse uma idade inválida.

Criando Exceções

```
class Throw {  
    public static void main (String args[]) {  
        try {  
            System.out.print("Como aprender ");  
            throw new Exception("MinhaExcecao"); //gera a exceção  
        }  
        catch (Exception MinhaExcecao) {  
            System.out.print("a linguagem Java?");  
        }  
    }  
}
```

Encapsulamento

O fator mais importante que distingue um módulo bem projetado de um mal projetado é o grau com que o módulo oculta de outros módulos os seus dados internos e outros detalhes da implementação.

Dessa forma, podemos concluir que uma das características essenciais de projetar um certo módulo, é definirmos a sua capacidade de ocultar todos os seus detalhes de implementação. Isso é importante, pois daria aos módulos uma comunicação somente através da sua API sem que um módulo não precise conhecer o funcionamento interno dos outros módulos.

O Encapsulamento oferece um controle de acesso aos atributos e métodos de uma classe. É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.

O Encapsulamento nos oferece a ideia de tornar o software mais flexível, fácil de modificar e de criar novas implementações.

Quando temos uma classe pública com seus métodos sendo diretamente acessados, dizemos que ela não oferece os benefícios do encapsulamento.

Encapsulamento

Porém, existe uma regra bastante clara sobre esse processo utilizando encapsulamento.

Criamos classes públicas contendo campos (variáveis) privados e métodos Acessores (getters) públicos, e caso a classe seja mutável devemos também fornecer os métodos modificadores (setters).

A melhor forma que acessarmos os atributos (variáveis) de uma classe é utilizando métodos.

Os métodos GET e SET são técnicas padronizadas para gerenciamento sobre o acesso dos atributos.

Nesses métodos determinamos quando será alterado um atributo e o acesso ao mesmo, tornando o controle e modificações mais práticas e limpas, sem contudo precisar alterar assinatura do método usado para acesso ao atributo.

Encapsulamento

Na criação dos métodos para acesso a esses atributos privados devemos colocar GET ou SET antes do nome do atributo. Porém, existem diferenças entre os métodos, pois modularizamos um procedimento para método SET e uma função para método GET.

Método get

Quando formos acessar, “pegar” alguns atributos da classe, devemos utilizar os métodos GET. Esse método sempre retornará um valor, seja ele String, int , double etc.

E dentro do método colocamos somente o retorno do atributo.

Encapsulamento

```
public String getNomeProduto() {  
    return nomeproduto;  
}
```

```
public int getQuantidade() {  
    return quantidade;  
}
```

```
public String getValorUnitario() {  
    return valorunitario;  
}
```

Encapsulamento

Método set

Para alterarmos, modificarmos os valores de um atributo da classe de maneira protegida, utilizamos os métodos SET.

Esse método não terá um retorno, pois o atributo será somente modificado, criando um método de tipo VOID, sem retorno.

Porém ele deve receber algum argumento para que possa ocorrer a devida alteração.

Dentro do método, colocamos o atributo da classe recebendo o valor recebido como parâmetro.

Encapsulamento

A utilização da cláusula THIS faz referência ao atributo da classe dentro da qual se está trabalhando.

```
public void setNomeProduto(String nomeproduto) {  
    this.nomeproduto = nomeproduto;  
}
```

```
public void setQuantidade(int quantidade) {  
    this.quantidade = quantidade;  
}
```

```
public void setValorUnitario(String valorunitario) {  
    this.valorunitario = valorunitario;  
}
```

Encapsulamento de atributos

```
public class Data{

    private int ano;

    private byte mes;

    private byte dia;

    public int getAno() {

return ano;

    }

    public void setAno(int a) {

if (a > 0){

    ano = a;

}

else

    System.out.println("Ano invalido");

}

    public byte getDia() {

return dia;

    }

}
```

```
    public void setDia(byte d) {

if (d > 0 && d <= 31){

    dia = d;}

else

    System.out.println("Dia invalido"); }

    public byte getMes() {

return mes;    }

    public void ajustarMes(byte m) {

if (m > 0 && m <= 12){

    mes = m; }

else

    System.out.println("Mes invalido"); }

    public boolean isAnoBissexto(){

if (((ano % 4 == 0) && (ano % 100 != 0)) ||
(ano % 400 == 0))

    return true;

else

    return false; } }
```

Encapsulamento de atributos

Podemos observar pela classe anterior que ao invés de modificar os atributos diretamente, devemos fazê-lo através dos métodos de acesso aos atributos

O nome dos métodos não precisa começar necessariamente com a palavra "set", você pode implementar um método com qualquer outro nome. Como exemplo disto, foi criado o método "ajustarMes", porém o mesmo poderia se chamar "setMes" também

Encapsulamento de atributos

Método is

- São métodos utilizados para leitura de atributos booleanos
boolean isConnected()
boolean isMaiorDeldade()

Vamos analisar o exemplo a seguir e a utilização do encapsulamento dos atributos

Encapsulamento

Muitas vezes os programadores reclamam que é um pouco chato criar os acessores e modificadores para cada um dos campos.

No entanto, as IDEs mais utilizadas já nos oferecem essas facilidades, tornando automática a sua geração. Para ilustrar isso vamos considerar o Eclipse.

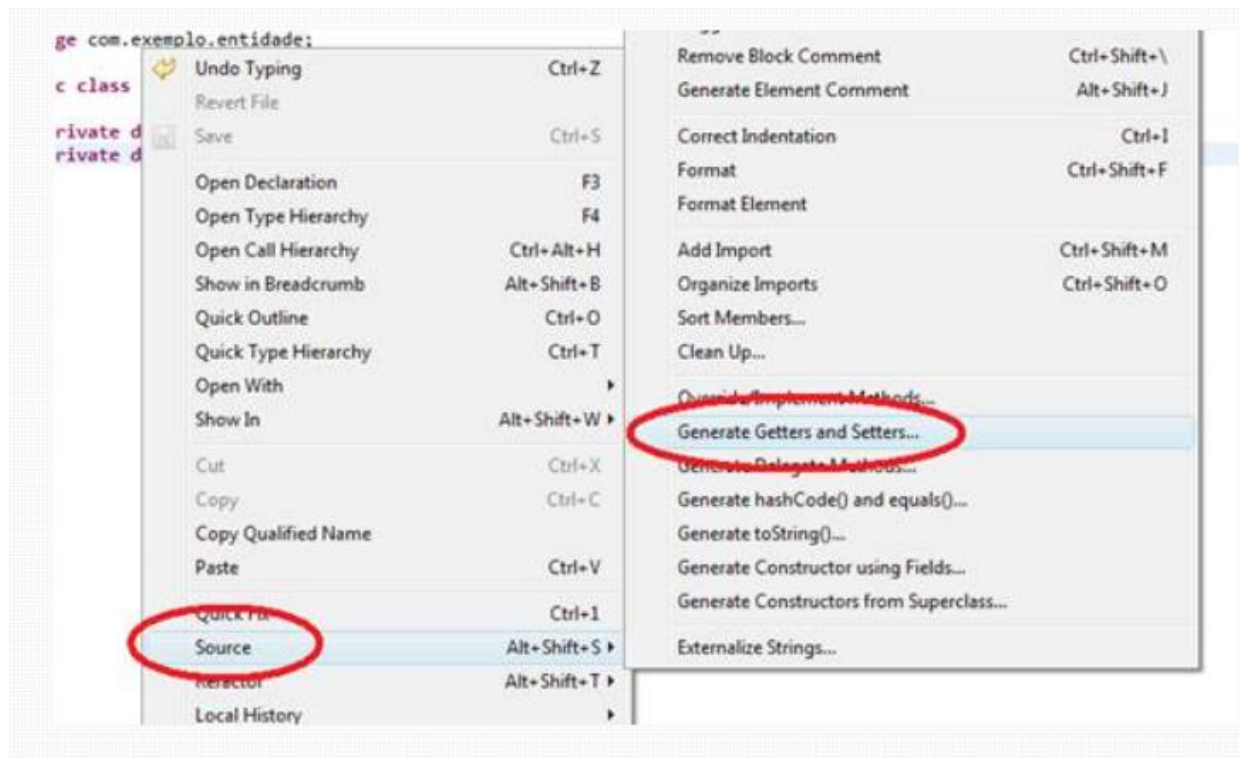
Após definir os campos da nossa classe e marcarmos eles como privados, como demonstra a imagem abaixo, vamos agora gerar os setters (modificadores) e getters (Acessores) da nossa classe.

Classe criada na IDE Eclipse.

```
1 package com.exemplo.entidade;
2
3 public class Ponto {
4
5     private double x;
6     private double y;
7
8 }
9
```

Encapsulamento

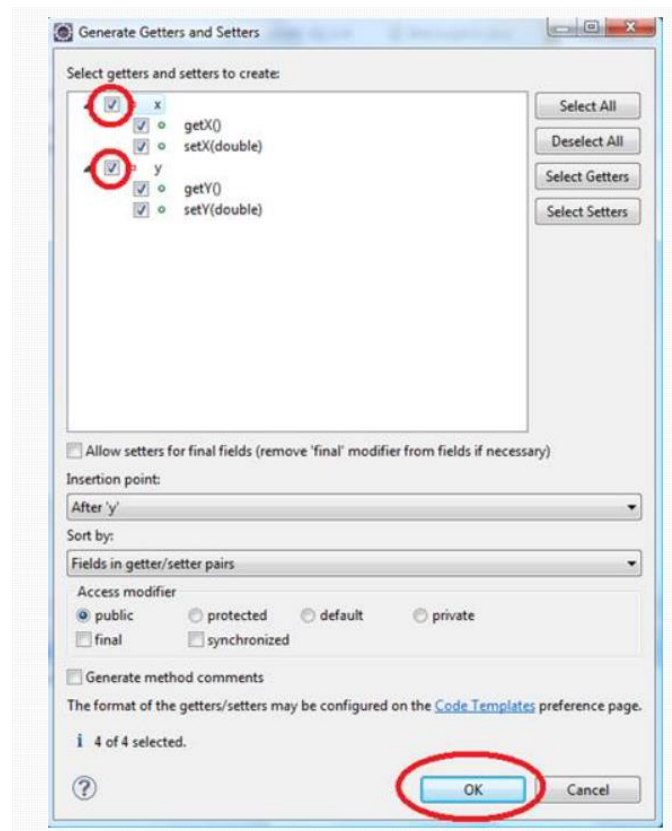
Após criar uma classe simples com dois campos, devemos gerar os setters e getters, para isso clique com o botão direito do mouse dentro da classe que foi criada e seleciona a opção “Source” e por fim “Generate Getters and Setters...” conforme ilustra a imagem abaixo:



Encapsulamento

Na próxima tela aparecerá na janela “Generate Getters and Setters” cada um dos campos da nossa classe criada. Ainda podemos expandir os campos e aparecerá um checkbox para selecionar se desejamos apenas criar setter ou getters.

Selecione o checkbox principal para que ele já selecione automaticamente os outros checkbox e clique em OK. A figura abaixo ilustra o processo:



Encapsulamento

```
import javax.swing.JOptionPane;

public class Principal {

    public static void main(String[] args) {

        String produto = JOptionPane.showInputDialog("Digite o nome do Produto");
        int quantidade = Integer.parseInt(JOptionPane.showInputDialog("Digite a quantidade "));
        float valor= Float.parseFloat(JOptionPane.showInputDialog("Digite o Valor Unitario "));

        Calcula c=new Calcula();
        c.setProduto(produto);
        c.setQuantidade(quantidade);
        c.setValor(valor);

        JOptionPane.showMessageDialog(null,"Total da Compra "+
        c.getTotal(),"Total",JOptionPane.INFORMATION_MESSAGE);

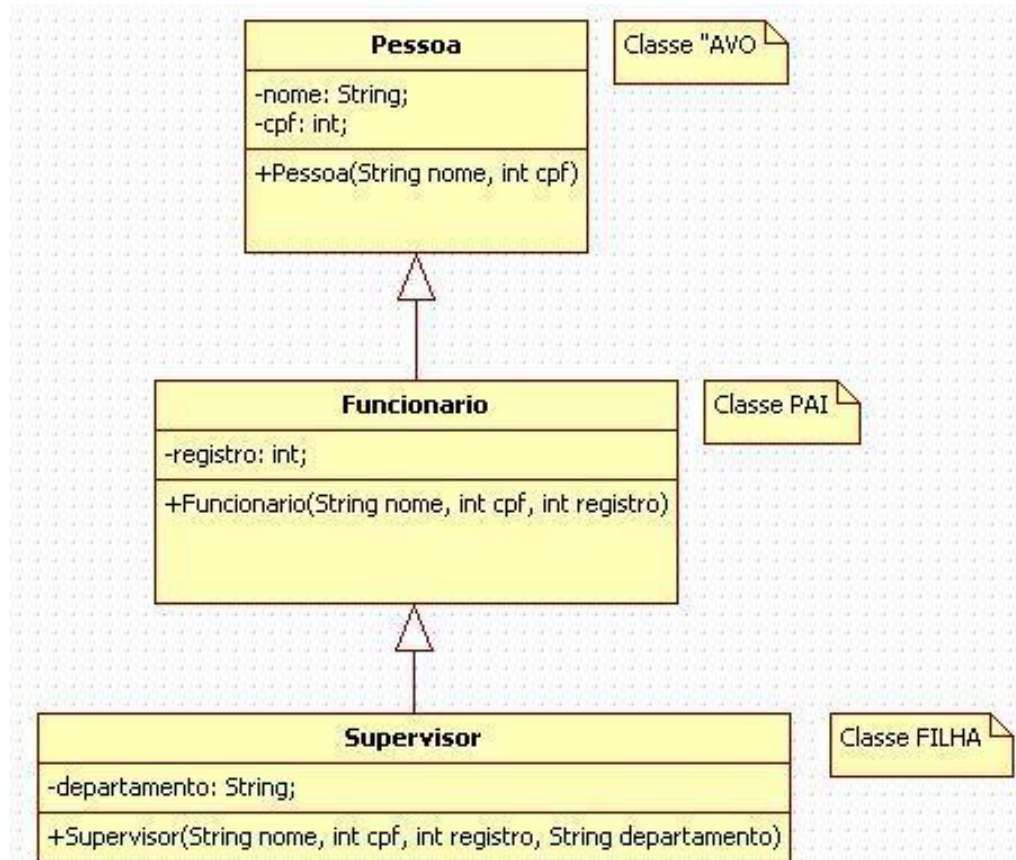
    }
}
```


Código Exemplo Usando Encapsulamento

```
public class Calcula {  
    private String produto;  
    private int quantidade;  
    private float valor;  
    private float total;  
  
    public String getProduto() {  
        return produto;  
    }  
  
    public void setProduto(String  
produto) {  
        this.produto = produto;  
    }  
  
    public int getQuantidade() {  
        return quantidade;  
    }  
  
    public void setQuantidade(int  
quantidade) {  
        this.quantidade = quantidade;  
    }  
  
    public float getValor() {  
        return valor;  
    }  
  
    public void setValor(float valor) {  
        this.valor = valor;  
    }  
  
    public float getTotal() {  
        return quantidade*valor;  
    }  
}
```

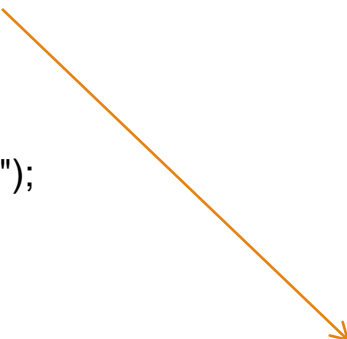
Herança

Uma classe pode Herdar atributos e métodos de outra classe. É como se o código de uma classe fosse reescrito numa outra classe que já ficaria com os seus próprios atributos métodos e mais os atributos e métodos de outra classe.



```
public class Ex20 {  
    public static void main(String[] args) {  
  
        int a=100,b=50,c=0;  
        Calc2 calc=new Calc2();  
        c= calc.soma(a,b);  
        calc.imprime(c, "Soma");  
        c=calc.multiplica(a,b);  
        calc.imprime(c,"Multiplica");  
    }  
}
```

```
public class Calc1 {  
    public int soma(int x, int y ){  
        return (x+y);  
    }  
  
    public int subtrai(int x, int y ){  
        return (x-y);  
    }  
  
    public int multiplica(int x, int y ){  
        return (x-y);  
    }  
  
    public void imprime(int tot,String tipo){  
        System.out.println(tipo + " = " + tot);  
    }  
}  
  
public class Calc2 extends Calc1 {  
    public int multiplica(int x, int y ){  
        return (x*y);  
    }  
  
    public int divide(int x, int y ){  
        return (x/y);  
    }  
}
```



```
public class Ex20 {
    public static void main(String[] args) {

        int a=100,b=50,c=0;
        Calc2 calc=new Calc2();
        c= calc.soma(a,b);
        calc.imprime(c, "Soma");
        c=calc.multiplica(a,b);
        calc.imprime(c,"Multiplica");
    }
}
```

Sobrecarga do método
imprime na herança
Produz o Polimorfismo

Vai ser utilizado o imprime
do Calc2 que é o do filho

```
public class Calc1 {
    public int soma(int x, int y ){
        return (x+y);
    }
    public int subtrai(int x, int y ){
        return (x-y);
    }
    public int multiplica(int x, int y ){
        return (x-y);
    }
    public void imprime(int tot,String tipo){
        System.out.println(tipo + " = " + tot);
    }
}

public class Calc2 extends Calc1 {
    public int multiplica(int x, int y ){
        return (x*y);
    }

    public int divide(int x, int y ){
        return (x/y);
    }

    public void imprime(int tot,String tipo){
        System.out.println("Total  "+ tipo + " = " + tot);
    }
}
```

Sobrecarga de métodos

Dizemos que métodos estão em **sobrecarga (overload)** se eles têm o mesmo nome e pertencem a mesma classe. Para isto, os métodos tem que ter parâmetros com tipos de dados que não sejam iguais de um método para outro. Essa diferença de quantidade de parâmetros e tipo de dados dos parâmetros, chamamos de assinatura de um método.

```
public int multiplica (int x, int y ){  
    return (x*y);  
}
```



Assinatura de um Método
Quantidade de parâmetros e tipos de dados

```
public int multiplica(int x ){  
    return (x*x);  
}
```

Classe Abstrata

A ideia da classes abstrata é representar o conceito de generalização. Dela serão derivadas as classes mais especializadas.

Da classe mais geral não serão instanciados objetos e sim das mais especializadas.

Na classe abstrata, não serão definidos todos os métodos mas eles serão simplesmente declarados e as classes derivadas deverão implementá-los.

Uma classe abstrata possui a palavra reservada **abstract** na declaração da classe.

A palavra reservada **abstract**, também vai ser utilizada prefixando o nome do método, indicando que o método é abstrato.

Em resumo:

Somente classes abstratas podem ter métodos abstratos.

Uma classe abstrata será usada para derivar outras classes dela.

Todos os métodos abstratos de uma classe abstrata tem que ser implementados.

```

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

int a=100,b=10;

Calc c=new Calc();

c.somar(a,b);

c.imprime();
    }
}

public abstract class Ex23 {

    public abstract void somar(int a,int b);
    public abstract void subtrair(int a,int b);

// public abstract void multiplicar(int a,int b);

    void imp(){
        System.out.println("Teste");
    }
}

```

```

public class Calc extends Ex23 {

    int tot=0;

    public void somar(int x,int y){
        tot=x+y;
    }

    public void subtrair(int x,int y){
        tot=x-y;
    }

    void imprime(){
        System.out.println("TOT "+tot);
    }
}

```

Uma classe abstrata pode ter métodos não abstratos e estes não são obrigados a serem implementados.

Classe Calendar

A classe Calendar é uma classe abstrata que provê métodos para conversão entre um instante específico no tempo e um conjunto de atributos tais como ano (YEAR), mês (MONTH), dia (DAY), hora (HOUR) e assim por diante

A classe Calendar possui um método chamado "getInstance", que é utilizado para gerar um objeto (instância) da classe. O método getInstance da classe Calendar retorna um objeto Calendar cujos atributos são iniciados com a data e hora atual

Classe Calendar

Exemplo:

```
Calendar agora = Calendar.getInstance();
```

OBS: Não é possível gerar um objeto da classe Calendar por meio do operador new

Classe Calendar

Obtendo valores para os atributos da classe Calendar:

```
public int get(int field)
```

O método get é utilizado para obter o valor presente (atual) do atributo passado como parâmetro.

```
c.get(Calendar.YEAR) ; // Este comando  
retorna o ano que está presente no atributo YEAR  
da classe Calendar
```

Classe Calendar

Alterando valores para os atributos da classe Calendar:

```
public void set(int field, int value)
```

O método set é utilizado para alterar o valor presente (atual) do atributo passado como parâmetro para o valor passado como parâmetro

```
c.set(Calendar.YEAR, 1990); // Este comando  
altera o valor do atributo ano (YEAR) para 1990
```

Classe Calendar

```
import java.util.Calendar;

public class TesteCalendar {

    public static void main (String args[]){

        Calendar c = Calendar.getInstance();

        System.out.println(c.get(Calendar.YEAR));

        System.out.println(c.get(Calendar.MONTH)+1);

        System.out.println(c.get(Calendar.DAY_OF_MONTH));

        System.out.println(c.get(Calendar.HOUR));

        System.out.println(c.get(Calendar.HOUR_OF_DAY));

        System.out.println(c.get(Calendar.MINUTE));

        System.out.println(c.get(Calendar.SECOND));

        c.set(Calendar.YEAR, 1990);

        c.set(Calendar.MONTH, 11);

        c.set(Calendar.DAY_OF_MONTH, 25);

        c.set(Calendar.HOUR_OF_DAY, 23);

        c.set(Calendar.MINUTE, 0);

        c.set(Calendar.SECOND, 0); }}


```

Interface

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para “obrigar” a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

Para criarmos uma interface em Java, utilizamos a palavra chave `interface` antes do seu nome.

Dentro das interfaces existem somente assinaturas de métodos e propriedades, cabendo à classe que a utilizará realizar a implementação das assinaturas, dando comportamentos práticos aos métodos.

Podemos declarar atributos, porém todos os atributos de uma interface são constantes publicas.

Interface

Quando uma classe implementa uma interface, esta classe obrigatoriamente precisa implementar todos os métodos declarados na interface.

Dentro da interface não podemos:

- implementar método
- construtor
- estender classe
- implementar outra interface

Interface

Para realizar a chamada/referência a uma interface por uma determinada classe, é necessário adicionar a palavra-chave `implements` ao final da assinatura da classe que irá implementar a interface escolhida.

Sintaxe:

```
public class nome_classe implements nome_interface
```

Onde:

- `nome_classe` – Nome da classe a ser implementada.
- `nome_Interface` – Nome da interface a se implementada pela classe.

Interface

Abaixo é possível ver um exemplo de duas interfaces

```
public interface Calc1{  
    public void somar(int a,int b);  
    public void subtrair(int a,int b);  
}
```

```
public interface Calc2{  
    public void multiplicar(int a,int b);  
    public void dividir(int a,int b);  
}
```


Interface

```
class Calc implements Calc1,Calc2{
```

```
int tot=0;
```

```
public void somar(int x,int y){
```

```
tot=x+y;
```

```
}
```

```
public void subtrair(int x,int y){
```

```
tot=x-y;
```

```
}
```

```
public void multiplicar(int x,int y){
```

```
tot=x+y;
```

```
}
```

```
public void dividir(int x,int y){
```

```
tot=x-y;
```

```
}
```

```
void imprime(){
```

```
System.out.println("TOT "+tot);
```

```
}
```

```
}
```

Interface

```
class TestaInterface {  
    public static void main (String args[]){  
        int a=100,b=10;  
        Calc c=new Calc();  
        c.somar(a,b);  
        c.imprime();  
    }  
}
```

Interface Usando Encapsulamento

Abaixo é possível ver um exemplo de uma interface chamada `FiguraGeometrica` com três assinaturas de métodos que virão a ser implementados pelas classes referentes às figuras geométricas.

```
public interface FiguraGeometrica {  
    public String getNomeFigura();  
    public int getArea();  
    public int getPerimetro();  
}
```

A seguir é possível ver duas classes que implementam a interface `FiguraGeometrica`, uma chamada `Quadrado` e outra `Triangulo`.

Interface Usando Encapsulamento

```
public class Quadrado implements FiguraGeometrica {  
    private int lado;  
  
    public int getLado() {  
        return lado;  
    }  
  
    public void setLado(int lado) {  
        this.lado = lado;  
    }  
}
```

Interface Usando Encapsulamento

@Override

```
public int getPerimetro() {  
    int perimetro = 0;  
    perimetro = lado * 4;  
    return perimetro;  
}
```

@Override

```
public String getNomeFigura() {  
    return "quadrado";  
}  
}
```

Interface Usando Encapsulamento

```
public class Triangulo implements FiguraGeometrica {
```

```
    private int base;
```

```
    private int altura;
```

```
    private int ladoA;
```

```
    private int ladoB;
```

```
    private int ladoC;
```

```
    public int getAltura() {
```

```
        return altura;
```

```
    }
```

```
    public void setAltura(int altura) {
```

```
        this.altura = altura;
```

```
    }
```

Interface Usando Encapsulamento

```
public int getBase() {  
    return base;  
}
```

```
public void setBase(int base) {  
    this.base = base;  
}
```

```
public int getLadoA() {  
    return ladoA;  
}
```

```
public int getLadoB() {  
    return ladoB;  
}
```

```
public void setLadoB(int ladoB) {  
    this.ladoB = ladoB;  
}
```

```
public int getLadoC() {  
    return ladoC;  
}
```

```
public void setLadoC(int ladoC) {  
    this.ladoC = ladoC;  
}
```

Interface Usando Encapsulamento

@Override

```
public String getNomeFigura() {  
    return "Triangulo";  
}
```

@Override

```
public int getArea() {  
    int area = 0;  
    area = (base * altura) / 2;  
    return area;  
}
```

@Override

```
public int getPerimetro() {  
    int perimetro = 0;  
    perimetro = ladoA + ladoB + ladoC;  
  
    return perimetro;  
}  
}
```


Interface Usando Encapsulamento

Como é possível ver acima, ambas as classes seguiram o contrato da interface `FiguraGeometrica`, porém cada uma delas a implementou de maneira diferente.

Ao contrário da herança que limita uma classe a herdar somente uma classe pai por vez, é possível que uma classe implemente varias interfaces ao mesmo tempo.

Por fim, interface nada mais que uma espécie de contrato de regras que uma classes deve seguir em um determinado contexto. Como em Java não existe herança múltipla, a interface passa a ser uma alternativa.

Arraylist

Uma classe ArrayList é uma matriz redimensionável, que está presente no pacote java.util. Embora os arrays integrados tenham um tamanho fixo, os ArrayLists podem alterar seu tamanho dinamicamente.

Elementos podem ser adicionados e removidos de um ArrayList sempre que houver necessidade, ajudando o usuário no gerenciamento de memória.

Enquanto num Arraylist podem ser adicionados e removidos elementos num num vetor ou matriz não podem.

ArrayList

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

```
public class Main {
```

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

```
        ArrayList<String> carros = new ArrayList<String>();
```

```
        carros.add("Volvo");
```

```
        carros.add("BMW");
```

```
        carros.add("Honda");
```

```
        carros.add("Hunday");
```

```
        System.out.println(carros);
```

```
    }
```

```
}
```

Arraylist

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

        ArrayList<String> carros = new ArrayList<String>();

        carros.add("Volvo");
        carros.add("BMW");
        carros.add("Honda");
        carros.add("Hunday");

        System.out.println(carros);

        System.out.println(carros.get(2)); // Honda

        carros.set(0, "Toyota"); Alterando a informação do primeiro elemento
        carros.remove(0); // remove o primeiro elemento  carros.clear() limpa todos

        System.out.println(carros.size()); / tamanho do vetor

    }

}
```

ArrayList

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

        ArrayList<String> carros = new ArrayList<String>();

        carros.add("Volvo");

        carros.add("BMW");

        carros.add("Honda");

        carros.add("Hunday");

        for (int i = 0; i < carros.size(); i++) {

            System.out.println(carros.get(i));

        }

    }

}
```

ArrayList

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

        ArrayList<String> carros = new ArrayList<String>();

        carros.add("Volvo");

        carros.add("BMW");

        carros.add("Honda");

        carros.add("Hunday");

        for (String i : carros) {

            System.out.println(i);

        }

    }

}
```

Arraylist

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {

        ArrayList<Integer> idade = new ArrayList<Integer>();

        idade.add(10);

        idade.add(15);

        idade.add(20);

        idade.add(25);

        for (int i : idade) {

            System.out.println(i);

        }

    }

}
```

ArrayList

```
import java.util.ArrayList;

import java.util.Collections;

public class Main {

    public static void main(String[] args) {

        ArrayList<Integer> idade = new ArrayList<Integer>();

        idade.add(10);

        idade.add(15);

        idade.add(20);

        idade.add(25);

        idade.add(55);

        idade.add(2);

        Collections.sort(idade); // Sort myNumbers

        for (int i : idade) {

            System.out.println(i);

        }

    }

}
```


LinkedList

A classe LinkedList é quase idêntica à ArrayList

```
import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {

        LinkedList<String> carros = new LinkedList<String>();

        carros.add("Volvo");

        carros.add("BMW");

        carros.add("Honda");

        carros.add("Hunday");

        for (String i : carros) {

            System.out.println(i);

        }

    }

}
```

LinkedList

A classe LinkedList é quase idêntica à ArrayList

A classe LinkedList é uma coleção que pode conter muitos objetos do mesmo tipo, assim como a ArrayList.

A classe LinkedList tem todos os mesmos métodos da classe ArrayList porque ambas implementam a interface List. Isso significa que você pode adicionar itens, alterar itens, remover itens e limpar a lista da mesma maneira.

No entanto, embora a classe ArrayList e a classe LinkedList possam ser usadas da mesma maneira, elas são construídas de maneira muito diferente.

A classe ArrayList possui um array regular dentro dela. Quando um elemento é adicionado, ele é colocado na matriz. Se o array não for grande o suficiente, um novo array maior é criado para substituir o antigo e o antigo é removido.

A LinkedList armazena seus itens em "contêineres". A lista possui um link para o primeiro container e cada container possui um link para o próximo container na lista. Para adicionar um elemento à lista, o elemento é colocado em um novo contêiner e esse contêiner é vinculado a um dos outros contêineres da lista.

LinkedList

É melhor usar uma **ArrayList** quando:

- Você deseja acessar itens aleatórios com frequência
- Você só precisa adicionar ou remover elementos no final da lista

É melhor usar uma **LinkedList** quando:

- Você só usa a lista percorrendo-a em vez de acessar itens aleatórios
- Você frequentemente precisa adicionar e remover itens do início, meio ou fim do List

LinkedList

Para muitos casos, o ArrayList é mais eficiente, pois é comum precisar de acesso a itens aleatórios na lista, mas o LinkedList fornece vários métodos para fazer certas operações com mais eficiência:

`addFirst ()` Adiciona um item ao início da lista.

`addLast ()` Adiciona um item ao final da lista

`removeFirst ()` Remove um item do início da lista.

`removeLast ()` Remove um item do final da lista

`getFirst ()` Pega o item no início da lista

`getLast ()` Pega o item no final da lista

LinkedList

```
import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {

        LinkedList<String> carros = new LinkedList<String>();

        carros.add("Volvo");

        carros.add("BMW");

        carros.add("Honda");

        carros.add("Hunday");

        for (String i : carros) {

            System.out.println(i);

        }

    }

}
```

HashMap

O ArrayList armazenam itens como uma coleção ordenada e você deve acessá-los com um número de índice (tipo int).

Um HashMap, entretanto, armazena itens em pares "chave / valor" e você pode acessá-los por um índice de outro tipo (por exemplo, uma String).

Um objeto é usado como uma chave (índice) para outro objeto (valor). Ele pode armazenar diferentes tipos: chaves de string e valores inteiros ou o mesmo tipo, como: chaves de string e valores de string:

```
import java.util.HashMap;
```

```
HashMap<String, String> capitais = new HashMap<String, String>();
```

HashMap

```
import java.util.HashMap;

public class Main {

    public static void main(String[] args) {

        HashMap<String, String> capitais = new HashMap<String, String>();

        // Aqui a chave vai ser Pais e Capital

        capitais.put("Inglaterra", "London");

        capitais.put("Alemanha", "Berlin");

        capitais.put("Brasil", "Brasília");

        capitais.put("USA", "Washington DC");

        System.out.println(capitais);

        System.out.println(capitais.get("Brasil"));

        capitais.remove("Alemanha"); // para limpar todos os itens capitais.clear();

        System.out.println(capitais.size());

    } }
```

HashMap

Para listar os dados através de um loop pelos itens de um HashMap , use o método `keySet ()` se quiser apenas as chaves e use o método `values ()` se quiser apenas os valores

```
import java.util.HashMap;

public class Main {

    public static void main(String[] args) {

        HashMap<String, String> capitais = new HashMap<String, String>();

        capitais.put("Inglaterra", "London");

        capitais.put("Alemanha", "Berlin");

        capitais.put("Brasil", "Brasília");

        capitais.put("USA", "Washington DC");

        for (String i : capitais.values()) {

            System.out.println(i); }

        for (String i : capitais.keySet()) {

            System.out.println("key: " + i + " value: " + capitais.get(i)); }

        } }
```


HashMap

```
import java.util.HashMap;

public class Main {

    public static void main(String[] args) {

        HashMap<String, Integer> pessoas = new HashMap<String, Integer>();

        // keys e values (Nome, Idade)

        pessoas.put("Jorge", 32);

        pessoas.put("Ana", 30);

        pessoas.put("Bianca", 33);


        for (String i : pessoas.keySet()) {

            System.out.println("key: " + i + " value: " + pessoas.get(i));

        }

    }
}
```

HashSet

Um HashSet é uma coleção de itens em que cada item é único e pode ser encontrado no pacote java.util. Neste caso, se formos listar os itens a BMW só vai aparecer uma única vez.

```
import java.util.HashSet;

public class Main {

    public static void main(String[] args) {

        HashSet<String> carros = new HashSet<String>();

        carros.add("Volvo");

        carros.add("BMW");

        carros.add("Ford");

        carros.add("BMW");

        carros.add("Mazda");

        System.out.println(carros);

        System.out.println(carros.contains("Mazda")); // checa se um item existe (true/false)

    } }
```

Java Iterator

A interface `Iterator` é utilizada para navegar dentro de coleções, como `ArrayList` e `HashSet`.

A interface `Iterator`, por sua vez, possui o método `hasNext()` para identificar se existem mais elementos na coleção.

Outro método importante desta classe é aquele utilizado para obter o próximo elemento da coleção e que possui a seguinte assinatura: `Object next()`

Java Iterator

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

```
import java.util.Iterator;
```

```
public class Main {
```

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

```
        ArrayList<String> carros = new ArrayList<String>();
```

```
        carros.add("Volvo");
```

```
        carros.add("BMW");
```

```
        carros.add("Ford");
```

```
        carros.add("Mazda");
```

```
        Iterator<String> it = carros.iterator();
```

```
        while(it.hasNext()) {
```

```
            System.out.println(it.next());
```

```
        }
```

```
    } }
```

Java Iterator

```
import java.util.ArrayList;  
import java.util.Iterator;
```

```
public class Main {  
    public static void main(String[] args) {  
        ArrayList<Integer> numeros = new ArrayList<Integer>();  
        numeros.add(12);  
        numeros.add(8);  
        numeros.add(2);  
        numeros.add(23);  
        Iterator<Integer> it = numeros.iterator();  
        while(it.hasNext()) {  
            Integer i = it.next();  
            if(i < 10) {  
                it.remove();  
            }  
        }  
        System.out.println(numeros);  
    }  
}
```

Java Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class TestIterator {

    public static void main(String[] args) {

        String nome1 = "Carlos";
        String nome2 = "Josias";
        String nome3 = "Marcos";
        String nome4 = "Armando";

        ArrayList nomes = new ArrayList();
        nomes.add(nome1);
        nomes.add(nome2);
        nomes.add(nome3);
        nomes.add(nome4);

        Iterator iterator = nomes.iterator();

        int iCont = 1;

        String nome;

        while (iterator.hasNext()){

            nome = (String) iterator.next();

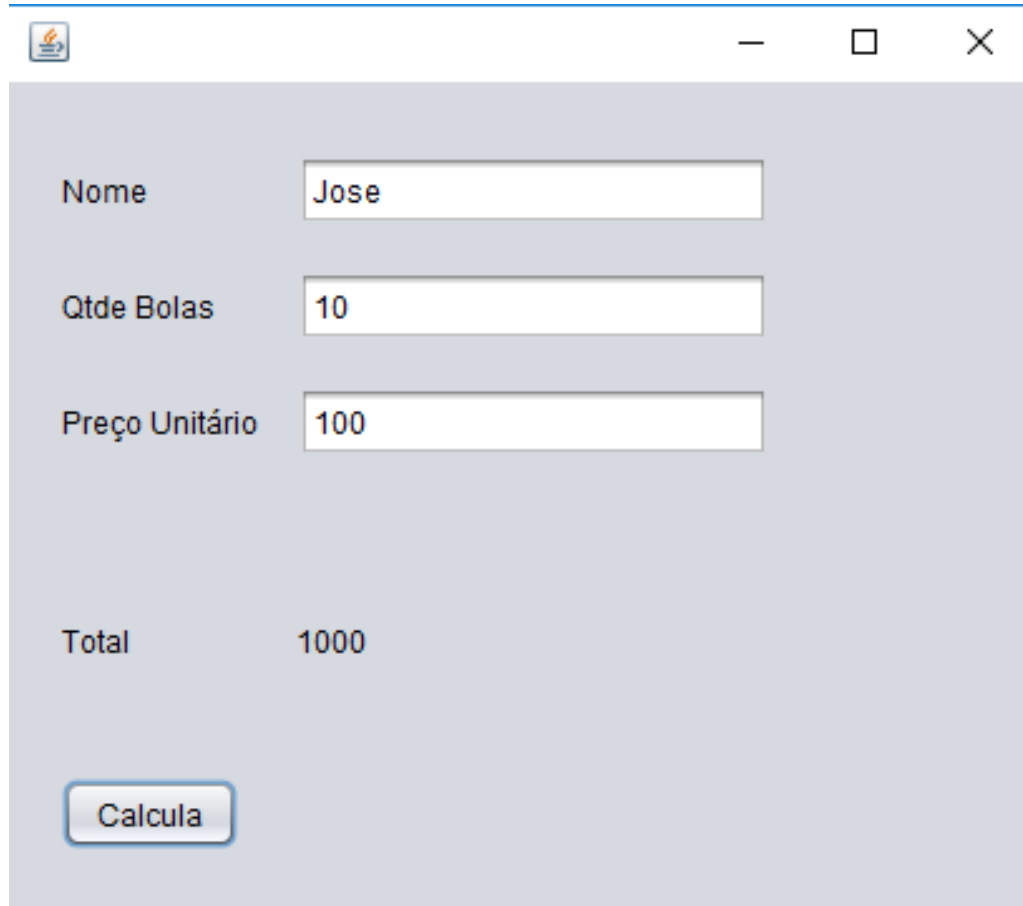
            System.out.println("nome[" + iCont++ + "]
= " + nome);

        }

    }

}
```

Criando Formulário no Netbeans

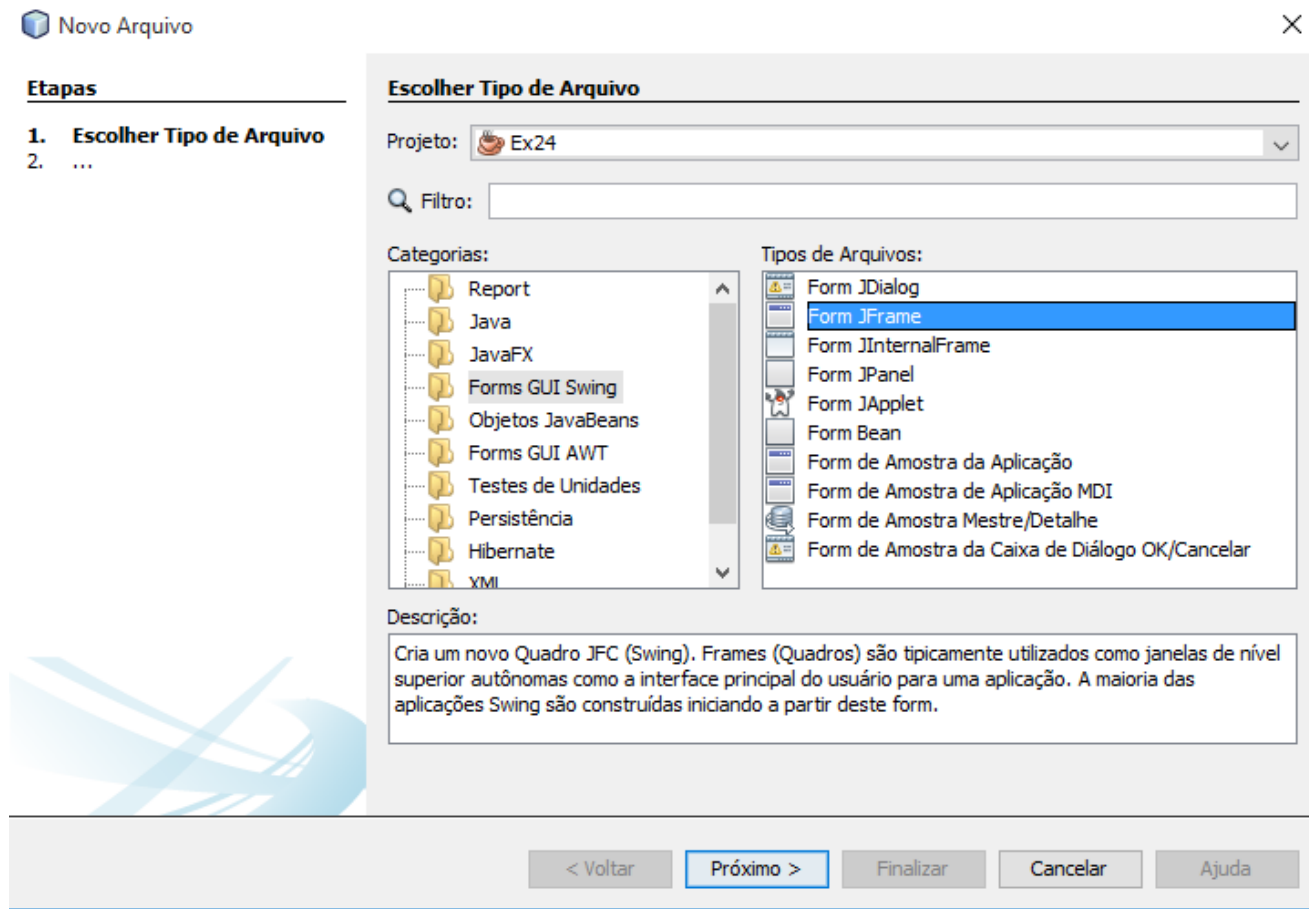


A screenshot of a Java Swing window titled "Calculadora" (Calculator). The window has a standard title bar with a maximize button, a close button, and a small icon on the left. The main content area is light gray and contains a form with the following elements:

- A label "Nome" followed by a text input field containing the text "Jose".
- A label "Qtde Bolas" followed by a text input field containing the text "10".
- A label "Preço Unitário" followed by a text input field containing the text "100".
- A label "Total" followed by the text "1000".
- A button labeled "Calcula" located at the bottom left of the form area.

Criando Formulário no Netbeans

- Crie um projeto novo no Netbeans , desmarcando a opção de criar a classe
- Adicione um Arquivo Novo

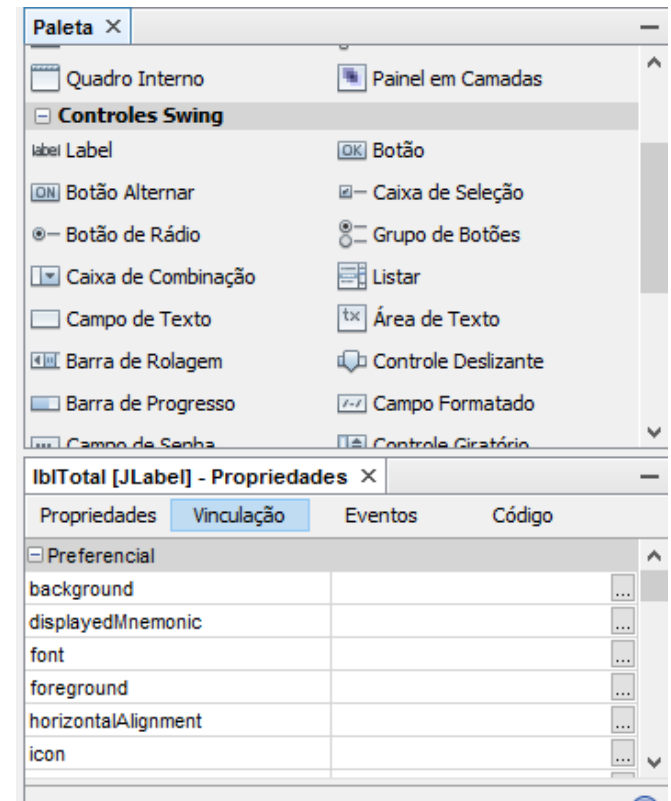
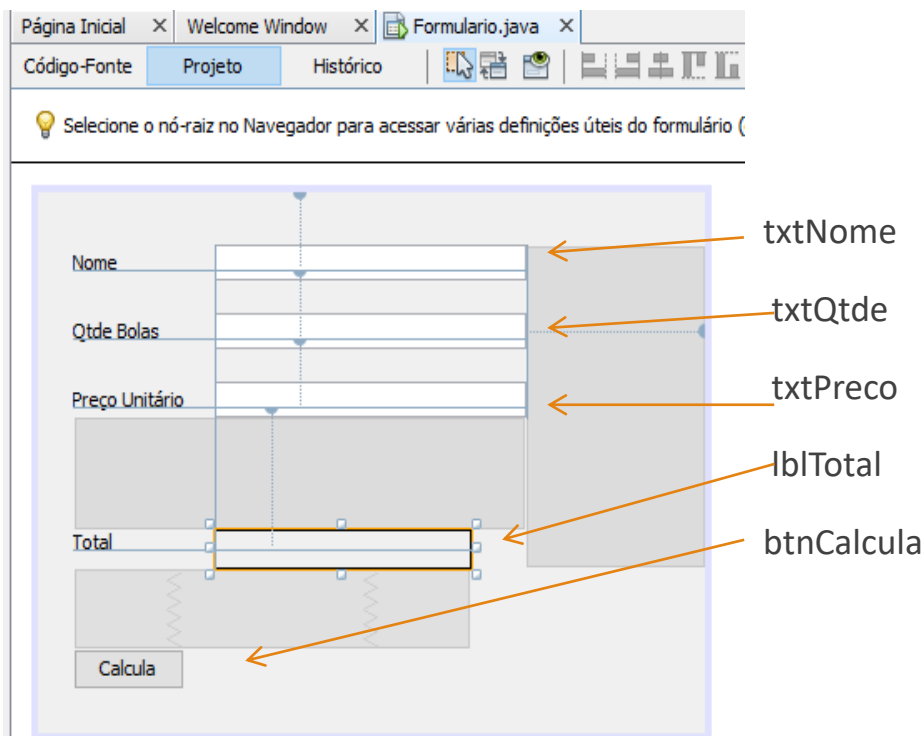


Criando Formulário no Netbeans

- Adicione os Controles Label , Campo Texto e Botão

Limpe os campos na propriedade Text de Cada um e renomei cada controle, clicando com o botão direito do mouse em cada um deles , renomeando para

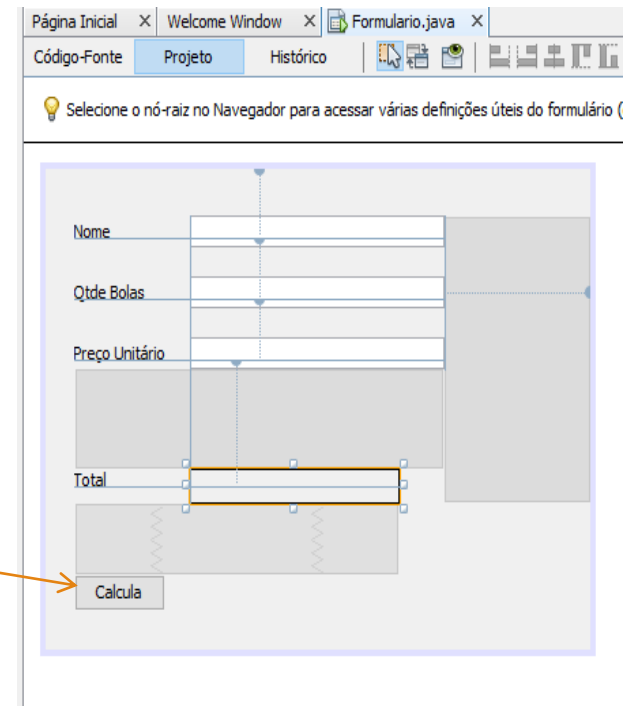
txtNome, txtQtde, txtPreco , lblTotal, btnCalcula



Criando Formulário no Netbeans

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
  
    int qtde=Integer.parseInt(txtQtde.getText());  
    int prunit=Integer.parseInt(txtPreco.getText());  
    int tot= qtde*prunit;  
    lblTotal.setText(Integer.toString(tot));  
}
```

De um duplo click no botão Calcula e adicione o código desejado



Abre outro Form

De um duplo click no botão Abre Form e adicione o código desejado

```
private void btnAbreActionPerformed(java.awt.event.ActionEvent evt) {  
    Consulta frame = new Consulta();  
    frame.setVisible(true);  
}
```

The image shows a Java Swing window with a light gray background. It contains three input fields with labels: 'Nome', 'Qtde Bolas', and 'Preço Unitário'. Below these is a 'Total' label and an empty input field. At the bottom, there is a horizontal bar containing two buttons: 'Calcula' and 'Abre Form'. The 'Abre Form' button is highlighted with an orange border and has a small blue square icon next to it, indicating it is selected or being edited. A dashed line connects the 'Abre Form' button to the code block above.

Usando o Swing

O **Swing é um framework** que disponibiliza um conjunto de elementos gráficos para ser utilizado na plataforma Java.

O Swing é compatível com o **Abstract Window Toolkit (AWT)**, mas trabalha de forma totalmente diferente.

A **API Swing**, diferente do **AWT**, não delega a tarefa de renderização ao sistema operacional, ele renderiza os elementos por conta própria.

Como a AWT é uma biblioteca de baixo-nível que depende de código nativo da plataforma ela traz alguns problemas de compatibilidade entre as plataformas, fazendo com que nem sempre o programa tenha a aparência desejada em todos os sistemas operacionais. Além disso, o **Swing** é mais completo e os programas têm uma aparência muito parecida, independente do sistema operacional que está sendo utilizado, possui uma enorme gama de controles extras disponíveis, tais como áreas de texto que nativamente podem mostrar conteúdo como RTF ou **HTML**, botões com suporte a imagens, sliders, selecionadores de cores, alteração do tipo de borda para os componentes, maior controle de como desenhar os mínimos detalhes de apresentação e muito mais. No entanto, a performance é um pouco pior devido a alta abstração, consumindo assim mais memória RAM.

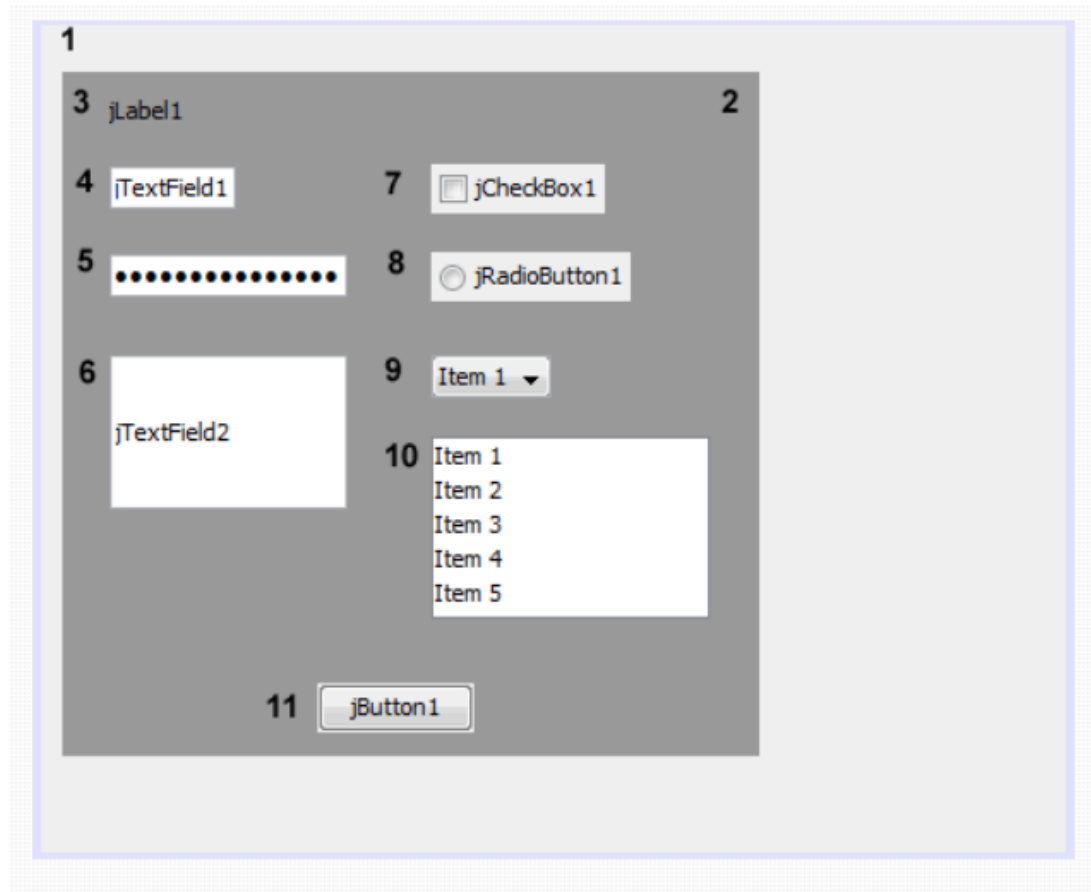
Principais Componentes do Swing

- **JFrame** representa a janela do programa com barra de título, ícone, botões de comando, etc. Entre os principais métodos temos o *pack()* que compacta a janela para o tamanho dos componentes, *setSize(int, int)* que define a largura e altura da janela, *setLocation(int, int)* que define a posição da janela na tela (x,y), *setBounds(int, int, int, int)* que define posição e tamanho, *setVisible(boolean)* que exibe a janela e *setDefaultCloseOperation(int)* que define o que ocorre quando o usuário tenta fechar a janela (as opções são: *DO_NOTHING_ON_CLOSE*, *HIDE_ON_CLOSE*, *DISPOSE_ON_CLOSE*, *EXIT_ON_CLOSE*).
- **JPanel** representa um tipo básico de container para inserção de componentes. Entre os principais métodos temos *add(Component, int)* que adiciona o componente definindo sua posição e *setLayout(LayoutManager)* que altera o tipo de layout.
- **JLabel** representa um rótulo de texto. Entre os principais métodos temos o *setText(String)* que altera o texto e *getText()* que retorna o texto atual.
- **JTextField** representa um campo de texto onde o usuário pode informar um texto em uma linha. Entre os principais métodos temos *setText(String)* que altera o texto e *getText()* que retorna o texto atual.
- **JPasswordField** representa um campo de texto protegido, subclasse de `JTextField`. O principal método é o *setEchoChar(char)* que define o caractere que aparece ao digitar um texto.

Principais Componentes do Swing

- **JTextArea** representa uma caixa onde o usuário pode informar várias linhas de texto. Entre os principais métodos temos o *setText(String)* que altera o texto, *getText()* que retorna o texto atual, *getSelectedText()* que retorna o texto selecionado pelo usuário e *insert(String, int)* que insere um texto na posição especificada.
- **JCheckBox** representa uma caixa de seleção e permite selecionar ou não uma opção. Entre os principais métodos temos o *setSelected(boolean)* que altera o estado da caixa de seleção e o método *isSelected()* que retorna *true* se a caixa estiver marcada e *false* se não estiver marcada.
- **JRadioButton** representa um componente que permite selecionar uma entre diversas opções. O *JRadioButton* é semelhante ao *JCheckBox*, inclusive com os mesmos construtores e métodos.
- **JComboBox** representa uma caixa de combinação, da qual o usuário pode selecionar uma opção. Entre os principais métodos temos o *addItem(Object)* que adiciona um item à lista de opções, *setEditable(boolean)* que permite ao usuário digitar uma opção, *getSelectedIndex()* que retorna a posição do item atualmente selecionado, *getSelectedItem()* que retorna o texto do item atualmente selecionado, *setSelectedIndex(int)* que seleciona o item da posição especificada e *setSelectedItem(Object)* que seleciona o objeto especificado na lista.
- **JList** representa uma lista de opções que permite a seleção de mais de um item simultaneamente. Entre os principais métodos temos o *setListData(Object[])* que preenche ou altera os itens de uma lista, *getSelectedValues()* que retorna um array de objetos contendo itens selecionados na lista.
- **JButton** representa um botão destinado a executar uma ação. Entre os principais métodos temos o *setText(String)* que altera o texto do botão e *setIcon(Icon)* que altera o ícone do botão.

Principais Componentes do Swing



Existem ainda diversos outros componentes Swing como o JSlider, JProgressBar, JToolBar, JTabbedPane entre outros.

Usando o Swing via comandos

```
import javax.swing.*;

import java.awt.event.*;

import java.awt.*;

JLabel lblNome = new JLabel("Nome");

JLabel lblEndereco = new JLabel("Endereco");

JTextField txtNome = new JTextField("");

JTextField txtEndereco = new JTextField("");

JButton cmdIncluir = new JButton("Incluir");

JButton cmdBuscar = new JButton("Buscar");

JLabel lblMensagem = new JLabel("Status: Incluir");
```

Colocando um Rotulo e um campo de Entrada de dados

Trabalhando com eventos no Swing

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

O ActionListener é o comando do Swing que colocamos códigos que seja executado num evento qualquer.

```
private class eBuscar implements ActionListener{  
  
    public void actionPerformed(ActionEvent e){  
  
        Categorias c = new Categorias();  
  
        c.setidcategoria(txtCliente.getText());  
  
        c.buscar();  
  
        txtEndereco.setText(c.getEndereco());  
  
        lblMensagem.setText(c.getStatus());  
  
    }  
  
}
```

Posicionando elementos na Tela

Os gerenciadores de layout são usados para decidir automaticamente a posição e o tamanho dos componentes adicionados. Na ausência de um gerenciador de layout, a posição e o tamanho dos componentes devem ser definidos manualmente.

O método `setBounds ()` é usado em tal situação para definir a posição e o tamanho. Para especificar a posição e o tamanho dos componentes manualmente, o gerenciador de layout do quadro pode ser nulo.

`setBounds ()`

O método `setBounds ()` precisa de quatro argumentos. Os primeiros dois argumentos são as coordenadas `x` e `y` do canto superior esquerdo do componente, o terceiro argumento é a largura do componente e o quarto argumento é a altura do componente.

```
setBounds(int x-coordinate, int y-coordinate, int width, int height)
```

Posicionando elementos na Tela

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class FlowLayoutExample {
    FlowLayoutExample(){
        JFrame frame = new JFrame("Flow Layout");
        JButton button,button1, button2, button3,button4;
        button = new JButton("button 1");
        button1 = new JButton("button 2");
        button2 = new JButton("button 3");
        button3 = new JButton("button 4");
        button4 = new JButton("button 5");
        frame.add(button);
        frame.add(button1);
        frame.add(button2);
        frame.add(button3);
        frame.add(button4);
        frame.setLayout(new FlowLayout());
        frame.setSize(300,300);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        new FlowLayoutExample();
    }
}
```

Posicionando elementos na Tela

```
import javax.swing.*;

import java.awt.*;

public class SetBoundsTest {

    public static void main(String arg[]) {

        JFrame frame = new JFrame("SetBounds");

        frame.setSize(375, 250);

        frame.setLayout(null);

        JButton button = new JButton("Hello Java"); // Criando um botão

        // Posicionando o botão e definindo tamanho

        button.setBounds(80,30,120,40);

        frame.add(button);

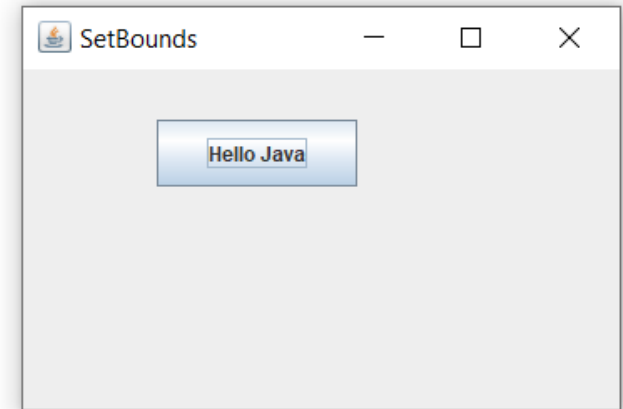
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);

    }

}
```



Usando Label, Text, Botões e Evento

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

import javax.swing.*;

public class Teste extends JFrame implements ActionListener

{

    JLabel    texto    = new JLabel("Graus Celcius: ");

    JTextField caixaTexto = new JTextField("Fahrenheit");

    JButton    botao    = new JButton("Calcular");


    public Teste()

    {

        this.setSize(300,150);

        //define tamanho para janela

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Para de executar ao fechar o programa ^
```

Posicionando elementos na Tela

```
this.setLayout(null);
```

```
//null Layout, permite colocar os componentes em qualquer lugar da janela
```

```
caixaTexto.setBounds(10,10,250,20);
```

```
//define as coordenadas da JTextField (posição X, posição Y, Largura, Altura)
```

```
this.add(caixaTexto);
```

```
//Adiciona o JLabel no JFrame
```

```
botao.setBounds(100,40,90,20);
```

```
//define as coordenadas do JButton (posição X, posição Y, Largura, Altura)
```

```
botao.addActionListener(this);
```

```
//Adiciona evento para este botão
```

```
this.add(botao);
```

```
//Adiciona o JButton no JFrame
```

Posicionando elementos na Tela

```
texto.setBounds(10, 70, 250, 20);
```

```
//define as coordenadas da JLabel (posição X, posição Y, Largura, Altura)
```

```
this.add(texto);
```

```
//Adiciona o JLabel no JFrame
```

```
this.setVisible(true);
```

```
//Deixa a janela visível
```

```
}
```

```
public void actionPerformed(ActionEvent e)
```

```
{
```

```
    try
```

```
    {
```

```
        // obtêm a entrada de usuário a partir da JTextField
```

```
        String firstNumber = caixaTexto.getText();
```

Posicionando elementos na Tela

```
// converte os valores recebido em string para double

double number1 = (double)Double.parseDouble(firstNumber);

//faz um cast para para um tipo primitivo

// conversão de gaus fahrenheit para graus celsius

//double sum = 5/9*number1 - 32;

double sum = (5.0/9.0)*(number1-32.0);

//Cuidar os pontos, senão ele indentifica com int e deixa o resultado como ZERO

//setar o resultado na JLabel

//Concatenar com uma String, pois não pode-se colocar diretamente o valor int

texto.setText("Graus Celcius: " + sum + "°C");

}

catch(Exception ex)

{

    JOptionPane.showMessageDialog(this, "Caracteres Inválidos no Campo JTextField");

    //Caso não for possivel converter o valor digirto, ele envia esta mensagem na tela

} }
```


Posicionando elementos na Tela

```
int x = ..., y = ..., w = ..., h = ...;
```

```
JComponent x = ...;
```

```
x.setSize(w, h);
```

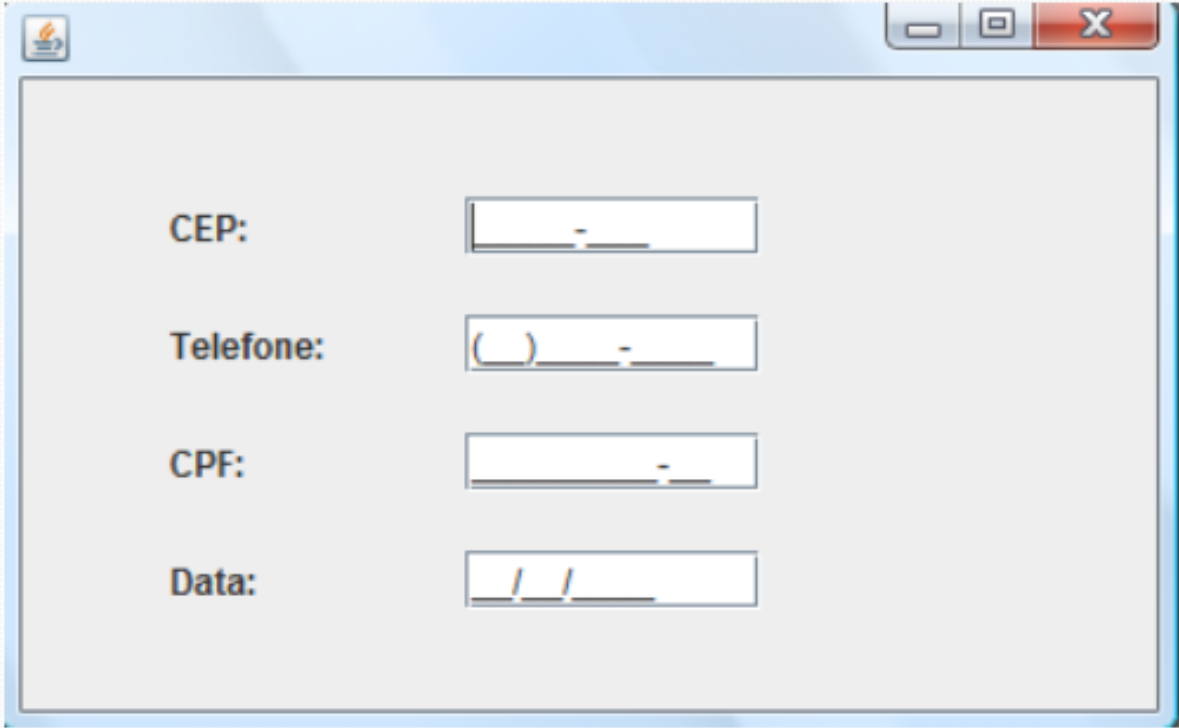
```
x.setLocation(x, y);
```

O `setBounds` é uma forma de definir a posição e o tamanho ao mesmo tempo, enquanto que o `setSize` define apenas o tamanho e o `setLocation` apenas a posição.

o `JFrame` é uma janela onde vc coloca diversos objetos nele e depois coloca o `JPanel` no `JFrame`

`JPanel` é um container que voce usa dentro do `JFrame`.

Campos com Mascaras



A screenshot of a web form with four input fields, each with a specific mask. The form is displayed in a window with a blue border and standard window controls (minimize, maximize, close) in the top right corner. The labels for the fields are in bold black text.

Label	Mask
CEP:	____-____
Telefone:	(____) ____-____
CPF:	____-____-____
Data:	____/____/____

Campos com Mascaras

```
import java.text.ParseException;

import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.text.MaskFormatter;

public class TestandoJTextField extends JFrame {

    private static final long serialVersionUID = 1L;

    public static void main(String[] args)
    {
        TestandoJTextField field = new TestandoJTextField();
        field.testaJFormattedTextField();
    }
}
```

Campos com Mascaras

```
private void testaJFormattedTextField() {  
    Container janela = getContentPane();  
    setLayout(null);  
  
    //Define os rótulos dos botões  
    JLabel labelCep = new JLabel("CEP: ");  
    JLabel labelTel = new JLabel("Telefone: ");  
    JLabel labelCpf = new JLabel("CPF: ");  
    JLabel labelData = new JLabel("Data: ");  
    labelCep.setBounds(50,40,100,20);  
    labelTel.setBounds(50,80,100,20);  
    labelCpf.setBounds(50,120,100,20);  
    labelData.setBounds(50,160,100,20);  
  
    //Define as máscaras  
    MaskFormatter mascaraCep = null;  
    MaskFormatter mascaraTel = null;  
    MaskFormatter mascaraCpf = null;  
    MaskFormatter mascaraData = null;
```

Campos com Mascaras

```
try{
    mascaraCep = new MaskFormatter("#####-###");
    mascaraTel = new MaskFormatter("(##)#####-####");
    mascaraCpf = new MaskFormatter("#####-##");
    mascaraData = new MaskFormatter("##/##/####");
    mascaraCep.setPlaceholderCharacter('_');
    mascaraTel.setPlaceholderCharacter('_');
    mascaraCpf.setPlaceholderCharacter('_');
    mascaraData.setPlaceholderCharacter('_');
}
catch(ParseException excp) {
    System.err.println("Erro na formatação: " + excp.getMessage());
    System.exit(-1);
}
```

Campos com Mascaras

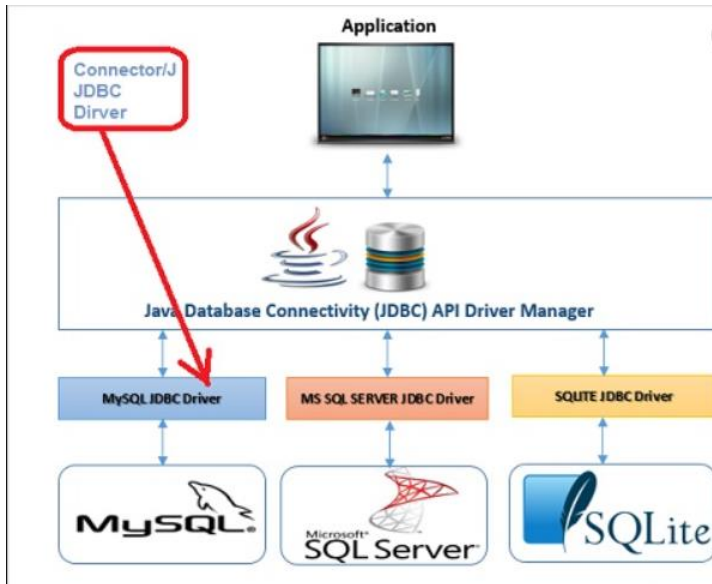
```
//Seta as máscaras nos objetos JFormattedTextField
    JFormattedTextField jFormattedTextCep = new
JFormattedTextField(mascaraCep);
    JFormattedTextField jFormattedTextTel = new
JFormattedTextField(mascaraTel);
    JFormattedTextField jFormattedTextCpf = new
JFormattedTextField(mascaraCpf);
    JFormattedTextField jFormattedTextData = new
JFormattedTextField(mascaraData);
    jFormattedTextCep.setBounds(150,40,100,20);
    jFormattedTextTel.setBounds(150,80,100,20);
    jFormattedTextCpf.setBounds(150,120,100,20);
    jFormattedTextData.setBounds(150,160,100,20);
```

Campos com Mascaras

```
//Adiciona os rótulos e os campos de textos com máscaras na tela
    janela.add(labelCep);
    janela.add(labelTel);
    janela.add(labelCpf);
    janela.add(labelData);
    janela.add(jFormattedTextCep);
    janela.add(jFormattedTextTel);
    janela.add(jFormattedTextCpf);
    janela.add(jFormattedTextData);
    setSize(400, 250);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
}

}
```

Para implementar o CONECTOR

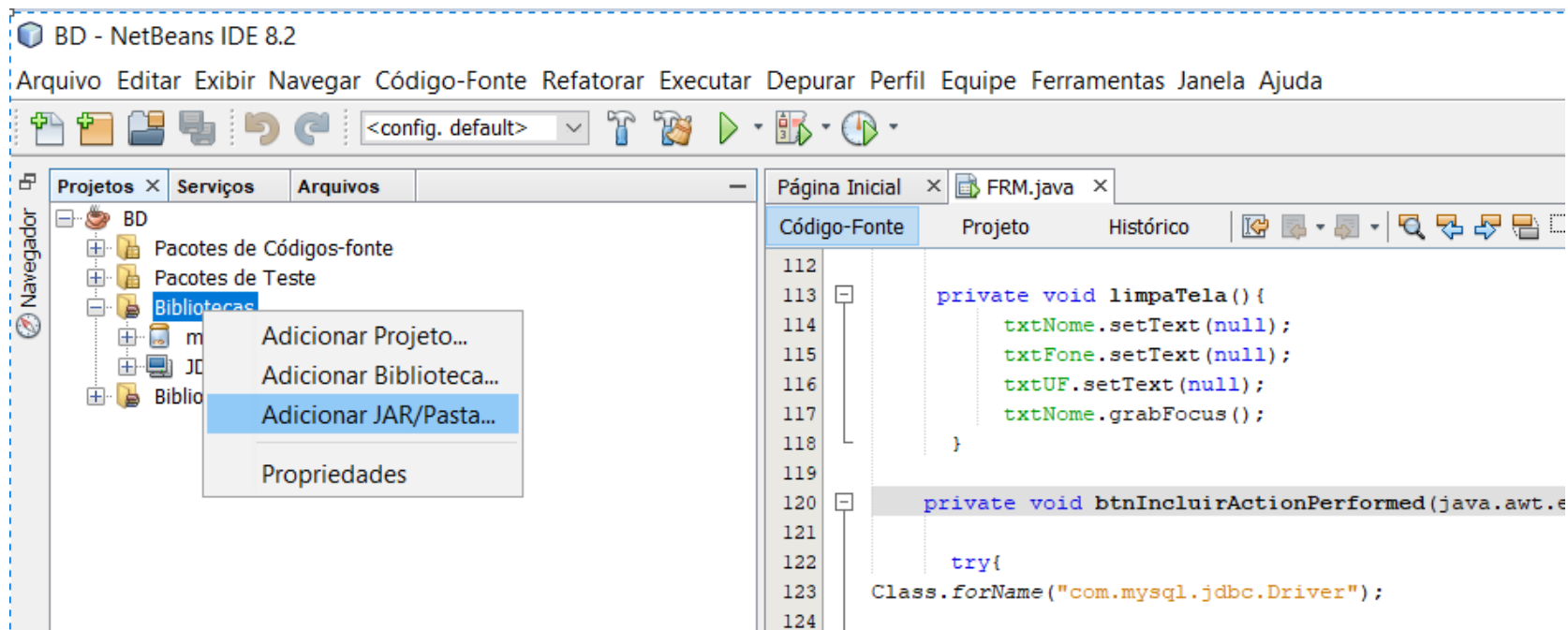


Para fazer a conexão entre Java e o MySQL é necessário utilizar um conector arquivo.JAR , como no caso do MySQL “MySQL Connector/J” que é o driver JDBC do MySQL. O Connector/J é um driver JDBC do tipo IV e contém todas as características de JDBC para utilizar MySQL.

Conector JDBC para Banco de Dados

Adicione o conector.jar na pasta JAR/PASTA de Bibliotecas, no Netbeans, como por exemplo o mysql-connector-java-8.0.23.jar

Verifique um conector compatível com o Servidor de Banco de Dados (SQL SERVER, ORACLE, MYSQL, POSTGREE..), e com a versão do mesmo, considerando também os conectores de acordo com o sistema operacional utilizado (Windows, Linux)



Trabalhando com Banco de Dados

Criar o Formulário

The image shows a Java Swing window with a light gray background. It contains two text input fields. The first field is preceded by the label 'Login' and followed by the text 'txtLogin'. The second field is preceded by the label 'Senha' and followed by the text 'txtSenha'. Below these fields are two buttons: 'OK' and 'Sair'. Underneath the 'OK' button is the text 'btnOK', and under the 'Sair' button is the text 'btnSair'.

```
private void  
btnSairActionPerformed(java.awt.event.ActionEvent evt)  
{  
    System.exit(0);  
}
```

Criar o Banco de Dados BD e as tabelas

No exemplo o banco de dados utilizado foi o MySQL

Usuarios → campos login e senha

Clientes → campos codCli, nome, fone, cidade, estado

```

private void btnOKActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
        String sql= "SELECT * FROM usuarios ";
        sql= sql+ "WHERE login = " + txtLogin.getText()+ "";
        sql= sql+ " and senha = " + txtSenha.getText()+ "";
        Statement st= cn.createStatement();
        ResultSet rs= st.executeQuery(sql);

        if(rs.next()){
            cn.close();
            FrmMenu frm = new FrmMenu();
            frm.setVisible(true);
        }
        else {
            cn.close();
            JOptionPane.showMessageDialog (null, "Usuario ou senha invalida ",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
        }
    }

    catch(Exception x){
        JOptionPane.showMessageDialog (null, "Falha na comunicação com Banco de Dados",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
    }
}

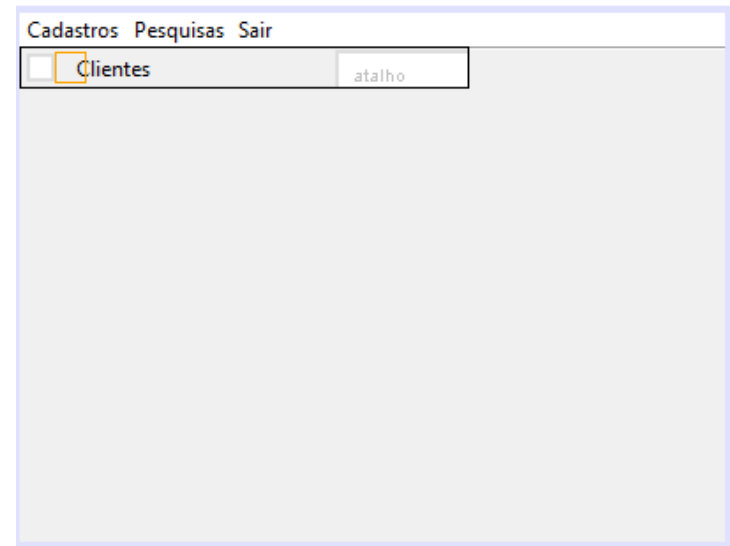
```

Criar o Formulário FrmMenu
com os itens :

mnuClientes

mnuPesquisa

mnuSair



```
private void mnuPesquisaMouseClicked(java.awt.event.MouseEvent evt) {  
    FrmListaClientes frm = new FrmListaClientes();  
    frm.setVisible(true);  
} dispose();  
  
private void mnuSairMouseClicked(java.awt.event.MouseEvent evt) {  
// fecha o formulario atual e volta para o menu anterior  
}  
  
private void mnuClientesActionPerformed(java.awt.event.ActionEvent evt) {  
    FrmClientes frm = new FrmClientes();  
    frm.setVisible(true);  
}
```

Criar o Formulário FrmClientes

The diagram illustrates the layout of the 'FrmClientes' form. It features a vertical list of labels on the left: 'Nome', 'Fone', 'Estado', and 'Cidade'. To the right of these labels are the corresponding input controls: a text box for 'Nome' (labeled 'txtNome'), a text box for 'Fone' (labeled 'txtFone'), a dropdown menu for 'Estado' (labeled 'cboEstado'), and a text box for 'Cidade' (labeled 'txtCidade'). Below these input fields is a horizontal separator line. To the right of this line is a button labeled 'Limpar'. Below the separator line is a row of five buttons: 'Incluir', 'Consultar', 'Alterar', 'Excluir', and 'Sair'. Below this row is another row of five buttons: 'btnIncluir', 'btnConsultar', 'btnAlterar', 'btnExcluir', and 'btnSair'. The form is enclosed in a rectangular border with a light gray background.

Colocar os códigos no FrmClientes

```
import java.sql.*;
import javax.swing.*;
public class FrmClientes extends
javax.swing.JFrame {
```

```
    private int codcli;
    public FrmClientes() {
        initComponents();
        montaComboEstado();
    }
```

```
private void montaComboEstado() {
    cboEstado.addItem("BA");
    cboEstado.addItem("MG");
    cboEstado.addItem("RJ");
    cboEstado.addItem("SP");
    cboEstado.setSelectedIndex(3);
}
```

```
private void limpaTela(){
    txtNome.setText(null);
    txtFone.setText(null);
    txtCidade.setText(null);
    cboEstado.setSelectedIndex(3);
    txtNome.grabFocus();
}
```

```
private void
btnSairActionPerformed(java.awt.event.ActionEvent evt)
{
    dispose();
}
```

```

private void btnIncluirActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        if (txtNome.getText().isEmpty()){
            // valida nome se foi preenchido
            txtNome.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher Nome",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }

        else if (txtFone.getText().isEmpty()){
            // valida fone se foi preenchido
            txtFone.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher Fone",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }
        else{
            Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
            Statement st = cn.createStatement();
            String sql="INSERT INTO CLIENTES (NOME,FONE,CIDADE,ESTADO) VALUES ('"+txtNome.getText() + "','"+
txtFone.getText()+ "','"+txtCidade.getText()+"','"+cboEstado.getSelectedItem()+"')";
            st.executeUpdate(sql);
            limpaTela();
        }
    }
    catch(Exception ValidaDados){
        //aborta a gravação e volta para o campo que deu erro de validação
    } }

```

```

private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        if (txtNome.getText().isEmpty()){
            // valida nome se foi preenchido
            txtNome.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher um Nome para pesquisa",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }
        Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
        String sql= "SELECT * FROM clientes ";
            sql= sql+ "WHERE nome like" + txtNome.getText()+ "%";
        Statement st= cn.createStatement();
            ResultSet rs= st.executeQuery(sql);
            if(rs.next()){
                limpaTela();
                codcli=rs.getInt("codcli");
                txtNome.setText(rs.getString("nome"));
                txtFone.setText(rs.getString("fone"));
                txtCidade.setText(rs.getString("cidade"));
                cboEstado.setSelectedItem(rs.getString("estado"));
                cn.close();
            }
            else {
                cn.close();
                JOptionPane.showMessageDialog (null, "Cliente não cadastrado",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
                limpaTela();
            }
        }

        catch(Exception ValidaDados){
            //aborta a consulta e volta para o campo que deu erro de validação
        } }

```



```

private void btnLimparActionPerformed(java.awt.event.ActionEvent evt) {
    limpaTela();
}

private void btnAlterarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        if (txtNome.getText().isEmpty()){
            // valida nome se foi preenchido
            txtNome.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher um Nome e clique em Consulta, depois faça as alterações ",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }
        else if (txtFone.getText().isEmpty()){
            // valida fone se foi preenchido
            txtFone.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher Fone", "Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }
        else{
            Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
            Statement st = cn.createStatement();
            String sql="UPDATE CLIENTES set nome='"+txtNome.getText()+"',";
            sql=sql+ "fone='"+txtFone.getText()+"',";
            sql=sql+"cidade='"+txtCidade.getText()+"',";
            sql=sql+"estado='"+cboEstado.getSelectedItem()+"'";
            sql=sql+" where codcli=" + codcli;
            st.executeUpdate(sql);
            limpaTela();
            cn.close();
        }
    }
    catch(Exception ValidaDados){
        //aborta a gravação e volta para o campo que deu erro de validação
    }
}

```

```

private void btnExcluirActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        if (txtNome.getText().isEmpty()){
            // valida nome se foi preenchido
            txtNome.grabFocus();
            JOptionPane.showMessageDialog (null, "Preencher um Nome e clique em Consulta, depois faça a
exclusão ", "Mensagem",JOptionPane.INFORMATION_MESSAGE);
            throw new Exception("ValidaDados");
        }
        else{
            int var=JOptionPane.showConfirmDialog(null,"Excluir este
Cliente","Mensagem",JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE);
            if (var==0){
                Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
                Statement st = cn.createStatement();
                String sql="DELETE FROM CLIENTES" ;
                sql=sql+" where codcli=" + codcli;
                st.executeUpdate(sql);
                limpaTela();
                JOptionPane.showMessageDialog (null, "Cliente excluido com sucesso ",
"Mensagem",JOptionPane.INFORMATION_MESSAGE);
            }
            else {
                limpaTela();
            }
        }
    }
    catch(Exception ValidaDados){
        //aborta a gravação e volta para o campo que deu erro de validação
    } }

```

Criar o Formulário FrmListaClientes

[illegible]

```
import java.sql.*;
import javax.swing.JOptionPane;
import
javax.swing.table.DefaultTableModel;
import java.util.*;
```

```
    private void
    btnSairActionPerformed(java.awt.event.Act
    ionEvent evt) {
        dispose();
    }
```

```

private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        Connection cn= DriverManager.getConnection("jdbc:mysql://localhost:3306/bd","root","");
        String sql= "SELECT NOME,FONE,CIDADE,ESTADO FROM clientes ";
        if (!txtNome.getText().isEmpty()){
            sql= sql+ "WHERE nome like" + txtNome.getText()+ "%";
        }
        sql=sql+ " order by nome";
        Statement st= cn.createStatement();
        ResultSet rs= st.executeQuery(sql);
        ResultSetMetaData rsmd = rs.getMetaData();
        Vector vetColunas = new Vector();
        for (int i=0;i< rsmd.getColumnCount();i++)
            vetColunas.add(rsmd.getColumnLabel(i+1));
        Vector vetLinhas= new Vector();
        while(rs.next()){
            Vector vetLinha= new Vector();
            for (int i=0;i< rsmd.getColumnCount();i++){
                vetLinha.add(rs.getObject(i+1));
            }
            vetLinhas.add(vetLinha);
        }
        tabClientes.getColumnModel().getColumn(1).setPreferredWidth(125000);
        tabClientes.setModel(new DefaultTableModel (vetLinhas,vetColunas));
    }
    catch(Exception e){
        JOptionPane.showMessageDialog (null, "Sem conexão com o Servidor de Banco de Dados",
        "Mensagem",JOptionPane.INFORMATION_MESSAGE);
    }
}

```