

# LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Java

# Linguagem de Programação Orientada a Objetos

## Conteúdo

2

### ▣ Formas de Avaliação:

- 1ª Avaliação - Prova Prática (até herança) – 4,0 pontos
- 2ª Avaliação - Prova Prática ou Projeto(GUI) – 4,0 pontos
- Exercícios e Atividades (cada uma avaliada de 0 a 10) – 2,0 pontos

# Linguagem de Programação Orientada a Objetos

## Conteúdo

3

### ▣ Introdução a Programação Orientada a Objetos

- Paradigmas da Programação Orientada a Objetos
- Conceitos básicos
- Conceitos de Estado, Comportamento e Identidade
- Implementação de uma classe e Criação de Objetos a partir da Classe
- Regras

### ▣ Herança

- Conceitos de SuperClasse e SubClasse
- Herança na prática
- Primeiro Contexto; Segundo Contexto; Terceiro Contexto; Quarto Contexto; Quinto Contexto; Sexto Contexto; Sub-Classes x Construtores;

# Linguagem de Programação Orientada a Objetos

## Conteúdo

4

- GUI – Graphical User Interface
- Métodos: Exemplos; Métodos da API; Modularização; Sobrecarga; Recursividade; Recursão x Iteração;
- Construtores: Construtores; Referência this; Sobrecarga de Construtores; Construtor-Padrão;
- Encapsulamento: Composição; Garbage Collector; Atributos e Métodos estáticos; Inicializadores Estáticos e Não Estáticos; Final;

# Linguagem de Programação Orientada a Objetos

## Conteúdo

5

- Associação:
- Polimorfismo: Entendendo e Codificando o Polimorfismo; Primeiro Contexto; Segundo Contexto; Terceiro Contexto – Classes Abstratas; Quarto Contexto;
- Exceções: Tratamento de exceções; `ArrayIndexOutOfBoundsException`; `StringIndexOutOfBoundsException`; `NullPointerException`; `ClassCastException`; `NumberFormatException`; Exceções criadas pelo usuário;
- Pacote `java.io`;
- `String`: Strings e expressões Regulares; Formatação de Strings;

# Linguagem de Programação Orientada a Objetos

## Conteúdo

6

### ▣ Tratamento de Exceções - Try / Catch

- Motivos para utilizar tratamento de exceções;
- Funções das Exceções;
- Exemplos de utilização de Exceções;
- Exemplos (estudo1.java; DivByZero.java; calc.java; exception.java);

# Linguagem de Programação Orientada a Objetos

## Introdução a Programação Orientada a Objetos

7

### ■ Objetivos:

- ❑ Conhecer a história do paradigma de programação orientada a objetos.
- ❑ Aprender o conceito de Classe e Objeto.
- ❑ Implementar uma Classe e Criar objetos a partir dela.

# Linguagem de Programação Orientada a Objetos

## Introdução a Programação Orientada a Objetos

8

- ▣ Classes, objetos, métodos e variáveis de instância:
  - ▣ Declarando uma classe com um método e instanciando um objeto de uma classe;
  - ▣ Declarando um método com um parâmetro;
  - ▣ Método com parâmetro: retornando e não retornando valor para o método de origem;
  - ▣ Método sem parâmetro: retornando e não retornando valor para o método de origem;



# Linguagem de Programação Orientada a Objetos

## Introdução a Programação Orientada a Objetos

9

- ▣ Apresentação de exemplo envolvendo os quatro casos vistos anteriormente;
- ▣ Exercícios envolvendo os quatro casos vistos anteriormente;
  - ▣ Defina:
    - Objeto:
    - Classe:
    - Atributos:
    - Comportamento:

### Exercícios propostos:

Para todos os casos abaixo, crie uma classe que resolva o solicitado pelo enunciado e outra classe, contendo o programa principal e que sirva para implementar a classe anteriormente criada. Lembre-se que toda a entrada de dados deverá ser feita pela classe que contém o método principal.

1. Elabore um programa em Java que contenha três métodos que recebam dois argumentos double e retornem: a soma, o produto e a divisão de ambos;
2. Elabore um programa em java que contenha um método booleano que teste se um ano é ou não bissexto. Anos divisíveis por 4 são bissextos, exceto para aqueles divisíveis por 100, ao menos que sejam também divisíveis por 400.

### Exercícios propostos:

- Para todos os casos abaixo, crie uma classe que resolva o solicitado pelo enunciado e outra classe, contendo o programa principal para implementar a classe anteriormente criada. Lembre-se que toda a entrada de dados deverá ser feita pela classe que contém o método principal.
3. Elabore um programa em java que contenha um método que aceite um valor inteiro e retorne o número com os dígitos invertidos. O valor deve ser solicitado ao usuário.
  4. Elabore um programa em java que imprima todos os números primos entre 1 e 10000.

### ■ Atributos x Métodos SET e GET (Modificadores);

- Toda vez que você declara uma variável dentro de um método, essa variável é considerada local. Assim que a execução desse método terminar, tais variáveis serão, automaticamente, destruídas. Um Objeto possui atributos (variáveis de instância) que serão portados com ele enquanto for utilizado pelo programa. Esses atributos existem antes de um método ser chamado por um objeto e depois de completada a execução do método.
- Uma classe contém um ou mais métodos que manipulam os atributos pertencentes a um objeto particular. Os atributos são representados como variáveis na declaração da classe. Tais variáveis são denominadas campos e são declaradas dentro de uma classe, mas fora do contexto de um método. Todo objeto mantém uma cópia de seus atributos, com valores, em memória.

# Linguagem de Programação Orientada a Objetos

## Introdução a Programação Orientada a Objetos

13

### ■ Métodos SET e GET;

- SET: terminologia utilizada para definir o método que recebe um valor externo à classe para determinado atributo da classe;
- GET: terminologia utilizada para definir o método que envia um valor de um atributo de uma classe para o meio externo à classe;
- Toda a alteração de atributos de uma classe deve ser feita por métodos que pertencem a própria classe;
- Palavra-chave public e private: Modificador de acesso que define o nível de encapsulamento do objeto. Veremos de forma mais detalhada no item Interface.

# Linguagem de Programação Orientada a Objetos

## Introdução a Programação Orientada a Objetos

14

- ▣ Construção de uma classe chamada PESSOA;
  - Atributos: Nome, Idade, Altura;
  - Métodos: atribuiNome; atribuiIdade; atribuiAltura; recebeNome; recebeIdade, recebeAltura; fazAniversario;
- ▣ Construção de um aplicativo para utilizar a classe criada anteriormente;

- Exercícios: Desenvolver a classe CASA com os seguintes parâmetros:

- Atributos:        porta1 : boolean;  
                      porta2 : boolean;  
                      porta3 : boolean;  
                      corcasa : String;

- Métodos:         abrirporta;  
                      fecharporta;  
                      pintarcasa;  
                      quantasportasestaoabertas;

### ■ Funcionamento:

- Inicialmente todas as portas estão fechadas e a casa está na cor branca;
- O método principal deve ter um menú na qual o usuário irá escolher a opção desejada;
- Quando o usuário abrir a porta, o programa deverá perguntar qual porta abrir. Caso a porta escolhida já estiver aberta, exibir a mensagem: A porta já está aberta.
- A mesma validação deverá ser feita quando o usuário fechar a porta;
- Quando o usuário escolher “Quantas portas estão abertas”, o programa deverá retornar o número de portas abertas;
- Quando o usuário escolher “Pintar a casa”, o programa deverá solicitar ao usuário a cor da casa, para posteriormente o programa atualizar a cor da casa;



# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

17

### ■ Funcionamento:

- Após cada opção executada, o programa deverá imprimir o status atual dos atributos da Casa;
- Criar uma opção para “Sair” do programa.

### ■ Elevador:

■ Crie uma classe Elevador em Java com as seguintes características:

■ andar\_atual (inteiro); n\_pessoas\_movim (inteiro); n\_pessoas\_momento (inteiro); manha (inteiro); tarde (inteiro); noite (inteiro); n\_viagens (inteiro); limite\_pessoas (inteiro); n\_andares (inteiro).

### □ Procedimentos:

■ Criar outra Classe em Java identificada como usa\_elevador, com um menu contendo as seguintes opções:

■ Movimentar Elevador; Subir Pessoas; Descer Pessoas; Estatísticas; Sair.

### ■ Elevador:

- Criar 1 Elevador;
- Ao escolher Movimentar Elevador o programa deverá solicitar o andar de destino.
- Ao escolher Entrar Pessoas o programa deverá solicitar o número de pessoas, o período que está funcionando (Manhã, Tarde ou Noite). Caso o número de pessoas existentes no elevador somado ao número de pessoas que estão entrando exceder o limite, o programa avisará que não poderão entrar o número de pessoas informado e irá solicitar nova entrada ao usuário, até que ele digite um número de pessoas válida.

### ■ Elevador:

- Ao escolher Sair Pessoas o programa deverá solicitar o número de pessoas e o período que está funcionando (Manhã, Tarde ou Noite). Caso o número de pessoas existentes no elevador seja inferior ao número de pessoas que estão saindo, o programa avisará que o número de pessoas informado é inválido e irá solicitar nova entrada ao usuário, até que ele solicite um número de pessoas válida.
- Ao escolher estatísticas o programa deverá informar:
  - Qual o período que se encontra o maior fluxo;
  - O número total de pessoas transportadas, por período.
- Criar os métodos necessários para implementar todos as opções acima.

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

21

- Herança através de exemplo prático:
  - Requisitos: Uma empresa de locação de veículos necessita de um sistema para armazenar as informações de sua frota de veículos. Para todos os tipos de veículos, é necessário armazenar informações como: placa, marca, modelo, ano, valor do km rodado, km inicial (antes da locação), km final (depois da locação). Essa locadora de veículos possui duas categorias de veículos para locação: Passeio e Carga. O total da frota é 15 veículos, sendo 10 do tipo Passeio e 5 do tipo Carga.
  - Para os veículos do tipo Passeio, é necessário armazenar se o veículo possui ar-condicionado e o número de portas (2 ou 4). Para os veículos do tipo Carga, é necessário armazenar a capacidade de carga em toneladas do veículo.

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

22

- Herança através de exemplo prático:
  - Todos os veículos cadastrados possuem um valor de locação. Esse valor de locação é calculado da seguinte forma: nº de km rodados \* valor do km rodado. O número de km rodados é obtido através do seguinte cálculo: km final – km inicial.
  - Nesse sistema os veículos do tipo Carga sofrem no cálculo de locação:
    - $(\text{nº de km rodados} * \text{valor do km rodado}) + 10\%$ . Além de armazenar as informações acima, o sistema deve permitir as seguintes consultas (listagens): Listagem de todos os veículos cadastrados; consulta de valor de locação mediante a digitação da placa do veículo; número de veículos com ar-condicionado;

# Linguagem de Programação Orientada a Objetos

Herança na prática.

23

- Herança através de exemplo prático:
  - Depois da apresentação dos requisitos do sistema, vamos começar a modelar o sistema dentro do paradigma orientado a objetos.
    - Identificar quais os objetos existentes nesse problema: É necessário armazenar informações sobre objetos do tipo Veiculo Carga e Veiculo Passeio.
    - Identificar atributos e comportamentos dos objetos

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

24

Veículo Passeio	Veículo Carga
<b>Atributos</b>	
Placa	Placa
Marca	Marca
Modelo	Modelo
Ano	Ano
valor do km rodado	valor do km rodado
km inicial	km inicial
km final	km final
Ar-condicionado	capacidade
Número de portas	
<b>Comportamentos ou métodos</b>	
Passeio ( )	Carga ( )
Métodos set e set para cada atributo	Métodos set e set para cada atributo
calculaValorLocacao( )	calculaValorLocacao( )



# Linguagem de Programação Orientada a Objetos

## Herança na prática.

25

- Herança através de exemplo prático:
  - Pela tabela anterior podemos identificar que existe um conjunto de atributos e métodos em comum entre os objetos Passeio e Carga. Diante disso, podemos optar por representar esses atributos e métodos em uma classe genérica (superclasse) chamada Veículo e representar os atributos e métodos específicos nas classes Passeio e Carga.
  - A seguir está a representação em UML das classes desse sistema.

# Linguagem de Programação Orientada a Objetos

Herança na prática.

26

## **Veiculo (Superclasse)**

```
placa: String  
marca: String  
modelo: double  
ano: int  
valorkmrod: double  
kminic: int  
kmfin: int
```

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

27

### **Veiculo (Superclasse)**

```
Veiculo()  
setPlaca(String splaca)  
setMarca(String smarca)  
setModelo(String smodelo)  
setAno(int ano)  
setValorKmRod(double dvalkm)  
setKminic(int ikmi)  
setKmFin(int ikmf)  
getPlaca():String  
getMarca():String  
getModelo():String  
getAno(): int  
getValorKmRod(): double  
getKminic(): int  
getKmFin: int  
calculaValorLocacao()
```

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

28

### Passeio (Subclasse)

arcond: boolean

nuportas: int

Passeio()

setArCond(boolean barcond)

setNuPortas(int inup)

getArCond(): boolean

getNuPortas(): int

### Carga (Subclasse)

capacidade: int

Carga()

setCapacidade(int icap)

getCapacidade():int

veiculo.java

passeio.java

carga.java

# Linguagem de Programação Orientada a Objetos

## Herança na prática.

29

### Exercício prático:

Passaro
corPena: String
imp() getCorPena():String setCorPena(String c)

Animal
nome: String Idade: int especie: String
impNome() getNome():String setNome(String n) getIdade():int setIdade(int i) getEspecie():String setEspecie(String e)

Mamifero
habitat: String temBico: boolean
imp() getHabitat():String setHabitat(String e) getTemBico(): boolean setTemBico(boolean b)

### Exercícios:



Universidade da Madeira

1. Crie uma subclasse `Fornecedor` à partir da classe `Pessoa` (desenvolvida em aulas anteriores) . Considere que cada instância da classe `Fornecedor` tem, para além dos atributos que caracterizam a classe `Pessoa`, os atributos `valorMaximo` e `valorDevido`. Implemente na classe `Fornecedor`, para além dos usuais métodos, um método `obterSaldo` que devolve a diferença entre os valores dos atributos da subclasse.

Depois de implementada a classe `Fornecedor`, crie um programa de teste adequado que lhe permita verificar o funcionamento dos métodos implementados na classe `Fornecedor` e os herdados da classe `Pessoa`.

### Exercícios:



Universidade da Madeira

2. Crie uma subclasse `Empregado` à partir da classe `Pessoa` (desenvolvida em aulas anteriores) . Considere que cada instância da classe `Empregado` tem, além dos atributos que caracterizam a classe `Pessoa`, os atributos `numeroSecao`, `salarioBase` e `iR` (porcentagem retida para o imposto de renda). Implemente a classe `Empregado`, além dos usuais métodos, um método `calcularSalario`.

### Exercícios:



Universidade da Madeira

3. Implemente a classe `Administrador` como subclasse da classe `Empregado`. Um determinado administrador tem como atributos, para além dos atributos da classe `Pessoa` e da classe `Empregado`, o atributo `ajudaDeCusto`. Note que deverá redefinir na classe `Administrador` o método herdado `calcularSalario` (o salário de um administrador é equivalente ao salário de um empregado usual acrescido da ajuda de custo). Escreva um programa de teste adequado para esta classe.



### Exercícios:



Universidade da Madeira

4. Implemente a classe Operario como subclasse da classe Empregado. Um determinado operário tem como atributos ValorProducao (que corresponde ao valor monetário dos artigos efetivamente produzidos pelo operário) e comissao, além dos atributos da classe Pessoa e da classe Empregado. Note que deverá redefinir nesta subclasse o método herdado calcularSalario (o salário de um operário é equivalente ao salário de um empregado usual acrescido da referida comissão). Escreva um programa de teste adequado para esta classe.

## Métodos

Segundo Serson (2007), os Métodos, também conhecidos em outras linguagens como procedimentos ou funções, permitem modularizar programas separando suas tarefas em unidades autocontidas.

Dentre os motivos que levam à utilização de métodos, podemos citar:

- Simplificação do problema;
- Maior gerenciabilidade e capacidade de reutilização do software;

## Métodos da API – Números randômicos.

Exemplo do método Random( )

```
Random aleatorio = new Random( );  
int valorAleatorio =aleatorio.nextInt();
```

[exemploMetodo1.java](#)

[exemploMetodo2.java](#)

## Exemplos de Modularização por meio de métodos.

Exemplo 1:

Elabore um programa que contenha um método que, fornecido um número inteiro, retorne o cubo deste número.

[exemploMetodo3.java](#)

Exemplo 2:

Elabore um programa que contenha um método que, fornecido dois números inteiros, retorne o menor deste dois números. Utilize o método `nextInt` de `Random` para fornecer inteiros aleatórios de 0 a 100.

[exemploMetodo4.java](#)

## Sobrecarga de Métodos.

Segundo Serson (2007), sobrecarga de métodos consiste no mecanismo na qual métodos de uma mesma classe possuem o mesmo nome, mas assinaturas diferentes.

A assinatura, por sua vez, define o tipo, a ordem e a quantidade de argumentos do método.

A sobrecarga permite, então, criar vários métodos com o mesmo nome e que realizem as mesmas tarefas, mas sobre tipos, ordens e quantidades diferentes de parâmetros.

[exemploClasseSobrecarga.java](#)

[classeSobrecarga.java](#)

## Sobrecarga de Métodos.

O exemplo abaixo trabalha com a sobrecarga de métodos com as assinaturas diferindo apenas pelas quantidades de argumentos. O primeiro método aceita dois números inteiros e o segundo aceita três inteiros.

[exemploCalculadora.java](#)

As chamadas de métodos não podem ser diferenciadas pelo tipo de retorno. O exemplo abaixo demonstra tal situação, quando dois métodos tem a mesma assinatura e diferentes tipos de retorno. Os métodos sobrecarregados podem ter diferentes tipos de retornos somente se tiverem diferentes listagens de argumentos.

### Exemplo de erro de sobrecarga:

```
Public class erro {  
    public int cubo (int x) {  
        return x * x * x;  
    }  
    public double cubo (int x) {  
        return x * x * x;  
    }  
}
```

### Exercícios de sobrecarga de métodos:

- 1) Elabore um programa em Java que solicite ao usuário três notas e que exiba um menu com pelo menos três formas de cálculo da média. Para resolver o problema utilize o recurso de sobrecarga de método para calcular a média das notas.



## Método Construtor

O método construtor é um método padrão que está presente na criação de todos os objetos a partir de uma classe. Permite que tais objetos sejam criados de forma correta, ou seja, com os parâmetros definidos de forma correta.

Possui algumas características que o diferencia dos demais métodos:

- Tem o mesmo nome da classe a que pertence;
- É o primeiro método executado quando o objeto é criado;
- Não retorna valor;
- Não pode ser chamado diretamente;
- Existe a possibilidade de criar vários métodos construtores com o mesmo nome, mas com assinaturas diferentes (diferentes parâmetros);
- É considerado padrão quando é executado sem a passagem de parâmetros;

## Exercícios - Método Construtor

- 1 – Crie uma classe chamada de Area. A classe possui atributos comprimento e largura, cada um deles configurado com 3. A classe deve ter um método que calcule o perímetro e a área, além dos métodos set e get para o comprimento e a largura. Escreva um outro programa ou classe para testar a classe Area. A área é a multiplicação dos lados de uma figura e o perímetro é a soma dos lados de uma figura.
- 2 - Crie e implemente a classe Carro com as seguintes propriedades: um veículo tem um certo consumo de combustível (medido em km/lt) e uma certa quantidade de litros no tanque. O consumo é especificado no construtor e o nível de combustível inicial é 0. Forneça um método movimentar que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina; e o método obterGasolina, que devolva o nível atual de gasolina; e adicionarGasolina, para abastecer o tanque. Crie um programa (classe) para executar a classe criada anteriormente.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

43

### ■ Objetivos:

- ❑ Acesso a conhecimentos básicos sobre o desenvolvimento de interface gráfica de aplicação, assim como sobre a conexão de aplicação com banco de dados;
- ❑ Conhecer os principais componentes Swing para o desenvolvimento de GUI;
- ❑ Aprender a manipular dados em um banco de dados via aplicação (software).

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

44

### ■ Entendendo uma GUI (graphical user interface)

Uma interface gráfica com o usuário GUI (graphical user interface) apresenta um mecanismo amigável para interagir com um aplicativo (é a tela do sistema). Uma GUI oferece ao aplicativo uma aparência e um comportamento.

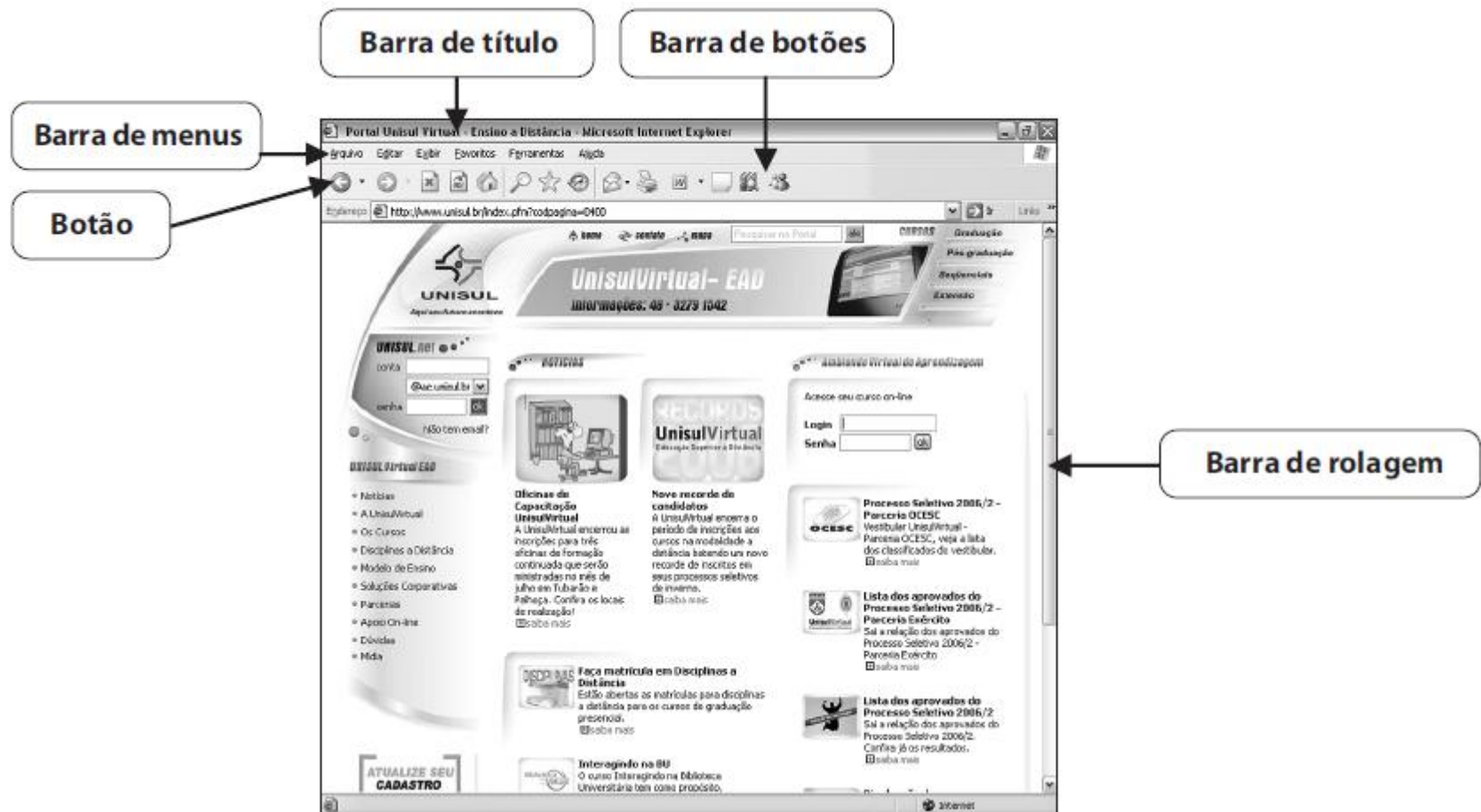
- Observe o exemplo de GUI, a seguir, visualizada através da janela do navegador Internet Explorer com seus componentes GUI rotulados.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

45

### ■ Entendendo uma GUI (graphical user interface)



# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

46

### ■ Entendendo uma GUI (graphical user interface)

A maioria dos aplicativos utiliza janelas (windows) ou caixas de diálogo para interagir com o usuário. Em geral, as caixas de diálogo são componentes GUI usadas para exibir mensagens importantes para o usuário ou para, em alguns casos, obter-se informações dele.

Utilizamos no semestre passado alguns deste tipo de componente (as caixas de diálogo) para fazer interações com o usuário (entrada e saída de dados).

Para exibir caixas de diálogo nos programas, use a classe `JOptionPane` que está no pacote `javax.swing` da API de Java (nesta disciplina, você explorará bastante as outras classes desse pacote).

A linguagem Java possui uma extensa biblioteca de classes prontas para os programadores usarem. Esta biblioteca é chamada de API (application programming interface). As classes nesta biblioteca são organizadas por funcionalidade, em estruturas chamadas pacotes (package).

### ■ showMessageDialog

Esse método permite a você exibir uma caixa de diálogo com uma mensagem e um botão. Ele pode ser customizado, ou seja, você pode alterar o texto da barra de título e o ícone da mensagem.

Para isso, você deve chamar esse método passando alguns argumentos adicionais. A forma abaixo é uma das maneiras de invocá-lo, embora existam outras que podem ser consultadas na documentação da API Java:

Sintaxe:

```
showMessageDialog(Component parentComponent, Object message,  
String title, int messageType)
```

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

48

```
showMessageDialog(Component parentComponent, Object message,  
String title, int messageType)
```

### Onde:

**parentComponent** – determina o Frame (este conceito será aprendido logo) no qual a caixa de diálogo será mostrada; se for null, então o Frame default (padrão) é usado.

**message** – mensagem a ser mostrada.

**title** – texto mostrado na barra de título.

**messageType** – determina o tipo de mensagem que aparecerá na caixa: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE` ou `PLAIN_MESSAGE` (sem ícone). Um ícone diferencia o tipo de mensagem.

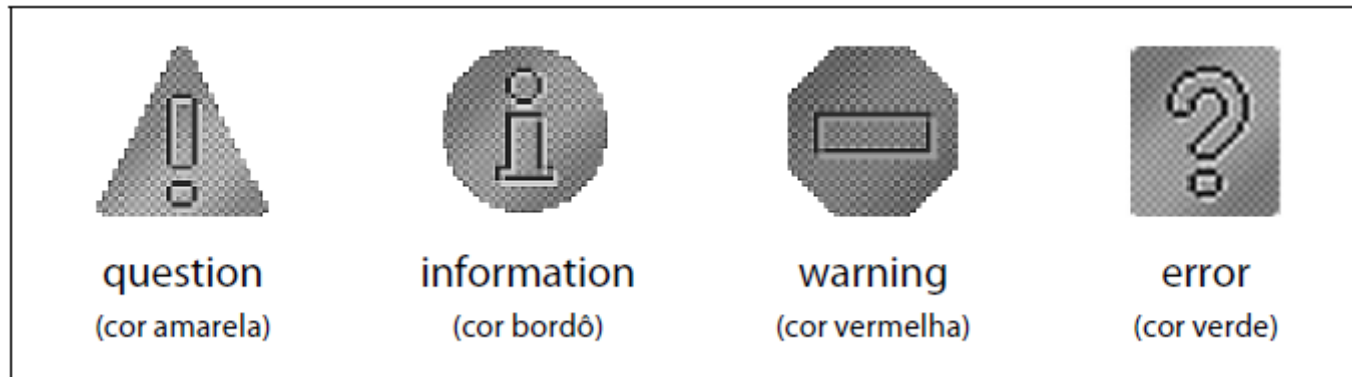


# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

49

De acordo com o tipo de mensagem (messageType) passado como argumento, os seguintes ícones podem aparecer na caixa de diálogo:



### ▣ showOptionDialog

Com este método, você pode mostrar uma caixa com botões específicos, ícone, mensagem e título. Você pode mudar o texto que aparece nos botões do diálogo e realizar outros tipos de customização.

Sintaxe:

```
showOptionDialog(Component parentComponent, Object message, String title,  
int optionType, int messageType, Icon icon, Object[] options, Object initialValue)
```

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

51

```
showOptionDialog(Component parentComponent, Object message, String title,  
int optionType, int messageType, Icon icon, Object[] options, Object initialValue)
```

### Onde:

**parentComponent** – determina o Frame (este conceito será aprendido logo) no qual a caixa de diálogo será mostrada; se for null, então o Frame default (padrão) é usado.

**message** – mensagem a ser mostrada.

**title** – texto mostrado na barra de título.

**optionType** – tipo de opções de botão que aparecerão: DEFAULT\_OPTION, YES\_NO\_OPTION, YES\_NO\_CANCEL\_OPTION, OK\_CANCEL\_OPTION.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

52

```
showOptionDialog(Component parentComponent, Object message, String title,  
int optionType, int messageType, Icon icon, Object[] options, Object initialValue)
```

### Onde:

**messageType** — determina o tipo de mensagem que aparecerá na caixa:  
ERROR\_MESSAGE, INFORMATION\_MESSAGE, WARNING\_MESSAGE,  
QUESTION\_MESSAGE ou PLAIN\_MESSAGE.

**Icon** - ícone mostrado na caixa;

**Option** - um array de objetos indicando as possíveis escolhas que o usuário pode fazer, normalmente um array de String com os nomes dos botões.

**initialValue** - opção default de botão que aparecerá na caixa. Deve ser usado somente se a opção options também for usada. Pode ser null.

### **Retorno:**

Um valor inteiro indicando a opção escolhida pelo usuário, ou `CLOSED_OPTION` se o usuário fechou a janela.

O código abaixo exemplifica a utilização desse método:mostrado na caixa;

`exemploShowOptionDialog.java`

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

54

### ■ `showInputDialog`

Ele permite uma entrada de dados por parte do usuário e algumas customizações. A novidade, mostrada aqui, é que o usuário pode escolher, ao invés de digitar, um entre vários valores estabelecidos previamente.

Esse método possui algumas variações, ou seja, ele pode ser chamado passando-se um conjunto reduzido de argumentos (como foi feito em POO) ou passando-se vários argumentos, o que possibilita uma customização maior.

Sintaxe:

```
showInputDialog(Component parentComponent, Object message, String title, int  
messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)
```

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

55

```
showInputDialog(Component parentComponent, Object message, String title, int  
messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)
```

### Onde:

**parentComponent** – determina o Frame (este conceito será aprendido logo) no qual a caixa de diálogo será mostrada; se for null, então o Frame default (padrão) é usado.

**message** – mensagem a ser mostrada.

**title** – texto mostrado na barra de título.

**messageType** – determina o tipo de mensagem que aparecerá na caixa: ERROR\_MESSAGE, INFORMATION\_MESSAGE, WARNING\_MESSAGE, QUESTION\_MESSAGE ou PLAIN\_MESSAGE.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

56

```
showInputDialog(Component parentComponent, Object message, String title, int  
messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)
```

**Onde:**

**icon** – imagem do tipo `Icon` pode ser mostrada.

**selectionValues** – array de objetos com vários valores possíveis de serem escolhidos pelo usuário. Normalmente um array de `String`.

**initialSelectionValues** – valor usado para inicializar o campo de entrada. É o valor que aparece no topo do campo.

**RETORNO:** Entrada ou escolha do usuário ou null se o usuário cancelar a entrada.

[exemploShowInputDialog.java](#)



### ■ Laboratório

- Elabore um programa em Java utilizando os recursos vistos anteriormente para converter números em diferentes bases numéricas. O programa deverá solicitar qual a conversão e o valor que se deseja converter. Exemplo:
  - Converter de Decimal para Binário
  - Converter de Decimal para Octal
  - Converter de Hexadecimal para Binário
  - Converter de Octal para Hexadecimal
  
- O resultado deverá utilizar os mesmos recursos acima citado.

### ■ Exceções

- Uma exceção é um erro que ocorre em tempo de execução, podendo ser decorrente, dentre outras coisas, de uma tentativa em referenciar um elemento inexistente de uma array, uma conversão ilegal de tipos primitivos ou simplesmente a tentativa de divisão de um número por 0.
- É uma indicação de um problema que ocorreu durante a execução do programa. É algo que ocorre raramente, daí o nome “exceção”. A proposta é estruturar o programa de maneira que o mesmo não venha a parar em função de um erro.
- Erros comuns: manipular um elemento da array que não existe; Quando o usuário digita um dado string e este é guardado em uma variável do tipo int;

### ■ Exemplos

- Digitar um número inteiro e outro double: no exemplo realizado. Ocorre então um erro identificado como `InputMismatchException`. Isso ocorre quando um método `Scanner.nextInt` recebe um double, que não representa o número válido.
- Digitar um número inteiro e outro 0: Ocorre então um erro identificado como `ArithmeticException`. Isso ocorre porque o programa está tentando realizar uma divisão por 0.

### ▣ Exemplos de programa com tratamento de exceção:

- Mecanismo que trata a divisão por 0.

exemploExcecao2.java

- Mecanismo que trata a incompatibilidade entre tipos.

exemploExcecao3.java

- Mecanismo que trata a os dois casos acima.

exemploExcecao4.java

### ▣ Outros tratamentos de exceção:

- `ArrayIndexOutOfBoundsException.`

`exemploExcecao5.java`

- `StringIndexOutOfBoundsException.`

`exemploExcecao6.java`

- `NullPointerException.`

`exemploExcecao7.java`

# Linguagem de Programação Orientada a Objetos

## Exceções

62

### ▣ Exemplo de exceção criada pelo usuário

`IdadeInsuficienteExcecao.java`

`CarteiraMotorista.java`

`testaMinhaExcecao.java`

### ■ Exercício:

#### ■ Classe Funcionario;

Finalidade: Armazenar dados de funcionário.

#### ■ Métodos

setNome(), setSalariob(), setSalariol(), getNome(), getSalariob (), getSalariol(),  
atualizaSalario().

#### ■ Criar exceções para os dados de entrada;

#### ■ Criar exceção para avisar quando o salário exceder o seu limite, ou seja, ficar negativo

### ▣ Exercício:

#### ■ Funcionamento:

- Criar um funcionário. Ao criar o funcionário, este receberá os atributos iniciais digitados pelo usuário.
- Criar um menú com as seguintes opções: 1. Incluir proventos / 2. Incluir descontos / 3. Sair;
- Controlar, através do tratamento de exceções, o salário líquido do funcionário;



### ■ Visão Geral dos componentes Swing

Alguns dos  
componentes  
mais utilizados

Componente	Descrição
JLabel	Exibe texto não editável ou ícones.
TextField	Permite ao usuário inserir dados através do teclado. Pode exibir texto editável ou não editável.
JButton	Botão.
JCheckBox	Opção associada a um texto que pode ou não ser selecionada. Podem-se selecionar várias opções de <i>checkbox</i> .
JRadioButton	Opção associada a um texto que pode ou não ser selecionada. Seleciona-se uma entre várias opções de <i>radio</i> .
JComboBox	Fornece uma lista de itens a partir da qual o usuário pode escolher um deles ou digitar um valor na própria caixa.
JList	Fornece uma lista de itens a partir da qual o usuário pode escolher um ou vários itens.
JPanel	Fornece uma área (painel, <i>container</i> ) onde outros componentes podem ser colocados e organizados.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

66

### Visão Geral dos componentes Swing



Exemplo de uma interface com os componentes listados acima

### Swing e AWT

Para construir as aplicações GUI, você deve utilizar dois conjuntos de componentes GUI no Java:

- Swing.
- AWT.

Antes do Swing ser construído no J2SE 1.2, as GUI de aplicações em Java eram construídas com componentes do Abstract Window Toolkit (AWT), que estão no pacote `java.awt`.

Quando uma aplicação GUI em Java é construída com componentes AWT e é executada em diferentes plataformas, esses componentes são exibidos diferentemente em cada uma dessas plataformas. Além disso, às vezes, a maneira como um usuário interage com um componente AWT difere entre as plataformas.

### Swing e AWT

Por exemplo, uma aplicação que contém um botão (objeto Button do pacote `java.awt`), ao ser executada no sistema operacional Windows, terá a mesma aparência dos botões do sistema operacional Windows. Se a aplicação for executada no sistema operacional do Macintosh, então esta aplicação terá a mesma aparência dos botões desse sistema.

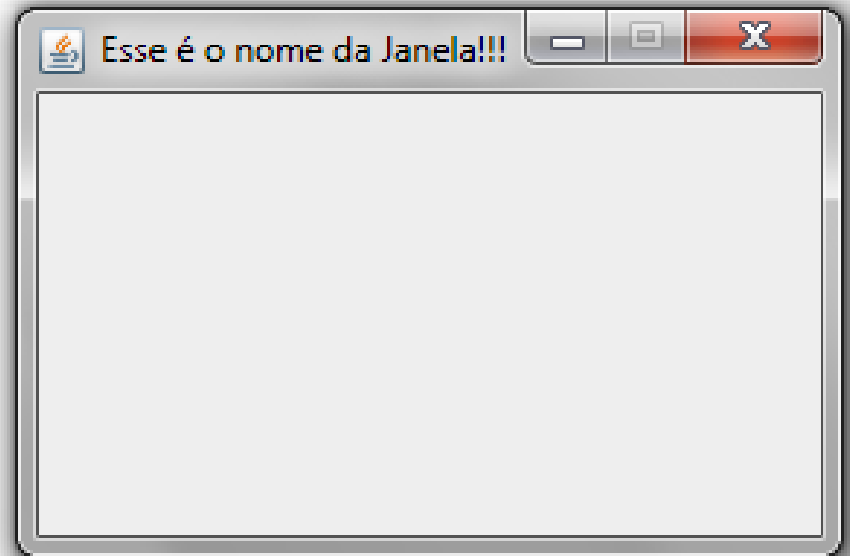
### **Criação da Primeira GUI - Componente JFrame**

Criar uma GUI mais elaborada envolve, primeiramente, a criação de um container, que é, por sua vez, um componente que pode receber outros componentes.

Em uma aplicação GUI desktop, normalmente o container usado é um frame (uma janela com barra de título, botões de minimizar, maximizar e fechar a janela).

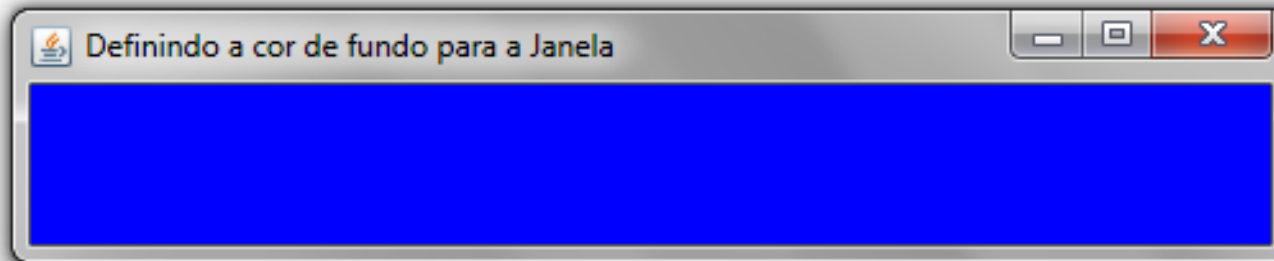
## Criação da Primeira GUI - Componente JFrame

Uma janela ou frame é uma instância da classe JFrame ou uma subclasse de JFrame. Tal classe fornece os atributos e comportamentos básicos de uma janela. Nesse container (janela ou frame) podem ser adicionados vários outros componentes GUI.



## Criação da Primeira GUI - Componente JFrame

Embora as propriedades mais importantes da janela do programa já tenham vistas anteriormente, a possibilidade de alterar a cor de fundo da janela só pode ser apresentada a seguir. Isso se deve ao fato de a cor de fundo ser definida não para a classe JFrame, mas para o painel de conteúdo, ou seja, o objeto Container que contém o controle da janela. O programa a seguir exibe uma janela com a cor azul definida como cor de fundo, observando que a cor de fundo do painel não afetará os demais controles da janela.

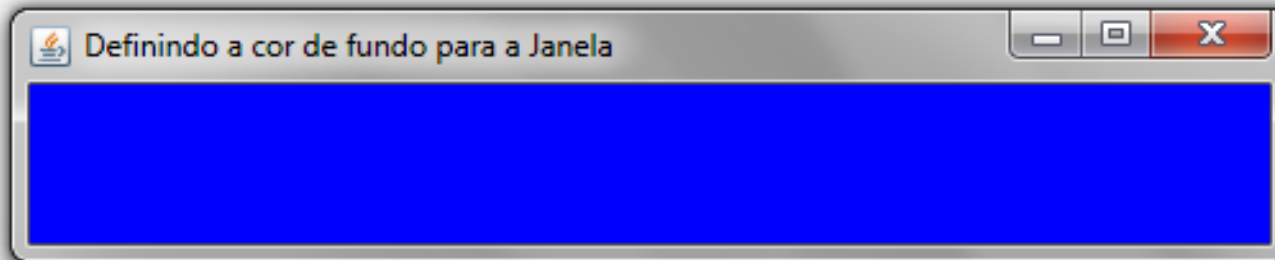


CorDeFundo.java

### Criação da Primeira GUI - Componente JFrame

Além de utilizar cores prédefinidas, há a possibilidade de criar novas cores baseadas no padrão RGB (Red, Green e Blue). Todas as tonalidades variam entre 0 e 255. Assim você pode colorir a tela de forma totalmente personalizada.

CorDeFundo2.java

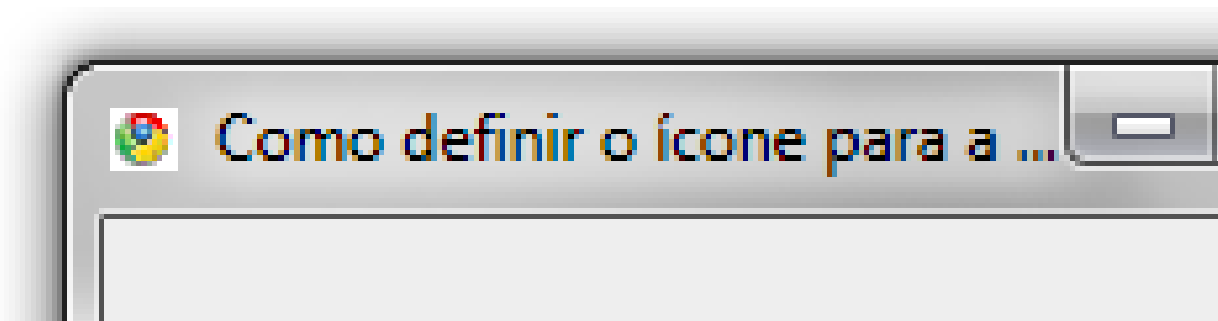




### Criação da Primeira GUI - Componente JFrame

O ícone padrão que é exibido em conjunto com as janelas é a xícara do Java. Ela sempre aparece no canto superior esquerdo da janela. Veremos como este ícone pode ser realizado, no sentido de personalizar da aplicação de acordo com a intenção do programador. O arquivo de imagem deve ser colocado no mesmo diretório onde o arquivo binário será gerado (arquivo com a extensão .class).

DefinirIcone.java



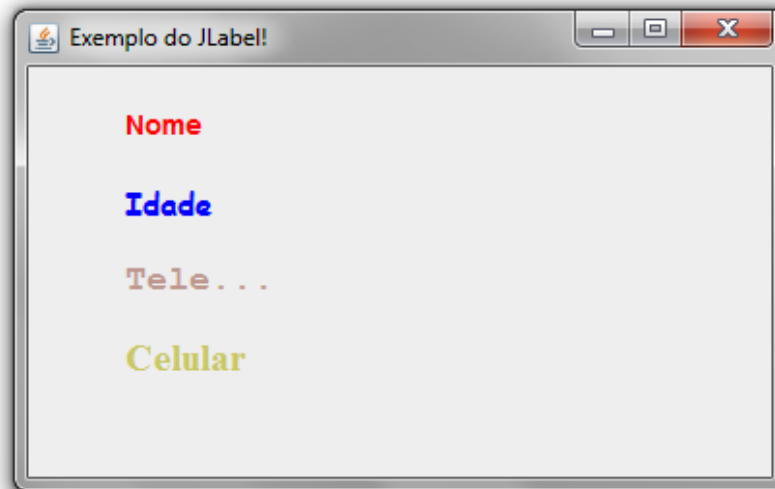
### **Criação da Primeira GUI - Componente JFrame**

Os objetos da classe `ImageIcon` podem ser criados por meio de nove construtores diferentes. Escolhemos por aquele que recebe o caminho e/ou nome da imagem como argumento.

Uma vez instanciado o objeto da classe `ImageIcon`, podemos fazer uso do método `getImage` dessa classe para obter a imagem e a definirmos como ícone da janela, por meio de uma chamada a `setIconImage` da classe `JFrame`.

## Componente JLabel

Chamado de rótulo, este componente raramente tem seu conteúdo alterado e, quando usado corretamente, possibilita manipulações bem interessantes. Vamos escrever um programa que permite instanciar a classe JLabel de maneiras diferentes. O programa abaixo adiciona o JLabel , posicionando na janela, alterando a cor e a fonte desse componente, além de, quando se passar o mouse por cima, exibe uma dica (tip).



ExemploJLabel.java

### **Componente JLabel – Métodos mais utilizados:**

`setBounds( left , top , altura , largura )`: define a posição (em pixels) da borda esquerda, do topo, o tamanho da altura e o tamanho da largura do JLabel;

`setForeground( cor)`: define a cor da letra dos componentes;

`setForeground(new Color(xx,xx,xx))`: definem cores personalizadas para o componente JLabel;

`setFont(fonte, estilo, tamanho)`: Definem a tipo da fonte, o estilo e o tamanho da fonte utilizada;

### Componente JLabel – Métodos mais utilizados:

`setBounds( left , top , altura , largura )`: define a posição (em pixels) da borda esquerda, do topo, o tamanho da altura e o tamanho da largura do JLabel;

O estilo pode ser definido como:

`Font.BOLD` – Fonte em negrito;

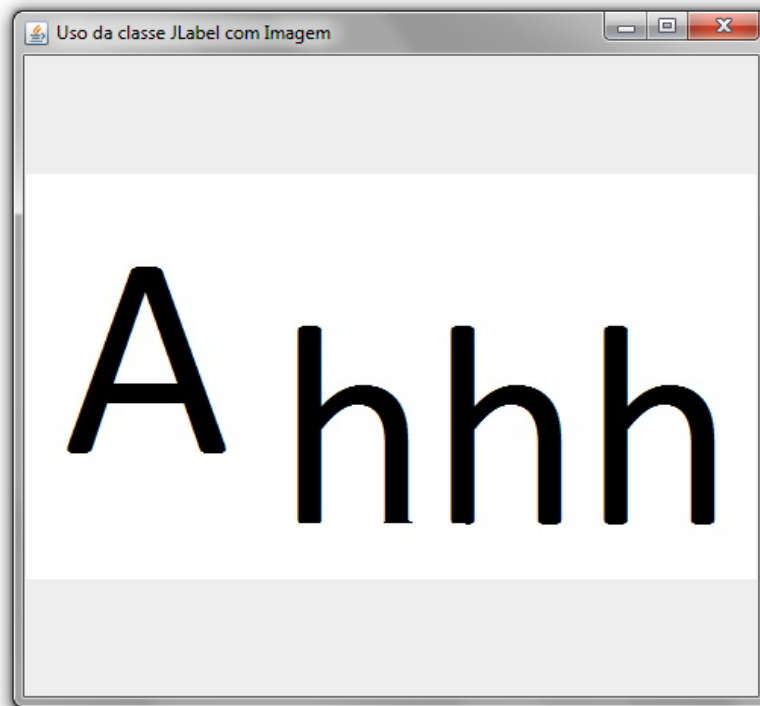
`Font.ITALIC` – Fonte em itálico;

`Font.PLAIN` – Fonte normal;

`Font.BOLD+Font.ITALIC` – negrito e itálico;

### Componente JLabel – Métodos mais utilizados:

É possível exibir imagens em rótulos, como instâncias da classe JLabel. Tais rótulos podem conter apenas imagens ou imagens e texto. O programa abaixo mostra apenas uma imagem adicionada com um JLabel na janela.



LabelImagem.java

### Componente JTextField

A classe JTextField possibilita a criação de caixas de textos de uma única linha. São frequentemente utilizados para receber informações digitadas pelo usuário.



### **Componente JTextField – Métodos mais utilizados:**

`setFont(new Font("nome da fonte",Font.ESTILO,TAMANHO)`: Define o nome da fonte, o estilo e o tamanho da fonte do `JTextField`;

`setHorizontalAlignment(JTextField.LEFT)`: define o alinhamento dentro da caixa de texto;

`setBackground(Color.blue)`: Altera a cor de fundo da caixa de texto;



### Componente JButton

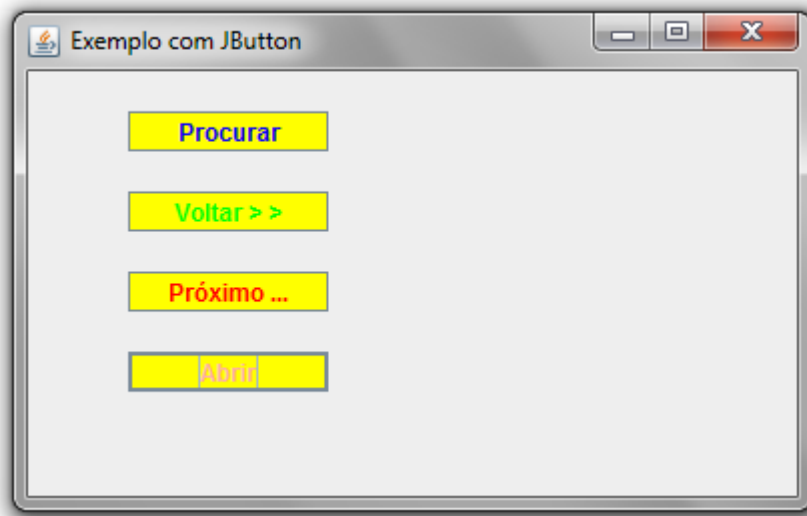
Assim como o JLabel, a classe JButton podem conter texto, texto e imagem ou apenas imagem.



### Componente JButton

Podemos utilizar vários botões dentro de uma mesma janela, alterando tanto a cor de fundo como a cor de texto. Existe a possibilidade de adicionar teclas de atalho para o botão. Tal recurso é realizado por meio da instrução:

```
botao1.setMnemonic(KeyEvent.VK_F9);
```



## Interface

Uma interface é uma espécie de classe ou estrutura que inicia com a palavra-chave `interface` ou `public interface` e possui somente métodos que são implicitamente `public abstract` e/ou atributos que são `public static final`.

## Tipos de Modificadores

**Public** - O modificador de acesso definido como `public` é aquele que permite o maior grau de acesso a uma classe ou método criado em Java. Isso significa que ele pode ser chamado (precedido da referência ao seu objeto) de qualquer outra classe. Dessa maneira, está se disponibilizando a interface do objeto para a sua manipulação.

## Tipos de Modificadores

**Private** - Um método ou variável declarada como um modificador de acesso private só pode ser acessado por uma instância da classe que declara o método ou variável. Ao tornar um atributo privado, estamos promovendo o encapsulamento, porque o atributo só poderá ser acessado por métodos do objeto.

**Protect** - O modificador definido como protected pode ser aplicado às variáveis e métodos criados em uma classe. Elementos em uma classe marcados como protected podem ser acessados por qualquer classe no mesmo pacote ou package, ou por suas classes derivadas, também chamadas de subclasses.

**Static** - um método com o modificador static pode ser chamado precedido apenas do nome da classe, sem que nenhum objeto daquela classe tenha sido criado; o atributo static faz com o que o atributo seja compartilhado por todos os objetos criados a partir da classe onde ele foi definido. Ou seja, o atributo é da classe e não de cada objeto.

## Tipos de Modificadores

**Final** - um método com o modificador final não pode ser redefinido, sobrescrito em uma subclasse; o atributo final é chamado de constante porque o seu valor nunca muda, é sempre fixo. O atributo final deve ser inicializado com um valor no momento da sua definição ou no método construtor da classe.

**Abstract** - uma classe abstract não pode ter objetos criados (instanciados) a partir dela; o método abstract não tem implementação, é um método vazio, sem código. Pode ser usado num contexto de herança, numa superclasse, quando a lógica do método é diferente para cada subclasse desta superclasse. Todo o método abstract deve ser obrigatoriamente redefinido (sobreescrito) por todas as subclasses da superclasse onde o método abstract está.

### Tipos de Modificadores

Os atributos `static final` (com valor fixo) e métodos `abstract` (sem implementação) serão os únicos itens são colocados numa interface.

Para quê serve e como é utilizada uma interface?

A interface é diferente de uma classe porque não se podem criar objetos a partir de uma interface. A interface é utilizada (implementada) por uma classe. Pode ser utilizada nas seguintes situações (1 e 2):

## Tipos de Modificadores

- 1) O Java não suporta herança múltipla, ou seja, uma classe (subclasse) não pode herdar ou estender mais que uma classe (superclasse). Uma alternativa para essa restrição são as interfaces, pois uma classe pode implementar (implements) várias interfaces.

Como todos os métodos de uma interface são abstract, eles devem ser obrigatoriamente redefinidos nas classes que implementam esta interface.

Uma classe pode implementar (uma espécie de herança) várias interfaces, resolvendo, portanto, a restrição de herança múltipla apresentada anteriormente. A indicação que uma classe implementa outra classe se dá pela palavra-chave implements.

Exemplo:

```
public class Quadrilatero implements CalculoFiguras,..... {  
}
```

## Tipos de Modificadores

Observe que `CalculoFiguras` é uma interface e que `Quadrilátero` é uma classe que implementa `CalculoFiguras` e mais outras interfaces, se houver necessidade.

Já que `Quadrilátero` implementa `CalculoFiguras`, ela tem que redefinir, obrigatoriamente, todos os métodos abstract de `CalculoFiguras`. Se isto não for feito, `Quadrilátero` se torna uma classe abstract e uma mensagem de erro aparece após a compilação, indicando que ela deve ser declarada como abstract.

Uma interface é utilizada quando existe um comportamento ou operação (método) padrão, cuja lógica é específica para cada classe e que deve ser executado por classes que não estão relacionadas somente através de Herança.

Acompanhe o exemplo:



### Exemplo:

O comportamento ou método `calculaArea()` deve ser executado pelos objetos das classes da hierarquia Quadrilátero (superclasse), Trapézio e Retângulo (subclasses) - (estas três classes estão relacionadas através de herança) - e, também, pelos objetos da classe Triângulo que não faz parte desse relacionamento. Note que a lógica para calcular a área dessas figuras é diferente para cada uma. A primeira solução seria implementar um método abstract (sem lógica) na superclasse Quadrilátero e redefini-lo com a lógica apropriada nas subclasses Trapézio e Retângulo.

Porém, se esse método ou comportamento for colocado somente na superclasse Quadrilátero, então só será visto e redefinido pelas subclasses Trapézio e Retângulo. A classe Triângulo não enxerga esse método, pois não faz parte da hierarquia de herança. O que fazer, então? Uma solução é criar uma interface e inserir esse comportamento (método) `calculaArea()` lá.

### Exemplo:

Você pode criar uma interface chamada `CalculoFiguras` e inserir o método `calculaArea()`. Lembre-se que métodos numa interface são sempre abstract (sem implementação). Assim, todas as classes (`Trapézio`, `Retângulo` e `Triângulo`) poderão implementar essa interface. Isto quer dizer que terão que, obrigatoriamente, redefinir o(s) método(s) abstract definidos nessa interface.

O código, abaixo, demonstra a implementação dessa interface. Esse código deve ser salvo num arquivo chamado `CalculoFiguras.java`. Depois de editar, o código deve ser somente compilado

```
public interface CalculoFiguras{  
    public abstract double calculaArea(); //os modificadores public abstract  
    não são obrigatórios  
}
```

### Exemplo:

Agora, várias classes podem implementar essa interface.

Seguindo o exemplo, as classes que devem enxergar e redefinir o método `calculaArea()` são Trapézio, Retângulo e Triângulo. Como Trapézio e Retângulo são subclasses de Quadrilátero, é esta superclasse que deve implementar a interface `CalculoFiguras`. Triângulo também deve implementar a interface `CalculoFiguras`.

Cada classe deve ser editada e compilada em um arquivo `.java` separado.

Não é necessário que você implemente esse código. A intenção é apenas demonstrar a utilização de uma interface.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

92

```
//superclasse Quadrilátero implementa a interface CalculoFiguras
public class Quadrilatero implements CalculoFiguras {
    //atributos e métodos
    ..
}
//subclasse Trapézio estende a classe Quadrilátero
public class Trapezio extends Quadrilatero{
    //atributos e métodos
    ..
    public double calculaArea(){
//...
    }
}
//subclasse Retângulo estende a classe Quadrilátero
public class Retangulo extends Quadrilatero{
    //atributos e métodos
    ..
    public double calculaArea(){
//...
    }
}
```

### Exemplo:

Note que somente a superclasse implementa a interface. Com este procedimento, o método `calculaArea()` deveria ser, obrigatoriamente, redefinido nesta superclasse.

Caso não seja redefinido nesta superclasse, como será o caso (porque não existe cálculo de área a ser implementado na superclasse `Quadrilatero`), então a superclasse `Quadrilátero` deve se tornar `abstract` e este método, por sua vez, é repassado através da herança para todas as subclasses, devendo ser obrigatoriamente redefinido em todas elas.

É exatamente nas subclasses que deve ser implementado o cálculo da área, já que ele é diferente para `Trapezio` e para `Retangulo`.

A classe `Triângulo` não pertence à hierarquia de herança das classes acima, mas ela também possui o comportamento ou método específico para calcular a área. Ela deve implementar a interface `CalculoFiguras` e, assim, tem que redefinir, obrigatoriamente, o método `abstract` dessa interface.

Confira no código seguinte:

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

94

```
//Classe Triangulo implementa a interface CalculoFiguras
public class Triangulo implements CalculoFiguras {
    //atributos e métodos

    ..
    public double calculaArea(){
        //...
    }
}
```

Em resumo, a interface é utilizada quando classes díspares (não relacionadas) precisam compartilhar métodos e atributos em comum. Assim, permite-se que os métodos de objetos (classes) que implementam uma interface sejam processados polimorficamente (uso de polimorfismo).

### Exemplo:

- 2) Outra aplicação de interface refere-se à possibilidade de utilizá-la como um repositório de constantes. Ou seja, você pode declarar numa interface um conjunto de constantes (valores fixos) que podem ser usados por várias classes.

Atenção! Lembre-se que todo atributo numa interface é `static` e `final`.

Observe o código abaixo:

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

96

```
public interface Constantes{ //esta interface deve ser salva no arquivo Constantes.java
    int UM = 1;
    int DOIS = 2;
    int TRES = 3;
}
public class TesteInterface{ //note que esta classe não precisa implementar a interface Constantes
    public static void main(String args[]){
        if (Constantes.UM == 1) //nome da interface.nome do atributo
            System.out.println("é igual a 1");
    }
}
```



## Tratamento de Eventos

Ao interagir com os componentes GUI de uma aplicação, o usuário indica a tarefa que deve ser executada pela aplicação. Por exemplo, em uma aplicação qualquer, ao clicar num botão com um rótulo Soma, provavelmente, a tarefa de somar alguns números será executada. Num editor de texto como o Word, o ato de clicar no botão com o ícone de um disquete (botão Salvar) indica que a tarefa de salvar o documento será executada.

Para que você consiga programar uma tarefa em uma aplicação GUI é necessário entender como funciona o tratamento de eventos. Quando o usuário interage com um componente GUI, um evento é disparado (um evento também é um objeto – aliás, tudo é objeto em uma linguagem orientada a objeto).

**Para recordar:** A classe **JTextField** (que faz parte do pacote javax.swing) representa o componente conhecido como campo de texto, onde o usuário pode inserir texto numa única linha através do teclado. O texto também pode ser exibido nesse componente sem que o usuário entre com ele. Para digitar o texto nesse componente é necessário que o “foco” esteja nele. Um componente recebe o foco quando você clica nele.

## Tratamento de Eventos

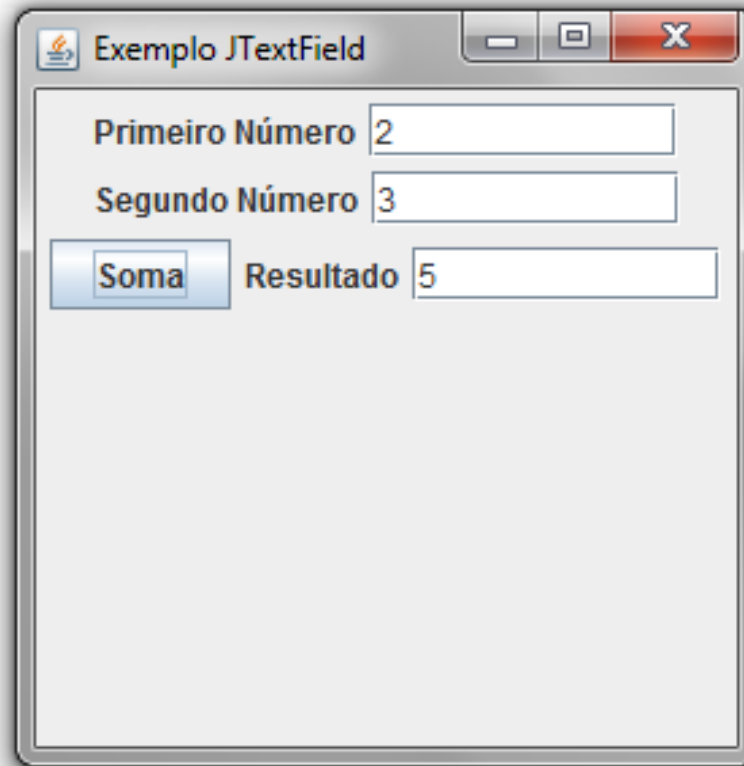
A classe **JButton** (que faz parte do pacote `javax.swing`) representa o componente botão, que é utilizado nas aplicações GUI para executar uma ação específica (ao ser clicado), como somar, salvar dados de um formulário, deletar dados, etc. Um botão pode exibir texto e imagem (objeto `Icon`) dentro dele. O texto é chamado de rótulo do botão.

A aplicação GUI, a seguir, demonstra o uso desses dois componentes numa interface onde o usuário deve entrar com dois números (nos componentes `TextField`) e clicar no botão Soma (`JButton`). Após o clique, será exibida a soma dos dois números em outro componente `TextField`.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

99



Veja o código da classe **eventoJTextField.java** onde a GUI é montada e executada:

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

100

A classe `TrataEventoBotao` implementa (implements) a interface já definida na API do Java chamada `ActionListener`. Ela deve implementar exatamente essa interface porque o evento gerado pelo clique no botão é do tipo `ActionEvent` (classe `ActionEvent`).

Quando o usuário clica no botão, um objeto de evento do tipo `ActionEvent` (classe `ActionEvent`) é gerado.

Quando o usuário pressiona a tecla `Enter` num componente `JTextEdit`, um objeto de evento do tipo `ActionEvent` também é gerado.

Quando o usuário digita caracteres em um campo de texto, um objeto de evento do tipo `KeyEvent` é gerado.

Quando o usuário passa o mouse sobre um componente, um objeto de evento do tipo `MouseEvent` é gerado, etc.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

101

Perceba que, conforme o tipo de interação com o componente, outro tipo de evento pode ocorrer. Em outras aplicações, você trabalhará com outros tipos de eventos.

O objeto de uma classe que implementa a interface `ActionListener` somente pode “ouvir” ou tratar eventos do tipo `ActionEvent`, desde que esse objeto seja registrado junto ao componente. Lembre-se que o objeto do tipo `TrataEventoBotao` é criado e é adicionado (registrado) ao componente `JButton button1`.

Assim, quando o usuário clicar no botão para realizar a soma, um objeto de evento do tipo `ActionEvent` será disparado e passado como parâmetro para o método do objeto ouvinte de evento, que foi registrado junto ao componente `JButton button1`. Assim, tudo o que foi programado nesse método será executado.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

102

### Exercícios:

- 1 – Com base nos exemplos vistos anteriormente, crie um programa em Java para executar o seguinte programa:



### Exercícios:

- 2 - Desenvolva o seguinte programa utilizando uma GUI com os componentes JTextField, JLabel e JButton aprendidos até o momento. O programa deve permitir que o usuário entre com 3 notas. Ao clicar no botão Média, a média deve ser calculada e apresentada num componente JLabel. Além disto, a situação do aluno - “Aprovado” se a média for maior ou igual a 7, “Exame” se a média for maior ou igual a 2 e menor que 7 e “Reprovado” se a média for menor que 2 - deve ser apresentada em outro componente JLabel.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

104

Você já viu anteriormente na linguagem Delphi que existem vários tipos de eventos (não somente `ActionEvent`) que podem ocorrer quando o usuário interage com um componente GUI. As informações sobre esse evento (objeto de evento) são armazenadas em um objeto (todo o objeto deriva de uma classe) que estende a classe `AWTEvent`.

Veja que nas linhas de importação da classe `GUIJTextField` foi importado a classe `ActionEvent` do pacote `java.awt.event`. Essa classe estende (é uma subclasse de) `AWTEvent`.

A seguir veremos uma figura que ilustra a hierarquia de classes a partir da classe `AWTEvent`. Todas essas classes representam todos os tipos de objetos de evento que podem ocorrer quando o usuário interage com um componente GUI.

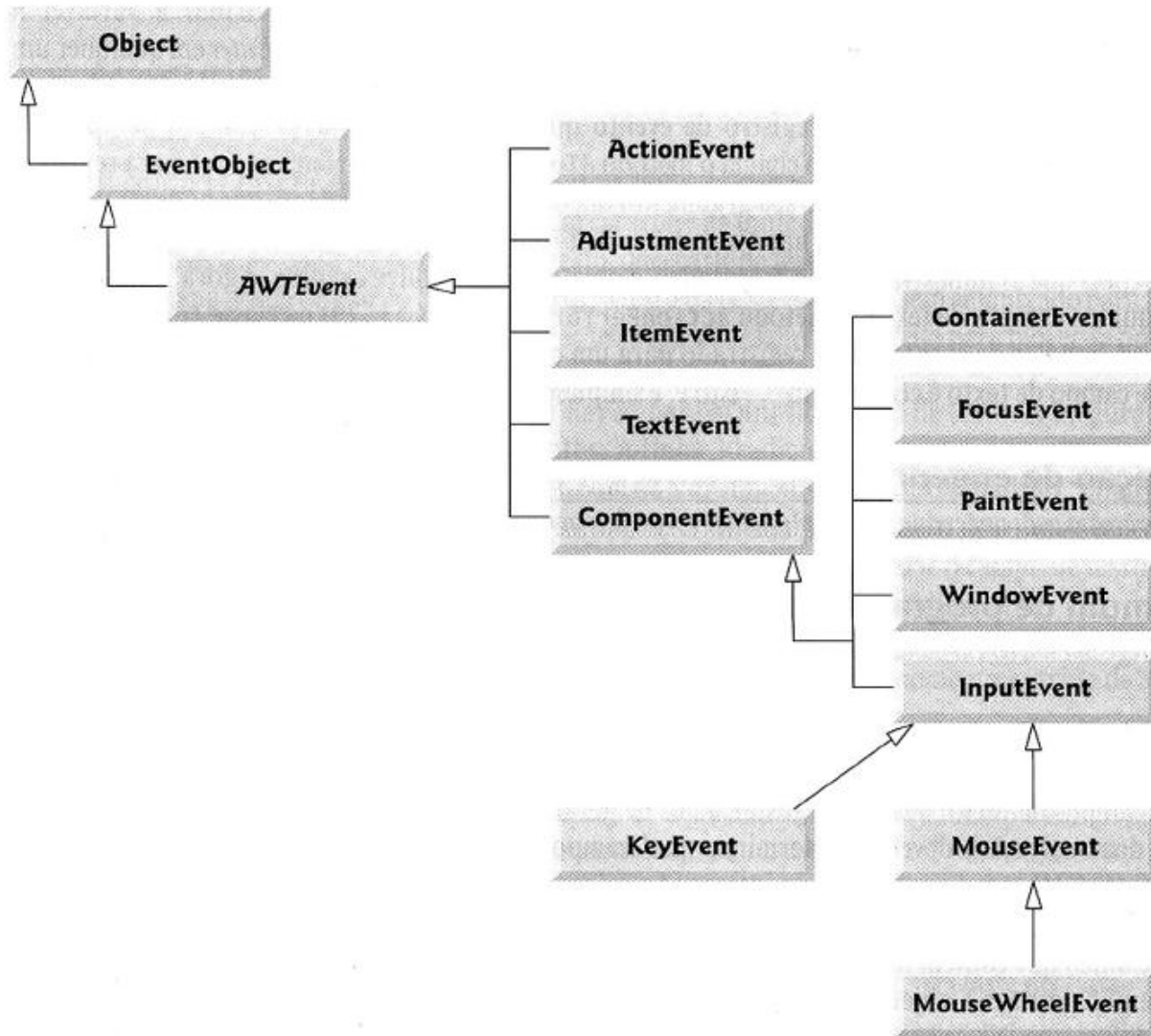
Nos próximos exemplos trabalharemos com muitos desses eventos.



# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

105



# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

106

Para cada tipo de objeto de evento, há uma interface ouvinte de evento apropriada. Para tratar esse evento, um objeto ouvinte de evento ou a classe dele precisa implementar a interface correspondente ao tipo de evento que ocorreu.

No caso da aplicação anterior, já se sabia que o evento era do tipo `ActionEvent`. Logo, a interface que a classe do objeto ouvinte de evento teve que implementar foi `ActionListener`.

Cada interface ouvinte de eventos especifica um ou mais métodos de tratamento de evento que devem ser declarados (redefinidos) na classe que implementa a interface. A interface `ActionListener` só especifica um método, `ActionPerformed()`, por isso, só é preciso redefinir esse método.

Um componente pode ter vários tipos de objeto ouvinte de evento registrados. No caso da aplicação anterior, só tinha um tipo de objeto ouvinte de evento registrado para cada componente.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

107

Quando o evento ocorre, o componente GUI que o usuário interagiu notifica (avisa) todos os seus objetos ouvintes de eventos (se houver mais de um registrado). Estes, depois de notificados, executam o método apropriado (aquele que foi definido dentro da sua classe).

### Componente JCheckBox

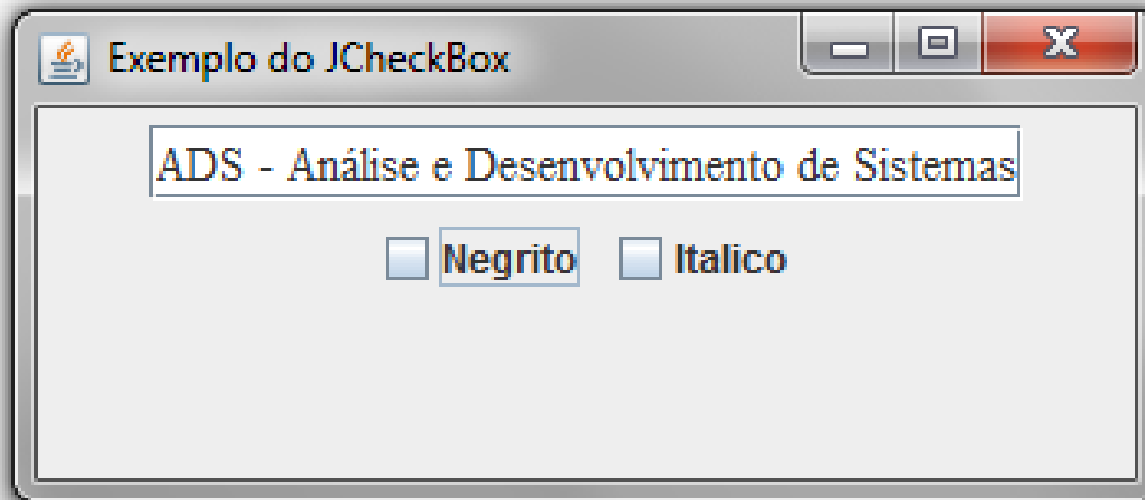
O componente JCheckBox é conhecido como componente caixa de seleção, pois sua interface exibe um pequeno quadro, onde o usuário pode selecionar (check) ou desmarcar a opção.

Quando o usuário interage com esse componente, ele mantém um estado interno selecionado ou não selecionado. Normalmente, esse componente é utilizado em situações onde o usuário pode marcar, escolher ou selecionar várias opções.

O usuário poderá interagir com os componentes JCheckBox marcando ou desmarcando as opções existentes. Quando um componente JCheckBox é marcado, um símbolo aparece e o estado do componente é configurado para selecionado. Se o usuário interage novamente, o símbolo desaparece e o estado do componente é configurado para não selecionado.

### Componente JCheckBox

O programa `exemploCheckBox1.java` apresenta como resultado a seguinte janela.



### Outros métodos do componente JCheckBox

- ❑ `void setEnabled(boolean b)` – habilita o JCheckBox.
- ❑ `void setText(String text)` – seta o label do JCheckBox.
- ❑ `void setSelected(boolean b)` – seta (ou marca) o checkbox

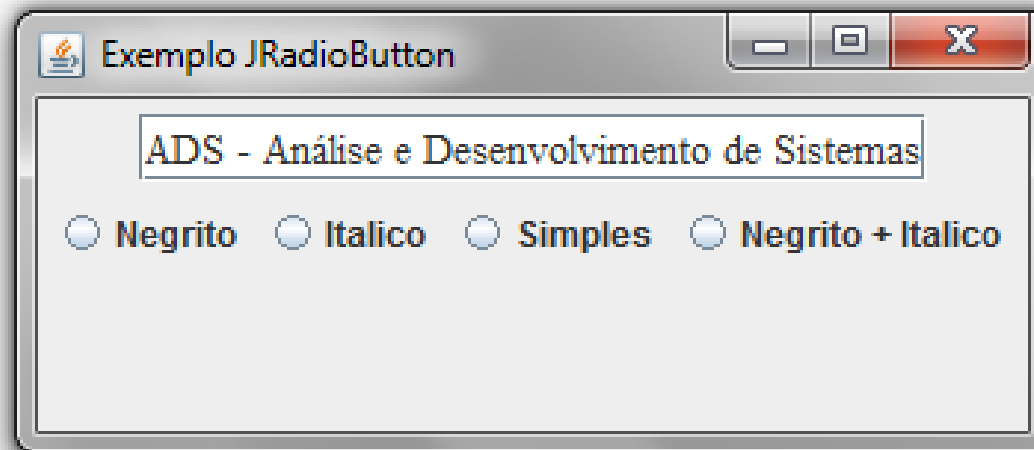
### Componente JRadioButton

O componente JRadioButton, também conhecido como botão de radio, é um componente que também mantém um estado interno selecionado ou não selecionado.

Normalmente, esse componente é utilizado com outros componentes JRadioButton, dentro de um grupo no qual somente uma opção deve ser escolhida, ou seja, um JRadioButton é selecionado. Assim, quando um componente é selecionado, o outro é desmarcado automaticamente.

## Componente JRadioButton

O programa `exemploRadioButton1.java` apresenta como resultado a seguinte janela.

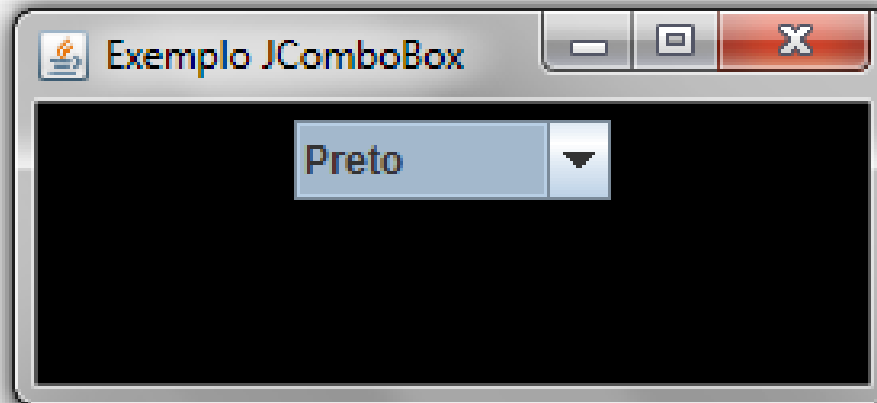




## Componente JComboBox

O componente JComboBox, conhecido como caixa de combinação ou caixa drop-down, fornece uma lista de itens das quais o usuário pode escolher uma única opção. Essa lista aparece quando a seta é clicada com o mouse. A interação do usuário com esse componente gera o objeto de evento do tipo `ItemEvent`, assim como `JCheckBox` e `JRadioButton`.

O programa `exemploComboBox1.java` apresenta como resultado a seguinte janela.



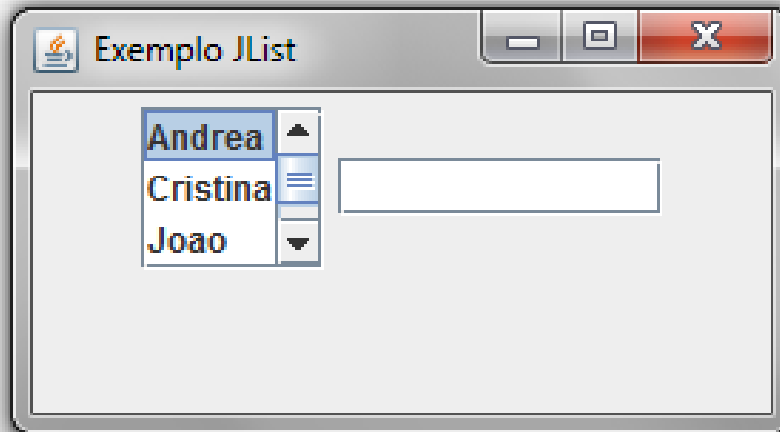
### Outros eventos do componente JComboBox

- ❑ `void addItem(Object anObject)` – adiciona um item à lista do combo.
- ❑ `Object getItemAt(int index)` – retorna o item pelo seu índice.
- ❑ `int getItemCount()` – retorna o número de itens.
- ❑ `Object getSelectedItem()` – retorna o item selecionado.
- ❑ `int getSelectedIndex()` – retorna o índice do item selecionado.
- ❑ `void insertItemAt(Object anObject, int index)` – insere um item em uma posição específica.
- ❑ `void removeItem(Object anObject)` – remove o item especificado.

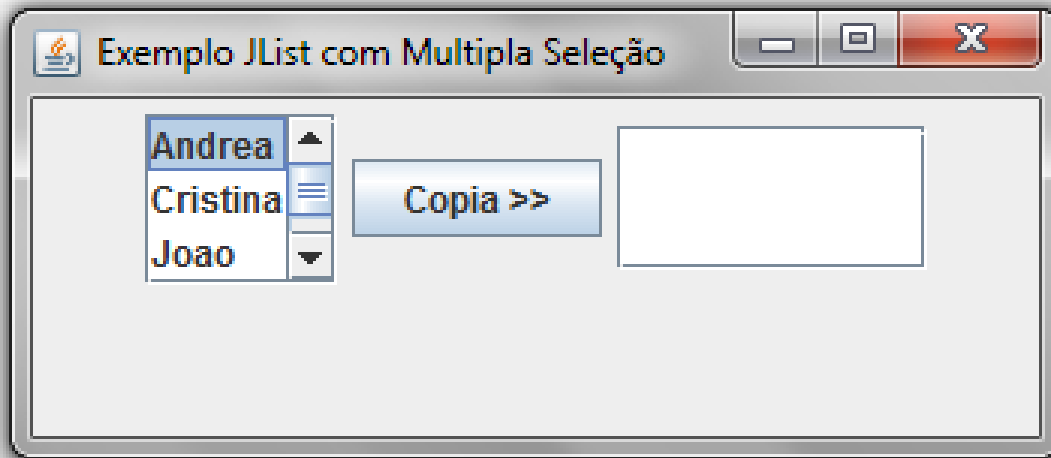
## Componente JList

O componente JList ou componente de lista permite que uma série de itens sejam exibidos numa caixa. O usuário pode escolher um ou mais itens dessa lista. JList permite a lista de seleção única, onde apenas um item pode ser escolhido, ou a lista de seleção múltipla, onde vários itens podem ser escolhidos.

A manipulação do componente JList gera objeto de evento do tipo ListSelectionEvent que deve ser tratado por um objeto que implementa a interface ListSelectionListener.



## Componente JList



exemploList2.java

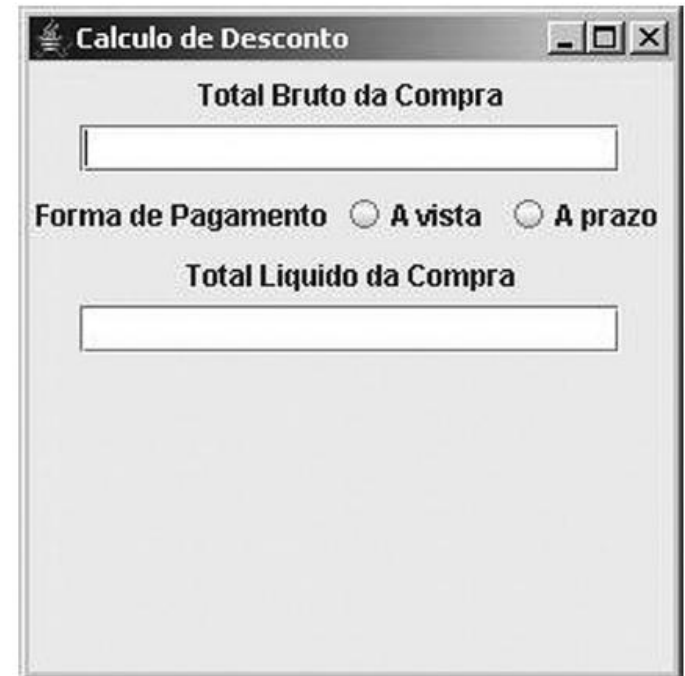
### Outros eventos do componente JList

- ❑ `void clearSelection()` – limpa a marca de seleção.
- ❑ `int getMinSelectionIndex()` e `int getMaxSelectionIndex()` – retorna o índice mínimo e índice máximo em uma seleção.
- ❑ `int getSelectedIndex()` e `int[] getSelectedIndices()` – retorna o índice dos itens selecionados.
- ❑ `Object getSelectedValue()` e `Object[] getSelectedValues()` – retorna os itens selecionados.
- ❑ `boolean isSelectionEmpty()` – retorna `true`, se não houver nada selecionado.

### Exercício

Desenvolva uma aplicação com a seguinte GUI.

Nessa aplicação, o usuário deve entrar com o total bruto da compra e escolher a forma de pagamento (à vista ou a prazo). Se for escolhida a forma de pagamento à vista, o usuário deve digitar o desconto no outro componente que vai aparecer somente nesse caso, conforme a GUI mostra na próxima página.



The screenshot shows a window titled "Calculo de Desconto" with a standard Windows-style title bar. Inside the window, there are three main sections: 1. "Total Bruto da Compra" with a text input field below it. 2. "Forma de Pagamento" with two radio buttons labeled "A vista" and "A prazo". 3. "Total Liquido da Compra" with a text input field below it. The window has a light gray background and a thin border.

### Exercício

Depois de digitar o desconto, o usuário apertará a tecla Enter nesse componente e o valor líquido aparecerá, calculado no campo total líquido.

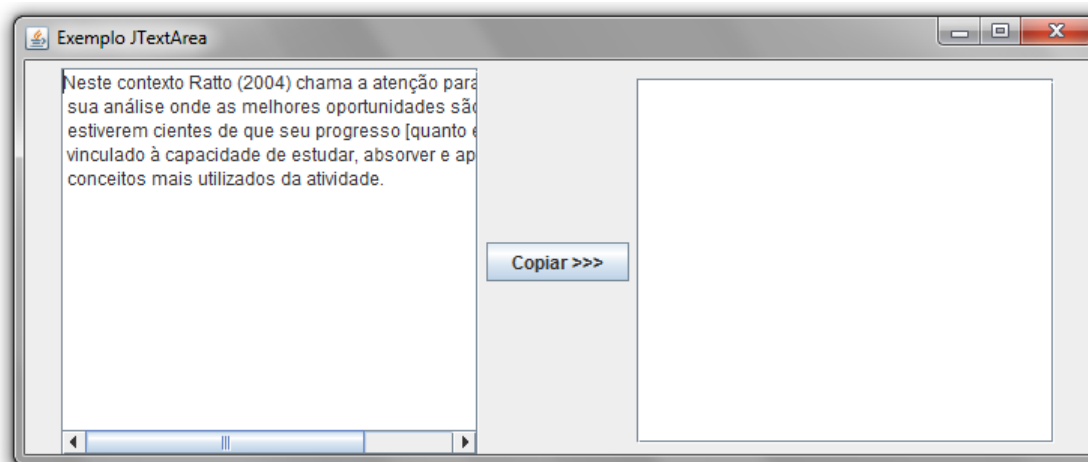
Caso seja escolhida a forma de pagamento a prazo, o componente de entrada do desconto não deve aparecer e o valor da compra deve ser mostrado no campo total líquido.

Dicas:

- 1) Para tornar visível ou não visível os componentes JLabel e JTextField use o método setVisible(). Passe o argumento true para deixar visível o componente e false para ocultar.
- 2) Nessa GUI, dois tipos de eventos devem ser tratados. ItemEvent, que é disparado quando o usuário interage com o componente JRadioButton, e (preste atenção!)(ActionEvent, quando o usuário interage com o componente JTextField, apertando a tecla Enter quando o foco está no componente.

## Componente JTextArea

- O componente JTextArea fornece uma área para manipular texto. Esse componente não gera eventos, assim como o componente JList de múltipla seleção. Nesse caso, é preciso existir outro componente que gere evento para se executar a ação desejada.
- O exemplo abaixo mostra uma aplicação onde um texto inicial é exibido em um dos componentes JTextArea. Veja que o usuário deve selecionar uma parte do texto e clicar no botão “Copiar”. O texto selecionado é copiado para o segundo componente JTextArea.



exemploJTextArea.java



### Gerenciador de Layout

Os gerenciadores de layout são utilizados para organizar os componentes GUI em um container. Os componentes GUI também podem ser organizados no container sem o uso de um gerenciador de layout.

Mas, nesse caso, o programador deve se preocupar com a posição absoluta do componente em relação ao container e também com o tamanho de cada componente. Isso pode ser extremamente trabalhoso!

Ao usar um gerenciador de layout você ganha rapidez para criar GUI's , mas pode perder algum controle sobre o tamanho e a posição exata de cada componente inserido da GUI.

Existe uma outra forma muito fácil e agradável de programar GUI. Use uma IDE (ambiente de desenvolvimento integrado) com recursos para desenhar GUI. Com esse tipo de ferramenta, você simplesmente escolhe o componente GUI numa barra de componentes, arrasta e solta o componente na posição que você quer da janela.

Existem vários gerenciadores de layout. Com eles, você poderá construir GUI's mais complexas, ou seja, com vários componentes inseridos e alinhados, da maneira que você deseja.

## FlowLayout

Já utilizamos esse gerenciador em atividades anteriores. É o gerenciador mais simples de ser utilizado. Com ele, os componentes são inseridos no container da esquerda para a direita e na ordem que você os adicionou no código-fonte. Quando algum componente alcança a borda do container, os demais são inseridos na próxima linha.

Esse gerenciador permite que os componentes sejam alinhados à esquerda, centralizados ou alinhados à direita, prevalecendo as características descritas anteriormente.

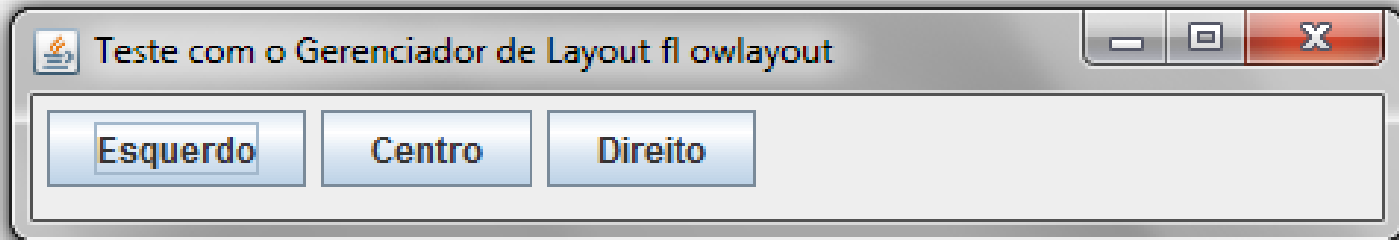
Se nenhum tipo de alinhamento for estabelecido, como foi o caso das nossas aplicações até agora, os componentes são centralizados. Esse gerenciador também respeita o tamanho dos componentes gerenciados por ele. No próximo exemplo veremos o uso do gerenciador FlowLayout de uma maneira um pouco diferente da usada até o momento.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

123

Na aplicação abaixo são exibidos apenas três botões, dispostos através do `FlowLayout`, inicialmente alinhados à esquerda.



`exemploFlowLayout.java`

## BorderLayout

O gerenciador `BorderLayout` é o gerenciador padrão de um `JFrame`. Assim, quando nenhum gerenciador for especificado, ele é utilizado. Esse gerenciador organiza os componentes GUI em cinco (5) regiões na tela: `NORTH` (norte), `SOUTH` (sul), `EAST` (leste), `WEST` (oeste) e `CENTER` (centro).

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

124

Com esse gerenciador, você pode inserir apenas cinco componentes no container, um em cada região. Não ache estranho, pois podemos inserir, nessas regiões, componentes (containers) como JPanel, que permitem que outros componentes GUI sejam adicionados a eles, construindo, com isso, GUIs mais complexas.

Os componentes inseridos nas regiões NORTH e SOUTH estendem-se horizontalmente para os lados do container e têm a altura do componente mais alto inserido nessas regiões.

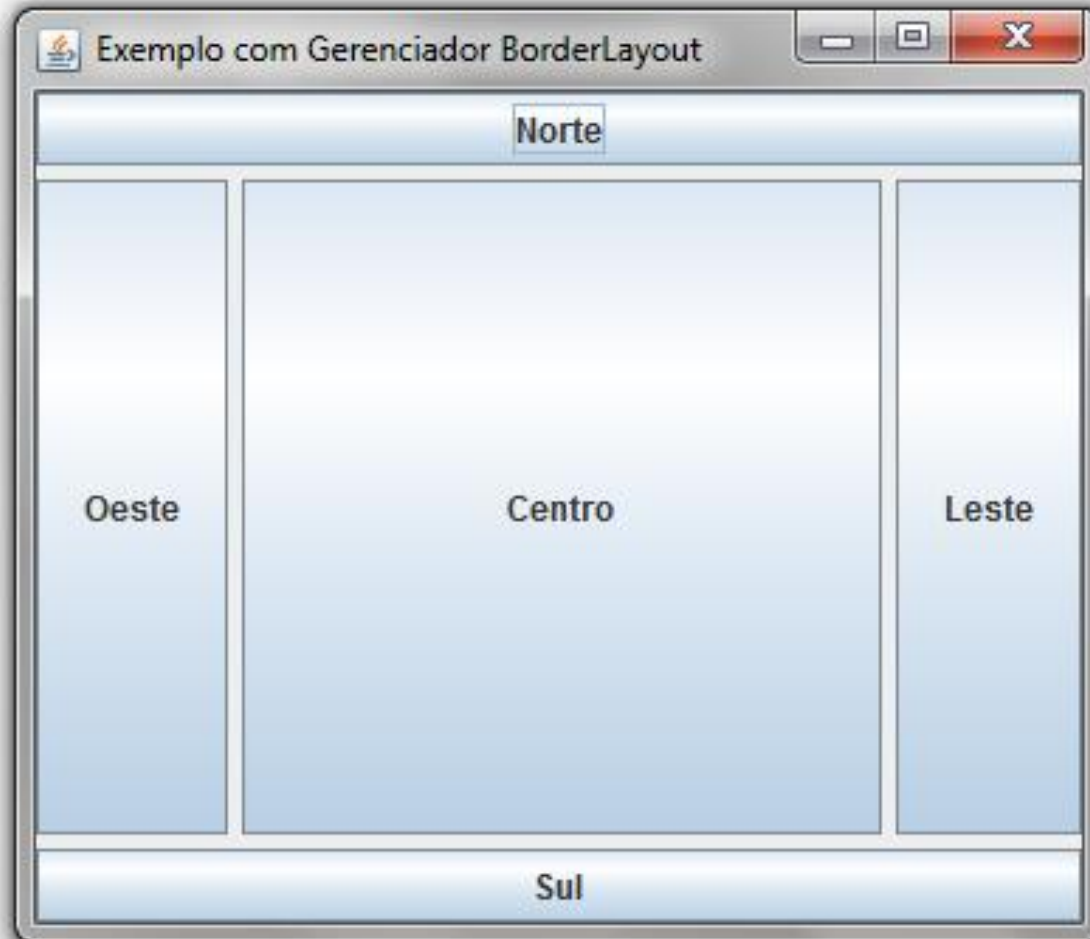
Os componentes inseridos nas regiões EAST e WEST estendem-se verticalmente entre as regiões NORTH e SOUTH e têm a mesma largura que os componentes colocados nas suas regiões. O componente inserido na região CENTER expande-se para ocupar todo o espaço restante do layout.

Se todas as regiões estão ocupadas, todo o espaço do container é preenchido. Se as regiões NORTH e SOUTH não estão ocupadas, os componentes das regiões EAST, WEST e CENTER estendem-se para ocupar essas duas regiões. Se as regiões EAST e WEST não estão ocupadas, o componente da região CENTER ocupa o espaço dessas duas regiões. Se a região CENTER não é ocupada, esta área fica vazia, ou seja, os componentes das outras regiões não ocupam esse espaço.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

125



exemploBorderLayout.java

## GridLayout

O gerenciador GridLayout permite que vários componentes GUI sejam inseridos dentro de uma grade ou tabela. A dimensão da grade, ou o número de linhas e colunas, é feita no construtor do objeto gerenciador de layout. Nesse gerenciador, cada componente tem a mesma largura e altura. Os componentes são inseridos a partir da primeira célula da primeira linha.

A próxima figura mostra a GUI de uma aplicação onde seis componentes JButton foram inseridos e organizados através do gerenciador GridLayout. Nessa aplicação, não foi programada qualquer ação para os componentes JButton. Ela objetiva apenas mostrar como funciona a disposição dos componentes segundo esse gerenciador de layout.

# Linguagem de Programação Orientada a Objetos

## GUI – Graphical User Interface

127



exemploGridLayout.java

## **JTabbedPane**

## **Componentes de Menu**



## Modificadores (Para relembrar)

Na linguagem Java, podemos utilizar algumas palavras-chave (keyword) para modificar o modo como são declaradas classes, métodos e variáveis (atributos). Veremos como aplicar cada um desses modificadores no texto a seguir.

Modificadores de acesso a membros (public, private, protected e acesso de pacote).

São considerados membros de uma classe as variáveis de instância (atributos) e os métodos da classe. Quando implementamos uma classe, sempre declaramos os seus membros (atributos ou métodos). No momento dessa declaração, utilizamos algum modificador de acesso. Os modificadores de acesso que podem ser utilizados são: public, private, protected e, quando nenhum modificador é especificado, se diz que o modificador de acesso é de pacote (package).

## Modificadores (Para relembrar)

É através dos modificadores de acesso, também conhecidos como qualificadores de acesso, que se controla o acesso aos membros da classe. Isso quer dizer que, os modificadores determinam como, ou melhor, de que lugar, esses membros (atributos e métodos) poderão ser acessados.

Digite código da classe abaixo:

funcionario.java

## Exercício

A seguinte classe tem dois métodos static. Implemente outra classe chamada UsaCalculos.java e demonstre como esses métodos static podem ser chamados dessa classe.

```
public class Calculos{
    public static int potencia(int base, int exp){
        int pote=1;
        for (int i=1;i<=exp;i++){
            pote = pote * base;
        }
        return pote;
    }

    public static int fatorial(int nu){
        int fat=1;
        for (int i=1;i<=nu;i++){
            fat = fat * nu;
        }
        return fat;
    }
}
```

## Associação

Ao modelar os atributos da classe Funcionário (lembre-se que ela representa os atributos e comportamentos de objetos Funcionário) podemos definir que o setor em que o funcionário trabalha é uma informação importante.

Também podemos pensar que essa informação é um objeto do problema (cadastrar informações sobre funcionários). Esse objeto possui atributos como código e nome do setor ao ser representado através de uma classe, possivelmente chamada de Setor, pode ser usado como atributo de outros objetos/Classe ou em outros programas.

Ao pensar em setor como um objeto, devemos representar esse objeto através de uma classe logo teremos mais uma classe no nosso problema. Ela pode ser chamada de Setor, pois representa os atributos e comportamentos de qualquer objeto Setor. Teremos, portanto, duas classes para representar os objetos do problema: Funcionário e Setor.

## Associação

O que ocasionou a criação da classe Setor foi pensarmos no atributo setor como um objeto. Portanto, se setor é um atributo de Funcionário e ele é do tipo Setor, a classe Funcionário possui um relacionamento com a classe Setor. Vamos chamar esse relacionamento de **Associação**.

A figura no próximo slide ilustra, em UML, o relacionamento de associação entre a classe Funcionário e a classe Setor.

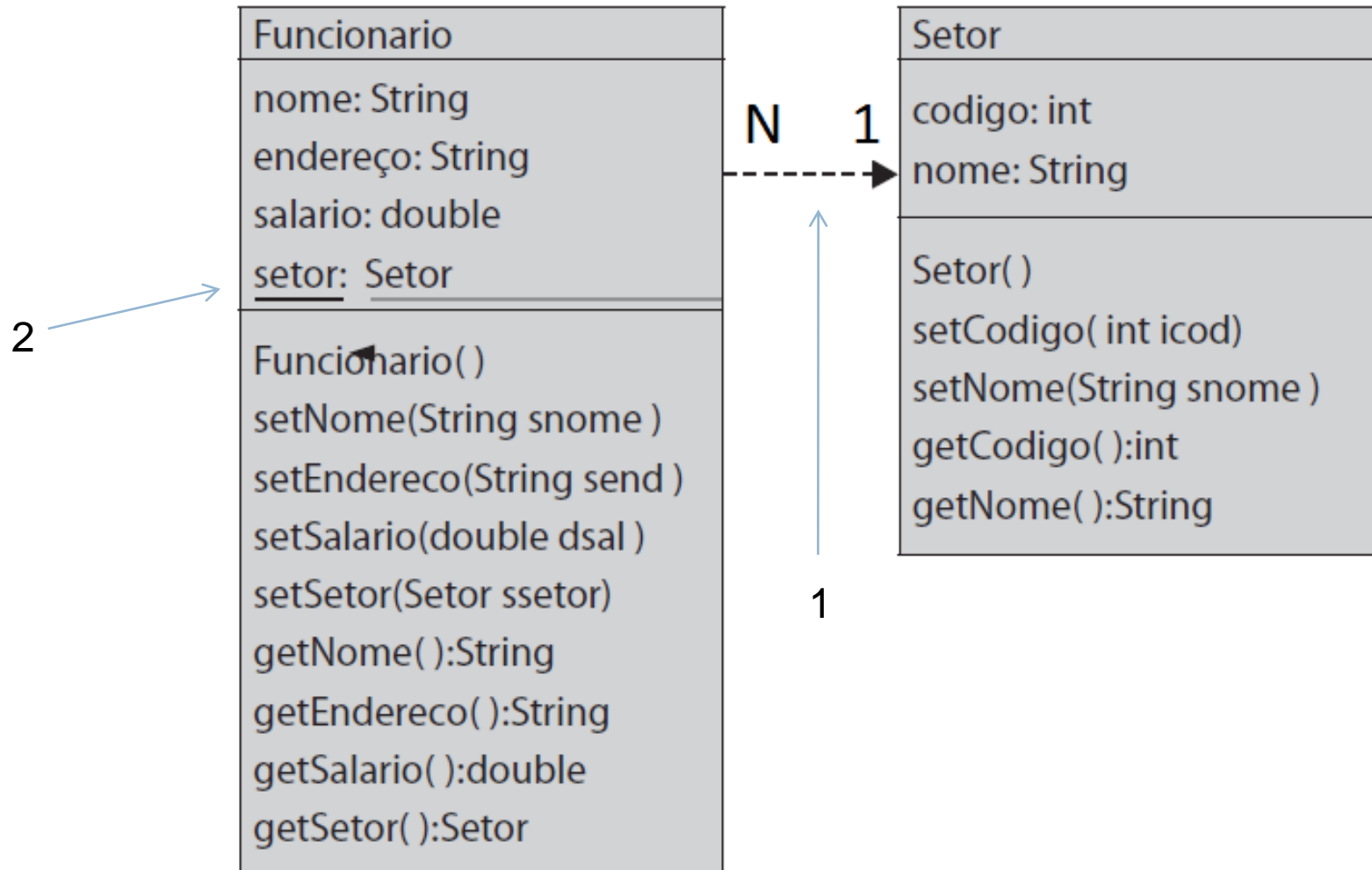
## Associação

- 1 Essa linha indica o relacionamento de associação entre as classes Funcionário e Setor. Os valores em cada extremidade indicam a multiplicidade desse relacionamento. A multiplicidade indica quantas vezes um objeto Funcionário está associado a um objeto Setor e vice-versa.
- 2 Observe que o atributo setor é do tipo Setor. Isso quer dizer que entre os atributos de um objeto Funcionário existirá uma referência para um objeto do tipo Setor.

# Linguagem de Programação Orientada a Objetos

## Associação

135



## Associação

Por exemplo: na figura anterior, um objeto Funcionário está associado a somente um objeto Setor (veja cardinalidade 1 na extremidade direita). Isso quer dizer que um objeto Funcionário fará referência a, no máximo, 1 objeto Setor. Por outro lado, um objeto Setor pode ser referenciado por vários objetos do tipo Funcionário.

A seta no sentido da direita indica que somente o objeto Funcionário tem conhecimento do objeto Setor.

setor.java  
ffuncionario.java



## Exercício – Controle de Afazeres

Proposta: Desenha o diagrama de classe e implementar os diagramas (Classes) em Java.

### Requisitos:

Ricardo pretende desenvolver um aplicativo para controlar as suas tarefas e colocar em seu Palm. As especificações da aplicação são as seguintes:

O cadastro de cada tarefa contém uma determinada prioridade, representado por um dado do tipo float. Além da prioridade, o cadastro deverá conter: nome da tarefa, a data máxima para execução, o percentual de conclusão da tarefa e o detalhamento da tarefa.

Para cada tarefa existe uma lista de itens que descrevem sua execução. Para cada item de execução, pode-se cadastrar.

- O percentual correspondente; A descrição da execução; A data de execução

## Definição de Dados do MySql

- Create Database <nome\_database>;
  - Cria um database no MySql
- Drop Database <nome\_database>;
  - Remove um Database no MySql
- Use <nome\_database>;
  - Acessa um Database no MySql
- Create table <nome\_tabela> (coluna1 tipo, ..., colunan tipo);
  - Cria uma tabela de banco de dados

## Definição de Dados do MySQL

- Drop table <nome\_tabela>;
  - Remove uma tabela do Banco de Dados;
- Alter table <nome\_tabela> Modify / Add coluna tipo;
  - Altera as colunas de uma tabela
- Create Index <nome\_indice> on nome\_tabela(coluna);
  - Cria um índice para a tabela
- Drop Index <nome\_indice> on tabela;
  - Remove o índice de uma tabela

## Definição de Dados do MySQL

- Drop table <nome\_tabela>;
  - Remove uma tabela do Banco de Dados;
- Alter table <nome\_tabela> Modify / Add coluna tipo;
  - Altera as colunas de uma tabela
- Create Index <nome\_indice> on nome\_tabela(coluna);
  - Cria um índice para a tabela
- Drop Index <nome\_indice> on tabela;
  - Remove o índice de uma tabela

## Conectando o Java ao Banco de Dados - Driver de Conexão

Uma funcionalidade essencial em qualquer software é a sua capacidade de se comunicar com um banco de dados, independentemente da linguagem de programação utilizada para implementação.

A linguagem de programação possui uma sintaxe própria e as ferramentas de banco de dados funcionam baseadas na linguagem SQL – o que significa que estas duas tecnologias não conseguem conversar entre si. É como se duas pessoas de idiomas diferentes comessem a conversar, sem que um entendesse o outro. Seria um problema, não acha?

### Conectando o Java ao Banco de Dados - Driver de Conexão



## Conectando o Java ao Banco de Dados - Driver de Conexão

No Java, os drivers ODBC recebem o nome de JDBC, ou seja, driver ODBC específico para Java. O driver JDBC é o driver ODBC que permite a conexão entre um banco de dados e o programa escrito em Java. Seria o intérprete entre um sistema desenvolvido em Java e um banco de dados, possibilitando a comunicação entre eles.



## Conectando o Java ao Banco de Dados - Driver de Conexão

Com base na ferramenta que será utilizada como banco de dados, acesse o site do fabricante e procure, na área de download, pelo driver de conexão JDBC. Para facilitar, acesse o link: <http://dev.mysql.com/downloads/connector/j/3.1.html>, para baixar o JDBC para MySQL. Esse driver será usado por você para conexão do software. Veja o site de download deste driver de conexão:



# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

145

The screenshot shows a Mozilla Firefox browser window with the address bar displaying <http://dev.mysql.com/downloads/connector/j/3.1.html>. The page is the MySQL Connector/J download page. The MySQL logo is at the top left, with the tagline "The world's most popular open source database". The navigation bar includes "Developer Zone", "Downloads" (selected), and "Documentation". Under "Downloads", there are tabs for "Current", "Archives", "Snapshots", and "Mirrors". The left sidebar lists various MySQL products, with "MySQL Connectors" expanded to show "Connector/J". The main content area is titled "Download Connector/J" and describes it as the official JDBC driver for MySQL. It includes links to "Online Documentation" and "Connector/J Documentation and Change History". A blue box on the right states: "MySQL open source software is provided under the GPL License. OEMs, ISVs and VARs can purchase commercial licenses." Below this, it says "Please report any bugs or inconsistencies you observe to our [Bugs Database](#). Thank you for your support!". A section titled "Generally Available (GA) Releases" features a "Connector-J 3.1.14" heading. It has a "Select Version:" dropdown menu with "3.1.14" selected and a "Select" button. Below it is a "Select Platform:" label. A light blue box on the right says "Looking for the latest GA version?". The browser's status bar at the bottom shows "Concluído" and "YSlow". The Windows taskbar on the right shows the date "31/05/2011" and time "21:50".

MySQL :: Download Connector/J - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

<http://dev.mysql.com/downloads/connector/j/3.1.html>

Mais visitados Primeiros passos Últimas notícias O Evento - Web Expo F...

Norton

MySQL :: Download Connector/J

MySQL The world's most popular open source database

Search Login Register

Developer Zone Downloads Documentation

Current Archives Snapshots Mirrors

### Download Connector/J

MySQL Connector/J is the official JDBC driver for MySQL.

Online Documentation

- Connector/J Documentation and Change History

Please report any bugs or inconsistencies you observe to our [Bugs Database](#).  
**Thank you for your support!**

MySQL open source software is provided under the GPL License.  
OEMs, ISVs and VARs can purchase commercial licenses.

Generally Available (GA) Releases

#### Connector-J 3.1.14

Select Version:

3.1.14 Select

Select Platform:

Looking for the latest GA version?

Concluído YSlow

PT 21:50 31/05/2011

# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

146

MySQL :: Select a Mirror to Start Downloading - mysql-connector-java-3.1.14.zip - Mozilla Firefox

Arquivo Editar Exibir Histórico Favoritos Ferramentas Ajuda

http://dev.mysql.com/downloads/mirror.php?id=13520

Mais visitados Primeiros passos Últimas notícias O Evento - Web Expo F...

Norton Cartões & Logins

MySQL :: Select a Mirror to Start Do...

Current Archives Snapshots Mirrors

### Select a Mirror to Start Downloading - mysql-connector-java-3.1.14.zip

**13 April 2011** - Individuals recently claimed to have hacked portions of the MySQL.com and Sun.com web sites. The results of Oracle's preliminary investigation do not indicate a wide compromise; however the attackers published a small number of user IDs, e-mails, and passwords. In an abundance of caution, Oracle recommends that users who had accounts on these systems change their passwords as soon as possible. Furthermore, users who employed the same passwords on other systems should change them as well. Guidelines for choosing strong passwords are readily available on the Internet, including on Wikipedia ([http://en.wikipedia.org/wiki/Password\\_strength](http://en.wikipedia.org/wiki/Password_strength)).

Please take the time to let us know about you.

If this is the first time you have downloaded from us, you will be sent a password to enable you to log into all of the MySQL web sites, including forums and bugs.

If you already have a MySQL.com account, save time by logging in now.

Returning Users	New Users
Save time by logging in	Proceed with registration
Email: <input type="text"/>	
Password: <input type="password"/>	
<a href="#">Forgot your password?</a>	

Primary mirrors hosted by:

- HURRICANE ELECTRIC INTERNET SERVICES
- University of Kent UKMIRROR service
- GWDG

Concluido

YSlow

21:52 31/05/2011

## Criando uma classe em Java para conexão com Banco de Dados

A conexão com um banco de dados, através da linguagem Java pode ser dividida em três processos distintos:

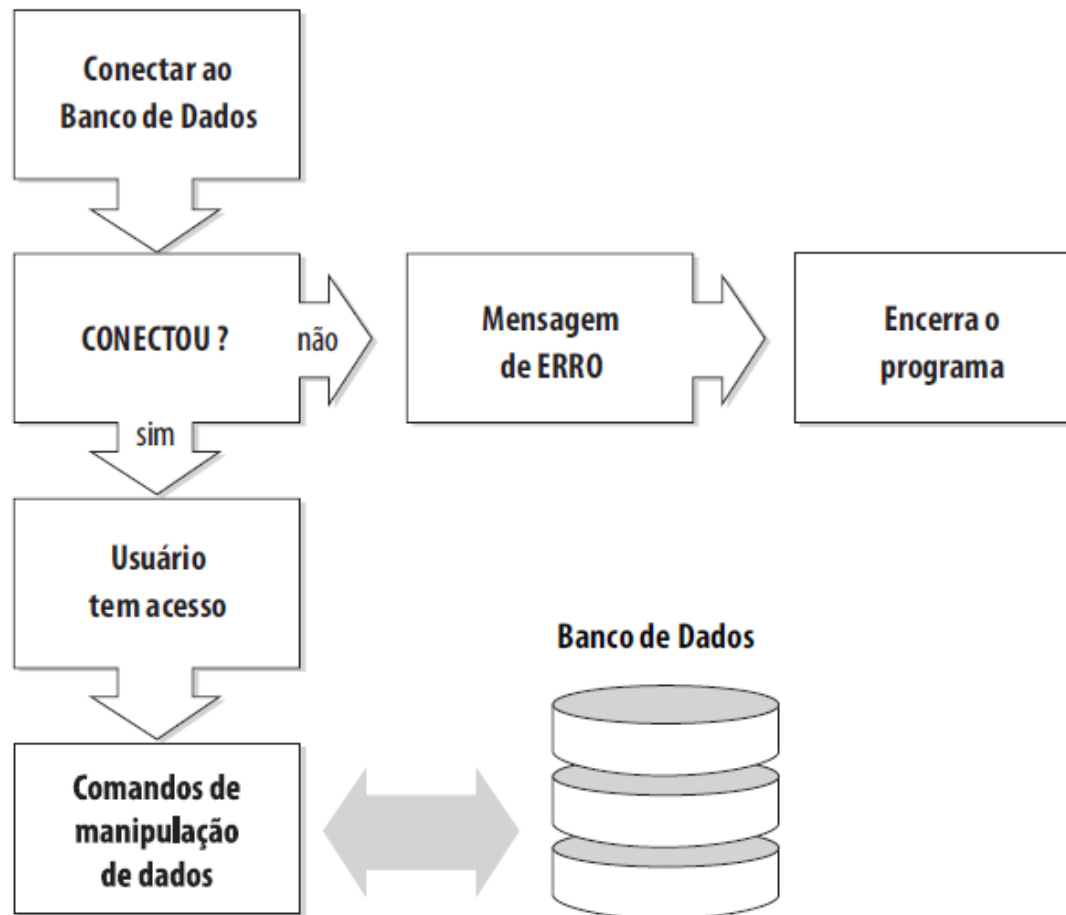
- conectar ao banco de dados;
- executar comandos com recuperação de dados – Select;
- executar comandos sem recuperação de dados – Insert, Update e Delete.

Vale destacar que, se o primeiro processo não for efetivado com sucesso, os demais não podem ser realizados.

# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

148



## Criando uma classe em Java para conexão com Banco de Dados

A conexão com o banco de dados se dá pela conexão JDBC, utilizando a classe chamada DriverManager, que poderia ser traduzido para administrador do driver. Esta classe pertence ao pacote java.sql e serve para:

- criar a conexão com o banco de dados;
- administrar o driver JDBC usado na conexão;
- controlar o acesso ao banco de dados via Login;
- controlar a troca de informação entre o driver e a ferramenta de banco de dados.

## **Criando uma classe em Java para conexão com Banco de Dados**

Como o DriverManager pertence ao pacote `java.sql`, sempre que você for utilizá-lo deverá incluir a linha `import java.sql.*` no código fonte da sua classe, pois este não é um pacote nativo do Java.

## Criando uma classe em Java para conexão com Banco de Dados

A sintaxe do uso desta classe é bem simples, assim como a finalidade dela é especificamente a conexão com o banco de dados, os seus atributos se referem a dados como:

- nome do servidor de acesso, a porta de acesso (3306 – mysql) e o nome do database;
- nome do usuário;
- senha de acesso ao banco de dados.

## Criando uma classe em Java para conexão com Banco de Dados

Para que a classe DriverManager receba estes valores, é necessária a utilização do método getConnection(), que será melhor detalhado a seguir.

Assim, a sintaxe de uso do DriverManager é:

`DriverManager.getConnection(URL, usuário, senha);` onde:

- URL: String;
- Nome do Usuário: String;
- Senha: String;



# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

153

Os nomes do usuário e da senha de acesso seguem as regras da definição do banco de dados. No caso do banco de dados MySQL instalado por você, será a mesma senha e usuário que você usa para se logar ao MySQL.

O método `getConnection()` retorna sempre um atributo do tipo `Connection`, que pode ser fechado por meio do método `close()`. A partir do momento que o banco de dados foi fechado, ele não pode mais ser manipulado, a não ser que uma nova conexão seja aberta através do `getConnection()`.

Sempre que você encerrar a aplicação, feche o banco de dados. Os comandos do `getConnection()` são sempre os mesmos, o que pode mudar são os parâmetros de nome da fonte, usuário e senha. Sendo assim, a conexão com banco de dados poderia ser representada por uma classe como:

# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

154

Conexao
<b>String</b> Usuario <b>String</b> Senha <b>String</b> Servidor <b>String</b> DataBase <b>Connection</b> Con <b>boolean</b> Conectado <b>ResultSet</b> Dados
<b>Conexao()</b> <b>Conectar()</b> <b>FecharConexao()</b> <b>ExpressaoSQL(String Comando)</b>

# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

155

**Usuário** – nome do usuário que se conectará ao MySQL;

**Senha** – senha de acesso ao MySQL;

**Servidor** – servidor de onde o MySQL será acessado;

**DataBase** – banco de dados que será acessado;

**Con** – classe do tipo Connection do próprio Java que faz a conexão com o banco de dados;

**Conectado** – campo lógico que indica se a conexão foi realizada;

**Dados** – classe do tipo ResultSet, do Java, que armazena os resultados de um comando “Select”;

**Conexao()** – método que instancia a classe Conexão;

**Conectar()** – método que faz a conexão com o banco de dados;

**FecharConexao()** – método que fecha a conexão com o banco de dados;

**ExpressaoSQL** – método que executa um comando de SQL.

# Linguagem de Programação Orientada a Objetos

## Conectando o Java ao Banco de Dados

156

Crie uma nova classe chamada `Conexao.java` para este projeto e edite o seguinte código-fonte:

# Linguagem de Programação Orientada a Objetos

## Conteúdo

157

- ▣ Introdução a GUI (Graphical User Interface)
- ▣ Componentes de uma GUI:
  - JFrame;
  - JLabel;
  - JButton; JButton com imagem;
  - JPasswordField;
  - JFormattedTextField;
- ▣ Tratamentos de Eventos em uma GUI
  - JButton – Tratamento de Eventos com classe interna anônima;
  - JCheckBox;
  - JCheckBox – Evento implementado com classe anônima;