unis
Grupo Educacional

Guia de Estudo

Programação Orientada a Objetos II

unis
Centro Universitário
Grupo unis

Instituição credenciada pelo MEC
Centro Universitário do Sul de Minas

Instituição Credenciada Unis
Credenciado pelo MEC
Portaria 4.385/05

SABE – Sistema Aberto de Educação

Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223

Instituição Credenciada pelo MEC – Portaria 4.385/05

Centro Universitário do Sul de Minas - UNIS
Unidade de Gestão da Educação a Distância – GEaD

Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG

005.26 Magalhães, Demétrio Reno.
M188g Guia de Estudo – Programação Orientada a Objetos II. Demétrio Renó Magalhães. Varginha: GEaD-UNIS/MG, 2008.
130p.

1. Objetos. Ch 1. 2. Classe 1. 3. Polimorfismo 1. 4.Herança 1. 5.Métodos 1. 6. Atributos I. Título.

REITOR
Prof. Ms. Stefano Barra Gazzola

GESTOR
Prof. Ms. Wanderson Gomes de Souza

Supervisora Técnica
Profª. Ms. Simone de Paula Teodoro Moreira

Design Instrucional
Prof. Celso Augusto dos Santos Gomes
Jacqueline Aparecida Silva

Coord. do Núcleo de Comunicação
Renato de Brito

Coord. do Núcleo de Recursos Tecnológicos
Lúcio Henrique de Oliveira

Equipe de Tecnologia Educacional
Profª. Débora Cristina Francisco Barbosa
Danúbia Pinheiro Teixeira

Revisão ortográfica / gramatical
Gisele Silva Ferreira

Autor
Demétrio Renó Magalhães

Graduado em Matemática Aplicada a Informática pela Fundação de Ensino e Pesquisa de Itajubá (FEPI) e Mestre em Engenharia Elétrica pela Escola Federal de Engenharia de Itajubá (EFEI). Professor no UnilesteMG nos cursos de Computação - Sistemas de Informação e Engenharias Mecânica, Elétrica, Produção, Sanitária e Ambiental e Materiais, pesquisador do Laboratório de Sistemas de Tempo Real (LTR), coordenador de pesquisa no curso de Computação – Sistemas de Informação, coordenador do Laboratório de Inteligência Computacional e Instrutor de Treinamentos.

ÍCONES

REALIZE. Determina a existência de atividade a ser realizada.
Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.

PESQUISE. Indica a exigência de pesquisa a ser realizada na busca por mais informação.

PENSE. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.

CONCLUSÃO. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.

IMPORTANTE. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.

HIPERLINK. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.

EXEMPLO. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.

SUGESTÃO DE LEITURA. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.

APLICAÇÃO PROFISSIONAL. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.

CHECKLIST ou PROCEDIMENTO. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.

SAIBA MAIS. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.

REVENDO. Indica a necessidade de rever conceitos estudados anteriormente.

SUMÁRIO

APRESENTAÇÃO

Prezado (a) aluno (a),

Este é o Guia de Estudos da disciplina Programação Orientada a Objetos II que iremos utilizar no curso de Sistemas de Informação do Centro do Sul de Minas – UNIS.

O guia o ajudará em suas atividades para melhor compreensão do conteúdo. Aqui você verá muitos códigos exemplos de programas.

As respostas dos exercícios propostos estão no final do guia.

Uma observação que faço para todos os alunos é que ao aprender programação, DIGITAR o programa é FUNDAMENTAL para um melhor aprendizado. Quando copiamos e colamos um código e ele funciona ficamos satisfeitos, mas não aprendemos como escrever o código.

A programação orientada a objetos possui vários comandos que às vezes parecem complexos e quando você digita o programa exemplo aprenderá muito mais e não terá dificuldades para resolver os exercícios.

Prof. Demétrio Renó Magalhães

EMENTA

Construção de interface gráfica com o usuário avançado; Introdução a applets; Tratamento de exceções; Conexão com banco de dados (JDBC); Desenvovimento de sistemas em camadas através do padrão MVC (Model-View-Controler); Persistência de Dados com Hibernate.

INTRODUÇÃO

A disciplina de Programação Orientada a Objetos II é bem ampla e nela estudaremos como programar interfaces gráficas avançadas usando menus com ícones, configurar tabelas de dados, frames internos, barras de ferramentas, formatação de dados. Também iremos estudar como construir applets em Java. Os applets são programas em Java que são executados no ambiente da Internet. Escreveremos alguns programas que serão executados em browsers. O tratamento de exceções também é um tópico que será abordado nesse guia. O tratamento de exceções torna os programas mais robustos. Banco de dados é uma das formas mais comuns de armazenamento de arquivos pelas corporações hoje em dia e esse também será um tópico estudado nessa disciplina.

Uma técnica utilizada na programação de computadores é a MVC (Model – View – Controler) em que a programação é bem definida e de fácil manutenção e finalmente veremos como programar em Java utilizando hibernate. Quando programamos com Hibernate podemos manipular bancos de dados sem escrever códigos da linguagem SQL no programa. Utilizamos métodos para manipular os dados e os códigos em HTML ficam separados da programação.

Bem isso é o que iremos trabalhar. Como podemos ver temos bastante trabalho pela frente.

Cada tema será abordado de forma objetiva com exemplos e exercícios para que você resolva e desenvolva a arte da programação.

Bem vindo a Programação Orientada a Objetos II

1 UNIDADE I

METAS

- Depois de ler a unidade o aluno deverá escrever códigos que trate erros através das classes de tratamento de exceções.

OBJETIVOS

- Definir e utilizar exceções em Java
- Utilizar manipulação de exceções na manipulação de arquivos em Java

PRÉ-REQUISITOS

- Para a realização dessa atividade o aluno deverá saber identificar situações que poderão causar erros e entender a lógica do programa. Será necessário também a utilização do *Kit de Desenvolvimento Java (JDK)* e o a IDE Eclipse.

1.1 MANIPULAÇÕES DE EXCEÇÕES EM JAVA

Uma exceção é diferente de um erro. Uma exceção é algo inesperado que pode acontecer durante a execuação de um programa e que podemos tratar não deixando que o sistema interrompa o funcionamento da máquina.
O tratamento de exceções é um mecanismo simples que torna os programas mais robustos. Um erro também é algo que pode acontecer durante a execução de um programa mas que não podemos recuperar.
Vejamos um exemplo:

- Falhas no tratamento de arquivos (tentar abrir arquivos que não existe, escrever em aquivo protegido...);
- Entrada inválidade dados;
- Falhas na comunicação entre processos;
- Erros aritméticos;
- Estouro de limites de arrays;
- Divisão por zero;
- etc.

Nos programas em que não temos um tratamento de exceções fazemos esse tratamento utilizando as instruções de decisão se...então (if...else).

Vejamos um exemplo:

```
em um programa ...
codErro = 0;
Comandos1
if (!erro1){      //Verifica se aconteceu um erro
  Comandos2;      //Executa Comandos2 se não aconteceu erro 1
  if (!erro2) {   //Verifica se aconteceu um outro tipo de erro
    Comandos3;    //Executa Comandos3 se não aconteceu erro 2
    Comandos3
```

```
    if (!erro3) { //Verifica se aconteceu um outro tipo de erro
          ...     //Caso não tenha acontecido o erro3 executa
outros comanos
    }
    else {
    codErro = -3; //Atribui um valor a variável codErro
       }
     } else {
      codErro = -2; //atribui um valor a variãvel codErro
     }
 } else {
    codErro = -1;   //atribui um valor a variável codErro
 }
 if (codErro < 0)   //Verifica o valor atual da variável
codErro
    tratarErro(codigoErro); //Trata o erro
    ...
```

Você pode observar que o código acima é bem confuso e fica bem poluído também.

Com o tratamento de exceções o código fica bem mais claro pois mostra claramente a separação entre o fluxo normal de execução do programa e o tratamento de condições excepcionais (exceções).

Vejamos agora um exemplo utilizando o tratamento de exceções

```
...
 try {

   Comandos1;
   Comandos2;
   Comandos3;
```

```
 } catch (erro1) {
   //Trata o erro1 capturado pela exceção

} catch (erro2) {
   //Trata o erro2 capturado pela exceção

} catch (erro3) {
   //Trata o erro3 capturado pela exceção

} finally{
   //Comando que SEMPRE será executado independentemente se
ocorreu ou não uma exceção.
    //Um exemplo seria a liberação de recursos (arquivos,
conexões....).
 }
 ...
```

O código acima mostra muito bem a diferença entre os comandos do programa e o tratamento de exceções.

O bloco **try** (tentar) é uma área chamada de risco, isto é, no bloco **try** devemos colocar os comandos que possívelmente poderão disparar uma exceção (abrir um arquivo por exemplo. Abrir um arquivo é uma operação de risco porque o arquivo pode não existir ou estar corrompido por exemplo. Sendo assim seu programa não irá parar de executar. Você pode enviar uma mensagem para o usuário dizendo o que ocorreu sem que o aplicativo interrompa o funcionamento da máquina.

O tratamento de exceções é feito nos blocos **catch** (captura) a exceção lançada e faz o tratamento.

O bloco **finally** colocado acima possui comandos também. O bloco **finally** é opcional.

Os comandos colocados dentro de um bloco **finally** são executados independentemente de ser ou não disparada uma exceção.

Geralmente utilizamos o bloco **finally** quando queremos liberar algum recurso como conexões com banco de dados, impressoras, arquivos etc..

A linguagem Java possui uma classe chamada **Exception** da qual utilizamos para tratamento de exceções. Exemplo: tratamento de arquivos, erro de divisão por zero, estouro no tamanho de um array etc.

Em um programa que faz a adição, subtração, multiplicação e divisão de dois números é necessário utilizar o tratamento de exceções?

Faça um programa com interface gráfica que faça a adição, multiplicação, subtração e divisão de dois números inteiros. Use o tratamento de exceções.

1.2 CRIANDO SUAS EXCEÇÕES

Na linguagem Java é possível construir nossas exceções quando as que fazem parte da linguagem não atendam nossa necessidade.
Para criar uma exceção devemos:

1. Criar uma classe com pelo menos um método que irá tratar a exceção.
2. O método deve ter a cláusula throws que irá disparar a exceção.

O código abaixo programa o método valorMenorQueDez(int x) para disparar uma exceção caso o valor de x seja menor que 10. Se o valor de x for maior ou igual a 10 nada irá acontecer. Vejamos o código:

```
class ClasseExcecao {
   void valorMenorQueDez(int x) throws MinhaExcecao {
     if (x  < 10) {
       throw new MinhaExcecao();
     }
   }
 }
```

Podemos observar no código acima que o método valorMenorQueDez que está na classe ClasseExcecao tem como parâmetro um valor do tipo inteiro chamado x e que ele "arremessa" (throws) uma exceção que está no método valorMenorQueDez da classe ClasseExcecao quando o valor de x for menor que dez. Em outras palavras, o método valorMenorQueDez **só aceita números que são maiores que 10** caso contrário ele dispara a exceção que está na classe MinhaExcecao utilizando o método valorDoParametro. Essa exceção é disparada toda vez que o valor de parâmetro passado para x for menor que 10.
Devemos agora escrever a classe que tem os métodos tratadores de exceção:

```
class MinhaExcecao extends Exception {
   public String valorDoParametro(){
     return "Valor menor ao valor solicitado";
   }
 }
```

A classe acima contém o método que será acionado quando uma exceção acontecer (i.e quando o valor de x for menor que 10). Nesse exemplo uma mensagem de erro será emitida ao usuário.
Agora escreveremos a aplicação. Vejamos:

```
public class TestaMinhaExcecao {
   public static void main(String[] args) {
     ClasseExcecao ce = new ClasseExcecao();
       try{
         ce.valorMenorQueDez(9);
       }catch(MinhaExcecao e){
       System.out.println(e.valorDoParametro());
       }
   }
 }
```

O código acima instancia um objeto da classe ClasseExcecao e invoca o método valorMenorQueDez(9) utilizando como parâmetro o valor 9. Essa invocação está entre um bloco **try** que é um bloco onde colocamos códigos que são passíveis de disparar uma exceção. Como o valor do parâmetro foi 9 que é menor que 10 então uma exceção é disparada e capturada pelo bloco catch que tem como argumento e que é do tipo MinhaExecao e então o método valorDoParametro é invocado através da variável de referência e a mensagem de texto é exibida para o usuário.

Podemos ter mais de um método dentro de uma classe que trata exceções. No exemplo anterior se o valor do parâmetro passado fosse um valor maior ou igual a 10 então nada aconteceria e o programa continuaria executando seu fluxo normalmente.

Pesquise sobre o tratamento de exceções para arrays. Qual o tratamento mais comum que utilizamos em arrays?

Faça um programa com interface gráfica que tenha dois botões um chamado carregar e outro chamado parar. Quando o usuário pressionar o botão carregar o programa deverá gerar um número inteiro e armazenar no vetor. O vetor deve ter 20 posições. Ao chegar na posição 5 se o número sorteado for impar

ele poderá ser colocado na posição do array. Caso o número seja par o programa deverá disparar uma exceção. Essa exceção não existe na linguagem Java. Você deve criar a exceção e utiliza-la.

O exercício acima é uma situação em que não existe classes de exceções para tratar os possíveis "erros" (não armazenar número par na posição 5).
O código abaixo é uma forma de resolver o exercício.

```
public class ClasseExcecao {
  void verificaPar(int pos, int x) throws MinhaExcecao {
    if ((pos == 5) && (x  % 2 == 0)) {
      throw new MinhaExcecao();
    }
  }
}
```

Primeiro construímos a classe chamada ClasseExcecao. Essa classe possui um método com os argumentos **int pos** e **int x.**
O argumento **int pos** irá verificar qual é a posição do vetor que não pode receber um número par.
O argumento **int x** irá verificar se o número que foi sorteado é um número par ou um número ímpar.
Neste caso, se a posição for 5 e o número sorteado for um número par (note o símbolo && indicando **e**) então será disparada uma exceção que será tratada na classe **MinhaExcecao.**
A classe **MinhaExcecao** está escrita abaixo:

```
class MinhaExcecao extends Exception {
  public String verificaPar(){
    return "Valor incorreto";
  }
}
```

A classe **MinhaExcecao** possui o método verificaPar(). Esse método é chamado quando acontece uma exceção no programa. Ela também estende da classe **Exception**. Aqui estamos tratando a exceção. O tratamento neste caso é enviar uma mensagem dizendo que existe um valor incorreto.
Depois de construir as duas classes anteriores vamos contruir o programa principal.

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
```

```
public class TrataMinhaExcecao{
  JFrame janela;
  JButton bto1;
  JButton bto2;
  JPanel painel;
  JLabel saida;

  TrataMinhaExcecao(){
    janela = new JFrame("Tratamento de exceções");
    janela.setLayout(new BorderLayout());
    painel = new JPanel();
    painel.setLayout(new FlowLayout());

    bto1 = new JButton("Carregar");
    bto2 = new JButton("Parar");
```

```
    saida = new JLabel();

    TrataEventoBotao evento = new TrataEventoBotao();
    bto1.addActionListener(evento);
    bto2.addActionListener(evento);

    painel.add(bto1);
    painel.add(bto2);

    janela.add(painel, BorderLayout.SOUTH);

    janela.setSize(400,200);
    janela.setLocation(200,300);
    janela.setVisible(true);

  }
  public static void main (String[] args){
        TrataMinhaExcecao tme = new TrataMinhaExcecao();
  }

  class TrataEventoBotao implements ActionListener{
        String resultado = "";
        int[] vetor = new int[20];
        final Random r = new Random();
        ClasseExcecao ce = new ClasseExcecao();

        public void actionPerformed(ActionEvent e){
              int numero;
              if(e.getSource() == bto1){
                    for(int i=0; i<20; i++){
                          try{
                                numero =
Math.abs(r.nextInt()%10);
```

```
                                        ce.verificaPar(i,numero );

                                        vetor[i] = numero;
                                        resultado = resultado +
Integer.toString(Math.abs(vetor[i]))+ " - ";
                                        saida.setText(resultado);
                                        janela.add(saida,
BorderLayout.CENTER);
                                }catch(MinhaExcecao e1){
i--;

JOptionPane.showMessageDialog(null,e1.verificaPar());
                                }
                            }
                        }
                        if(e.getSource() == bto2){
                                System.exit(0);
                        }
                }
  }
}
```

A classe **TrataMinhaExcecao** possui cinco elementos gráficos para a construção da interface gráfica.

O construtor da classe **TrataMinhaExcecao** constrói a janela define o leiatute da janela. Essa janela foi configurada com o gerenciador de leiaute **BorderLayout** (aquele que divide a janela em Norte, Sul, Leste, Oeste e Centro). No espaço Sul foi adicionado um painel com gerenciador de leiaute **FlowLayout** (aquele que os elementos inseridos ficam um após o outro). Os botões foram adicionados no leiaute do painel que está na posição Sul e um **label** foi adicionado na posição Centro.

A classe interna chamada **TrataEventoBotao** que implementa a interface **ActionListener** também foi escrita nesse código.

Dentro da classe interna **TrataEventoBotao** foi instanciada uma **String** chamada **resultado** que foi inicializada com espaço em branco.

Um vetor chamado **vetor** do tipo inteiro com 20 posições também foi instanciado dentro da classe **TrataEventoBotao**.

A variável de instância **r** do tipo **Random** também foi declarada para armazenar os valores sorteados pelo método **nextInt()** da classe **Random**.

Finalmente a **ClasseExcecao** que foi construída no código anterior também foi instanciada usando como variável de referência **ce**.

Como a classe **TrataEventoBotao** implementa a interface **ActionListener** então o método **actionPerformed (ActionEvent e)** foi implementado dentro da classe **TrataEventoBotao**.

O método **actionPerformed** declara uma variável inteiro chamada numero.

Ele também verifica se o **bto1** foi pressionado através da comparação

if (e.getSource() == bto1)

Se essa comparação for verdadeira então é iniciado um **looping** usando a estrutura de repetição **for**. Ela inicializa o inteiro **i** de zero e verifica se ele é menor que 20 (ele irá contar de 0 a 19 que são 20 posições).

Dentro do **looping** é usado a cláusula **try** onde colocamos o código que pode disparar uma exceção. A variável **numero** recebe um número sorteado (método **nextInt()**). Esse número vai até 10 (**r.nextInt() % 10**). Depois de sorteado o número usamos apenas seu valor absoluto (**abs**). Para fazer isso usamos o método **abs**(absoluto) da classe **Math** que está no pacote **java.util** que foi importado anteriormente.

Depois de sortear o número devemos saber se ele foi sorteado para ser colocado na quinta posição e se ele é um número par ou ímpar. Se ele for um número par então uma mensagem de erro será enviada para o usuário. Para fazer isso usamos o método **verificaPar(i, numero)**. Esse método foi construido na classe **ClasseExcecao** escrito anteriormente.

Depois o vetor na posição **i** recebe o número. Nesse caso quando o número sorteado for par ele volta o índice do vetor em uma posição e mosta a mensagem.

O usuário irá pressionar **Ok** e outro número será sorteado se ele for par então a mensagem será mostrada novamente. Quando número sorteado for ímpar então o vetor será preenchido.

Quando botão **bto2** for pressionado **(if(e.getSource() == bto2)** o sistema é fechado.

O exercício anterior mostra como usar exceções que não são próprias da linguagem Java.

1.3 EXEMPLO DE EXCEÇÕES NA MANIPULAÇÃO DE ARQUIVOS

A leitura e escrita de arquivo em Java são fáceis. Como Java é uma linguagem Orientada a Objetos então devemos usar as classes que se tornarão objetos para trabalharmos com entrada e saída (input/output) ou IO. A linguagem Java trata as Streams que são fluxos de dados e tem controle sobre ele.

Para utilizarmos as classes que manipulam arquivos em Java devemos importar o pacote java.io possui as classes que fazem a leitura e gravação de arquivos.

As instâncias da classe java.io.File os caminhos para os locais onde o arquivo será gravado ou lido no sistema operacional. Vejamos um exemplo onde é criada uma instância da classe File direcionada para /aula/arquivos/meuteste.txt.

```
File arquivo = new File("/aula/arquivos/teste.txt");
```

O código abaixo mostra instanciar um objeto que aponta para um arquivo que está no diretório atual.

```
File arquivo = new File("teste.txt");
```

O exemplo abaixo mostra instanciar um objeto que aponta para um arquivo que está no diretório anterior ao código do programa.

```
File arquivo = new File("../teste.txt");
```

Quando utilizamos o Sistema Operacional Windows devemos utilizar duas barras invertidas para navegar entre diretórios (pastas).

1.3.1 ESCREVENDO EM ARQUIVOS

Uma instância da classe java.io.FileWrite permite que possamos escrever em um arquivo. Vejamos:

```
FileWriter escrever = new FileWriter(new
File("teste.txt"),true);
FileWriter escrever = new FileWriter(new
File("teste.txt"));
```

O parâmetro **true** utilizado no exemplo acima indica que podemos adicionar dados a um arquivo já existente e que o arquivo não será sobrescrito.

Para escrevermos em um arquivo utilizamos instâncias da classe java.io.PrintWriter que possui os métodos print e println utilizados para escrever no arquivo. Vejamos a instância da classe PrintWriter:

```
PrintWriter saida = new PrintWriter(escrever);
PrintWriter saida = new PrintWriter(escrever,true);
```

O parâmetro **true** significa que os dados serão enviados para o arquivo toda vez que ocorrer uma chamada do método println(). Quando não colocamos o parâmetro **true** significa que os dados só são enviados quando enviarmos uma quebra de linha, fecharmos o arquivo ou atualizarmos o arquivo. O código abaixo mostra como devemos escrever em um arquivo:

```
escrever.println("Aula de Java");
escrever.println("desde 2008");
escrever.println("Curso de Java");
```

Depois de escrever no arquivo, devemos fecha-lo.

```
saida.close();
escrever.close();
```

Exemplo

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

 public class EscreveArquivo{
   public static void main(String[] args){
     try{
       FileWriter canal = new FileWriter(new
File("meuarquivo.txt"),true);
       PrintWriter escreve = new PrintWriter(canal);
       escreve.println("aqui tem um texto");
       escreve.println("continua o texto aqui");
       escreve.close();
       canal.close();
     }
     catch(Exception e){
       e.printStackTrace();
     }
   }
 }
```

O código acima abre um canal de comunicação com um arquivo chamado **meuarquivo.txt** utilizando a classe **FileWriter** através da variável de referência **canal**.

A classe **PrintWriter** instancia uma variável escreve que irá escrever no canal de comunicação aberto entre a aplicação e o arquivo em disco. O método **println** da classe **PrintWriter** é utilizado para fazer a escrita no arquivo. Depois de escrito no arquivo devemos fechar a escrita e o canal de comunicação utilizando o método **close().**

Por que os comandos de abertura do arquivo e escrita no arquivo estão entre as cláusulas **try** e **catch**?

1.3.2 LENDO UM ARQUIVO ESCRITO

Para fazermos uma leitura de um arquivo texto devemos utilizar a classe **FileReader** do pacote java.io (**java.io.FileReader**).

Os construtores da classe FileReader são:

- `FileReader reader = new FileReader(new File("texto.txt"));`
- `FileReader reader = new FileReader("texto.txt");`

Primeiro devemos instanciar um objeto da classe **BurreferedReader** (java.io.BufferedReader) que irá fornecer o método **readLine()** que faz a leitura das linhas do arquivo. Construtores da classe BufferedReader estão abaixo:

- `BufferedReader leitor = new BufferedReader(reader);`
- `BufferedReader leitor = new BufferedReader(reader, 1*1024*1024);`

É possível especificar o tamanho do buffer desejado utilizano o construtor da classe **BufferedReader**.

O **buffer** é utilizado para minimizar o número de pedidos de io para ler blocos de dados maiores do arquivo de uma vez só. A vantagem é que se aumentarmos o número, mais dados serão lidos de uma única vez diminuindo o número de acesso ao disco. A desvantagem é que se o número for muito alto o consumo de memória será muito alto.

Exemplo

```
import java.io.*;
```

```
 public class LeArquivo{
   public static void main (String[]args){
     try{
        BufferedReader le = new BufferedReader(new
FileReader("meuarquivo.txt"));
       while(le.ready()){
         String linha = le.readLine();
         System.out.println(linha);
       }
       le.close();
     }catch(IOException e){
       e.printStackTrace();
     }
   }
 }
```

Exceções são ações inesperadas que podem acontecer dentro de um programa.

O tratamento de exceções é uma técnica importante na programação. Ela torna o código mais simples e enxuto.

O tratamento de exceções em Java possui uma cláusula chamada **try** em que colocamos códigos que podem disparar uma exceção.

A cláusula **catch** faz o tratamento dessa exceção.

Conclusão

A manipulação de arquivos em Java necessita do tratamento de exceções.

Para abrir um arquivo usamos o método construtor `File arquivo = new File("nome_do_arquivo");`

Para deixarmos o arquivo pronto para escrita usamos a classe FileWriter.

Para escrevermos no arquivo usamos a classe PrintWriter

Para lermos o arquivo usamos as classes BufferedReader e FileReader.

- Faça um programa para calcular a divisão de dois números quaisquer.
- Excute o programa e faça a divisão de 79 dividido por 0.
- Capture a exceção lançada utilizando o método printStackTrace() da classe Exception.
- Faça com que o foco seja novamente apontado para o denominador.

- Escreva um programa que possua uma interface gráfica com um menu.
- O menu deverá ter as opções Arquivo e os itens Novo, Salvar e sair.
- Uma área de texto deverá compor a janela de seu programa.
- Quando você pressionar o item de menu Novo, a janela de texto deverá ser habilitada tornando possível a escrita.
- Quando você pressionar o item de menu Salvar, deverá aparecer uma janela solicitando o nome do arquivo.
- Depois de preenchido o nome do arquivo este deverá ser gravado no disco.

APONTAMENTOS SOBRE A PRÓXIMA UNIDADE

Na próxima unidade você irá utilizar o tratamento de exceções para manipulação de banco de dados.

2 UNIDADE

METAS

- Depois de ler a unidade o aluno deverá escrever códigos que manipulem banco de dados utilizando a linguagem Java.
- Estudar o padrão de projeto MVC.
- Estudar a ferramenta hibernate.

OBJETIVOS

- Utilizar drivers para banco de dados
- Incluir, consultar, excluir e alterar dados em um banco de dados.
- Reconhecer programas que utilizam o padrão de projeto MVC
- Escrever programas que utilizam a ferramenta hibernate para acessar banco de dados.

PRÉ-REQUISITOS

Para a realização dessa atividade o aluno deverá:

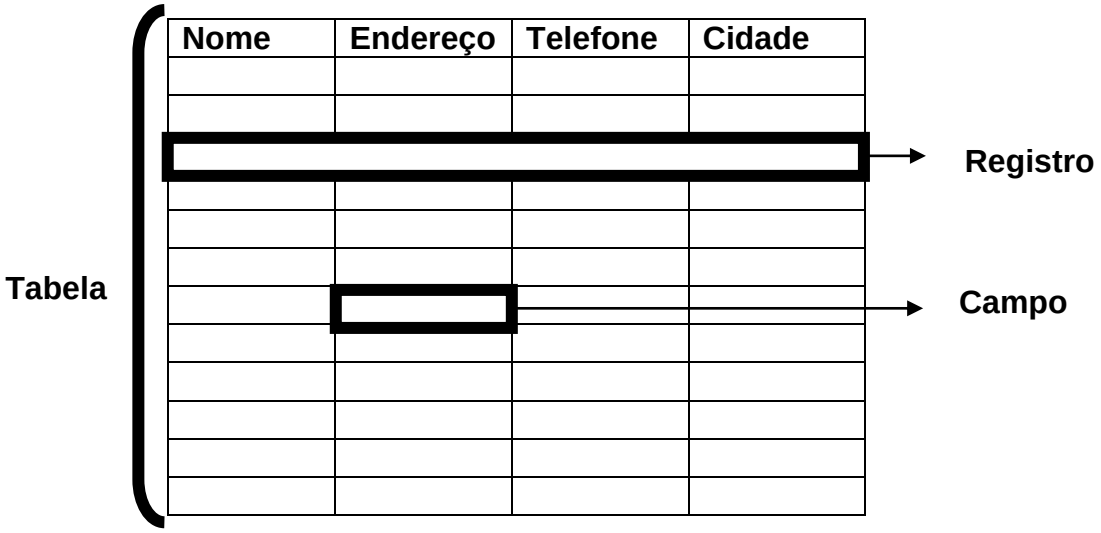
- Saber contruir banco de dados.
- Saber contruir tabelas em um banco de dados.
- Conhecer as estruturas de controle da e de repetição da linguagem Java.

2.1 BANCO DE DADOS EM JAVA

Sistemas geralmente necessitam de armazenamento de dados. Os dados podem ser armazenados em arquivos ou bancos de dados. Os bancos de dados são programas que armazenam dados. Os Sistemas Gerenciadores de Banco de Dados fazem o gerenciamento desses dados.

Os dados em um banco de dados são armazenados em tabelas e essas tabelas são compostas por registros e os registros são compostos por campos.

Vejamos a figura abaixo:



O MySQL é um Sistema Gerenciador de Banco de Dados também conhecido pelo acrônimo SGBD.

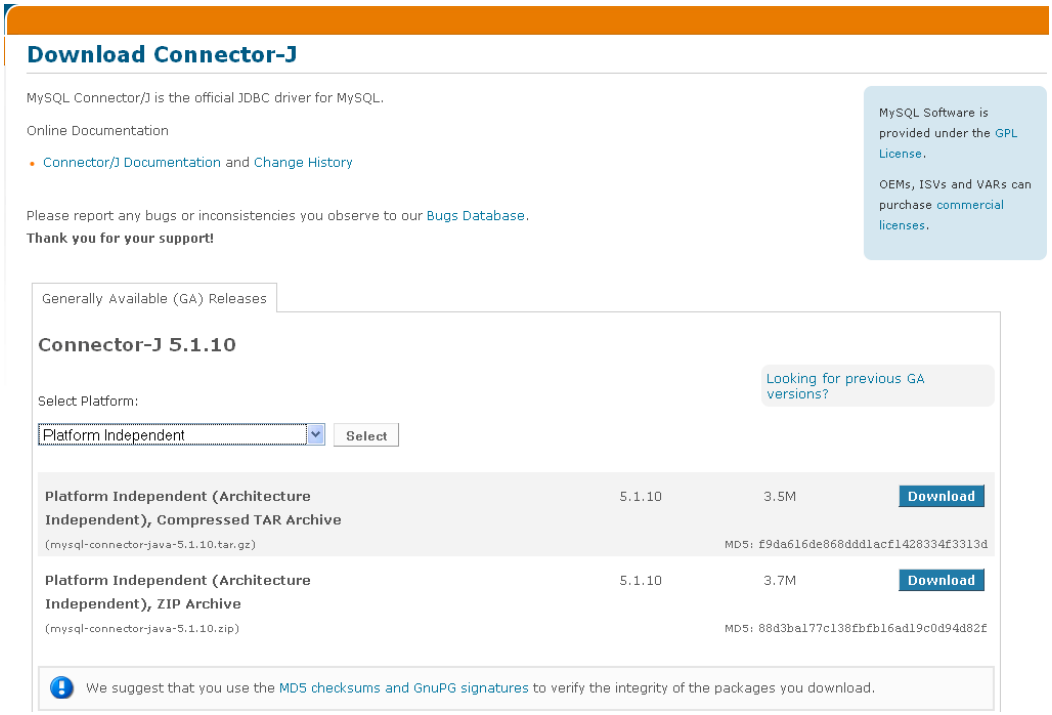
O MySQL pode ser baixado em
http://dev.mysql.com/downloads/

Ao desenvolver um sistema precisamos ter uma funcionalidade essencial que é a habilidade para comunicar-se com um repositório de dados.

Os Bancos de Dados são repositórios de dados e a linguagem Java dispõe de uma API para acessar repositórios de dados chamada Java DataBase Connectivity API ou JDBC API.

Durante o curso iremos utilizar a IDE Eclipse para construir programas que fazem comunicação com um banco de dados MySQL.

Vamos baixar o driver de comunicação entre a linguagem Java e o banco de dados MySQL. O driver está disponível em: http://dev.mysql.com/downloads/connector/j/5.1.html



Você tem das opções para baixar o driver Connector-J 5.1.10 que são *Compressed TAR archive* ou *ZIP archive*.

Depois de selecionar qual o tipo de arquivo você vai baixar, selecione de qual servidor será feito o *download*.

North America

University of Waterloo Computer Science Club, Canada	HTTP	FTP
Rafal Rzeczkowski/ Hamilton, ON, Canada	HTTP	FTP
University of Wisconsin / Madison, WI, United States of America	HTTP	FTP
Argonne National Laboratory / Chicago, IL, United States of America		FTP
Hurricane Electric / San Jose, CA, United States of America	HTTP	
Semaphore Corporation, Seattle, WA, United States of America	HTTP	FTP

South America

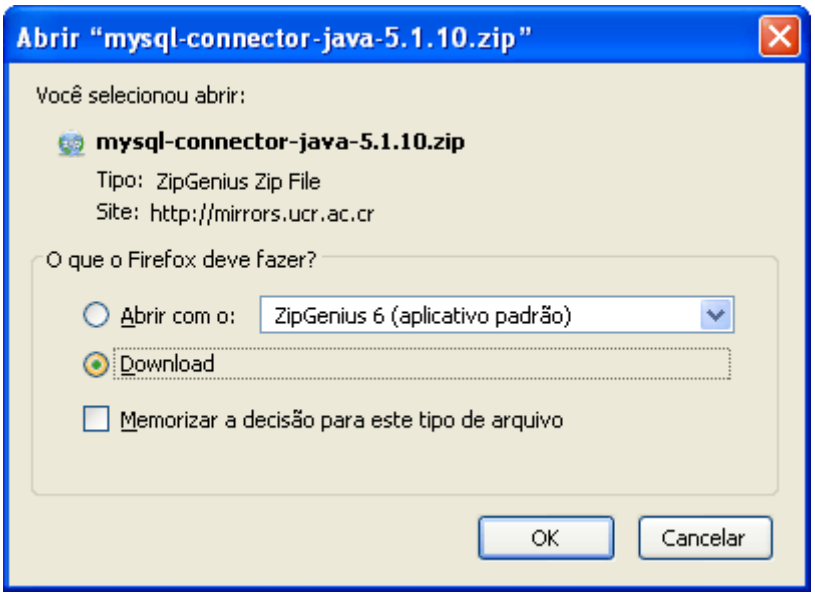
Universidad de Costa Rica, Costa Rica	HTTP	FTP

Asia

Armenian Datacom Company, Armenia	HTTP	
sPD Hosting, Israel	HTTP	
Mirimar Networks, Israel	HTTP	
JAIST, Japan	HTTP	FTP
Internet Initiative Japan Inc., Japan	HTTP	FTP
Kyung Hee University Linux User Group, Republic of Korea	HTTP	FTP
ezNetworking Solutions Pte. Ltd., Singapore	HTTP	FTP
National Taiwan University, Taiwan	HTTP	FTP
mirror.tw (Taiwan Mirror), Taiwan	HTTP	FTP
Providence University, Taiwan	HTTP	FTP
Computer Center, Shu-Te University / Kaohsiung, Taiwan	HTTP	FTP
National Sun Yat-Sen University, Taiwan	HTTP	FTP
FPT Telecom, Viet Nam	HTTP	FTP

Você tem duas opções para baixar o arquivo: http ou ftp. Escolha uma das opções e baixe o arquivo.

A janela abaixo será mostrada para que você possa baixar o arquivo.



Depois de pressioar o botão *Ok* o *download* será iniciado.

Depois de baixar o arquivo, faça a instalação.

Depois de instalado o driver de comunicação entre a linguagem Java e o Banco de Dados MySQL devemos então baixar o MySQL.

O MySQL é um sistema gerenciador de banco de dados muito utilizado.

Veja mais sobre o MySQL em http://www.mysql.com/

Vamos utilizar o pacote XAMPP para manipular nossos bancos de dados com o Java. O XAMPP é um pacote de distribuição que possui o Apache que é um servidor, o MySQL, PHPMyAdmin entre outros softwares interessantes para programação.

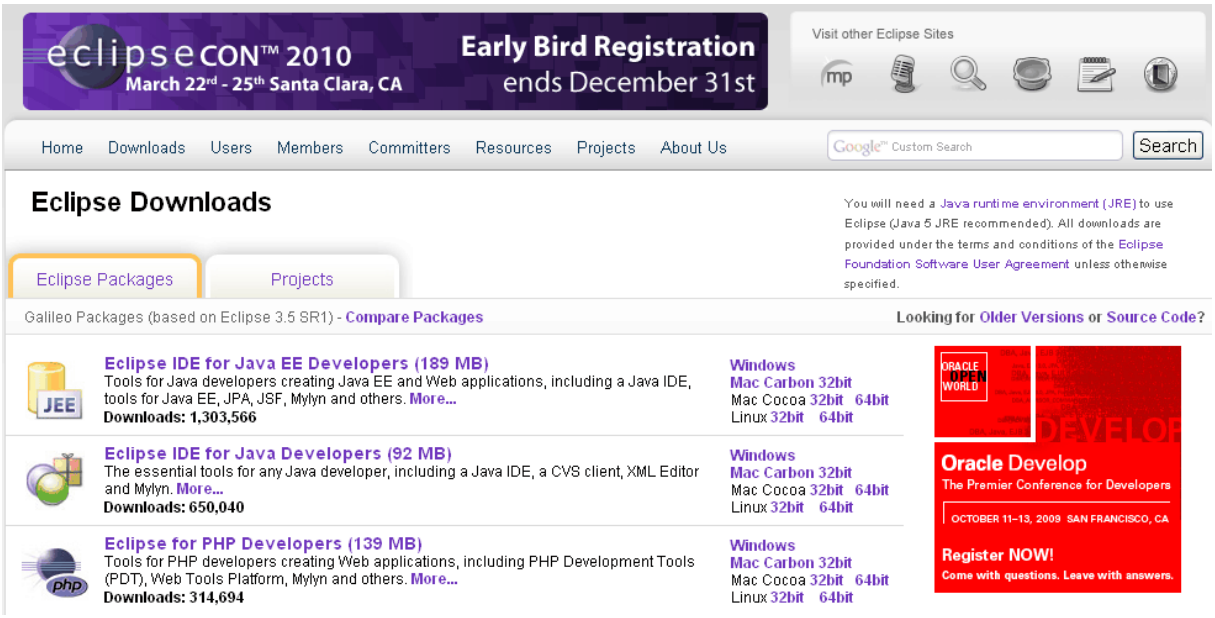
Vamos baixar o XAMPP em:

http://www.apachefriends.org/pt_br/xampp-windows.html

depois faça a instalação.

Vamos baixar o Eclipse em:

http://www.eclipse.org/downloads/

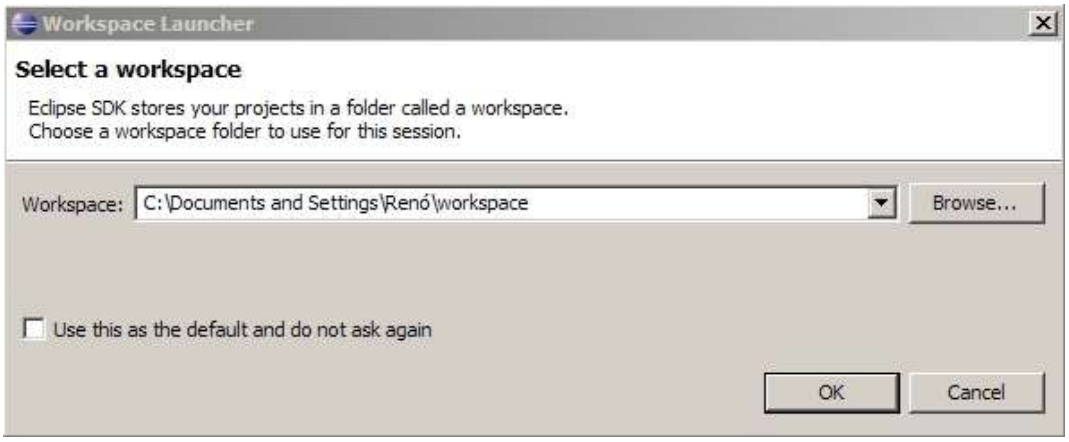


O Eclipse é uma IDE (ambiente de desenvolvimento integrado) que utilizamos para facilitar a programação. Baixe o arquivo *Eclipse IDE for Java Developers (92 MB)*.

O *Eclipse for PHP Developers* também é muito interessante para quem programa em PHP.

Depois de todos os softwares necessários para a manipulação de banco de dados em Java foram baixados e instalados, devemos configurar o ambiente.

1. Descompacte o arquivo em C:\Eclipse (por exemplo);
2. Inicie o eclipse iniciando o arquivo C:\eclipse\eclipse.exe e a janela abaixo será mostrada



3. Indique onde seus códigos fonte serão gravados;

Logo após a janela do Eclipse será aberta. A tela inicial mostra a janela de boas vindas "Welcome". Podemos fechar essa janela para iniciar os trabalhos.

Depois de instalado (ou melhor o Eclipse copiado no seu computador) vamos iniciar os trabalhos.

Vamos criar nosso primeiro projeto no Eclipse

a. Clique em File > New > Java Project
b. Digite o nome do projeto. Exemplo ProjetoBancoDados.
c. Clique em Finish.

7. Criando uma classe no Eclipse

a. Clique em File > New > Class
b. Digite o nome da classe. Exemplo: Pessoa
c. Clique em Finish.

8. Acessando o MySQL no Eclipse

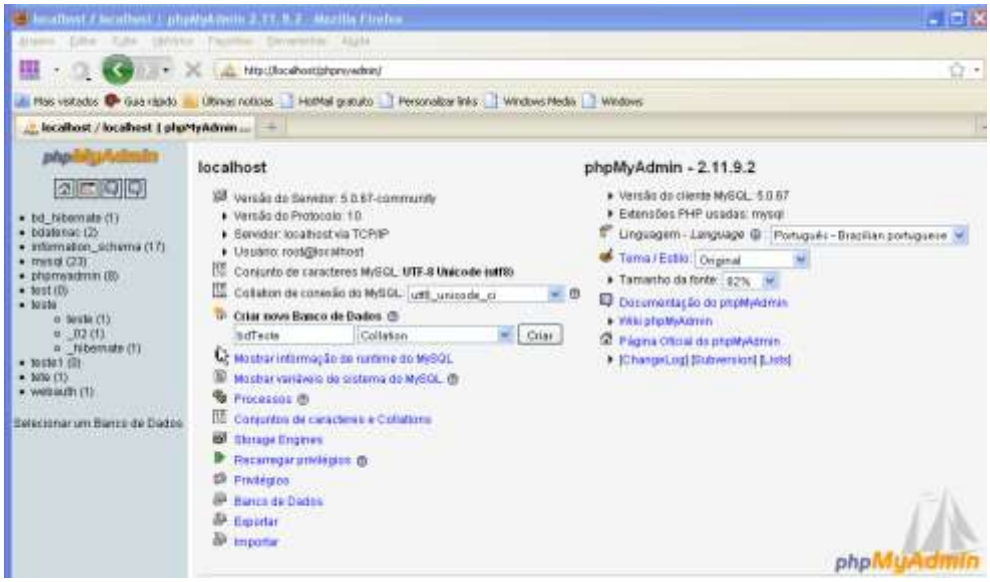
a. Crie uma pasta em C: com o nome ConectorMySQL.
b. Descompacte o arquivo em C:\ConectorMySQL
c. Clique com o botão direito do mouse em cima do nome do projeto criado;

d. Selecione Build Path > Add External Archives...

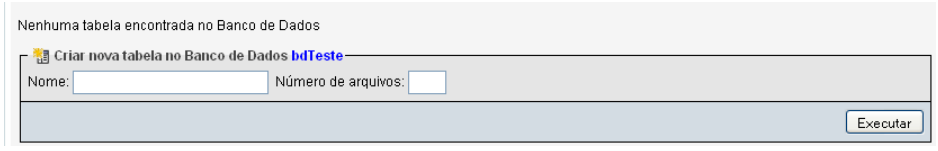
e. Selecione o arquivo mysql-connector-java-5.1.6-bin.jar;

Vamos escrever um código para verificar se a instalação foi feita com sucesso.

1. Inicie o XAMPP.
2. Inicie o Apache (dentro do XAMPP).
3. Inicie o Mysql (dentro do XAMPP).
4. Inicie seu navegador.
5. Digite na URL do navegador (http://localhost/phpmyadmin)
6. A janela abaixo será mostrada:



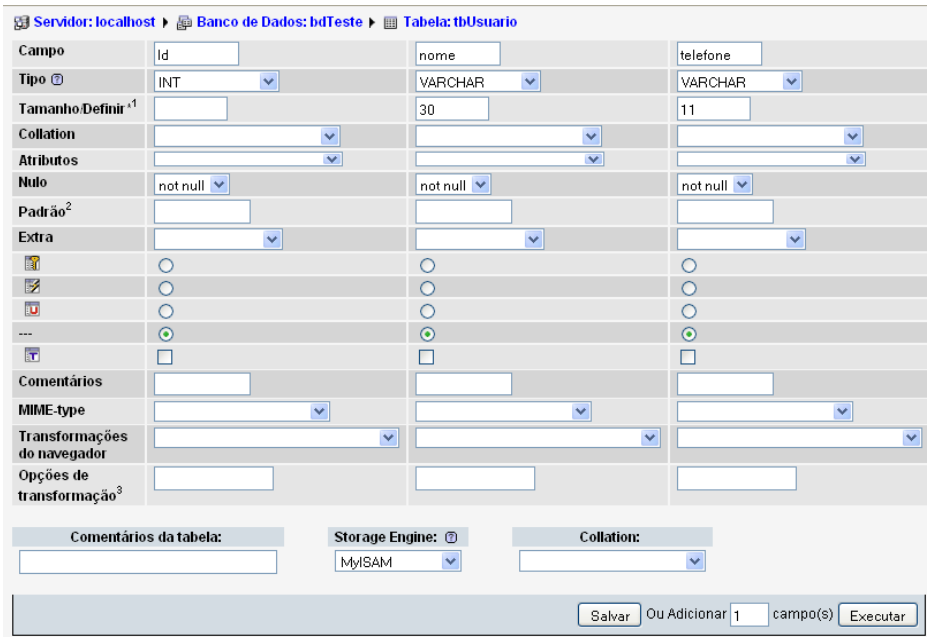
7. Escreva um nomo para **Criar novo banco de dados** e pressione o botão **Criar**.
8. A próxima janela irá solicitar que você atribua um nome para a tabela.



9. Digite **tbUsuario** preencha o campo **Número de arquivos** (aqui você deve colocar quantos campos sua tabela possuirá) com **3** e pressione

executar para que a tabela seja criada.

10. Preencha os campos com o nome dos campos da sua tabela com id (int), nome (varchar tamanho 30) e telefone (varchar tamanho 13). Clique em **Salvar**



11. Agora você já tem sua tabela criada no banco de dados.

Depois de criada a tabela no banco de dados vamos escrever o código em Java para fazermos a conexão com o banco de dados criado.

Digitar o código de um programa é muito importante para seu aprendizado.

O código abaixo mostra como fazer uma inclusão no banco de dados MySQL utilzando a linguagem Java.

Digite o código abaixo para fazer a inclusão de um dado em um banco de dados utilizando a Linguagem Java.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class InserePessoa{

  public static void main(String[] args){
    //Variáveis de método
    Connection conexao = null;
    Statement stm = null;
    String msg = "";
    String sql = "";

    try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         conexao=
DriverManager.getConnection("jdbc:mysql://localhost:3306/bdTest
e","root", "");
         stm = conexao.createStatement();
         msg = "Conexao realizada com sucesso";
         System.out.println(msg);

         //Insere pessoas
         sql = "insert into tbUsuario (id,nome,telefone)values
(1,'Gustavo','3188769890')";

         //executa a sql de inserção
         stm.executeUpdate(sql);

         System.out.println("Inserção concluída com sucesso!");

    }catch (Exception e){
      msg = "Erro de conexão " + e.getMessage();
```

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

O código acima faz a inserção de **id**, nome e telefone em uma tabela de um banco de dados.

O código faz a importação das classes que serão usadas no desenvolvimento do programa que são

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

A classe **Connection** possui os métodos necessários para a criação da conexão da aplicação Java com o banco de dados.

A classe **DriveManager** possui os métodos necessários para registrar o **driver** de comunicação da aplicação com o banco de dados.

A clsse **Statement** possui métodos para execução de comandos.

No método principal (**main**) forma inicializadas as variáveis:

conexao do tipo **Connection** com **null**

stm do tipo **Statement** com **null**

msg e **sql** do tipo **String** com uma **string** sem texto (espaço)

Um tratamento de exceções é feito no programa usando a cláusula **try**. Dentro dessa cláusula carregamos o **driver** para que a linguagem Java saiba que ele existe.

Quando utilizamos o método **Class.forName()**, o carregador de classes inicializa a classe. A classe possui um bloco inicializador que irá registrar essa classe como um **driver JDBC**, que comunica o **java.sql.DriverManager**.

O **java.sql.DriverManager** verifica para cada **Driver** registrado, se é possível fazer a conexão . Se for possível a conexão é aberta (pelo **Driver**), e retornada. Se não for possível fazer a conexão uma exceção é lançada.

A **String sql** recebe uma instrução escrita na linguagem **sql** que insere um **id, nome** e **telefone** na tabela chamada **tbUsuario**.

o método **executeUpdate()** da classe **Statement** executa essa linha de comando em sql que foi armazenada na **string**.

Caso aconteça algo inesperado no código (exceção) então uma mensagem de erro é enviada para o usuário (cláusula **catch**).

O código abaixo mostra os dados inseridos no programa anterior.

Digite o código abaixo para fazer uma consulta no banco de dados.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class ConsultaPessoa{

  public static void main(String[] args){
    Connection conexao = null;
    Statement stm = null;
    String msg = "";
    String sql = "";
    ResultSet rs = null;
    try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
```

```
      conexao=
DriverManager.getConnection("jdbc:mysql://localhost:3306/bdTest
e","root", "");
      stm = conexao.createStatement();
      msg = "Conexao realizada com sucesso";
      System.out.println(msg);
      sql = "select * from tbUsuario";
      rs = stm.executeQuery(sql);
      while (rs.next()){
       int id = rs.getInt(1);
        String nome = rs.getString(2);
        String telefone = rs.getString(3);
        System.out.println(id);
        System.out.println(nome);
        System.out.println(telefone);
      }
    }catch (Exception e){
      msg = "Erro de conexão " + e.getMessage();
      System.out.println(msg);
    }
  }
}
```

O código acima é inicializado como o código anterior (inserção). A **string** que é executada ao invés de ser uma **string** de inserção de dados é uma **string** de consulta de dados. Veja que o comando **SQL** inserido na **string** é para fazer uma consulta.

O método **executeQuery(sql)** executa o comando de consulta de dados e armazena o resultado em uma variável (**data set**) chamada **rs** que é do tipo **ResultSet**. Essa variável que é do tipo **ResultSet** possui um método chamado **next()** que vai para o próximo registro da tabela. Então ela irá extrair todos os registros da tabela. Ela possui também um método chamado **getInt(posição)** que "pega" o campo da tabela. Se posição for igual a 1 ela irá "pegar" o primeiro campo do registro, se o argumento posição for igual a

2 ela irá "pegar" o segundo campo do registro e assim sucessivamente. Esses campos que foram resgatados da tabela são atribuídos as variáveis do tipo **string id**, nome e telefone respectivamente e depois mostradas para o usuário utilizando o comando **System.out.println().**

Caso aconteça algo inesperado (exceção) a cláusula **catch** caputra a exceção e dispara uma mensagem de erro.

No exemplo acima foi utilizada a classe **Exception** e que é a classe que possui todas as exceções (genérica) e foi tratada somenteo erro de conexão.

Depois de digitar o programa acima pesquise e escreva o que faz a linha:
Class.forName("com.mysql.jdbc.Driver").newInstance();
jdbc:mysql:
localhost:3306
root

Vamos agora alterar o conteúdo do banco de dados.

Depois que os dados foram gravados no banco de dados, faça a alteração dos dados. Para ver como fazer a alteração dos dados digite o código abaixo:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class AlteraPessoa{

  public static void main(String[] args){
    Connection conexao = null;
    Statement stm = null;
    String msg = "";
    String sql = "";

    try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
```

```
            conexao=
DriverManager.getConnection("jdbc:mysql://localhost:3306/bdTest
e","root", "");
            stm = conexao.createStatement();
            msg = "Conexao realizada com sucesso";
            System.out.println(msg);

            //faz a alteração
            sql = "UPDATE tbUsuario SET telefone = '0213144556677'
WHERE telefone = '3188769890'";
            stm.executeUpdate(sql);
            System.out.println("Atualização executada
corretamente");

        }catch (Exception e){
          msg = "Erro de conexão " + e.getMessage();
          System.out.println(msg);
        }
      }
}
```

Para finalizar devemos fazer uma exclusão dos dados que estão gravados no banco de dados. Escreva o código abaixo para excluir um dado do banco de dados.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class ExcluiPessoa{

  public static void main(String[] args){
    Connection conexao = null;
    Statement stm = null;
```

```
    String msg = "";
    String sql = "";

    try {
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         conexao=
DriverManager.getConnection("jdbc:mysql://localhost:3306/bdTest
e","root", "");
         stm = conexao.createStatement();
         msg = "Conexao realizada com sucesso";
         System.out.println(msg);

         //faz a exclusão
         sql = "DELETE FROM tbUsuario WHERE nome='Gustavo'";
         stm.executeUpdate(sql);
         System.out.println("Exclusão executada corretamente");

    }catch (Exception e){
      msg = "Erro de conexão " + e.getMessage();
      System.out.println(msg);
    }
  }
}
```

Depois de ter digitado os códigos acima podemos observar que existe uma pequena diferença entre a consulta de dados e a exclusão de dados. Que diferença é essa?

2.1.1 ACESSO A BANCO DE DADOS VIA JDBC

Vamos analisar o código:

Usamos os comandos try e catch para capturar as exceções caso elas aconteçam.

```
 try  {
   //Registra o driver
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").getInstance();
   //Estabelece uma conexão
   Connection con =
DriverManager.getConnection("jdbc:odbc:bdDisco",
"usuario", "senha");
 }catch (SQLException e) {
  //Caso haja alguma operação inesperada uma exceção é gerada
para
informar.
  e.printStackTrace();
 }
```

Depois que estabelecemos a conexão podemos executar comandos SQL para manipular os dados no banco de dados. Vamos ver como devemos realizar uma consulta sobre o id, nome e telefone de cada usuário no banco de dados. Para isso podemos usar 3 interfaces para executar comandos SQL no banco de dados.

a. Podemos utilizar a interface Statement, que permite a execução dos comandos fundamentais de SQL (SELECT , INSERT , UPDATE ou DELETE).

b. A interface PreparedStatement nos permite usufruir de SQL armazenado ou pré-compilado no banco, quando o banco de dados suportar este recurso.

c. E a CallableStatement, e permite executar procedimentos e funções armazenados no banco quando o banco suportar este recurso.

Vamos utilizar a interface Statement.

Depois de estabelecida a conexão com o banco de dados utilizamos o método **createStatement** de **con** para criar o **Statement**

```
 Statement stm = con.createStatement ();
 // Criando o comando SQL
 String SQL = " Select id, nome, telefone from tbUsuario";
```

A interface ResultSet irá captar os resultados da execução dos comandos SQL no banco de dados. Esta interface apresenta uma série de métodos para prover o acesso aos dados:
Depois de criado o Statement podemos executar a query no banco de
Dados.

```
ResultSet rs = stm.executeQuery(SQL);
```

O método next() irá informar se existe resultados e vai para a próxima linha disponível para recuperação de dados

```
while (rs.next()) {
```

Os métodos get "pegam" os dados de acordo com o tipo SQL do dado. As variáveis id, nome e telefone contêm os valores retornados pela **query**.

```
   int id = rs.getInt(id);
   String nome = rs.getString("nome");
   String telefone = rs.getString("telefone");
   System.out.println("ID: ", + id + " Nome: " + nome +
"Telefone: " + telefone);
 }
```

Devemos agora liberar os recursos alocados pelo banco de dados para a execução do código. Podemos fazer isso fechando o Statement, que libera os recursos associados à execução dessa consulta, mas deixa a conexão aberta para a execução de uma próxima consulta, ou fechando diretamente a conexão, que encerra a comunicação com o banco de dados. Para termos certeza de que

vamos encerrar esta conexão mesmo que uma exceção ocorra, deixaremos o fechamento para a cláusula **finally ()** como mostrado no código abaixo.

```
finally {
   try {
     con.close();
   } catch (SQLException onConClose) {
     System.out.println("Houve erro no fechamento da conexão
");
     onConClose.printStackTrace();
   }
 }
```

Escreva um programa para criar clássico controle de DVDs. O programa deverá utilizar interface gráfica e fazer as operações de inclusão. As informações gravadas deverão ser Identificador do filme, Nome do filme, ator principal.

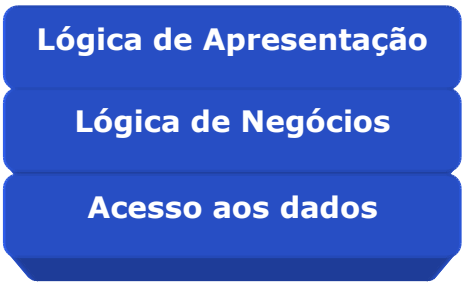
Faça o programa, insira alguns filmes e veja o resultado no banco de dados.

2.2 INTRODUÇÃO AO MODEL VIEW CONTROLLER

O Model View Contreller ou MVC é um *design pattern* ou um padrão de projeto utilizado no desenvolvimento de aplicações. Os padrões de projeto são úteis para resolver problemas de modelagem de projetos. A idéia principal de quem usa um padrão de projeto para programar é ter uma aplicação segura, de fácil manutenção, reutilizável, eficiente e tudo isso em prazos cada vez menores.

Quando uma aplicação é organizada em camadas é fundamental para a independência entre os componentes e essa independência atingirá certamente a escalabilidade, reutilização, eficiência e facilidade de manutenção.

As aplicações monolíticas utilizadas antigamente eram aplicações que seriam executadas somente em uma máquina. Este aplicativo na maioria das vezes tinha todas as funcionalidades em um único módulo e esse módulo possuía muitas linhas de código e a manutenção não era nada simples. A entrada de dados, lógica de negócio, verificação e acesso ao banco de dados estavam todos dentro de um mesmo lugar. Vejamos o esquema de uma aplicação monolítica:



As aplicações em duas camadas surgiram posteriormente porque enxergou-se a necessidade de compartilhar a lógica de acesso a dados entra vários usuários de uma só vez (simultaneamente). Nessa estrutura a base de dados foi colocada em uma máquina dedicada e separada das maquinas que executavam as aplicações. Nessa configuração os aplicativos ficam instalados em estações de trabalho do cliente com toda a lógica da aplicação. O gerenciamento de versões é um grande problema para essa configuração. Cada alteração feita nos aplicativos precisam ser instalados novamente nas estações dos clientes.

A figura abaixo mostra a arquitetura em duas camadas:

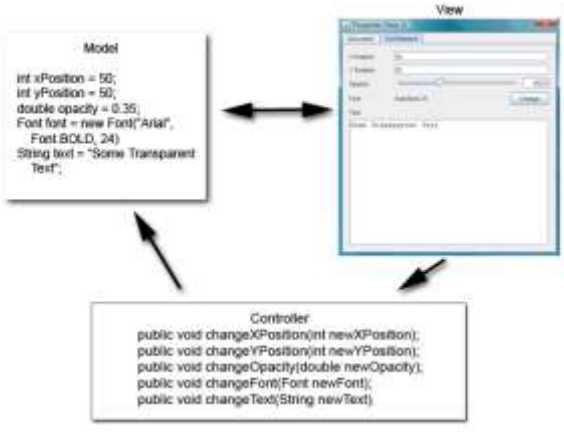
A internet mais uma vez mudou a forma de se programar. Nesse caso a estrutura que eram feitas em duas camadas tiveram que se adaptar para trabalhar em três camadas. A lógica de negócio foi separada da interface com o usuário. Partindo do princípio que os usuários da internet podem acessar a mesma aplicação sem ter que instalar algumas aplicações em suas máquinas.

Para implementar uma aplicação utilizando esse padrão de projeto são utilizados *frameworks.*

A camada Model tem por responsabilidade tratar a lógica de negócio e seu estado. Esse modelo possui todos os objetos de negócio onde as regras são implementadas e onde também é suportada todos os requisitos e funcionalidades do sistema sem que haja acoplamento das partes do sistema relacionadas ao fluxo da aplicação que corresponde ao controle.

A camada View possui os JSP´s e os Servlets e representam a visão da aplicação. Têm por objetivo a geração de interface com o usuário ou com outros sistemas e sua principal responsabilidade é manter a interface gráfica da aplicação. Nessa camada não são desenvolvidas as regras de negócio. Sendo assim todo o processamento é feito pelo Model e então repassado para a View.

A camada Controller responde a eventos que são geralmente eventos de usuário.



Fonte: http://java.sun.com/developer/technicalArticles/javase/mvc/#1

A forma de organização dos componentes dentro do padrão MVC dá credibilidade e facilita o desenvolvimento das aplicações, pois promove a implementação de problemas em cada componente de forma isolada. Sendo assim se algum componente não mais funcionar é possível trocar o componente do mesmo sem maiores prejuízos para o sistema.
Por poderem ser separados, a implementação dos componentes podem ser feitas separadamente utilizando plataformas e sistemas diferentes mantendo uma definição de interface em meio os componentes e preservando essas interfaces. O MVC divide os objetos com o intuito de aumentar sua reutilização e flexibilidade.

A camada de visualização tem a função apenas de mostrar a informação. Ela não se interesa em saber quem produziu a informação. Ela possui os elemetos de exibição no cliente tais como HTML, ASP e XML por exemplo. Essa camada é a interface dcom o usuário e também é usada para receber a entrada de dados e apresentar o resultado

A camada de regras de negócio é importante porque ela é a resopnsável por tudo o que a aplicação fizer. A camada de regras de negócio modela os dados e o comportamento por trás do porcesso de negócios além de ser reponsável pelo armazenamento, manipulação e geração de dados.

A camada de controle é a que determina o fluxo da aplicação sendo uma camada intermediária entre a camada de apresentação e lógica.

2.3 INTRODUÇÃO AO HIBERNATE

Quando desenvolvemos softwares corporativos (Orientados a Objetos) e temos que fazer um acesso ao banco de dados, geralmente, esse banco de dados é um banco de dados relacional. Isso gera desconforto para o programador e aumenta o tempo de execução do projeto.

O Hibernate, uma ferramenta que faz o mapeamento objeto/relacional (i.e. programação orientada a objetos com banco de dados relacional) na linguagem Java.

A técnica de mapeamento relacional faz o mapeamento de uma representação de dados de um modelo de objeto para dados de modelo relacional baseado em SQL e no JDBC.

O Hibernate pode diminuir e muito o trabalho do desenvolvedor nas tarefas mais simples de programação e armazenamento de dados. Com o Hibernate as linhas SQL da aplicação são todas retiradas ficando em arquivo separado.

Preparando o ambiente

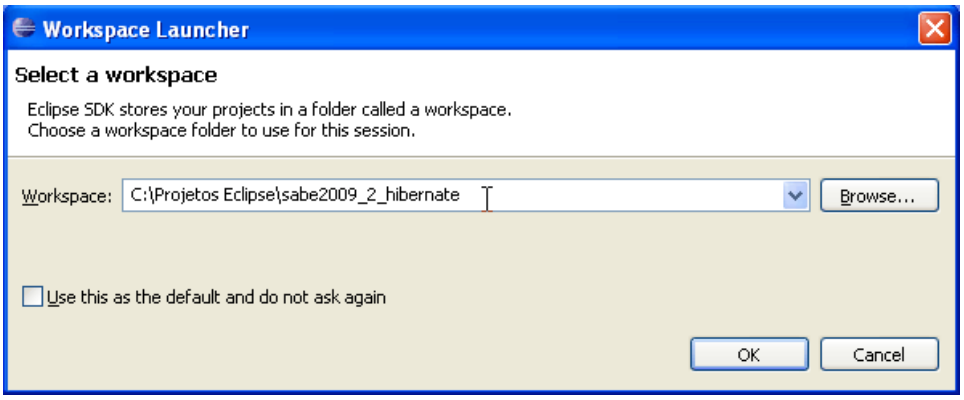
1. Baixar os arquivos
    a. hibernate-3.2.0.ga.zip
    b. hibernate-annotations-3.2.0.GA.zip
    c. mysql-connector-java-5.1.8.zip
2. Descompactar os arquivos em uma pasta qualquer.
3. Iniciar o XAMPP (Apache e MySQL)
4. Criar um banco de dados no MySQL chamado banco_hibernate.

Vamos iniciar nosso exemplo criando um controle de pessoas. Para isso devemos criar uma classe chamada Pessoa.

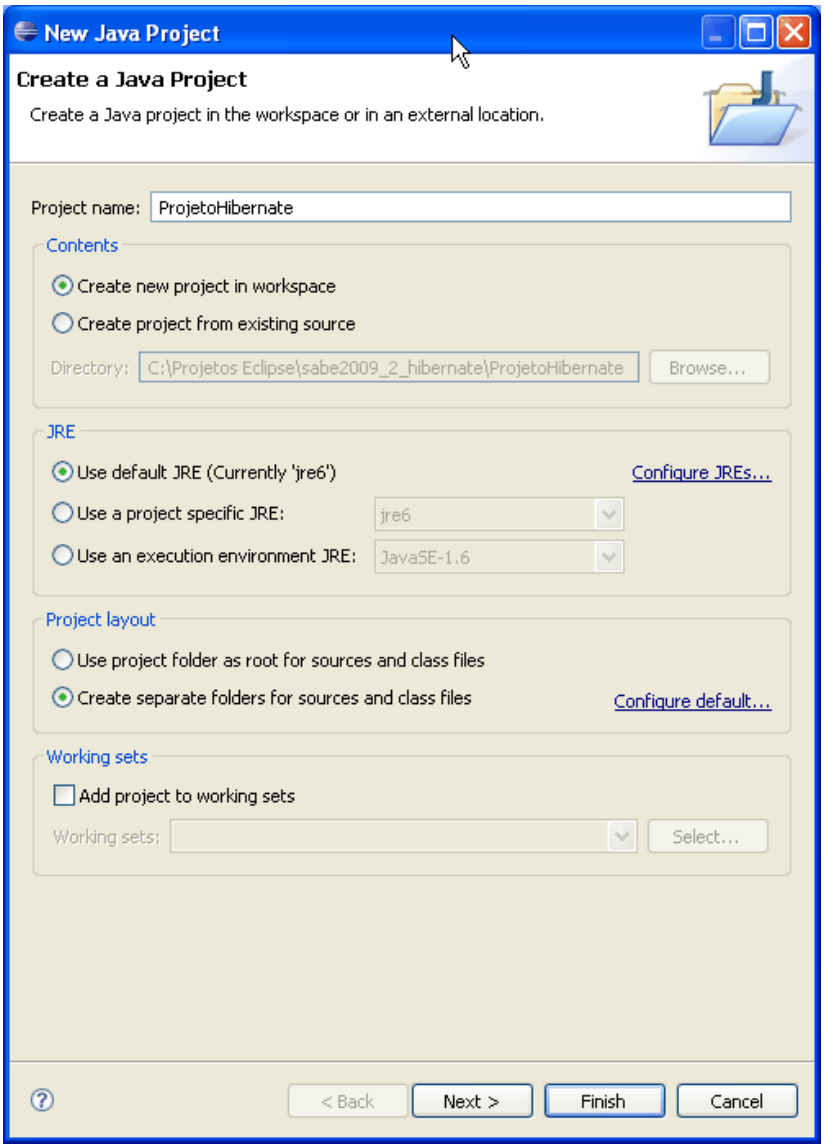
Vejamos como criar uma classe Pessoa no Eclipse.

Essa atividade é um pouco longa mas é feita uma única vez somente dentro do projeto. Vamos configurar o Eclipse para o desenvolvimento do projeto e criar nossa classe com as anotações.

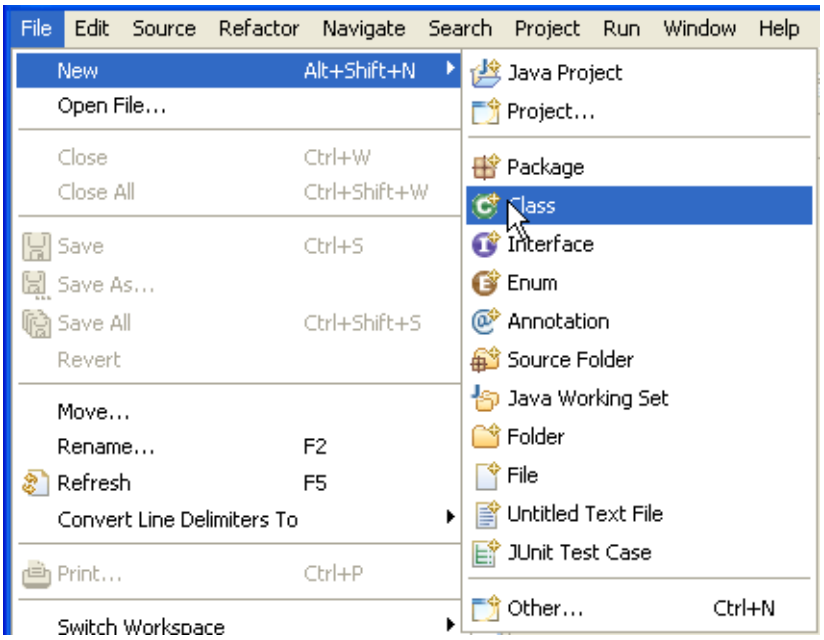
1. Iniciar o Eclipse
2. Criar a nova Workspace

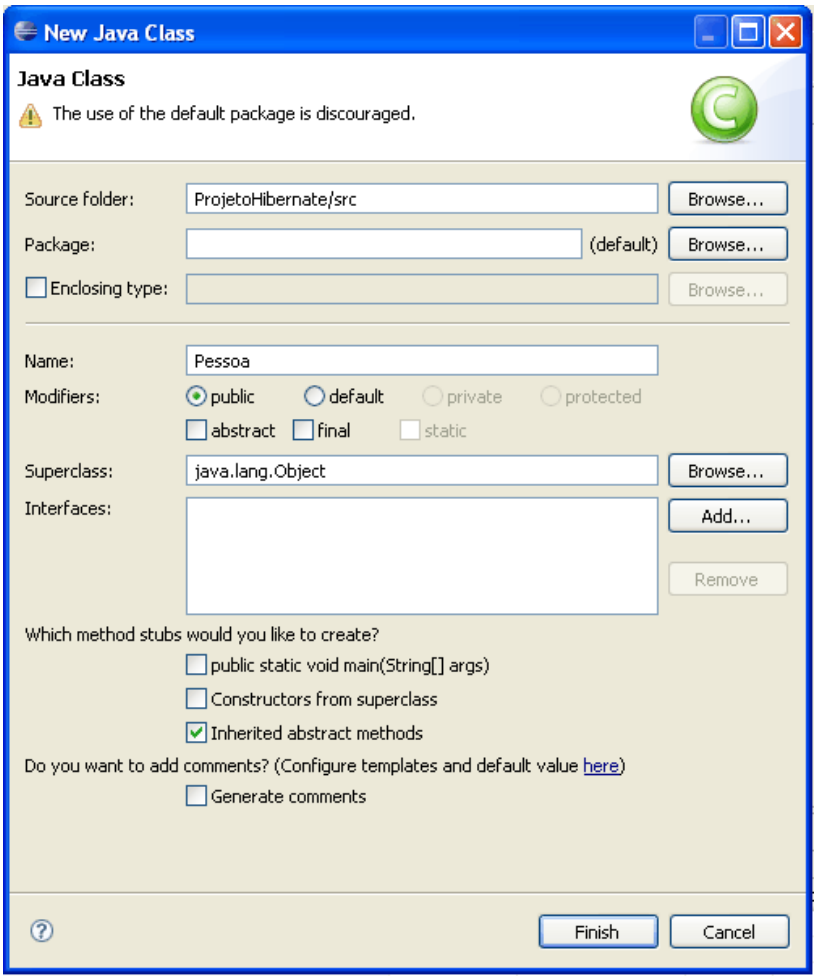


3. Iniciar um novo projeto (Java Project) chamado ProjetoHibernate



4. Criar uma nova classe chamada Pessoa





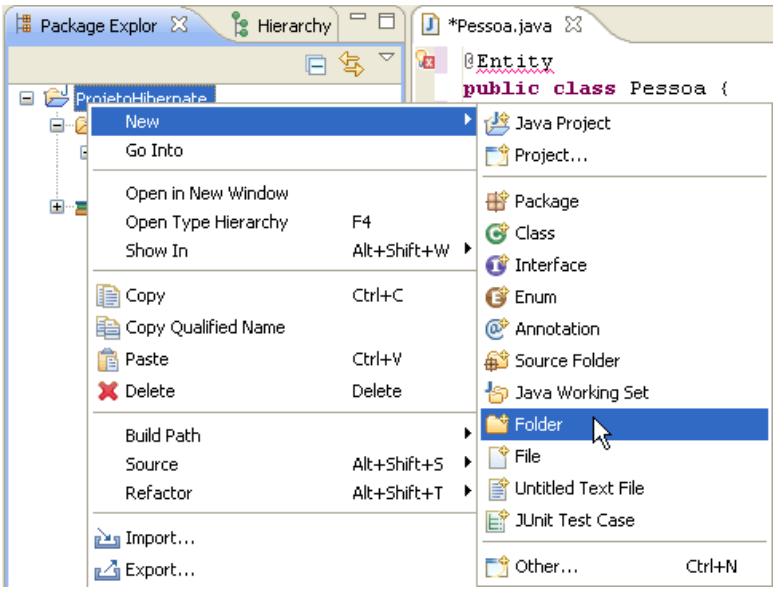
5. Escrever o código na classe Pessoa.

```
import java.io.Serializable;
```

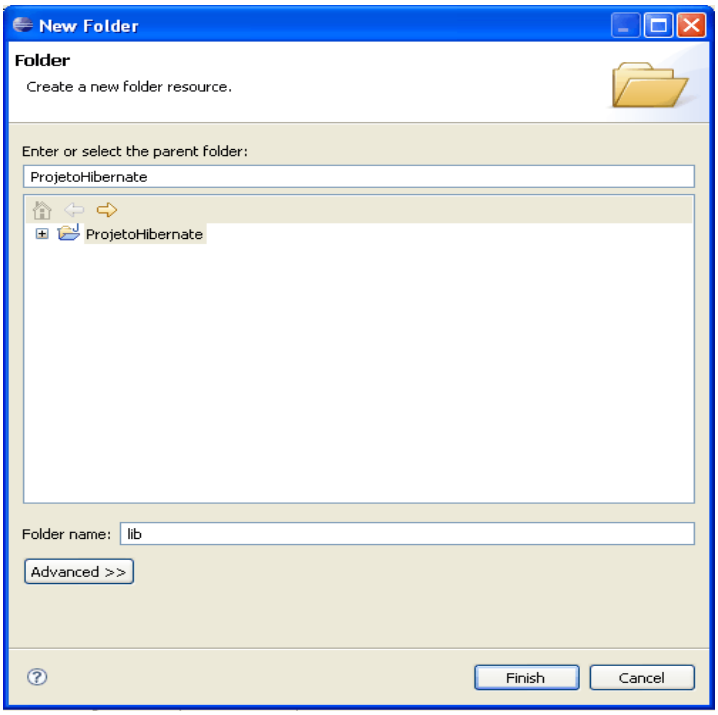
```
import javax.persistence.Id;
import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.Table;
import javax.persistence.Entity;

 @Entity
 @Table (name="pessoa")
 public class Pessoa implements Serializable{

      private static final long serialVersionUID =
1L;

      @Id @GeneratedValue
      private Long id;

      private String nome;


@Column(name="telefone", nullable=true, length=11)
      private String telefone;

      public Long getId() {
            return id;
      }

      public void setId(Long id) {
            this.id = id;
      }

      public String getNome() {
            return nome;
      }

      public void setNome(String nome) {
            this.nome = nome;
      }
```

```
    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
}
```

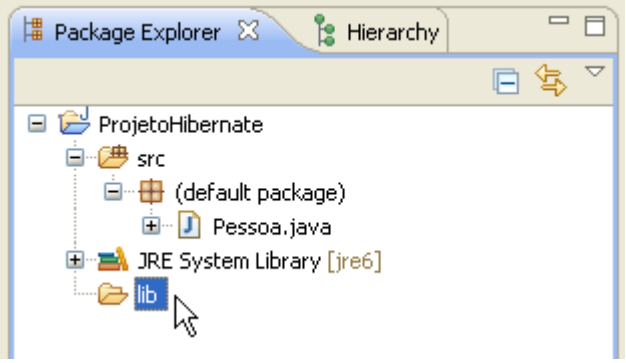
6. Até agora você apenas escreveu códigos em Java. Agora devemos criar uma pasta lib em nosso projeto. Esta pasta irá possuir os arquivos necessários (.jar) para a correta utilização do Hibernate. Vamos criar a pasta lib no projeto. Para criar a pasta lib no projeto: Clique com o botão direito no nome do seu projeto > New > Folder



7. Coloque o nome da pasta (lib)



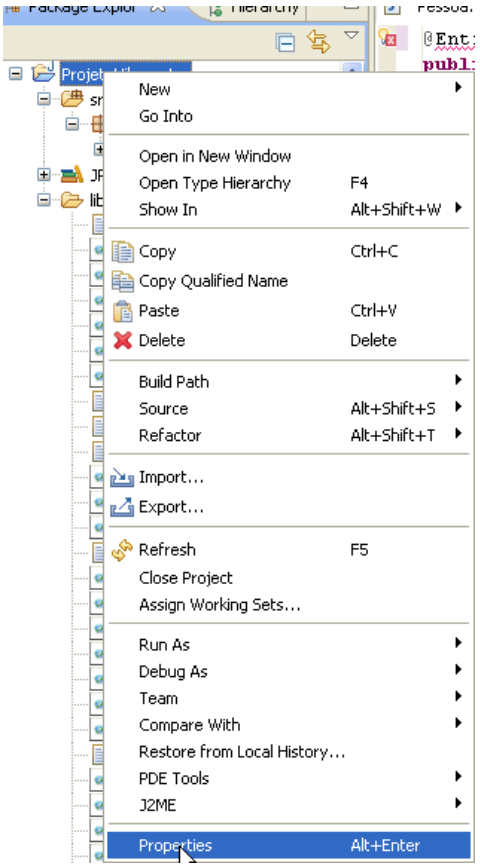
8. No lado esquerdo da janela do Eclipse você tem a pasta lib



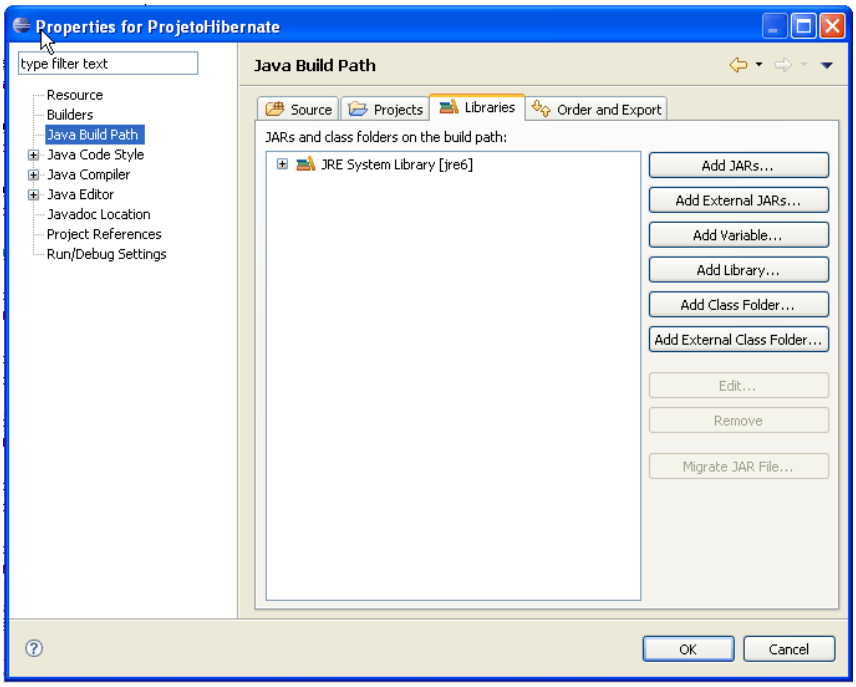
9. Agora iremos copiar os arquivos importantes para a execução no hibernate na pasta lib. Você já deve ter descompactado os arquivos hibernate-3.2.0.ga.zip, hibernate-annotations-3.2.0.GA.zip e mysql-connector-java-5.1.8.zip em uma pasta qualquer.
10. Usando o Windows Explorer, copie o arquivo mysql-connector-java-5.1.8-bin.jar da pasta mysql-connector-java-5.1.8 para a pasta lib do seu projeto. A pasta lib do seu projeto está em Workpspace (que você criou)/ProjetoHibernate/lib
11. Copie o arquivo hibernate-annotations.jar da pasta hibernate-

annotations-3.2.0.GA para a pasta lib do seu projeto

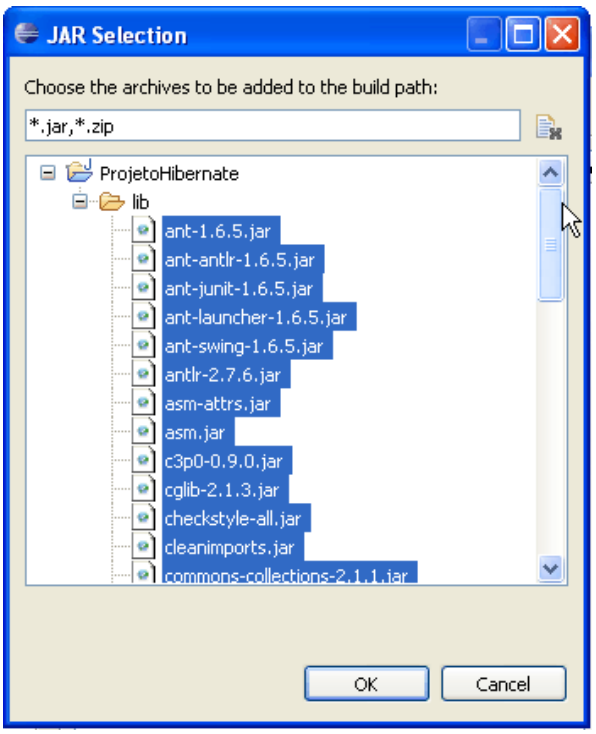
12. Copie os arquivos da pasta hibernate-annotations-3.2.0.GA/lib para a pasta lib do seu projeto.
13. Copie o arquivo hibernate3.jar da pasta hibernate-3.2 para a pasta lib do seu projeto.
14. Copie todos arquivos que estão na pasta hibernate\hibernate-3.2\lib para a pasta lib do seu projeto.
15. Vá para o Eclipse e pressione F5 para atualizar a pasta lib do seu projeto.
16. Agora vamos colocar os arquivos da pasta lib do seu projeto no classpath da aplicação. Os arquivos que estão dentro de Classpath são vistos por todas as classes da aplicação. Clique com o botão direito do mouse no nome do projeto > Properties



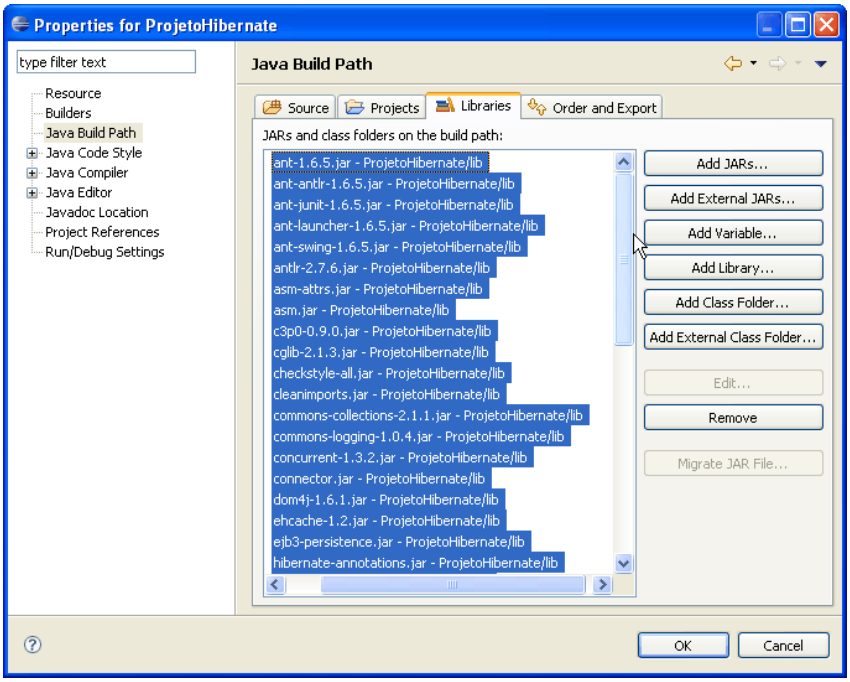
17. Selecione Java Build Path e a guia Libraries



18. Clique em AddJARs... selecione todos os arquivos que estão na pasta lib da sua aplicação. (Marque o primeiro arquivo, pressione shift, role a barra para baixo e marque o ultimo arquivo – sem soltar a tecla shift). Pressione Ok. Assim todos arquivos da pasta lib do seu projeto serão colocados no classpath.



19. Você terá a janela abaixo:



20. Pressione Ok
21. O ambiente já está preparado.
22. Inicie o Xampp, inicie o Apache e o Mysql
23. No browser digite localhost/phpmyadmin
24. Crie um novo banco de dados chamado bd_hibernate.
25. Agora vamos programar o Java para criarmos as tabelas no banco e inserir dados na tabela através do Hibernate.

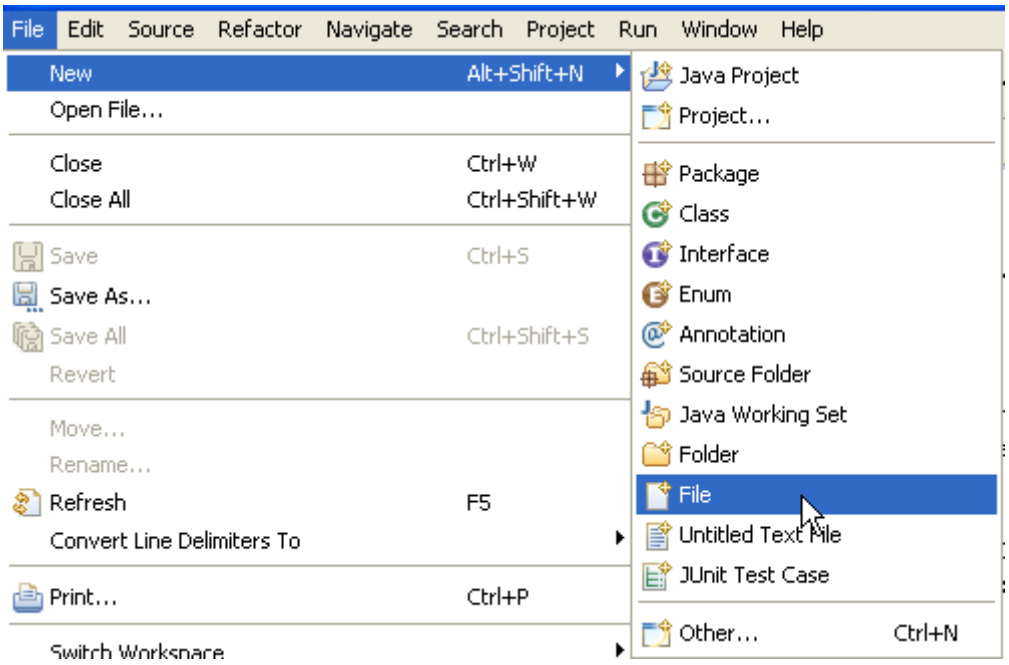
2.4 PROGRAMANDO COM O HIBERNATE

Precisamos adicionar à classe pessoa algumas anotações.

Existe diferença entre Anotações e Comentários. As anotações são textos que colocamos no código e esses textos são utilizados na compilação do programa. Os comentários são ignorados quando o programa é compilado.

Vamos agora criar o arquivo de propriedades do Hibernate. O arquivo deverá se chamar **hibernate.properties** e deverá ser **salvo na pasta src** do seu projeto.
Para criar um arquivo (não uma classe) no Eclipse, clique em File > new > File como mostra a figura abaixo.

Usar o mesmo nome das classes e dos atributos nas tabelas dos bancos de dados ***evita*** vários problemas na programação.



Escreva o código abaixo no seu arquivo **hibernate.properties**

```
hibernate.dialect =
org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class =
com.mysql.jdbc.Driver
hibernate.connection.url =
jdbc:mysql://localhost/bd_hibernate
hibernate.connection.username = root
hibernate.connection.password =
hibernate.show_sql = true
hibernate.format_sql = true
```

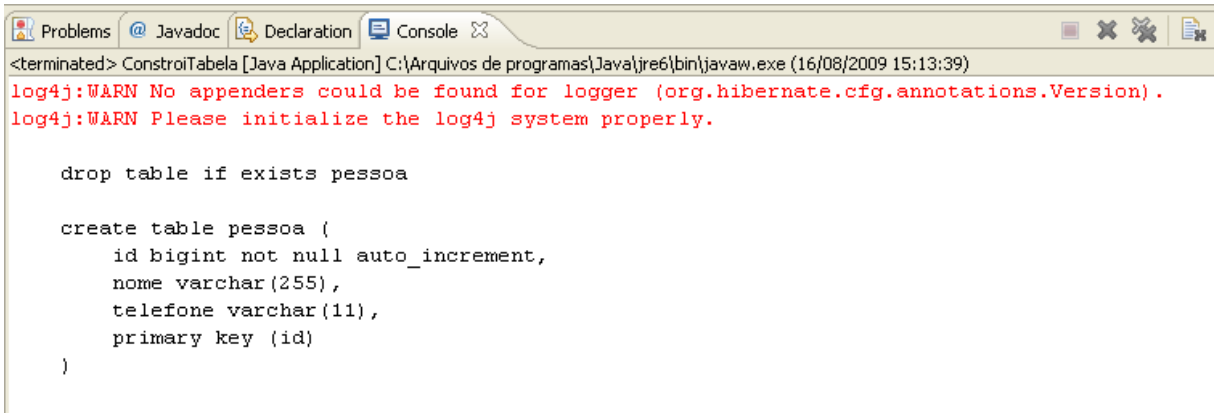
O nome do arquivo deverá ser **hibernate.properties**.

Vamos agora criar uma classe que irá construir a **tabela** no banco de dados **bd_hibernate**.

O arquivo que iremos construir é uma classe Java irá se chamar ConstroiTabela.java.

```
import
org.hibernate.cfg.AnnotationConfiguration;
import
org.hibernate.tool.hbm2ddl.SchemaExport;

 public class ConstroiTabela {
        private static void
create(AnnotationConfiguration cfg) {
         new SchemaExport(cfg).create(true,
true);
     }
     public static void main(String[] args){
         AnnotationConfiguration cfg = new
AnnotationConfiguration();
      try{

create(cfg.addAnnotatedClass(Pessoa.class));
         }
         catch(Exception e){
             e.printStackTrace();
         }
     }
}
```

A saída deverá ser como a figura abaixo:



Veja que na saída do programa são mostradas duas linhas de *warning* (Atenção). Essas linhas são mostradas porque não configuramos nenhum arquivo de log para o Hibernate. Logo abaixo é mostrado o código em SQL que cria a tabela pessoa. Observe também que esse código em SQL primeiro apaga qualquer tabela que tiver o nome de pessoa dentro do banco de dados e depois cria a tabela novamente.

Se você não quiser que essas linhas de warning sejam mostradas copie o arquivo log4j.properties que está na pasta **src** de onde o Hibernate foi descompactado para a pasta **src** do seu projeto. Ele mostrará várias linhas dizendo o que foi executado.

Neste momento você deverá criar a fábrica de sessões (SessionFatory). As sessões criadas têm por responsabilidade conectar ao banco de dados para gravar e buscar objetos do mesmo.

Para verificar sua conexão digite o código abaixo:

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import
org.hibernate.cfg.AnnotationConfiguration;
```

```
public class VerificaConfiguracao {
     public static void main(String[] args)
{
          AnnotationConfiguration cfg = new
AnnotationConfiguration();

     cfg.addAnnotatedClass(Pessoa.class);
          SessionFactory factory =
cfg.buildSessionFactory();
          // cria a sessão
          Session session =
factory.openSession();
          // fecha a sessão
          session.close();
          factory.close();
     }
}
```

Agora vamos criar outro arquivo que terá como objetivo instanciar uma SessionFactory do Hibernate e nos devolver Sessions do Hibernate quando solicitarmos.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import
org.hibernate.cfg.AnnotationConfiguration;
public class UtilitarioHibernate {
     private static SessionFactory factory;
     static {
          AnnotationConfiguration cfg = new
AnnotationConfiguration();

     cfg.addAnnotatedClass(Pessoa.class);
          factory =
```

```
            cfg.buildSessionFactory();
                 }
                 public Session getSession() {
                      return factory.openSession();
                 }
            }
```

O bloco

```
static {
          AnnotationConfiguration cfg = new
AnnotationConfiguration();
          cfg.addAnnotatedClass(Pessoa.class);
          factory = cfg.buildSessionFactory();
     }
```

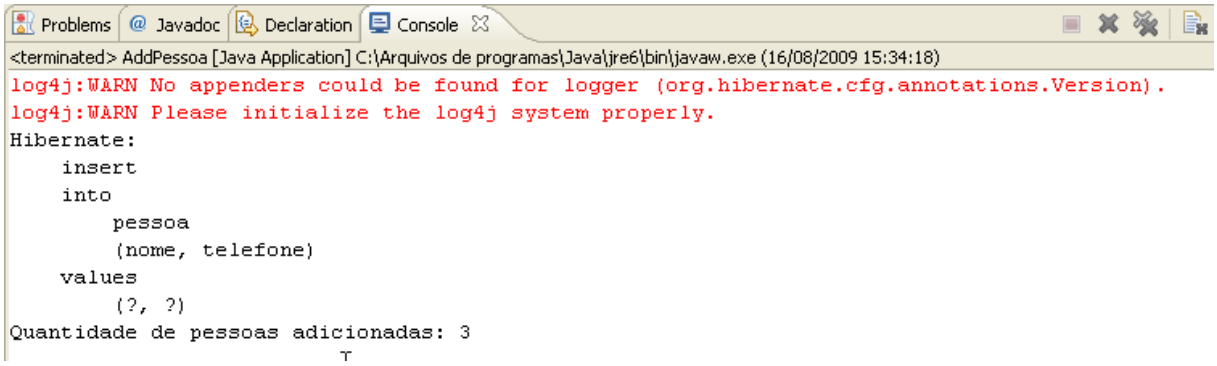
Configura o Hibernate e pega uma SessionFactory. O bloco estático é sempre executado quando a classe é carregada pelo carregador de classes e não mais carregado depois.
O método getSession mostrado no código acima (completo) devolve uma Session (através de SessionFactory).

Vamos agora adicionar uma pessoa ao banco de dados. Vejamos código:

```
            import org.hibernate.Session;

            public class AddPessoa {
                 public static void main(String[] args)
            {
                      Pessoa p = new Pessoa();
                      p.setNome("Elvis Presley");
                      p.setTelefone("88348988");
                      Session session = new
            UtilitarioHibernate().getSession();
```

```
            session.save(p);
            System.out.println("Quantidade de
pessoas adicionadas: " + p.getId());
            session.close();
        }
}
```

A saída do código acima será:



Observe que o arquivo log4j.properties foi retirado da pasta src da aplicação. Caso contrário a saída seria muito grande devido aos comentários colocados.

Para buscar uma pessoa no banco de dados podemos utilizar o código abaixo:

```
import org.hibernate.Session;

public class BuscaPessoa {
 public static void main(String[] args){
  Session session = new
UtilitarioHibernate().getSession();
 Pessoa achou = (Pessoa)
session.load(Pessoa.class, 25L);
 System.out.println("Nome encontrado: "
+achou.getNome());
        }
}
```

Note que a linha Pessoa achou = (Pessoa) session.load(Pessoa.**class**, 25L); O 25L é o id da pessoa na tabela. Na verdade 25 é o id da pessoa L diz para a linguagem que é um Long (confirmando o tipo de dados que foi utilizado para id).

Caso você coloque um número diferente para o id uma exceção será disparada.

2.5 A CLASSE DAO

Data Access Object ou simplesmente DAO faz com que o acesso ao banco de dados seja isolado em classes relativamente simples cuja que será instanciada. Sua instância é um objeto responsável por acessar dados no banco de dados. Vamos construir nossa classe DAO para melhorarmos o acesso dos dados no nosso banco de dados.

```
import org.hibernate.Session;

public class PessoaDAO {
     private Session session;
     public PessoaDAO (Session session) {
          this.session = session;
     }
     public void incluir (Pessoa p) {
          this.session.save(p);
     }
     public void excluir (Pessoa p) {
          this.session.delete(p);
     }
     public Pessoa consultar (Long id) {
          return (Pessoa)
this.session.load(Pessoa.class, id);
     }
     public void atualizar (Pessoa p) {
          this.session.update(p);
```

```
        }
    }
```

Para usarmos a classe PessoaDAO vamos criar outra classe.

```
import org.hibernate.Session;
import org.hibernate.Transaction;

import org.hibernate.Session;
public class IncluirPessoas {
  public static void main(String[] args){
    Session session = new
UtilitarioHibernate().getSession();
     PessoaDAO dao = new
PessoaDAO(session);
     Pessoa pessoa = new Pessoa();

     //Incluir pessoas
     pessoa.setNome("Raul Seixas");
     pessoa.setTelefone("88776565");
     dao.incluir(pessoa);

     session.close();
  }
}
```

Ao executarmos o código acima teremos a possível saída

```
log4j:WARN No appenders could be found for logger
(org.hibernate.cfg.annotations.Version).
log4j:WARN Please initialize the log4j system
properly.
Hibernate:
    insert
```

```
    into
        pessoa
        (nome, telefone)
    values
        (?, ?)
```

Se você buscar um id inválido qual será a saída?

Para atualizar as pessoas na tabela do banco de dados acrescente o código abaixo na sua classe ExecutaPessoaDAO.

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class AlterarPessoas {
  public static void main(String[] args){
    Session session = new
UtilitarioHibernate().getSession();
    PessoaDAO dao = new PessoaDAO(session);
    Pessoa pessoa = new Pessoa();
```

```
    //Alterar pessoas
    Transaction tr =
session.beginTransaction();
    pessoa.setId(53L);
    pessoa.setNome("Maria Betânia");
    pessoa.setTelefone("99666688");
    dao.atualizar(pessoa);
    tr.commit();
    session.close();
  }
}
```

Vamos ver como devemos programar para excluir uma pessoa da tabela

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class ExcluirPessoas {
     public static void main(String[]
args){
          Session session = new
UtilitarioHibernate().getSession();
          PessoaDAO dao = new
PessoaDAO(session);
          Pessoa pessoa = new Pessoa();

          //Exclui pessoas
          Transaction tr =
session.beginTransaction();
          pessoa.setId(31L);
          dao.excluir(pessoa);

          tr.commit();

          session.close();
     }
}
```

A linha

```
pessoa.setId(31L);
```

configura o Id da pessoa que será excluída.

Para consultarmos uma pessoa no banco escrevemos o código abaixo:

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class ConsultarPessoas {
  public static void main(String[] args){
    Session session = new
UtilitarioHibernate().getSession();
    PessoaDAO dao = new PessoaDAO(session);

     //Consultar pessoas
     System.out.println("Id:"+dao.consultar(27L)
.getId());
     System.out.println("Nome:"+dao.consultar(27L)
.getNome());
     System.out.println("Telefone:"+dao.consultar(27L)
.getTelefone());
  }
}
```

2.6 USANDO A CLÁUSULA WHERE

Vimos que até agora conseguimos apenas consultar um objeto no banco de dados. Vamos ver agora como devemos fazer para consultar um conjunto de objetos.

Para fazer consultas em objetos iremos utilizar o HQL (Hibernate Query Language). O HQL faz consultas em objetos e simplifica o relacionamento entre tabelas.

Veja mais sobre o HQL em
http://docs.jboss.org/hibernate/stable/core/reference/en/html/queryhql.html

Vamos ver um exemplo utilizando o HQL

```
import java.util.List;
import org.hibernate.Session;

public class HQLBuscaPessoas {
  public static void main(String[] args){
    Session session = new
UtilitarioHibernate().getSession();
    List<Pessoa> lista = null;
    lista = session.createQuery("from
Pessoa where id > 30").list();
    for (Pessoa atual : lista) {
      System.out.print(atual.getNome());
      System.out.print(" -
"+atual.getTelefone());
      System.out.println();
    }
  }
}
```

2.7 EXERCÍCIO

Faça um programa para controlar o cadastro de CD´s. Os campos da tabela são id, nome do CD e quantidade de músicas.
Faça as operações de inclusão, exclusão, alteração e consulta para os CD´s .

Escrever aplicações Java para banco de dados é necessário que tenhamos um driver correspondente para o banco de dados.
Os métodos da linguagem Java facilitam a escrita de aplicativos com acesso a banco de dados.
O padrão de projeto MVC facilita e organiza a contrução de grandes aplicações.

O hibernate facilita a manipulação de banco de dados pois retira o código SQL do corpo da aplicação.
A configuração do hibernate é um processo extenso mas quando configurado facilita a manipulação de banco de dados e torna o código mais limpo.

APONTAMENTOS SOBRE A PRÓXIMA UNIDADE

A próxima unidade iremos trabalhar com as interfaces gráficas avançadas na qual iremos desenhar interfaces gráficas com recursos avançados da linguagem Java.

3 UNIDADE

METAS

Depois de ler o capítulo você será capaz de escrever interfaces gráficas avançadas e irá manipular o posicionamento do mouse na janela, identificar as teclas do mouse, identificar o que foi pressionado no teclado, criar menus poupups, configurar as aparências das janelas, utilizar frames internos, utilizar classes adaptadoras

OBJETIVOS

- Manipular o posicionamento do mouse
- Identificar as teclas do mouse
- Identificar o que foi pressionado no teclado
- Construir menus poupups
- Configurar as aparências das janelas
- Utilizar frames internos para o desenvolvimento de aplicações
- Utilizar classes adaptadoras

PRÉ-REQUISITOS

- Saber escrever códigos para a construção de interfaces gráficas
- Manipular eventos
- JDK (Kit de desenvolvimento Java) instalado.
- IDE Eclipse instalada

3 INTERFACE GRÁFICA AVANÇADA

3.1 IDENTIFICANDO POSIÇÕES DO MOUSE NA JANELA

Agora veremos mais alguns programas que exploram a interface da linguagem Java. Essa unidade é bem fácil e interessante.

O mouse é um periférico muito utilizado nos programas atuais devido as janelas que as novas aplicações apresentam.

O pacote java.awt possui interfaces para controlar os movimentos do mouse na tela.

O programa abaixo mostra como identificar as posições do mouse, se o botão está pressionado ou não etc.

O programa abaixo mostra uma janela e quando o mouse é passado por cima dela então uma janela é mostrada indicando que o mouse está sobre a janela.

Pressione **<ENTER>** para fechar a janela.

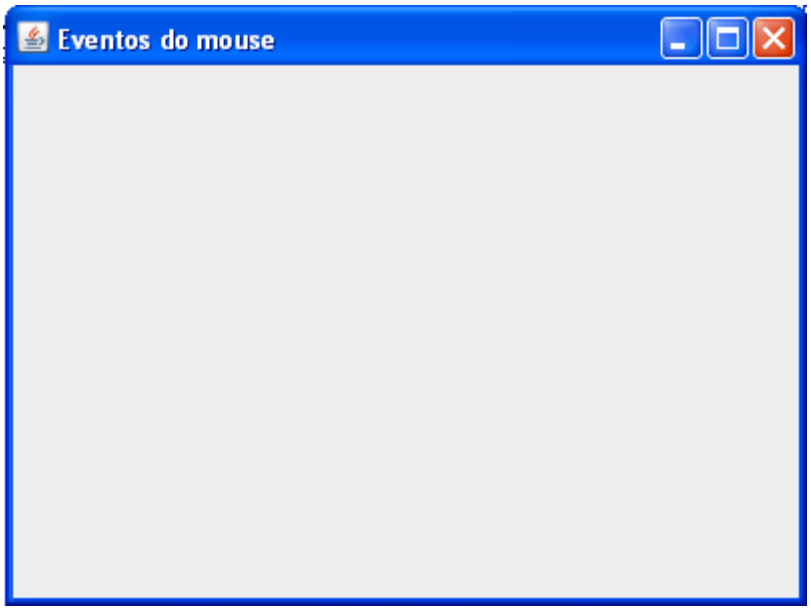
Copie e execute o programa abaixo para verificar as possibilidades de programação com o mouse.

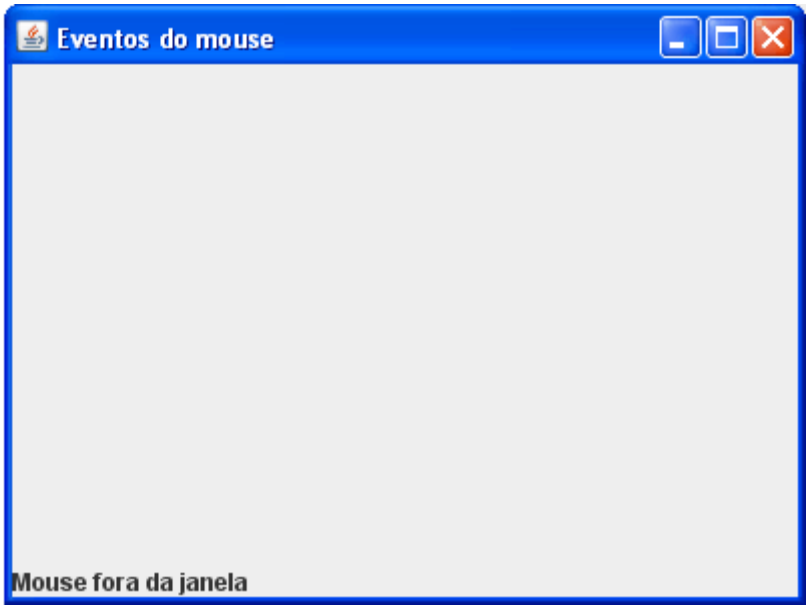
```
import java.awt.BorderLayout;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

public class TrataEventoMouse implements
MouseListener, MouseMotionListener {
   private JFrame janela;
```

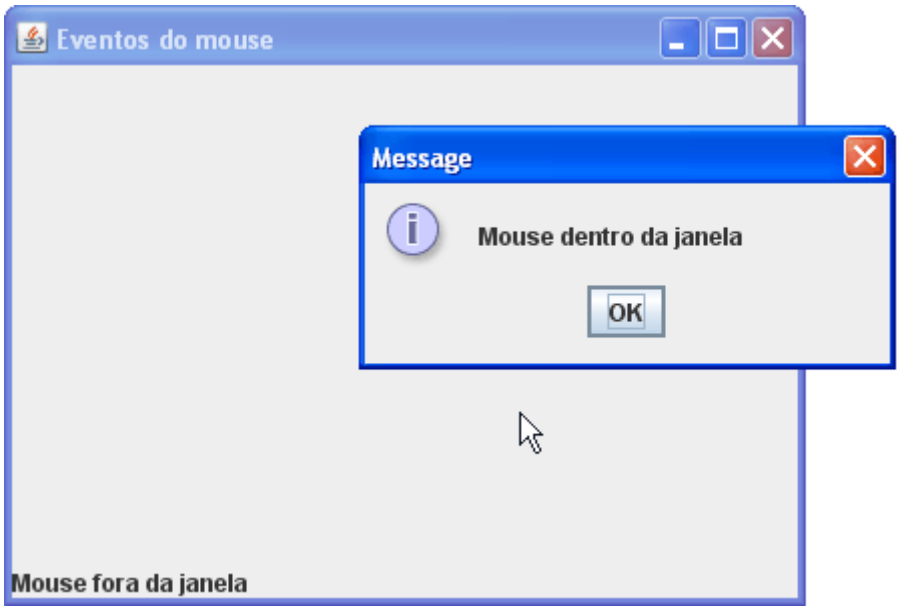
```
   private JLabel barraStatus;

   public TrataEventoMouse()
   {
       barraStatus = new JLabel();
       janela = new JFrame("Eventos do
mouse");
       janela.setLayout(new
BorderLayout());
       janela.add( barraStatus,
BorderLayout.SOUTH );
       janela.addMouseListener(this);
       janela.addMouseMotionListener(this);
       janela.setLocation(200,300);
       janela.setSize( 400, 300 );
       janela.setVisible( true );
   }

   public void mouseClicked( MouseEvent
event )
   {
      barraStatus.setText( "Posição inicial
em: " + event.getX() +
         ", " + event.getY());
   }
   public void mousePressed( MouseEvent
event )
   {
      barraStatus.setText( "Botão
pressionado: " + event.getX() +
         ", " + event.getY());
   }
   public void mouseReleased( MouseEvent
event )
```

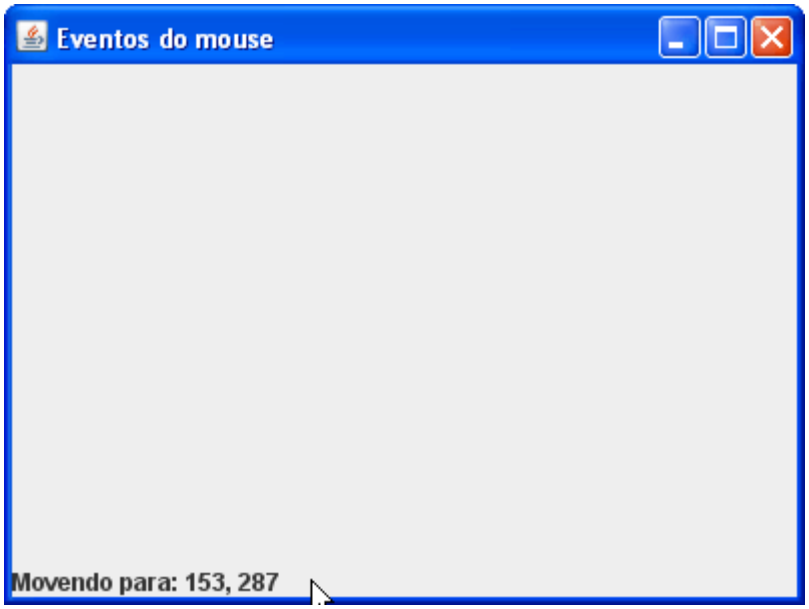
```
   {
      barraStatus.setText( "Botão solto em:
" + event.getX() + ", " + event.getY());
   }
   public void mouseEntered( MouseEvent
event )
   {
      JOptionPane.showMessageDialog( null,
"Mouse dentro da janela" );
   }
   public void mouseExited( MouseEvent
event )
   {
      barraStatus.setText( "Mouse fora da
janela" );
   }
   public void mouseDragged( MouseEvent
event )
   {
      barraStatus.setText( "Arrastando em:
" + event.getX() +
         ", " + event.getY());
   }
   public void mouseMoved( MouseEvent event
)
   {
      barraStatus.setText( "Movendo para: "
+ event.getX() +
         ", " + event.getY());
   }
}
```

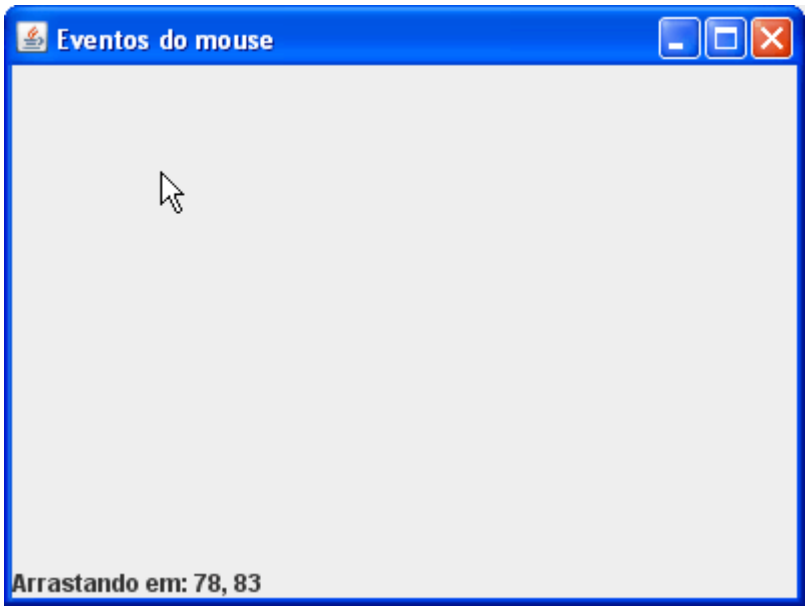
Veja as saídas do programa acima:

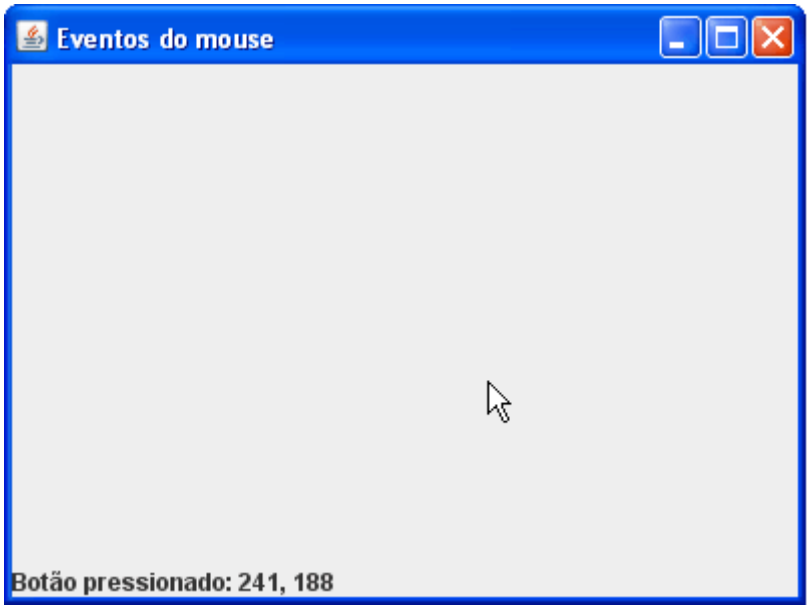

Eventos do mouse


Eventos do mouse
Mouse fora da janela


Eventos do mouse
Message
Mouse dentro da janela
OK
Mouse fora da janela


Eventos do mouse
Movendo para: 153, 287





Como você deve ter observado a classe TrataEventoMouse implementa as interfaces MouseListener e MouseMotionListener. Os métodos escritos no programa são os métodos dessas interfaces.

Altere o código anterior para que quando o mouse foi movido para coordenadas iguais uma janela de diálogo seja disparada dizendo que as coordenadas são iguais.

3.2 IDENTIFICANDO OS BOTÕES DO MOUSE

Algumas aplicações que você for desenvolver deverá identificar qual dos botões do mouse foi pressionado e também onde esse botão foi pressionado (principalmente em jogos).

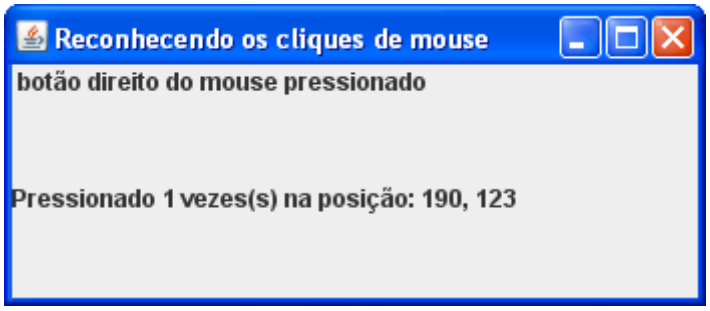
O exemplo abaixo identifica qual botão do mouse foi pressionado, quantas vezes ele foi pressionado e em que posição ele foi pressionado.

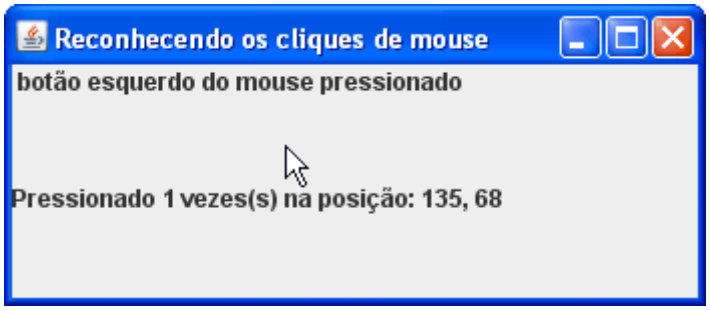
Digite o código abaixo para ver o resultado.

```
import java.awt.BorderLayout;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class DiferenciaBotoes {
   private int x, y;
   JFrame janela;
   JLabel status1, status2;

   public DiferenciaBotoes()
   {
      janela = new JFrame("Reconhecendo os
cliques de mouse");
       janela.setLayout(new
BorderLayout());

      janela.addMouseListener(new
EventoClick());
      janela.setSize( 350, 150 );
      janela.setVisible( true );
```

```
      status1 = new JLabel();
      status2 = new JLabel();
   }

   public static void main( String args[] )
   {
        DiferenciaBotoes application = new
DiferenciaBotoes();
   }


   private class EventoClick extends
MouseAdapter {

      public void mouseClicked( MouseEvent
event )
      {
         x = event.getX();
         y = event.getY();

         status1.setText("Pressionado " +
event.getClickCount() + " vezes(s) na
posição: " + x + ", "+ y);
         janela.add(status1,
BorderLayout.CENTER);


         if ( event.isMetaDown() )
            status2.setText(" botão direito
do mouse pressionado");

         else if ( event.isAltDown() )
            status2.setText("botão do
```

```
centro pressionado");

            else
              status2.setText(" botão
esquerdo do mouse pressionado");

          janela.add(status2,
BorderLayout.NORTH);
        }
    }
}
```





Vamos fazer um pequeno jogo com esse programa. Você deve alterar o programa para que quando o mouse for pressionado na posição 122 e 44 uma mensagem de texto deverá ser mostrada para o usuário. Quem achar a posição primeiro ganha o jogo.

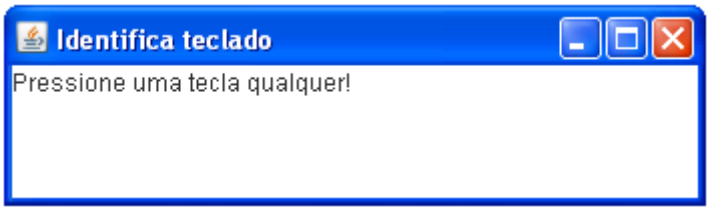
3.3 IDENTIFICANDO O TECLADO

Utilizar as teclas de função em uma aplicação pode ser muitro importante para o usuário, pois facilita a utilização do software. O programa abaixo mostra como tratar os eventos para manipulação das teclas de função e de outras teclas também.

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

public class IdentificaTeclado implements
KeyListener {
   private String texto1 = "", texto2 = "",
texto3 = "";
   private JTextArea textArea;
   private JFrame janela;

   public IdentificaTeclado()
   {
      janela = new JFrame("Identifica
teclado");
      textArea = new JTextArea(30,45);
      textArea.setText("Pressione uma tecla
qualquer!");
      textArea.setEnabled(false);
      janela.add(textArea);

      janela.addKeyListener(this);
      janela.setLocation(300,200);
      janela.setSize(350, 100);
      janela.setVisible(true);
   }
```

```
   public void keyPressed(KeyEvent event)
   {
      texto1 = "Tecla pressionada: " +
KeyEvent.getKeyText( event.getKeyCode());
      saida(event);
   }
   public void keyReleased( KeyEvent event
)
   {
      texto1 = "Tecla pressionada: " +
KeyEvent.getKeyText( event.getKeyCode());
      saida( event);
   }
   public void keyTyped( KeyEvent event )
   {
      texto1 = "Tecla pressionada: " +
event.getKeyChar();
      saida(event);
   }
   private void saida(KeyEvent event)
   {
      texto2 = "Esta tecla " +
(event.isActionKey() ? "" : "não " ) + "é
uma tecla de função";
      String temp =
KeyEvent.getKeyModifiersText(
event.getModifiers() );

      texto3 = "Tecla modificadora: " +
(temp.equals( "" ) ? " " : temp);
      textArea.setText(texto1 + "\n" +
texto2 + "\n" + texto3 + "\n");
   }
   public static void main(String args[])
```

```
    {
        IdentificaTeclado teclado = new
IdentificaTeclado();
    }
}
```



Depois de executar o programa acima você sabe quais métodos capturam as teclas.

Qual método captura as teclas de funções?

Qual método captura as teclas modificadoras? (CTRL ou SHIFT)

Qual método captura as outras teclas?

O que faz a linha

`texto3 = "Tecla modificadora: " + ( temp.equals( "" ) ? " " : temp );` no programa acima?

3.4 COPIAR TEXTO SELECIONADO PARA OUTRA ÁREA DE TEXTO

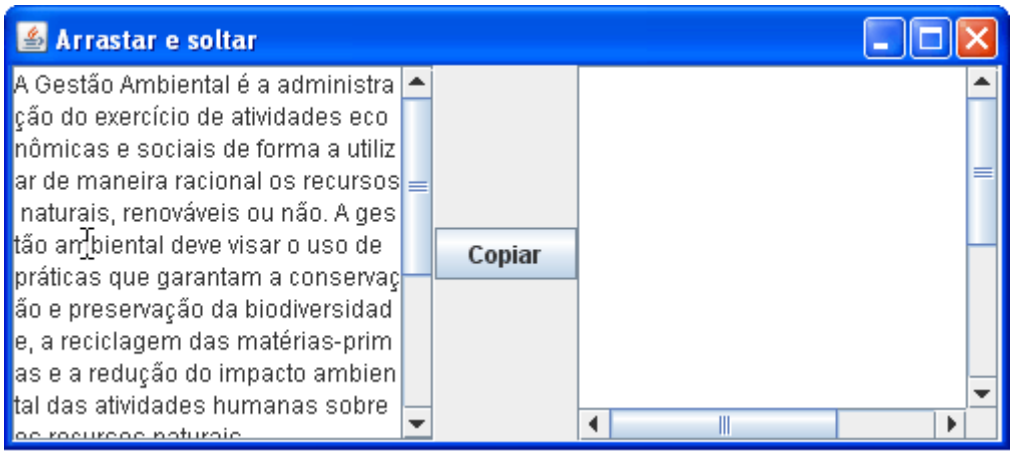
Copiar textos é uma facilidade que encontramos em alguns programas. Essa técnica diminui muito o trabalho dos usuários. O programa que faremos agora fará com que possamos selecionar um texto e copiá-lo de uma janela para outra.

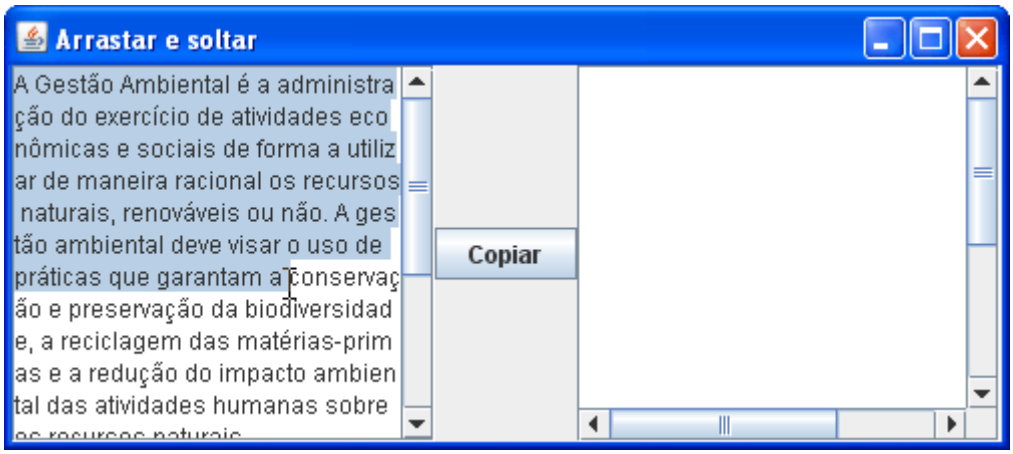
```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class CopiarTexto{
   private JTextArea txArea1, txArea2;
   private JButton btoCopia;
   private JFrame janela;
   public CopiarTexto()
   {
      janela = new JFrame("Arrastar e
soltar");
      Box areaTexto =
Box.createHorizontalBox();
      String texto = "A Gestão Ambiental é
a administração do exercício de atividades
econômicas e sociais de forma a utilizar de
maneira racional os recursos naturais,
renováveis ou não. A gestão ambiental deve
visar o uso de práticas que garantam a
conservação e preservação da
biodiversidade, a reciclagem das matérias-
primas e a redução do impacto ambiental das
atividades humanas sobre os recursos
naturais. Sed vitae velit at turpis
convallis sagittis vitae vehicula ipsum
Nulla pellentesque, ante non condimentum
ultrices, massa dui accumsan risus, et
vehicula est elit nec est. Nunc sed tortor
nunc, quis interdum nisi. Vestibulum
volutpat lacinia purus nec consequat. Duis
auctor tellus sit amet enim porta auctor.
```

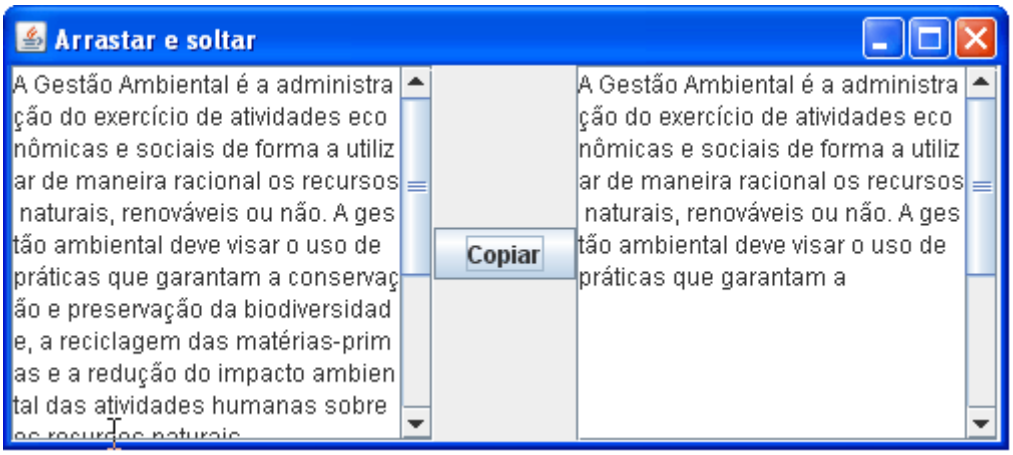
Donec porttitor pellentesque consequat. Fusce eget metus at augue volutpat tempus. Phasellus placerat magna in nibh pulvinar dignissim eget quis augue. Curabitur pellentesque erat vulputate nibh convallis laoreet. Curabitm cursus tempor risus. Fusce tincidunt dignissimhendrerit. Sed placerat erat iaculis odio dapibus feugiat. Sed at lorem justo. Nullam semper, quam eu pellentesque. Proin commodo nulla at nunc feugiat malesuada.";

```
      txArea1 = new JTextArea( texto, 20,
25 );
      txArea1.setLineWrap(true);
      areaTexto.add( new JScrollPane(
txArea1 ) );
      btoCopia = new JButton( "Copiar" );
      btoCopia.addActionListener( new
ActionListener() {
      public void actionPerformed(
ActionEvent event )
            {
               txArea2.setText(
txArea1.getSelectedText() );
               txArea2.setLineWrap(true);
            }
         }
      );

      areaTexto.add( btoCopia );
      txArea2 = new JTextArea( 20, 25 );
      txArea2.setEditable( false );
      areaTexto.add( new JScrollPane(
txArea2 ) );
```

```
            janela.add(areaTexto);
            janela.setLocation(300,200);
            janela.setSize(425, 200);
            janela.setVisible(true);
        }
        // execute application
        public static void main( String args[] )
        {
            CopiarTexto as = new CopiarTexto();
        }
}
```







Lendo o programa do item 3.4 responda:

a) Qual comando faz a quebra de linha na caixa de texto?

b) Copie a linha que copia a linha ou texto da janela txArea1

3.5 UTILIZANDO MENU POPUP

Os menus PopUp são muito práticos e facilitam a usabilidade de um programa.

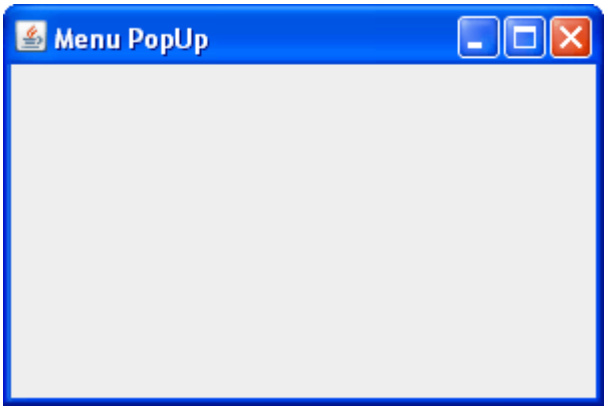
O código abaixo mostra como construir os menus PopUp´s

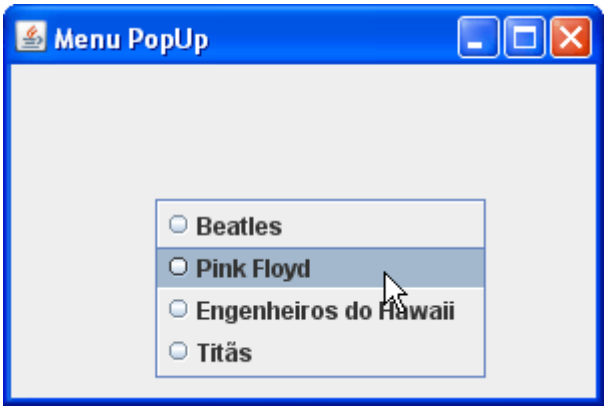
```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPopupMenu;
import javax.swing.JRadioButtonMenuItem;

public class MenuPopUp{
    private JRadioButtonMenuItem items[];
    private String itens[] = { "Beatles",
"Pink Floyd", "Engenheiros do Hawaii",
"Titãs" };
    private JFrame janela;
```

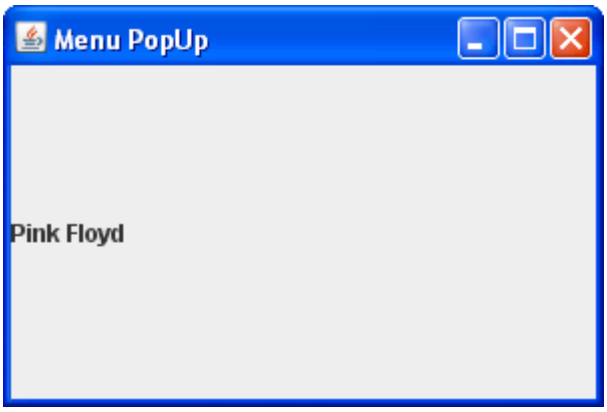
```
    private JLabel lblSaida;

    private JPopupMenu popupMenu;

    public MenuPopUp()
    {
        janela = new JFrame("Menu PopUp");
        lblSaida = new JLabel();
        janela.add(lblSaida);
        ItemHandler evento = new
ItemHandler();
        ButtonGroup btoGrupo = new
ButtonGroup();
        popupMenu = new JPopupMenu();
        items = new JRadioButtonMenuItem[4];
        for ( int i = 0; i < items.length;
i++ ) {
            items[i] = new
JRadioButtonMenuItem( itens[i] );
            popupMenu.add( items[i] );
            btoGrupo.add( items[i] );

items[i].addActionListener(evento);
        }
        janela.setLocation(400, 300);
        janela.addMouseListener(
            new MouseAdapter() {
                public void mousePressed(
MouseEvent event )
                {
                    checkForTriggerEvent( event
);
                }

                public void mouseReleased(
MouseEvent event )
                {
                    checkForTriggerEvent( event
```

```
);
            }

            private void
checkForTriggerEvent( MouseEvent event )
            {
               if ( event.isPopupTrigger()
)

                  popupMenu.show(
event.getComponent(),
                     event.getX(),
event.getY() );
            }

         }

      );

      janela.setSize( 300, 200 );
      janela.setVisible( true );
   }

   public static void main( String args[] )
   {
      MenuPopUp menu = new MenuPopUp();
   }
   private class ItemHandler implements
ActionListener {
      public void actionPerformed(
ActionEvent event )
      {
         for ( int i = 0; i < items.length;
i++ )

            if ( event.getSource() ==
items[ i ] ) {
```

```
                    lbl Saida.setText(itens[i]);

                    return;
                }
            }
        }
    }
```







3.6 CONFIGURANDO AS APARÊNCIAS

O pacote javax.swing da linguagem Java é considerado um pacote peso leve porque para construir suas janelas elas não dependem do sistema operacional. Podemos por exemplo escrever um programa com aparência do Motif e este programa ser executado no ambiente Windows. Essa técnica é mostrada no código abaixo. Escreva o código abaixo para facilitar o entendimento.

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

public class ConfiguraAparencia{
    private JFrame janela;
    private String opcoes[] = {"Windows",
"Metal", "Motif" };
    private UIManager.LookAndFeelInfo
visual[];
    private JRadioButton rdSistema[];
    private ButtonGroup grupo;
    private JButton btoBotao;
    private JLabel lblTexto;
    private JComboBox combo;

    public ConfiguraAparencia(){
        janela = new JFrame("Mudando as
aparências");
        JPanel superior = new JPanel();
        superior.setLayout( new FlowLayout() );
        lblTexto = new JLabel( "Esta é a
aparência Metal", SwingConstants.CENTER );
        superior.add(lblTexto);
        btoBotao = new JButton( "Botão" );
        superior.add(btoBotao);
        combo = new JComboBox(opcoes);
        superior.add( combo );
       janela.add( superior, BorderLayout.NORTH
);
```

```
        rdSistema = new
JRadioButton[opcoes.length ];
        JPanel  southPanel  = new JPanel();
        southPanel.setLayout(new FlowLayout());
        rdSistema = new
JRadioButton[opcoes.length];
        grupo = new ButtonGroup();
        Evento evento = new Evento();
        for ( int i = 0; i < rdSistema.length;
i++ ) {
          rdSistema[i] = new
JRadioButton(opcoes[i] );
          rdSistema[i].addItemListener(evento);
          grupo.add( rdSistema[i] );
          southPanel.add( rdSistema[i] );
         }

        janela.add( southPanel,
BorderLayout.SOUTH );
        visual =
UIManager.getInstalledLookAndFeels();
        janela.setLocation(400, 200);
        janela.setSize( 500, 200 );
        janela.setVisible( true );
        rdSistema[0].setSelected( true );
    }

        private void changeTheLookAndFeel(int
valor)
    {
        try {
            UIManager.setLookAndFeel(
                visual[valor].getClassName() );

SwingUtilities.updateComponentTreeUI(janela);
        }

        catch ( Exception exception ) {
```

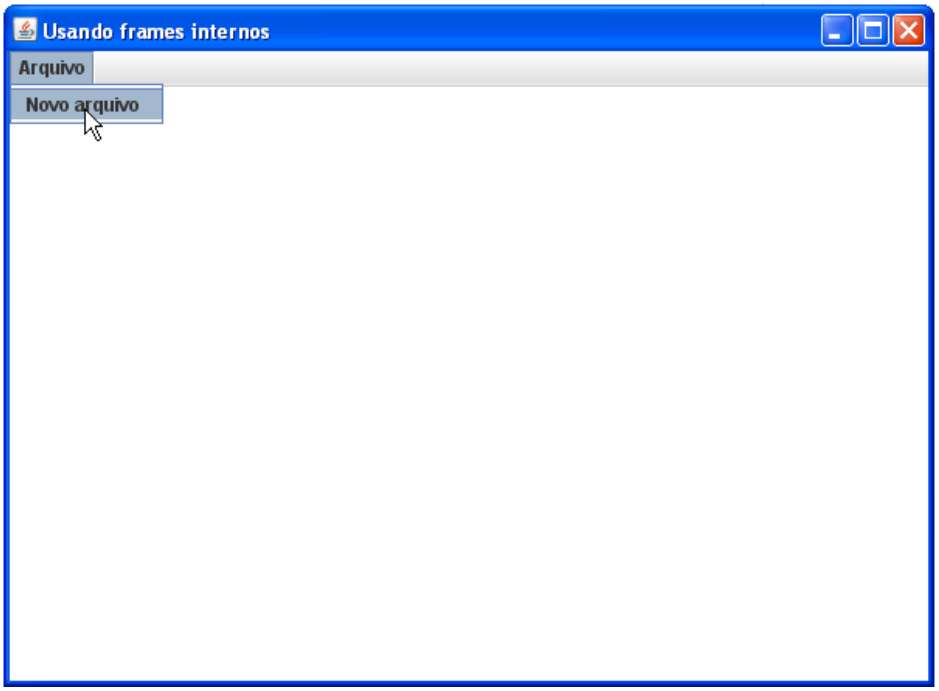
```
            exception.printStackTrace();
        }
    }

    public static void main( String args[] )
    {
        ConfiguraAparencia application = new
ConfiguraAparencia();
    }
    private class Evento implements
ItemListener {
        public void itemStateChanged( ItemEvent
event )
        {
            for ( int i = 0; i <
rdSistema.length; i++ )
                if ( rdSistema[i].isSelected() )
{
                    lblTexto.setText( "Esta é a  "
+ opcoes[i]);
                    combo.setSelectedIndex(i);
                    changeTheLookAndFeel(i);
                }
        }
    }
}
```

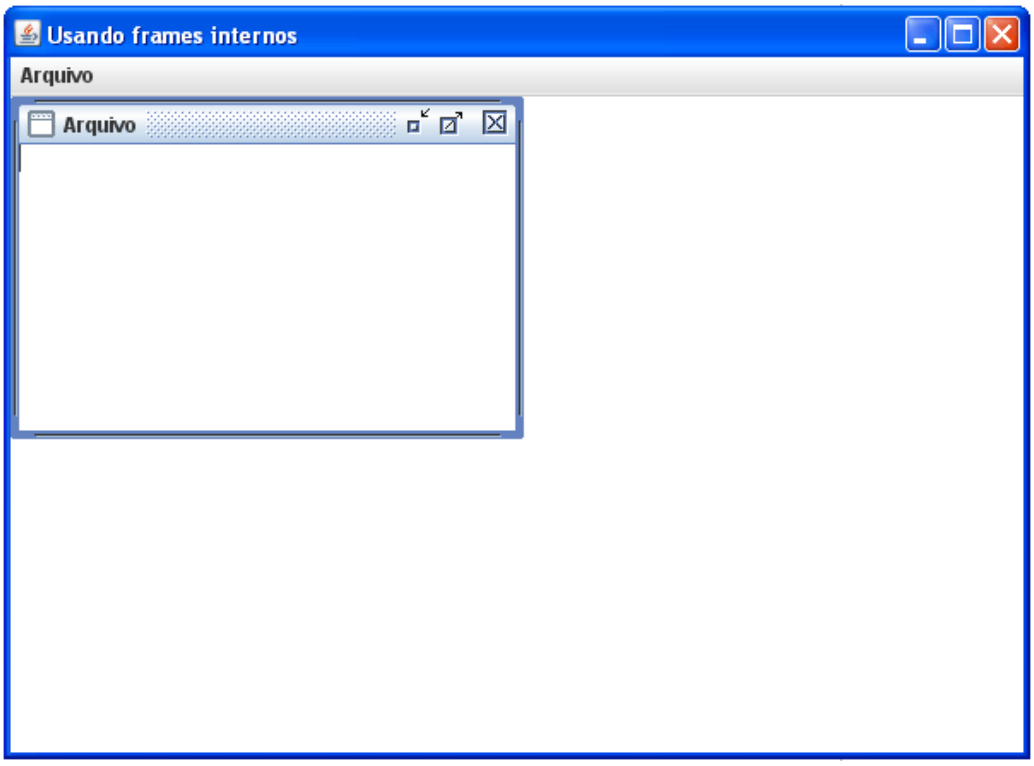
3.7 FRAMES INTERNOS

Os frames internos são muito úteis na construção de aplicações gráficas. Eles facilitam a usabilidade do programa. Frames Internos são janelas que são abertas dentro de uma janela maior chamada Desktop. Essas pequenas janelas podem ser arrastadas dentro da janela principal facilitando a usabilidade do software. Vejamos no exemplo abaixo como utilizar os frames internos.

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

public class FrameInterno{
   private JDesktopPane desktop;
   private JFrame janela;
   private JTextArea areaTexto;
   public FrameInterno()
   {
      janela = new JFrame("Usando frames
internos");
      areaTexto = new JTextArea();
      JMenuBar barraMenu = new JMenuBar();
      JMenu mnuArquivo = new JMenu(
"Arquivo" );
      JMenuItem mnuItemNovo = new
JMenuItem( "Novo arquivo" );
      mnuArquivo.add(mnuItemNovo);
      barraMenu.add(mnuArquivo);
      janela.setJMenuBar(barraMenu);
      desktop = new JDesktopPane();
      janela.add(desktop);
      mnuItemNovo.addActionListener(
            new ActionListener() {
           public void actionPerformed(
ActionEvent event ) {
            JInternalFrame frameInterno =
new JInternalFrame("Arquivo", true, true,
true, true );
               Container container =
frameInterno.getContentPane();
               frameInterno.add(areaTexto);
```

```
frameInterno.setSize(300,200);
                desktop.add( frameInterno );
                frameInterno.setVisible(
true );
            }
         }
      );
      janela.setLocation(400, 200);
      janela.setSize( 600, 440 );
      janela.setVisible( true );
   }
   public static void main( String args[] )
   {
      FrameInterno fi = new FrameInterno();
   }
}
```





Troque alternadamente o parâmetro true da linha

```
JInternalFrame frameInterno = new
JInternalFrame("Arquivo", true, true, true,
true );
```

para false e veja o qua acontece.

3.8 CLASSES ADAPTADORAS

Como vimos, quando implementamos uma interface devemos reescrever **todos** os métodos da interface. As classes adaptadoras fornecem uma implementação padrão de cada método da interface. Esta implementação padrão são métodos com corpo vazio. Sendo assim o programador pode **herdar** (não implementar) a classe adaptadora e sobrescrever somente os métodos que interessam para seus eventos.

Exemplo:

```
import java.awt.BorderLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class Mouse extends JFrame{
    private JLabel estado;
    public Mouse(){
      super("Demonstração do uso do mouse");
      estado = new JLabel();
getContentPane().add(estado, borderLayout.SOUTH);
      //instancia a classe interna
      Posicao m = new Posicao();
      this.addMouseListener(m);
      setSize(300,100);
      setVisible(true);
    }
    //construção da classe interna
    private class Posicao '''extends'''
MouseAdapter{
```

```
            public void mouseClicked(MouseEvent e){
            estado.setText("Botão pressionado em
"+e.getX()+";"+e.getY());
        }
    }
    public static void main(String[] args){
            Mouse m = new Mouse();
    }
}
```

Note que a classe interna Posição **herdou** da classe MouseAdapter e sendo assim só reescreveu o método mouseClicked().

Algumas interfaces possuem classes adaptadoras equivalentes. Veja a tabela abaixo:

Interface	Classe Adaptadora
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

Observação importante

Uma observação importante é que não existe *ActionAdapter* porque a interface *ActionListener* só possui um método chamado *actionPerformed()* e sendo assim não justifica uma classe adaptadora para ele.

4 UNIDADE

METAS

Depois de ler o capítulo você será capaz escrevre aplicações em Java que serão executadas em browsers e hospedadas na internet.

OBJETIVOS

- Escrever programas em Java que são executados em browsers e são hospedados na intenet.
- Identificar o ciclo de vida de um applet

PRÉ-REQUISITOS

- Conhecimento das classes da interface gráfica JFrame
- Conhecimento de construção de códigos em HTML

4. INTRODUÇÃO AOS APPLETS

Os applets são programas escritos na linguagem Java excutados em um navegador. Os applets tornam as páginas mais interativas com o usuário. Com os applets é possível escrever páginas com sons, exibir gráficos, realizar cálculos etc.

Um applet é um arquivo escrito na linguagem Java que gera um arquivo .class e esse arquivo .class que fica hospedado em um servidor web é carregado em um arquivo .html e sua execução é local.

Para chamar um applet em um arquivo .html usamos a tag <applet> </applet> e ela possui alguns parâmetros.

O código abaixo mostra a sintaxe da tab <applet> para inserir um applet em uma página .html.

```
<APPLET CODE=[java applet] WIDTH=[largura]
HEIGHT=[altura]>
</APPLET>
```

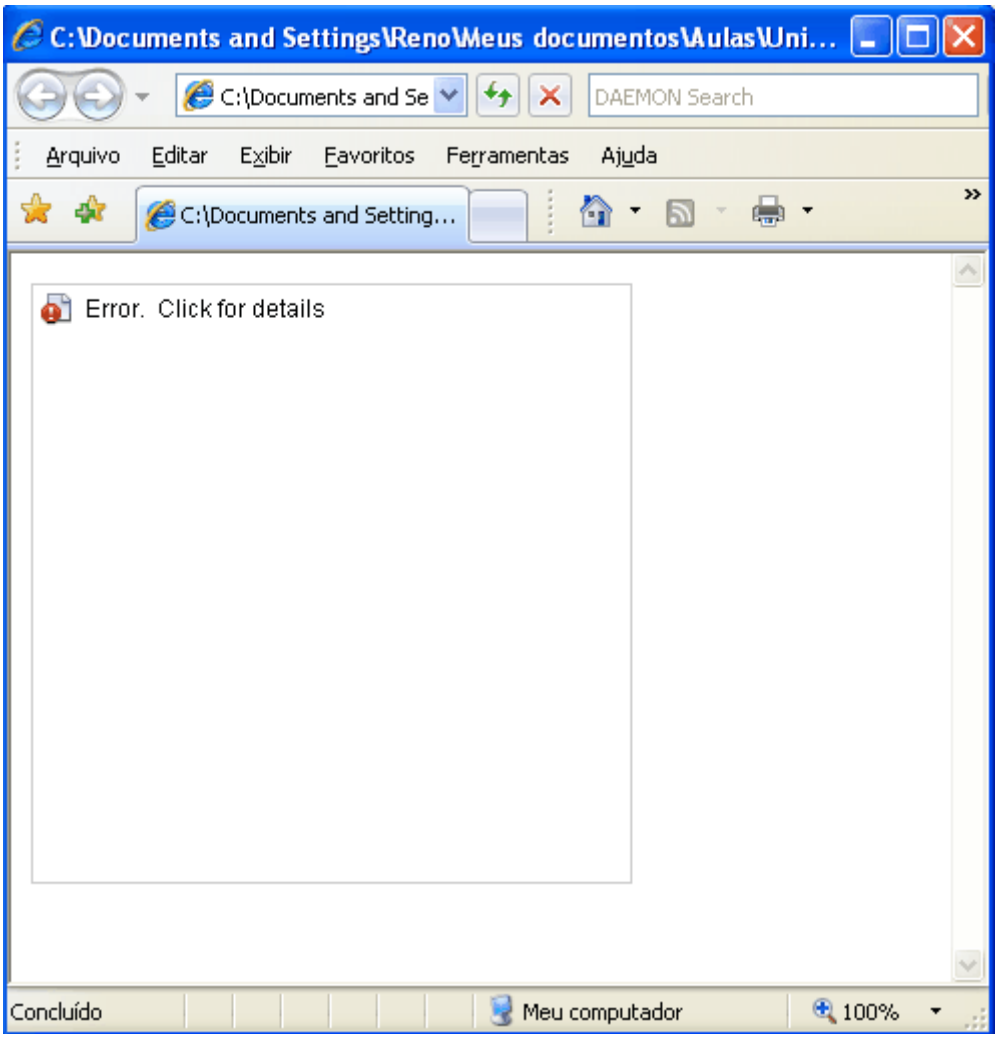
```
<APPLET CODE=nome_da_classe.class  WIDTH =
120  HEIGHT = 150>
</APPLET>
```

Observe que a tag applet forma uma janela com 120 pixels de largura e 150 pixels de altura. Essa janela será onde o applet será executado. Essa janela pode ter o tamanho que você quiser, basta configurar os parâmetros altura e largura.

O exemplo abaixo mostra o exemplo completo de uma tag applet inserida em um arquivo .html.

```
<HTML>
  <BODY>

    <APPLET CODE="nome_da_classe.class"
HEIGHT = 300 WIDTH = 300>
```

```
    </APPLET>
  </ BODY>
</ HTML>
```



Note que na janela acima existe uma linha que delimita o tamanho que a *applet* irá ocupar na página.

Podemos também enviar parâmetros para os applets usando o argumento param. Vejamos o exemplo:

```
<APPLET CODE = nome_da_classe.class
WIDTH=360 HEIGHT=40   NAME=emite_msg>
  <PARAM NAME = texto value="texto que
```

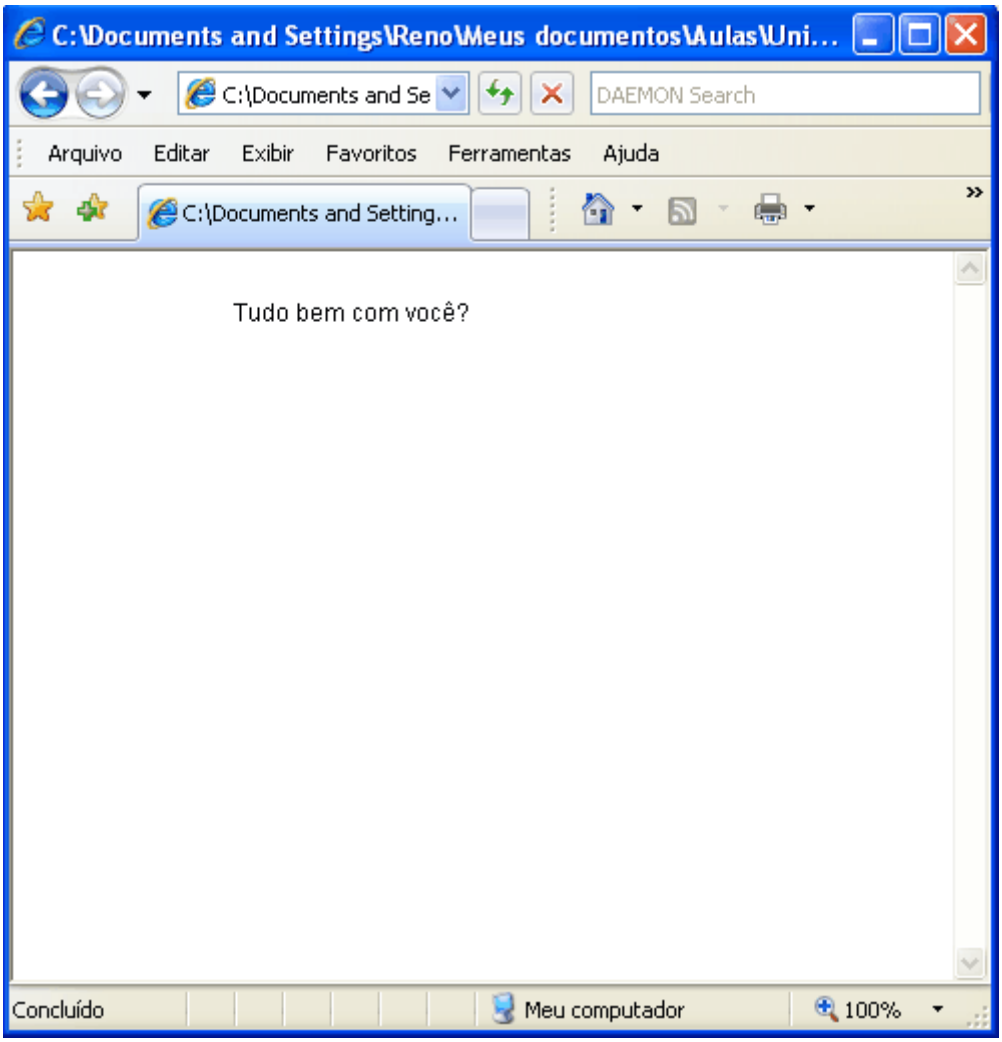
```
aparecerá">
</APPLET>
```

Se você tentar executar o código acima verá somente um espaço reservado para mostrar o applet. Veja que no código precisamos de uma arquivo .class que contenha o texto a ser mostrado.

Vamos construir um applet chamado MeuPrimeiroApplet.java. O código está abaixo:

```
import java.applet.*;
import java.awt.*;

public class MeuPrimeiroApplet extends
Applet {
   public void paint(Graphics g) {
      g.drawString("Tudo bem com você?",
100, 20);
   }
}
```

```
<HTML>
  <BODY>
    <APPLET CODE="MeuPrimeiroApplet.class"
HEIGHT=180 WIDTH=20>
    </APPLET>
  </BODY>
</HTML>
```



Veja que para escrever na tela foi utilizado o método drawString da classe Graphics que está no pacote Java.awt.

Veja também que a classe herda da classe Applet que está no pacote java.applet.

Aqui não utilizamos o método main para iniciar o applet e nem o comando System.out.println( ) para escrever um texto na tela.

O método drawString tem como primeiro argumento o texto a ser escrito e logo depois a posição desse texto dentro da janela já desenhada no HTML que irá armazenar o applet.

Podemos passar parâmetros no HTML para o arquivo Java. Vejamos o código abaixo.

Código HTML

```
<html>
  <head>
  <title>Applets com parâmetros</title>
  </head>
  <body>
     <APPLET CODE =
AppletComParametro.class WIDTH = 360 HEIGHT
= 40 NAME = emite_msg>
        <PARAM NAME = texto value="Esse
texto é do parâmetro!">
     </APPLET>
  </body>
</html>
```

Código Java

```
import java.applet.Applet;
import java.awt.Graphics;
public class AppletComParametro extends
Applet {
   String mensagem;
   public void paint(Graphics g) {
      mensagem = getParameter("texto");
       g.drawString(mensagem, 100, 20);
   }
}
```

O exemplo abaixo foi retirado do site da Sun Microsystems em http://java.sun.com/applets/jdk/1.4/demo/applets/DrawTest/example1.html

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

import java.util.Vector;
```

```
public class Desenho extends Applet{
    DrawPanel panel;
    DrawControls controls;

    public void init() {
     setLayout(new BorderLayout());
     panel = new DrawPanel();
        controls = new DrawControls(panel);
     add("Center", panel);
     add("South",controls);
    }

    public void destroy() {
        remove(panel);
        remove(controls);
    }

    public static void main(String args[]) {
     Frame f = new Frame("DrawTest");
     DrawTest drawTest = new DrawTest();
     drawTest.init();
     drawTest.start();

     f.add("Center", drawTest);
     f.setSize(300, 300);
     f.show();
    }
    public String getAppletInfo() {
        return "A simple drawing program.";
    }
}

class DrawPanel extends Panel implements
```

```
MouseListener, MouseMotionListener {
    public static final int LINES = 0;
    public static final int POINTS = 1;
    int      mode = LINES;
    Vector lines = new Vector();
    Vector colors = new Vector();
    int x1,y1;
    int x2,y2;

    public DrawPanel() {
     setBackground(Color.white);
     addMouseMotionListener(this);
     addMouseListener(this);
    }

    public void setDrawMode(int mode) {
     switch (mode) {
       case LINES:
       case POINTS:
         this.mode = mode;
         break;
       default:
         throw new IllegalArgumentException();
     }
    }
    public void mouseDragged(MouseEvent e) {
        e.consume();
        switch (mode) {
            case LINES:
                x2 = e.getX();
                y2 = e.getY();
                break;
            case POINTS:
            default:
```

```
                colors.addElement(getForeground());
                lines.addElement(new Rectangle(x1, y1,
e.getX(), e.getY()));
                x1 = e.getX();
                y1 = e.getY();
                break;
            }
            repaint();
        }

        public void mouseMoved(MouseEvent e) {
        }

        public void mousePressed(MouseEvent e) {
            e.consume();
            switch (mode) {
                case LINES:
                    x1 = e.getX();
                    y1 = e.getY();
                    x2 = -1;
                    break;
                case POINTS:
                default:
                    colors.addElement(getForeground());
                    lines.addElement(new
Rectangle(e.getX(), e.getY(), -1, -1));
                    x1 = e.getX();
                    y1 = e.getY();
                    repaint();
                    break;
            }
        }

        public void mouseReleased(MouseEvent e) {
```

```
            e.consume();
            switch (mode) {
                case LINES:
                    colors.addElement(getForeground());
                    lines.addElement(new Rectangle(x1, y1,
e.getX(), e.getY()));
                    x2 = -1;
                    break;
                case POINTS:
                default:
                    break;
            }
            repaint();
        }

        public void mouseEntered(MouseEvent e) {
        }

        public void mouseExited(MouseEvent e) {
        }

        public void mouseClicked(MouseEvent e) {
        }

        public void paint(Graphics g) {
         int np = lines.size();

         /* draw the current lines */
         g.setColor(getForeground());
         for (int i=0; i < np; i++) {
             Rectangle p = (Rectangle)lines.elementAt(i);
             g.setColor((Color)colors.elementAt(i));
             if (p.width != -1) {
              g.drawLine(p.x, p.y, p.width, p.height);
```

```
            } else {
             g.drawLine(p.x, p.y, p.x, p.y);
            }
        }
        if (mode == LINES) {
            g.setColor(getForeground());
            if (x2 != -1) {
                   g.drawLine(x1, y1, x2, y2);
            }
        }
    }
}
class DrawControls extends Panel implements
ItemListener {
    DrawPanel target;

    public DrawControls(DrawPanel target) {
     this.target = target;
     setLayout(new FlowLayout());
     setBackground(Color.lightGray);
     target.setForeground(Color.red);
     CheckboxGroup group = new CheckboxGroup();
     Checkbox b;
     add(b = new Checkbox(null, group, false));
     b.addItemListener(this);
     b.setForeground(Color.red);
     add(b = new Checkbox(null, group, false));
     b.addItemListener(this);
     b.setForeground(Color.green);
     add(b = new Checkbox(null, group, false));
     b.addItemListener(this);
     b.setForeground(Color.blue);
     add(b = new Checkbox(null, group, false));
     b.addItemListener(this);
```

```
        b.setForeground(Color.pink);
        add(b = new Checkbox(null, group, false));
        b.addItemListener(this);
        b.setForeground(Color.orange);
        add(b = new Checkbox(null, group, true));
        b.addItemListener(this);
        b.setForeground(Color.black);
        target.setForeground(b.getForeground());
        Choice shapes = new Choice();
        shapes.addItemListener(this);
        shapes.addItem("Lines");
        shapes.addItem("Points");
        shapes.setBackground(Color.lightGray);
        add(shapes);
      }
      public void paint(Graphics g) {
        Rectangle r = getBounds();
        g.setColor(Color.lightGray);
        g.draw3DRect(0, 0, r.width, r.height, false);

            int n = getComponentCount();
            for(int i=0; i<n; i++) {
                Component comp = getComponent(i);
                if (comp instanceof Checkbox) {
                    Point loc = comp.getLocation();
                    Dimension d = comp.getSize();
                    g.setColor(comp.getForeground());
                    g.drawRect(loc.x-1, loc.y-1,
d.width+1, d.height+1);
                }
            }
      }
    public void itemStateChanged(ItemEvent e) {
      if (e.getSource() instanceof Checkbox) {
```

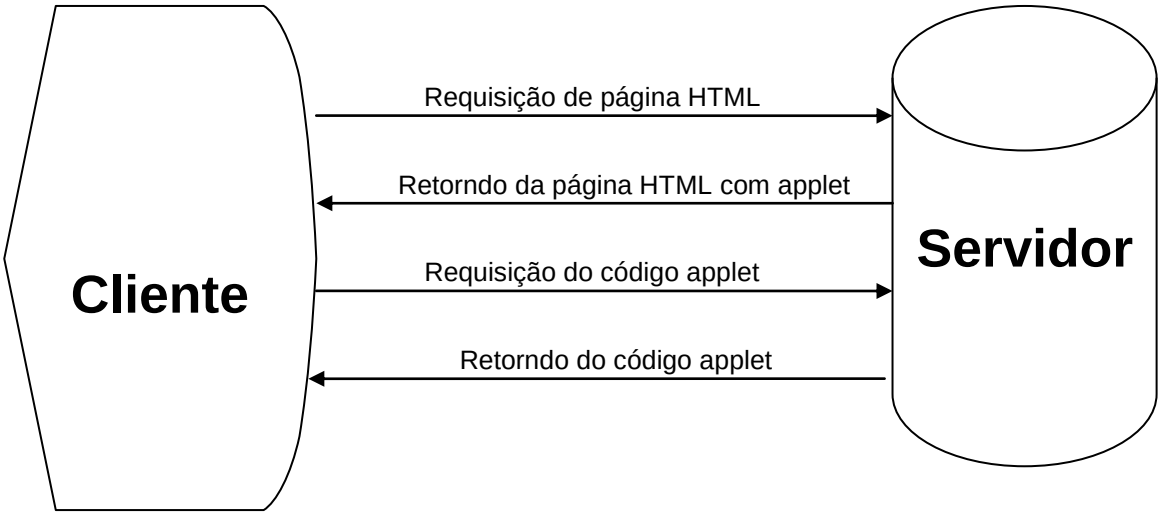
```
target.setForeground(((Component)e.getSource()).getFor
eground());
    } else if (e.getSource() instanceof Choice) {
      String choice = (String) e.getItem();
      if (choice.equals("Lines")) {
     target.setDrawMode(DrawPanel.LINES);
      } else if (choice.equals("Points")) {
     target.setDrawMode(DrawPanel.POINTS);
      }
    }
  }
}
```

Ele constrói um editor de desenho (parecido com *Paint*) para que o usuário faça desenhos dentro do seu site.

Coloque o arquivo class dentro da pasta bin (junto com o arquivo .class) Se for hospedar em um servidor também coloque os arquivos .html junto com os arquivos .class

4.1 CICLO DE VIDA

A execução de código Java no navegador é, como já vimos, definida em uma página no servidor que é carregada e executada no cliente. A tag HTML especifica onde está o código a ser executado do applet (.class) e o posicionamento da excução deste código na página. O funcionamento básico de um applet é mostrado na figura abaixo:



Funcionamento de uma requisição ao servidor de um código applet

Para executar o applet possui quatro métodos: init( ), start( ), stop( ) e destroy( ).

O ciclo de vida de um applet é mostrado na figura abaixo:

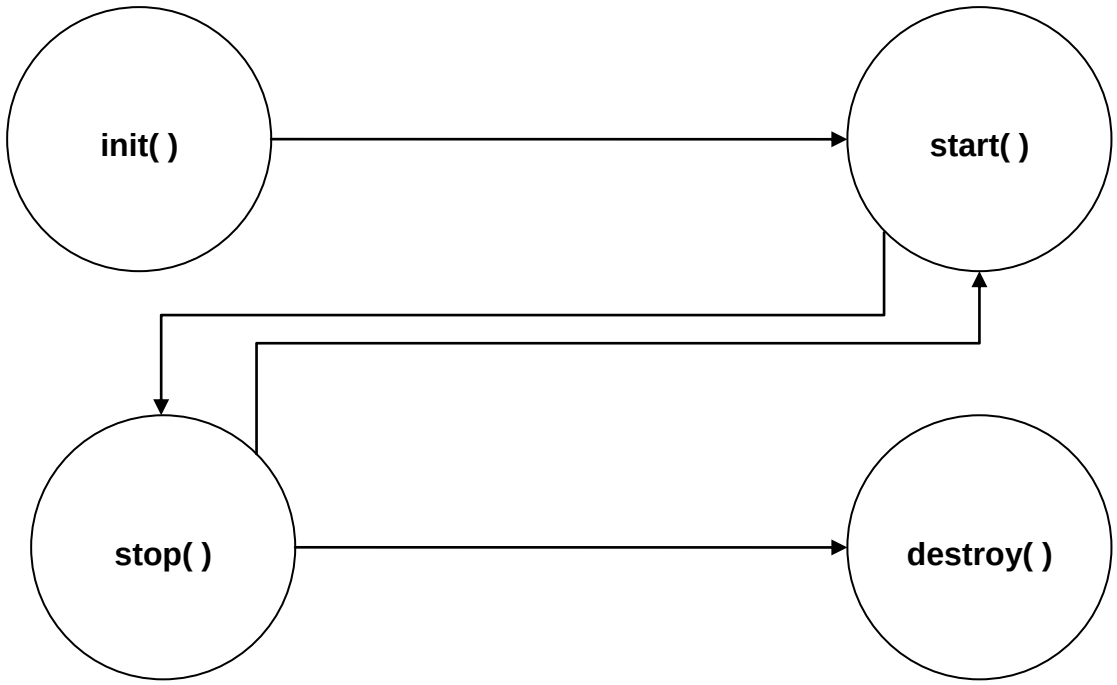


Figura mostra o ciclo de vida de um applet

O método init( ) é quando o applet é chamado pelo navegador (carregado). Este ciclo é responsável pela inicialização do applet (processamento de parâmtros, criação de componentes da interface e carregamento dos componentes da interface).

O método start( ) o applet já está pronto para sua execução. O método start( ) é chamado logo depois do método init( ) e toda vez que o usuário retorna a página que contém o applet.

O método stop( ) é chamado pelo navegador informando que o applet deve interromper sua excução (antes da destruição do applet pelo método destroy( ) ou quando a página que contém o applet é substituída). As atividades interrompidas pelo método stop( ) serão retomadas com a execução do método start( ).

O método destroy( ) é chamado pelo navegador para que o applet libere os recursos que tenham sido alocados por ele. O método stop( ) é sempre chamado antes do método destroy( ).

```
import java.applet.Applet;
import java.awt.Graphics;
public class PrimeiroApplet extends Applet
{
     public void init()
     {
     }

     public void stop()
     {
     }

     public void paint(Graphics g)
     {
       g.drawString("Olá",20,20);
       g.drawString("Escrevi meu primeiro
Applet",40,40);
     }
}
```

Veja que no programa acima:

Importamos a classe Applet que é requerida quando você vai construir um applet.

Importamos também a classe Graphics que possui o método paint( ). Esse método é utilizado para escrever na tela.

O método init( ) é automaticamente chamado quando construímos um applet. Ele é necessário mas não precisa ter nada no corpo (neste exemplo).

O método stop( ) é chamado quando o applet é encerrado i.e. quando o usuário vai para outra página ou sai do navegador. Neste exemplo não precisamso colocar nada dentro dele.

paint( ) é o método padrão para desenhar textos ou figuras na tela.

4.2 INTERFACE GRÁFICA EM APPLETS

Podemos escrever applets com interfaces gráficas. O código abaixo mostra como inserir um *Label* em um applet. Os *labels* são utilizados para "rotularmos" a janela, isto é, inserir textos nas janelas gráficas. No eclipse digite o código abaixo e veja a saída.

```
import java.awt.*;
import java.applet.*;
public class ComponentesBasicos extends
Applet
{
     Button btoOk;
     TextField txtTexto;
     CheckboxGroup grupoRadio;
     Checkbox rd1;
     Checkbox rd2;
     Checkbox checkBox;
     public void init()
     {
        setLayout(null);
        btoOk = new Button("Botão");
```

```
        txtTexto = new TextField("Campo
texto",100);
        grupoRadio = new CheckboxGroup();
        rd1 = new Checkbox("Opção 1",
grupoRadio, false);
        rd2 = new Checkbox("Opção 2",
grupoRadio,true);
        checkBox = new
Checkbox("Opção",false);
        btoOk.setBounds(20,20,100,30);
        txtTexto.setBounds(20,70,100,40);
        rd1.setBounds(20,120,100,30);
        rd2.setBounds(140,120,100,30);
        checkBox.setBounds(20,170,100,30);
        add(btoOk);
        add(txtTexto);
        add(rd1);
        add(rd2);
        add(checkBox);
    }
}
```

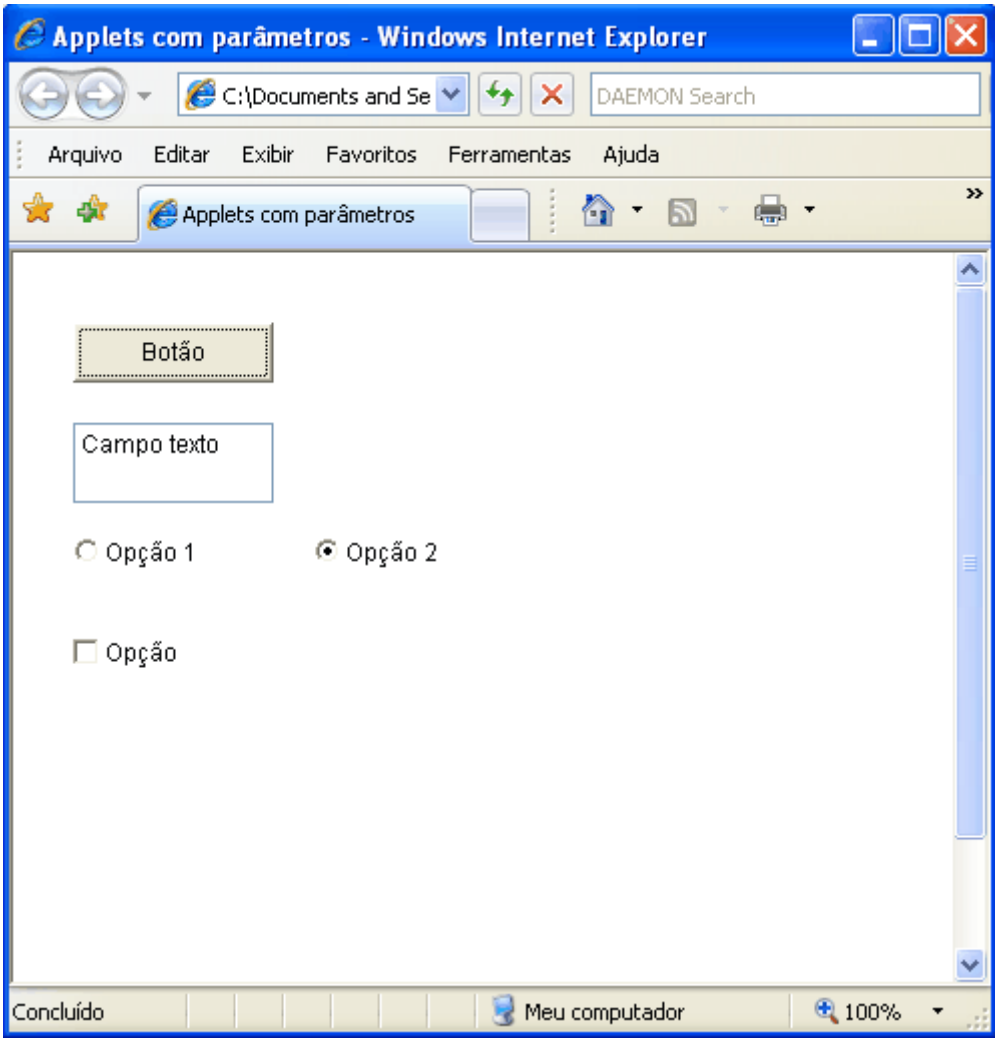
O código acima os componentes básicos da interface gráfica em um applet.

Para inserir o código acima em uma página HTML devemos escrever o código abaixo:

```
<html>
  <head>
  <title>Applets com parâmetros</title>
  </head>
  <body>
     <APPLET CODE =
ComponentesBasicos.class WIDTH = 80 HEIGHT
= 400 MARGIN-LEFT=400>
```

```
    </APPLET>
  </body>
</html>
```

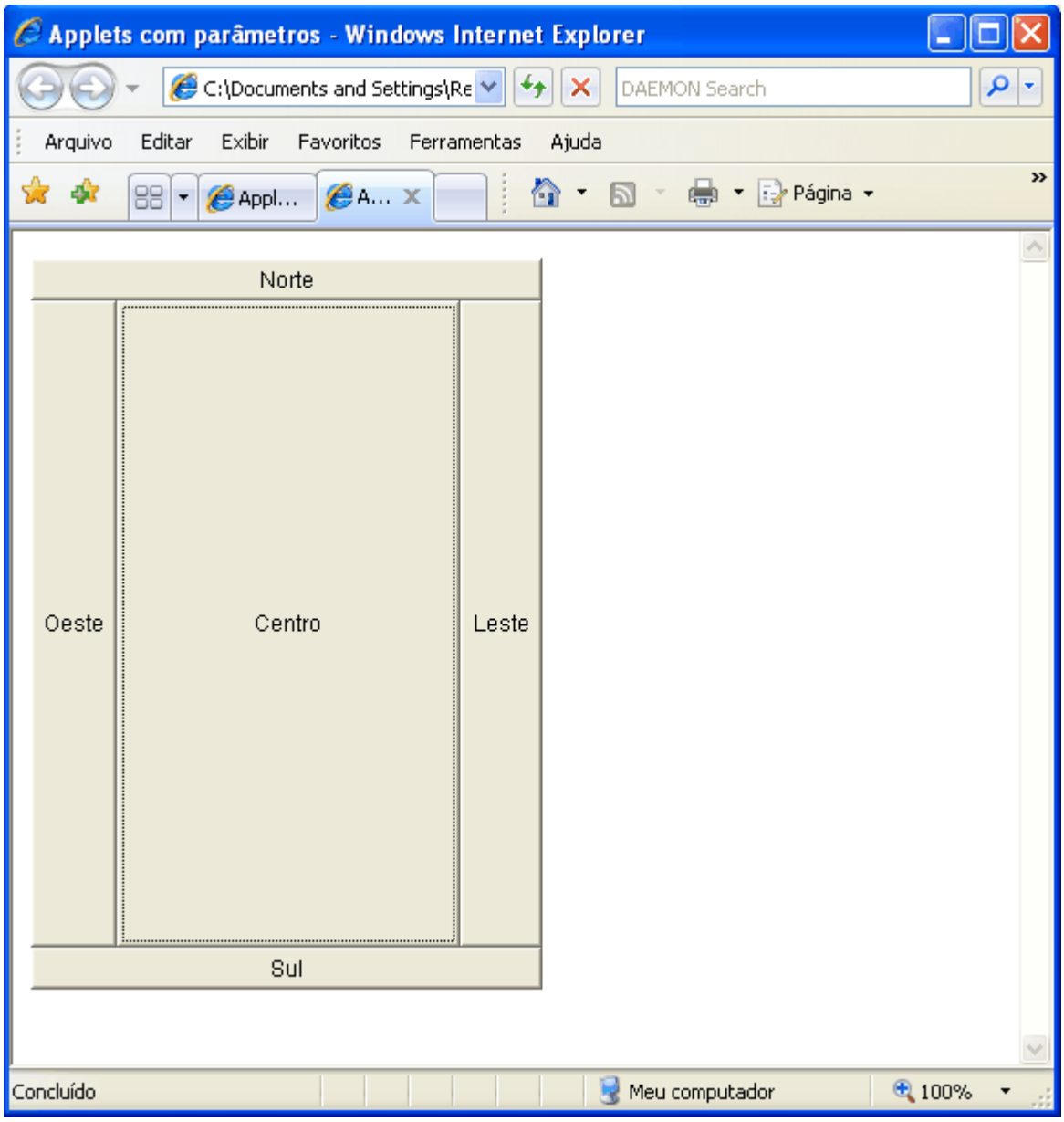
O arquivo HTML irá ficar dentro da pasta bin (onde está o arquivo .class)



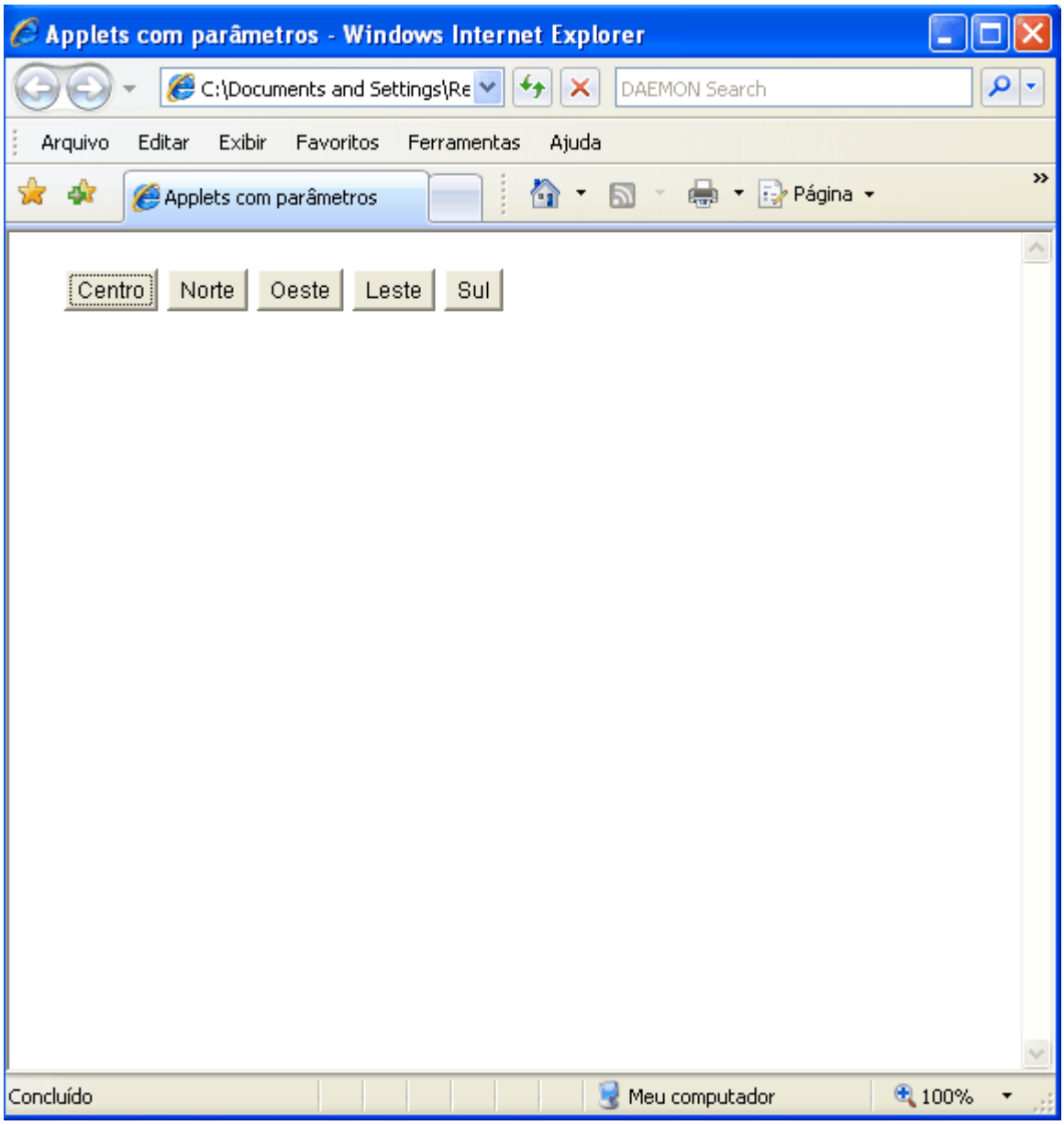
Vamos agora usar os gerenciadores de leiaute em applets

```
import java.awt.*;
import java.applet.*;
public class GerenciadoresLeiauteApplet
extends Applet
    {
```

```
    Button bto1;
    Button bto2;
    Button bto3;
    Button bto4;
    Button bto5;

    public void init()
    {
       setLayout(new BorderLayout());
       bto1 = new Button("Centro");
       bto2 = new Button("Norte");
       bto3 = new Button("Oeste");
       bto4 = new Button("Leste");
       bto5 = new Button("Sul");

       add(bto1,"Center");
       add(bto2,"North");
       add(bto3,"West");
       add(bto4,"East");
       add(bto5,"South");
    }
}
```



Se excluirmos o leiaute para *FlowLayout* teremos a seguinte saída:



4.3 GERENCIADORES DE EVENTOS EM APPLETS

Agora que você já sabe inserir componentes da interface gráfica e os gerenciadores de leiaute em um applet vamos ver agora como iremos tratar os eventos de usuário.

Vamos ver o comportamento dos componentes gráficos básicos e do mouse.

4.3.1 ACTIONLISTENER

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class EventoComponentes extends
Applet implements ActionListener
{
     Button btoMudaCor;
     Button btoOutroBotao;
     TextField txtTexto;
     CheckboxGroup grupoRadio;
     Checkbox rd1;
     Checkbox rd2;
     Checkbox rd3;
     public void init()
     {
          setLayout(new FlowLayout());
          btoMudaCor = new Button("Altere a
cor");
          btoOutroBotao = new
Button("Botão");
          txtTexto = new TextField("Um
texto aqui",35);
          grupoRadio = new CheckboxGroup();
          rd1 = new Checkbox("Vermelho",
grupoRadio,false);
          rd2 = new Checkbox("Azul",
grupoRadio,true);
          rd3 = new Checkbox("Verde",
grupoRadio,false);
          add(btoMudaCor);
          add(btoOutroBotao);
```

```
            add(txtTexto);
            add(rd1);
            add(rd2);
            add(rd3);

btoMudaCor.addActionListener(this);

btoOutroBotao.addActionListener(this);
      }
      public void paint(Graphics g)
      {
            if (rd1.getState())
g.setColor(Color.red);
            else if (rd2.getState())
g.setColor(Color.blue);
            else g.setColor(Color.green);

g.drawString(txtTexto.getText(),20,100);
      }

      public void
actionPerformed(ActionEvent evt)
      {
            if (evt.getSource() ==
btoMudaCor)
                  repaint();

            if (evt.getSource() ==
btoOutroBotao)
            {
                  btoOutroBotao.setLabel("Um
texto aqui");
                  txtTexto.setText("Foi
pressionado outro botão");
```

```
                repaint();
            }
        }
    }
```

Veja que o tratamento de eventos em applets é igual ao tratamento de eventos em aplicações desktop.

4.3.2 MOUSELISTENER

O exemplo abaixo mostra um retângulo na cor azul na janela do navegador. Quando o botão esquerdo do mouse é pressionado dentro do retângulo então ele mostra as coordenadas x e y do mouse e escreve que o mouse foi clicado dentro do retângulo. Caso o botão esquerdo do mouse seja pressionado fora do retângulo é mostrado também as coordenadas x e y do mouse e um texto dizendo que o mouse foi pressionado fora do retângulo. Digite o código abaixo e execute-o para ver os resultados.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class EventosDeMouse extends Applet implements
MouseListener
{
    int x;
    int y;
    int
xRetangulo,yRetangulo,larguraRetangulo,alturaRetangulo
;
    boolean mouseDentroDoRetangulo;
    boolean clicadoDentroDoRetangulo;

    public void init()
```

```
        {
                xRetangulo = 20;
                yRetangulo = 20;
                larguraRetangulo = 300;
                alturaRetangulo = 200;
                addMouseListener(this);
        }

        public void paint(Graphics g)
        {
                g.setColor(Color.blue);

g.fillRect(xRetangulo,yRetangulo,larguraRetangulo,altu
raRetangulo);
                g.setColor(Color.white);
                g.drawString("("+x+","+y+")",x,y);

                if
(clicadoDentroDoRetangulo)g.drawString("Seu clique foi
dentro do retângulo",20,220);
                else g.drawString("Seu clique foi fora do
retângulo",20,120);

                if (mouseDentroDoRetangulo) g.drawString("O
mouse está dentro do applet",20,160);
                else g.drawString("O mouse está fora do
applet",20,160);
        }

        public void mouseClicked (MouseEvent e) {
         x = e.getX();
         y = e.getY();
         if (x > xRetangulo && x <
xRetangulo+larguraRetangulo && y >yRetangulo &&  y <
```

```
yRetangulo+alturaRetangulo)  clicadoDentroDoRetangulo
= true;
      else
           clicadoDentroDoRetangulo = false;
      repaint();
     }

     public void mousePressed (MouseEvent me) {}

     public void mouseReleased (MouseEvent me) {}

     public void mouseEntered (MouseEvent me) {
      mouseDentroDoRetangulo = true;
      repaint();
     }
     public void mouseExited (MouseEvent me) {
      mouseDentroDoRetangulo = false;
      repaint();
     }
}
```

Os métodos abaixo deverão sempre estar presentes quando você implementa a interface MouseListener

```
public void mouseClicked (MouseEvent me) {}
public void mouseEntered (MouseEvent me) {}
public void mousePressed (MouseEvent me) {}
public void mouseReleased (MouseEvent me) {}
public void mouseExited (MouseEvent me) {}
```

4.3.3 ABRINDO URL

Este applet permite que o usuário digite um endereço web e pressione o botão abrir para que uma nova aba seja aberta no nagegador.

```
import java.awt.*;
import java.net.*;
import java.applet.*;
import java.awt.event.*;
public class AbrePaginaWebApplet extends Applet
implements ActionListener
{
     TextField campoURL;
     Button btoAcesso;
     boolean erroURL;
     URL usuarioURL;
     public void init()
     {
          setLayout(new FlowLayout());
          campoURL = new TextField("www.sabe.br");
          btoAcesso = new Button("Abrir");
          campoURL.addActionListener(this);
          btoAcesso.addActionListener(this);
          add(campoURL);
          add(btoAcesso);
     }
     public void paint(Graphics g)
     {
          if (!erroURL)
               g.drawString("Escreva o endereço que
quer abrir",20,80);
          else
          {
               g.drawString("URL informada:
"+usuarioURL,20,80);
               g.drawString("URL inválida",20,100);
          }
     }
```

```
        public void actionPerformed(ActionEvent act)
        {
              erroURL = false;
             String str = campoURL.getText();

             if (str.length() > 6)
             {
               if
(!str.substring(0,7).toUpperCase().equals("HTTP://"))
                  str = "Http://" + str;
             }
             else str = "Http://" + str;

             if (str.indexOf(".com") == -1)
                  erroURL = true;

             try {
                  usuarioURL = new URL(str);
             }

             catch (Exception e)
             {
                  erroURL = true;
             }
             campoURL.setText(usuarioURL.toString());
             if (!erroURL)


getAppletContext().showDocument(usuarioURL,"_blank");
              repaint();
        }

}
```

Conclusão

As interfaces gráficas facilitam a utilização do software pelo usuário

Para escrever um Applet devemos importar a classe applet que está no pacote java

Os Applets são executados em browsers e hospedados na internet.

Os Applets possuem um ciclo de vida.

Os métodos de inicialização de um applet (init()) é diferente do método de inicialização de uma aplicação (main()).

As interfaces gráficas utilizadas nos applets são iguais as usadas em aplicativos desktop desenvolvidos em Java.

Os gerenciadores de evento em Applets são iguais aos usados em aplicações desktop.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] – Deitel, H.M., Java, como programar, 4.ed. – Porto Alegre: Bookman, 2003.

[2] – Sierra, K., Bates, B., Certificação Sun para progrmador java 5: Guia de estudo, 4.ed. – Rio de Janeiro: AltaBooks, 2006.

[3] – Puga, S., Rosseti, G. Lógica de programação e estrutura de dados, com aplicações em Java, São Paulo: Prentice Hall, 2003.

[4] – Barnes, D.J., Kölling, M., Programação orientada a objetos com java: Uma introdução utilizando o Blue J., Sã o Paulo: Pearson Prentice Hall, 2004.

[5] – Thompson, M.A., Java 2 & banco de dados: aprenda na pratica a usar java e SQL para acessar banco de dados relacionais., São Paulo: +rica, 2002.

[6] – Keogh, J., Giannini, M., OOP Desmystified, McGraw-Hill/Osborne, 2004.

[7] – Leopoldino, F. L., Instalando o J2SE 5.0 JDK no Windows 2000/XP, site na internet no endereço: http://www.guj.com.br/content/articles/installation/j2sdkinstall.pdf acessado em 16/06/2007.

[8] – Wikipédia, site na internet acessado em 16/06/2007 http://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos

SINTES, Anthony. Aprenda Programação Orientada a Objetos em 21 dias: , . São Paulo: Makron Books, 2002.

SANTOS, Rafael. Introdução à programação orientada a objetos usando Java: Editora Campus, 2003.

HORSTMANN, Cay S.; CORNELL, Gary. Core Java 2: Fundamentos - Vol. 1. 7 ed. Alta Books, 2005.

Caelum: Ensino e Soluções em Java, site na internet no endereço: www.caelum.com.br, acessado em 19 de Dezembro de 2007.