

# Programação para Web



Prof. Dr. Sérgio Donizetti Zorzo  
Universidade Federal de São Carlos



# Desenvolvimento Web

---

Conjunto de atividades relacionadas ao desenvolvimento de aplicações que se utilizam da World Wide Web (www) ou uma intranet, como:

- Desenvolvimento de regras para comércio eletrônico;
- Layout de sites;
- Conteúdo;
- Codificação no Cliente e no Servidor;
- Configuração de Servidor Web;
- Banco de Dados.

Para programadores web, o termo desenvolvimento web refere-se apenas aos aspectos de construção de sites dinâmicos



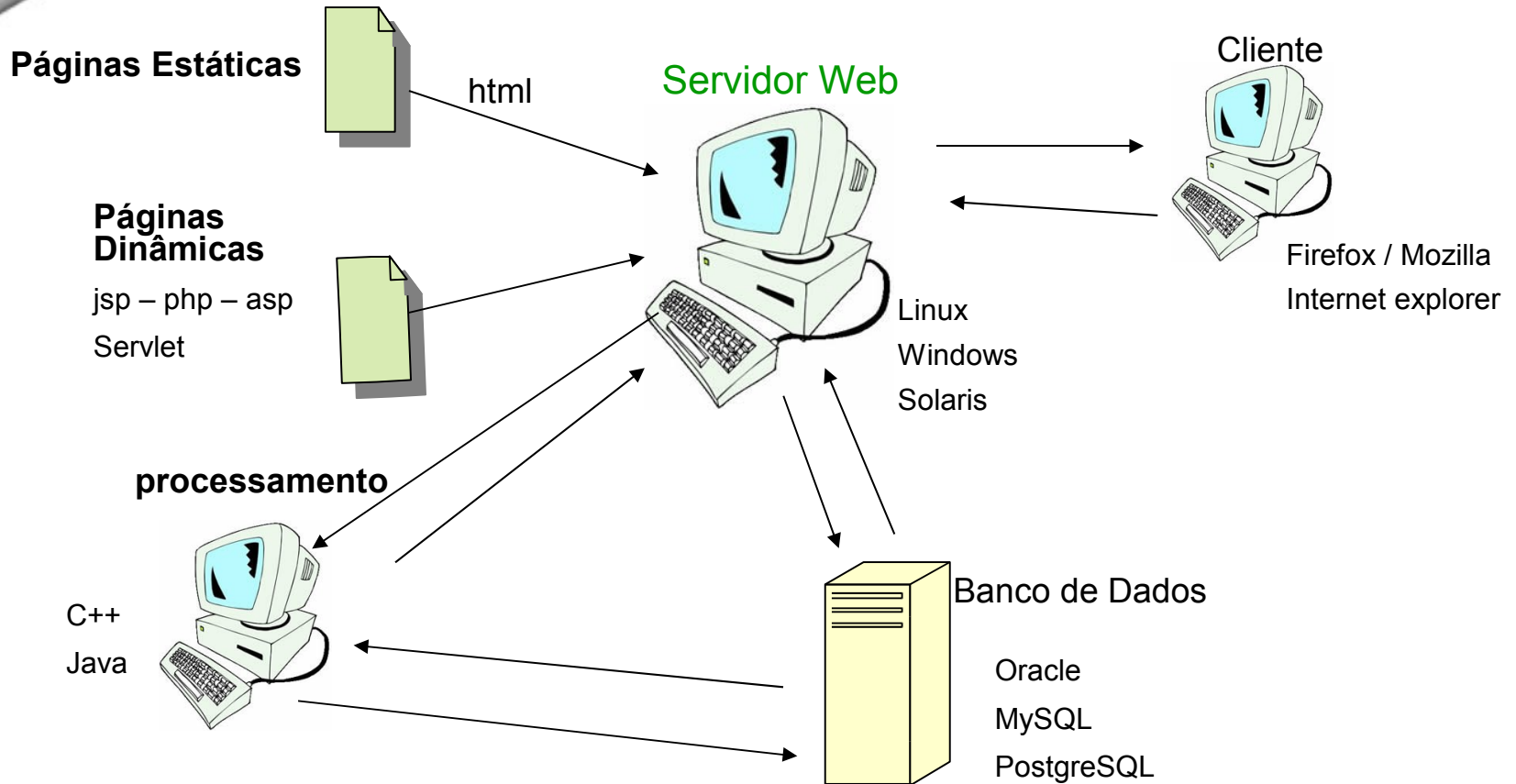
# Desenvolvimento Web

---

- Ambiente de Desenvolvimento
- Ambiente de Utilização do Sistema
- Arquitetura do Sistema
- Tecnologia utilizada
- Ferramentas



# Arquitetura de uma Aplicação Web





# Tecnologia Cliente-Servidor

---

## CLIENTE – CSB/ Client Side Building

Processamento no cliente. Consome tempo de carga.

Limitação de processamento no cliente.

Linguagem dependente de navegador.

Linguagens disponíveis: JavaScript, Applets.

## SERVIDOR – SSB/ Server Side Building

Processamento no servidor. Retorna ao cliente páginas HTML.

Capacidade maior para armazenamento e processamento.

Segurança para manipulação de informações (no servidor).

Linguagens disponíveis: JSP, Servlets, PHP, PERL-CGI, ASP.

# Padrão para o desenvolvimento de uma aplicação Web



Desenvolvimento do Sistema em 3 Camadas (MVC – Model / View / Control):

## **Camada de apresentação**

recebe informações do usuário, envia aos componentes de negócio para o processamento, recebe os resultados fornecidos pelos componentes de negócio, e apresenta-os ao usuário.

## **Camada de lógica de negócio**

O processamento é especificado e as regras de negócios mantidas. Regras de negócios realizam a validação de dados, verificação de logins, busca em banco de dados e algoritmos de transformação.

Une a camada de apresentação com as camadas de dados.

## **Camada de acesso a dados**

permite o acesso a uma variedade de fontes de informações, incluindo dados relacionais e não relacionais, e uma interface de programação fácil de usar que é independente de ferramenta e de linguagem.



# Configurações do Ambiente de Desenvolvimento

---

## **Sistema Operacional**

Linux, Windows, Solaris

## **Compiladores**

JDK, PHP


## **Servidor Web**

Apache, Tomcat, JBoss, Glassfish

## **SGBD (SQL)**

MySQL, SQL Server, PostgreSQL

# Formulários - comunicação entre cliente e servidor



```
<FORM name=" " action=" " method=" " enctype=" ">  
  conteúdo do formulário . . .  
</FORM>
```

name: identificador do formulário –

action: nome do script que receberá os dados do formulário no servidor

method: método de envio dos dados – get ou post

Enctype: formato envio dos dados – default é urlencoded

Existem vários tipos de campos de entrada no formulário, como:

- campos de entrada de texto
- menus de múltipla escolha ou escolha única
- botões sim-ou-não
- botões para submissão ou limpeza de formulário





# Formulários - métodos de comunicação: GET e POST

**GET:** informações do formulário são enviadas junto e ao final da URL ativada.

**POST:** informações do formulário são enviadas após a ativação da URL, sendo que o servidor entende que novas informações serão enviadas pelo cliente.

As informações do formulário serão tratadas na URL referenciada pela tag **ACTION**, que será responsável por responder a solicitação do formulário.

*ACTION="/cgi-bin/post-query"*

*ACTION="http://www.dc.ufscar.br/~zorzo/resp.jsp"*

O formulário:

```
<FORM ACTION= "exemplo.jsp" METHOD= "GET" >  
  Marcação de campos de entrada e HTML em geral  
</FORM>
```

Será ativado por:

*http://localhost/diretorioatual/exemplo.jsp?campo1=valor1&campo2=valor2*



# Formulários -

## *<input>* define elemento do formulário

- **Campo de Texto**

- `<input type= " text " name= "tx1" value= " " maxlength= " " >`
- exibe na tela um campo para entrada de texto com apenas uma linha
- *value* – valor pré-definido que será exibido na visualização
- *size* – tamanho (em caracteres) do elemento na tela
- *maxlength* – tamanho máximo do texto (em caracteres)

- **Campo de Texto com Máscara**

- `<input type="password" name="ps1" value="" maxlength="">`
- semelhante ao anterior, mas os caracteres digitados serão trocados por \*

- **Checkbox**

- `<input type="checkbox" name="ch1" value="valor" checked >`
- utilizado para campos de múltipla escolha
- *value* – valor que será enviado ao servidor quando o formulário for submetido, no caso de estar marcado
- *checked* – (estado inicial) quando presente, o elemento já aparece marcado



# Formulários -

## *<input>* define elemento do formulário

- **Radio Button**

- `<input type="radio" name="" value="" checked>`
- utilizado para campos de múltipla escolha, podendo marcar apenas uma opção
- *value* – valor que será enviado ao servidor quando o formulário for submetido, no caso de estar marcado
- *checked* – estado inicial – quando presente, o elemento já aparece marcado  
=> lembre-se que todos os radio button devem ter o mesmo name

- **Submit Button**

- `<input type="submit" name="" value="" >`
- utilizado para enviar os dados do formulário
- *value* – texto que irá aparecer no botão de envio

- **Reset Button**

- `<input type="reset" name="" value="" >`
- utilizado para fazer todos do formulário retornar ao valor original – usado para "limpar" a página contendo o formulário parcialmente preenchido.
- *value* – texto que irá aparecer no botão de reset/limpeza.



# Formulários - <textarea> e <select>

- **TextArea**

- `<textarea cols="" rows="" name="" wrap="" > texto </textarea>`
- exibe na tela uma caixa de texto, com o tamanho definido em cols e rows
- *wrap* – como tratar as quebras de linha: "soft", "hard" ou "off"- quebra apenas na tela, quebra na tela e no servidor, e sem quebra de linha.

- **Select**

- `<select name="" size="" multiple>  
    <option value=""> texto </option>  
</select>`
- exibe na tela uma lista de seleção – se size for 1 e não houver o parâmetro multiple –exibe uma tela um "combo box".
- *size* – número de linhas exibidas – default = 1
- *multiple* – permite a seleção de dois ou mais itens (opcional)
- *option* – cada item option adiciona um item a lista para seleção
- *value* – valor a ser enviado ao servidor se aquele elemento for selecionado (default=text).
- *text* – valor a ser exibido para o item – é o texto entre <option> e </option>



# Formulário

```
<FORM METHOD = "POST" ACTION= "RESP.JSP" >
  Nome: <input type="TEXT" name="NOME" size=40> <p>
  Sexo: <input type="RADIO" name="SEXO" value="MASC"> masc
        <input type="RADIO" name="SEXO" value="FEM"> fem
  <p> Esportes de Interesse:
  <input type="CHECKBOX" name="fut" value="X"> FUTEBOL
  <input type="CHECKBOX" name="vol" value="X"> VOLEIBOL
  <input type="CHECKBOX" name="bas" value="X"> BASQUETEBOL
  <p><TEXTAREA name="sugestoes" rows=4 cols=64>
    Digite as suas sugestoes:
  </TEXTAREA>
  <p>Sistema Utilizado
  <SELECT name="SISTEMA" size=1>
    <option value="w98"> Windows 98
    <option value="unx"> Unix
  </SELECT>
  <input type="RESET" value="Limpar Campos">
  <input type="SUBMIT" value="Enviar Formulario">
</FORM>
```



# Linguagem Java

- Uma linguagem de Programação Orientada a Objeto
- Independente de Plataforma
- Não utiliza ponteiros
- Tem alocação dinâmica de memória em tempo de execução
- Utiliza *Multithreading*
- Mesma sintaxe usada em diferentes tecnológicas

Alguns usuários Java:

Banco do Brasil, CEF, SERPRO, Banco Central, Banco Itaú  
Hong Kong Telecom (rede de TV interativa)  
J. P. Morgan (banco de investimentos)  
Fannie Mac (maior empresa de hipotecas dos EUA )  
CSX (maior companhia ferroviária dos EUA)

# Produtos para Desenvolvimento de Aplicações com a plataforma Java



**JSDK ou JDK:** kit de desenvolvimento de software Java; contém , dentre outros, os aplicativos:

- javac* - compilador do código Java,
- javadoc* – gera documentação (html),
- appletviewer* -visualizador de applets.
- java* – máquina virtual Java

A versão 1.5 (*Java Tiger*) traz novas construções sintáticas para a linguagem (atual – *Java Mustang* 1.6.0\_05)

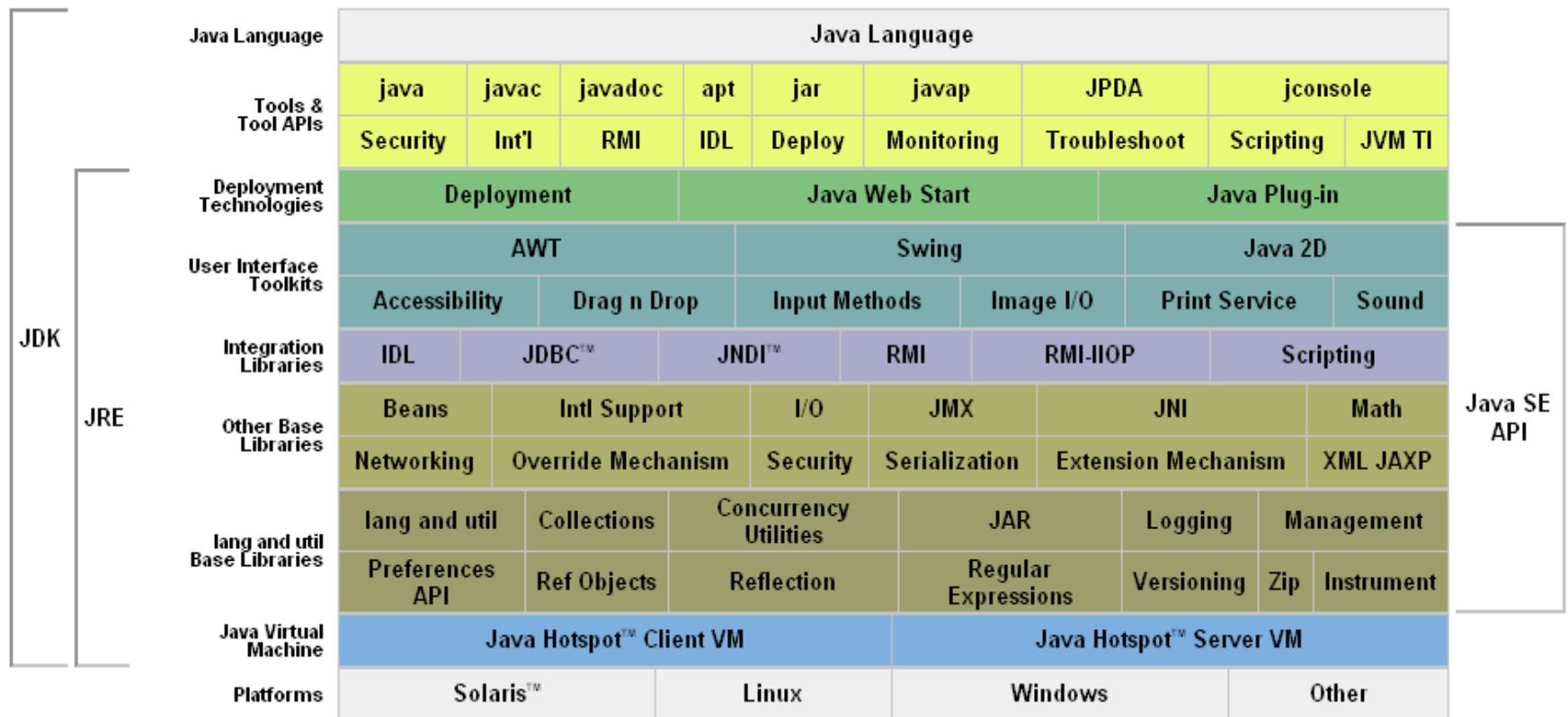
**JVM:** *Java Virtual Machine* - responsável pela interpretação do Código Intermediário Java (*bytecode*)

**JRE:** *Java Runtime Environment* - Identifica o ambiente de execução Java existente. Inclui a Máquina Virtual Java e as classes que formam a plataforma Java

# Plataforma Java SE 6



## Java™ SE 6 Platform at a Glance





# Programação para Web



Instalação do software  
JDK e NetBeans

# Java SDK

- 
- Disponível para download no site:

```
http://java.sun.com/javase/downloads/  
http://www.comp.ufscar.br/latosensu/
```

- Última versão disponível
  - JDK6 *update* 6
  - Disponível para Windows, Linux e Solaris

# Java SDK

- Download do JDK6 para Linux

`jdk-6u6-linux-i586-rpm.bin - 63.51 MB`

e salve o arquivo na pasta /tmp

- Atribua permissão de execução:

```
[root@aula]# cd /tmp  
[root@aula tmp]# chmod +x jdk-6u6-linux-i586-rpm.bin
```

# Java SDK

- Execute o arquivo de instalação

```
[root@aula]# ./jdk-6u6-linux-i586-rpm.bin
```

```
Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA SE DEVELOPMENT KIT (JDK), VERSION 6

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE
SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION
THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY
CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS
(COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT
CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU
ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY
SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE
AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE
TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE
AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT
CONTINUE.

1. DEFINITIONS. "Software" means the identified above in
binary form, any other machine readable materials
(including, but not limited to, libraries, source files,
header files, and data files), any updates or error
corrections provided by Sun, and any user manuals,
programming guides and other documentation provided to you
--Mais--
```

# Java SDK

- Digite `yes` para aceitar os termos de licença do JDK

for the JAVA SE DEVELOPMENT KIT (JDK), VERSION 6


SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.

1. DEFINITIONS. "Software" means the identified above in binary form, any other machine readable materials (including, but not limited to, libraries, source files, header files, and data files), any updates or error corrections provided by Sun, and any user manuals, programming guides and other documentation provided to you

Do you agree to the above license terms? [yes or no]

yes ☐

# Java SDK


- 
- Após o término da instalação, o JDK foi instalado na pasta:

```
/usr/java/jdk1.6.0_06
```

- Foi criado um link simbólico para pasta do JDK em:


```
/usr/java/latest
```

# Java SDK

- 
- É necessário inserir as variáveis de ambiente em um arquivo dentro da pasta `/etc/profile.d/`:

```
[root@aula]# cd /etc/profile.d/  
[root@aula]# gedit jdkconfig.sh
```

# Java SDK

- 
- Insira as linhas no arquivo criado `jdkconfig.sh`:

```
JAVA_HOME=/usr/java/latest  
PATH=$JAVA_HOME/bin:$PATH  
CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib  
export JAVA_HOME PATH CLASSPATH
```



# Java SDK

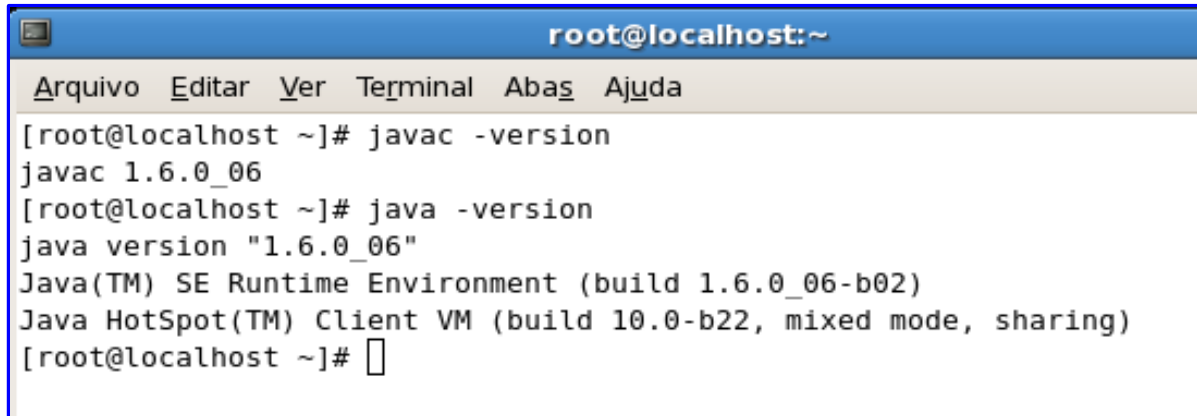
- Atribua permissão de execução:

```
[root@aula]# cd /etc/profile.d/  
[root@aula]# chmod +x jdkconfig.sh
```

# Java SDK

- Para testar, digite:

```
[root@aula]# java -version  
[root@aula]# javac -version
```

A terminal window titled 'root@localhost:~' with a menu bar containing 'Arquivo', 'Editar', 'Ver', 'Terminal', 'Abas', and 'Ajuda'. The terminal shows the execution of 'javac -version' and 'java -version' commands. The output for 'javac -version' is 'javac 1.6.0\_06'. The output for 'java -version' is 'java version "1.6.0\_06"', 'Java(TM) SE Runtime Environment (build 1.6.0\_06-b02)', and 'Java HotSpot(TM) Client VM (build 10.0-b22, mixed mode, sharing)'. The prompt '[root@localhost ~]#' is followed by a cursor.

```
root@localhost:~  
Arquivo  Editar  Ver  Terminal  Abas  Ajuda  
[root@localhost ~]# javac -version  
javac 1.6.0_06  
[root@localhost ~]# java -version  
java version "1.6.0_06"  
Java(TM) SE Runtime Environment (build 1.6.0_06-b02)  
Java HotSpot(TM) Client VM (build 10.0-b22, mixed mode, sharing)  
[root@localhost ~]#
```

# NetBeans

- 
- Disponível para download no site:

```
http://www.netbeans.info/downloads/  
http://www.comp.ufscar.br/latosensu/
```

- Última versão disponível
  - NetBeans 6.1
  - Disponível para Windows, Linux, MacOS e Solaris

# NetBeans

- 
- Download do NetBeans para Linux

**netbeans-6.1-linux.sh - 178.1 MB**

e salve o arquivo na pasta /tmp

Atribua permissão de execução:

```
[root@aula]# cd /tmp  
[root@aula]# chmod +x netbeans-6.1-linux.sh
```

# NetBeans

- Execute o arquivo de instalação

```
[root@aula]# ./netbeans-6.1-linux.sh
```

```
[root@localhost opt]# chmod +x netbeans-6.1-linux.sh
[root@localhost opt]# ./netbeans-6.1-linux.sh
Configuring the installer...
Searching for JVM on the system...
Extracting installation data...
Running the installer wizard...
□
```

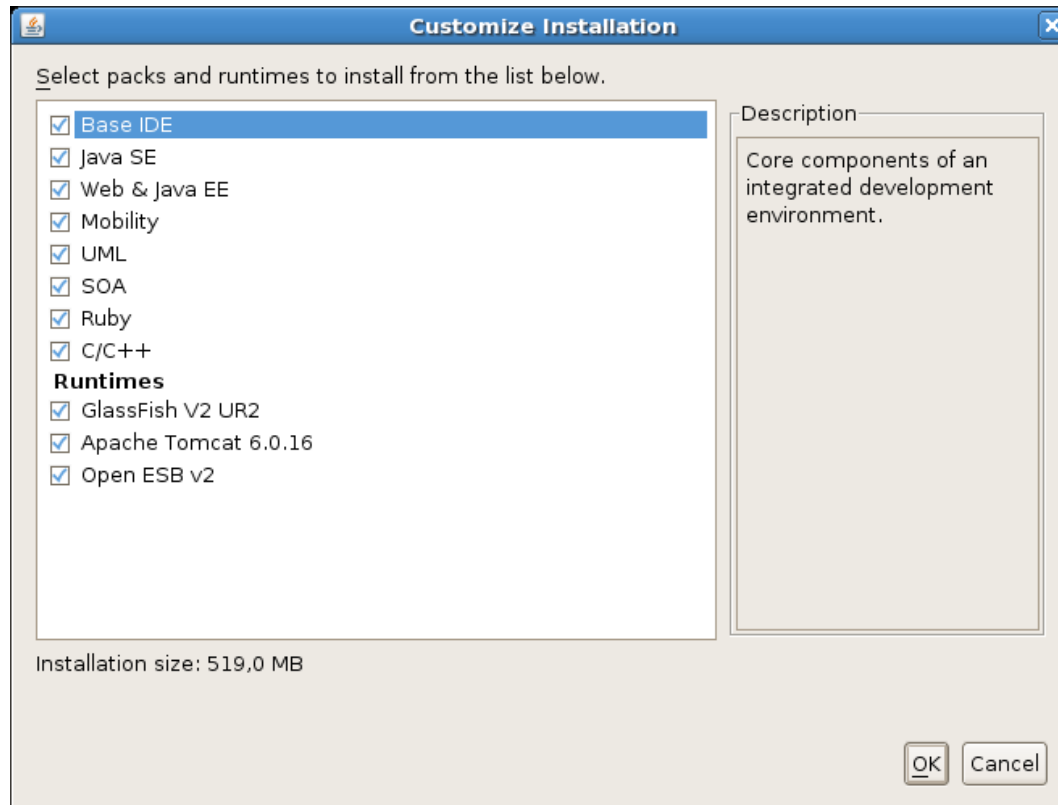
# NetBeans

## ■ Tela inicial da instalação



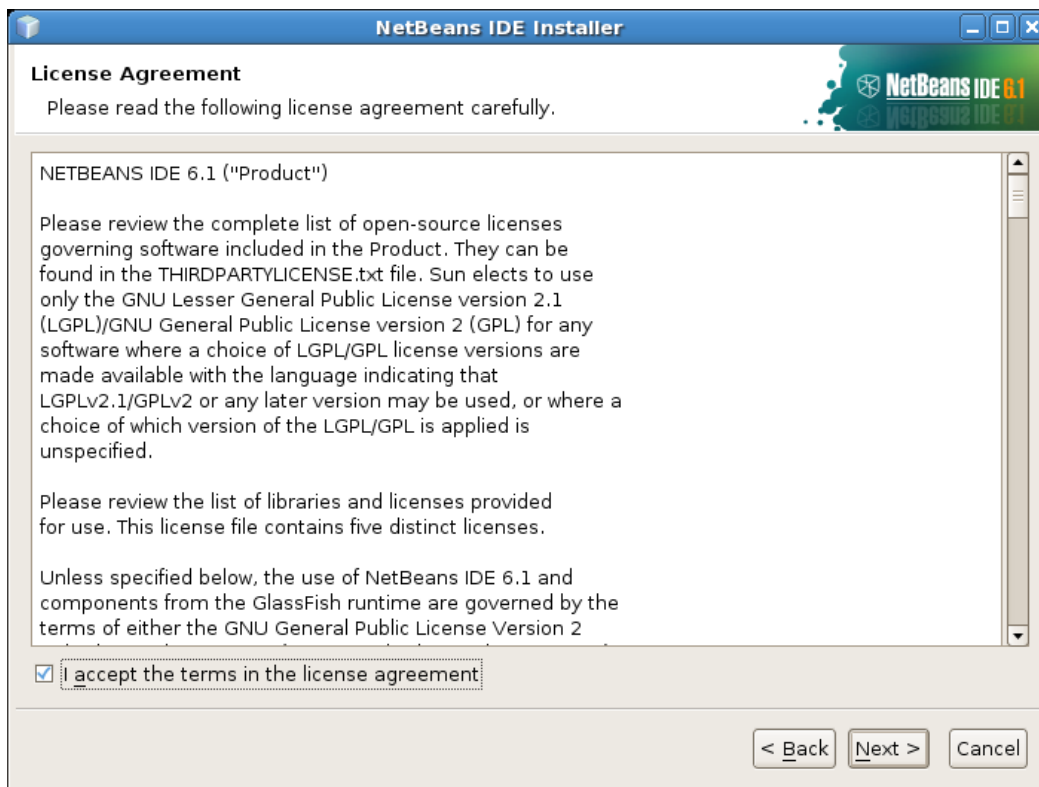
# NetBeans

## ■ Personalizando a instalação



# NetBeans

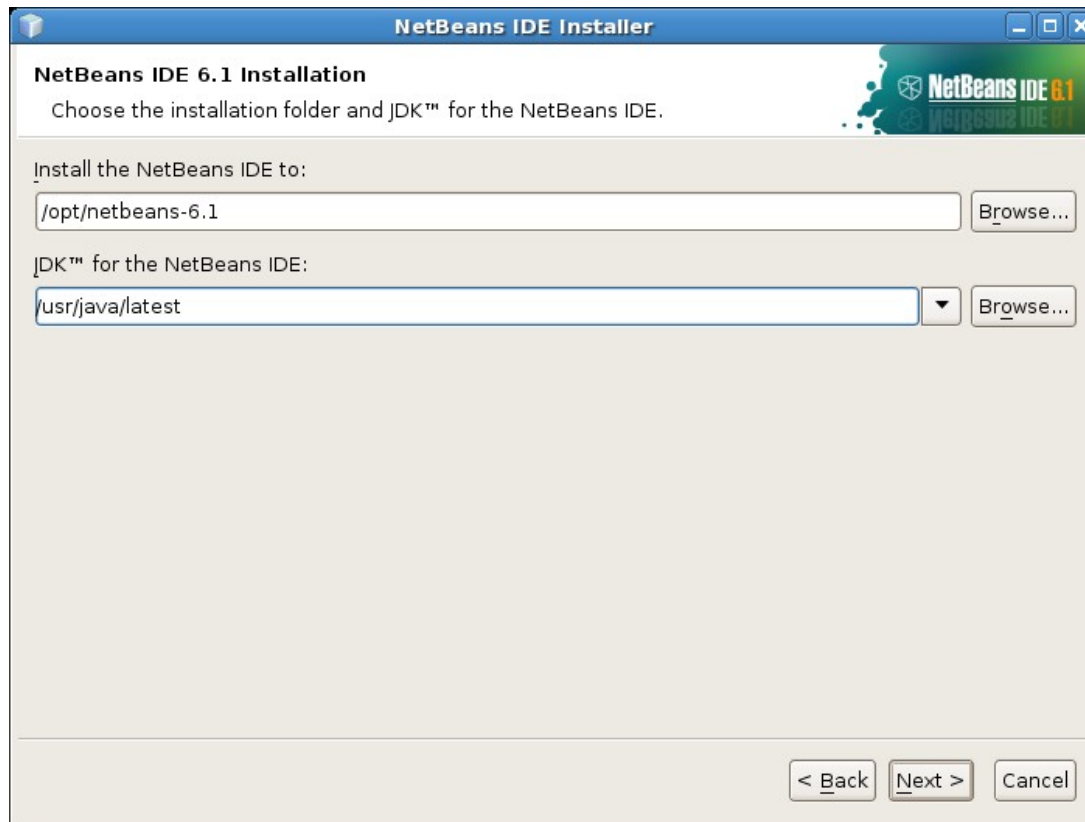
## ■ Contrato de licença





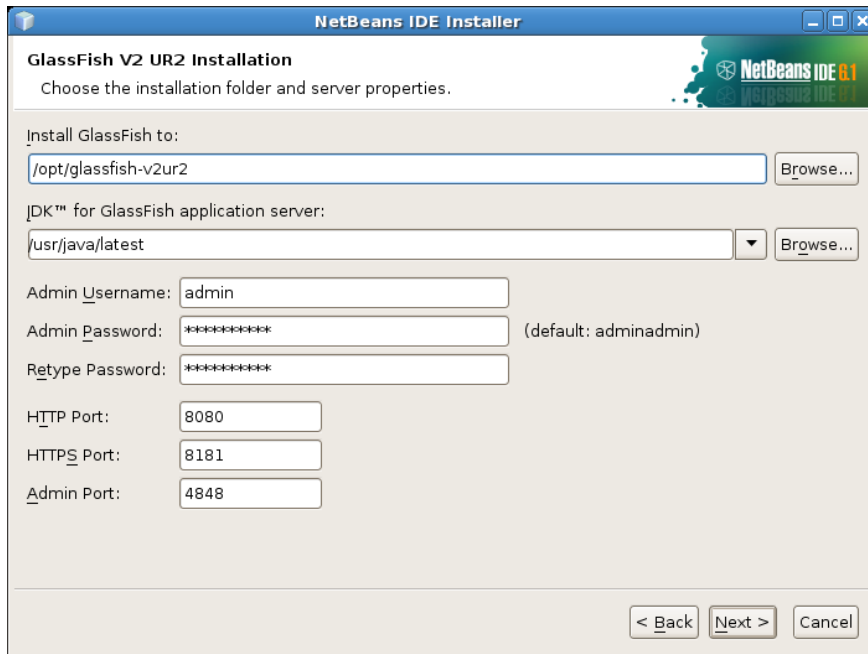
# NetBeans

- Diretório de instalação e seleção do JDK



# NetBeans

## ■ Servidores Glassfish e Tomcat



NetBeans IDE Installer

**GlassFish V2 UR2 Installation**  
Choose the installation folder and server properties.

Install GlassFish to:

JDK™ for GlassFish application server:

Admin Username:

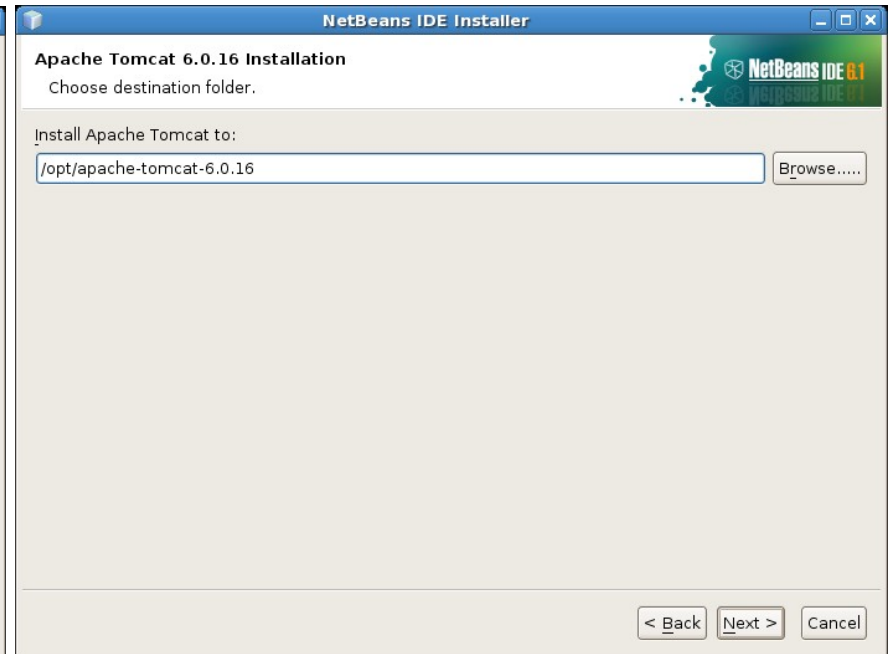
Admin Password:  (default: adminadmin)

Retype Password:

HTTP Port:

HTTPS Port:

Admin Port:



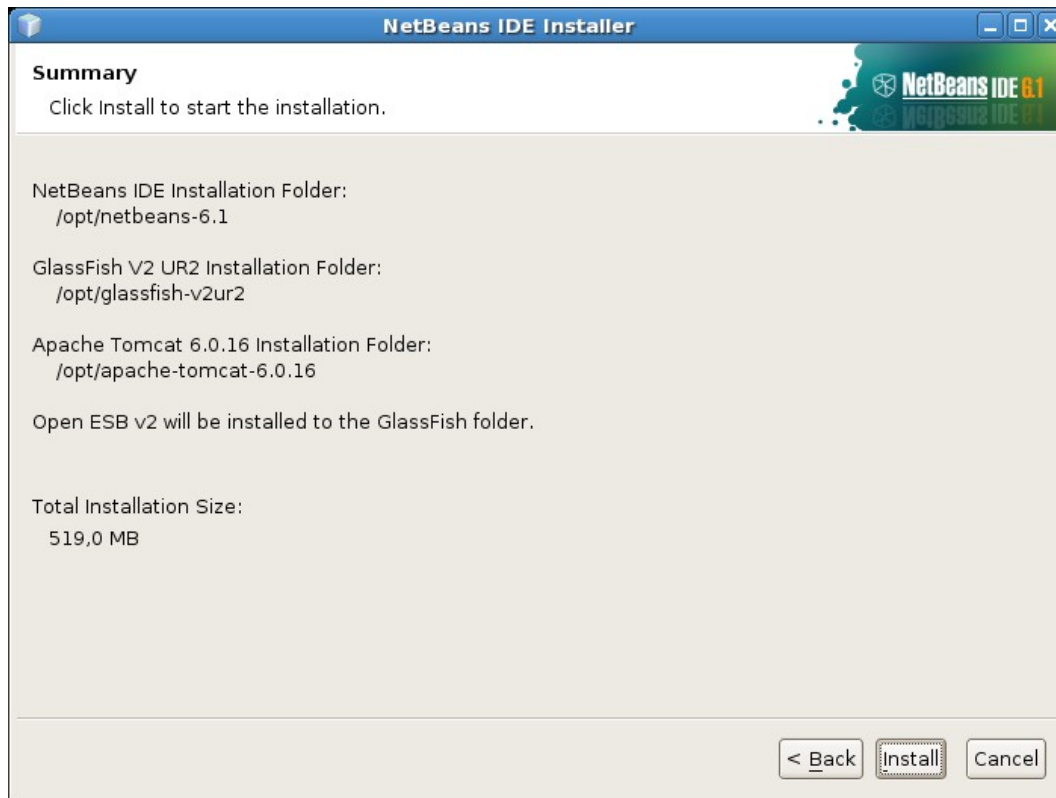
NetBeans IDE Installer

**Apache Tomcat 6.0.16 Installation**  
Choose destination folder.

Install Apache Tomcat to:

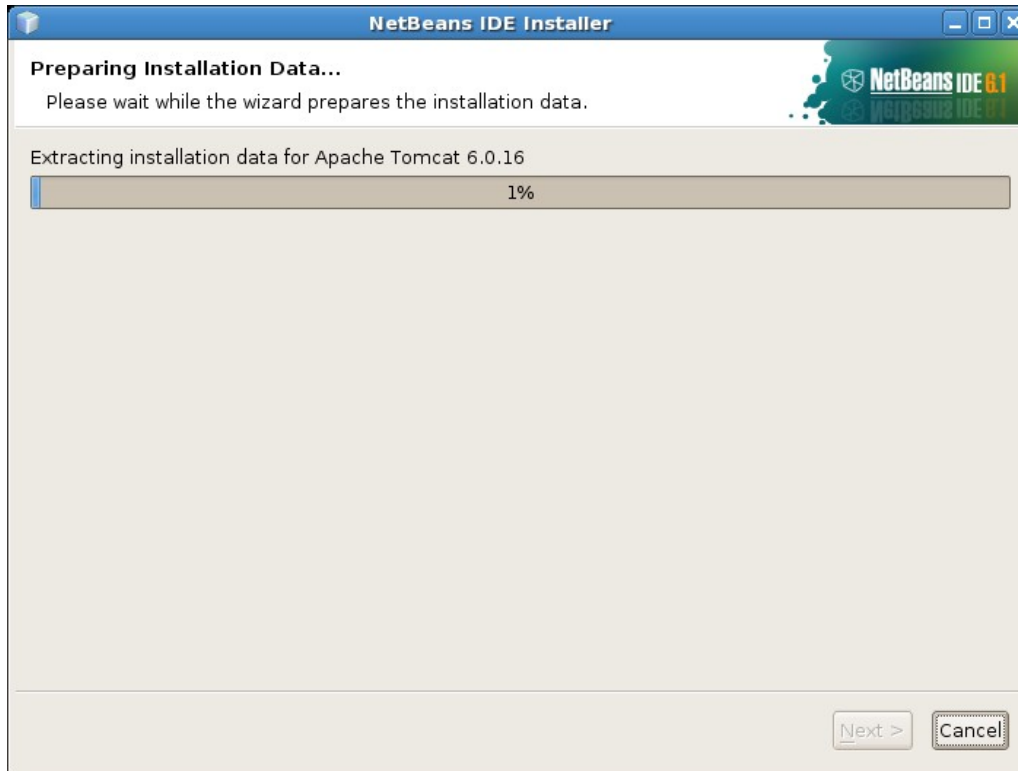
# NetBeans

## ■ Resumo da instalação



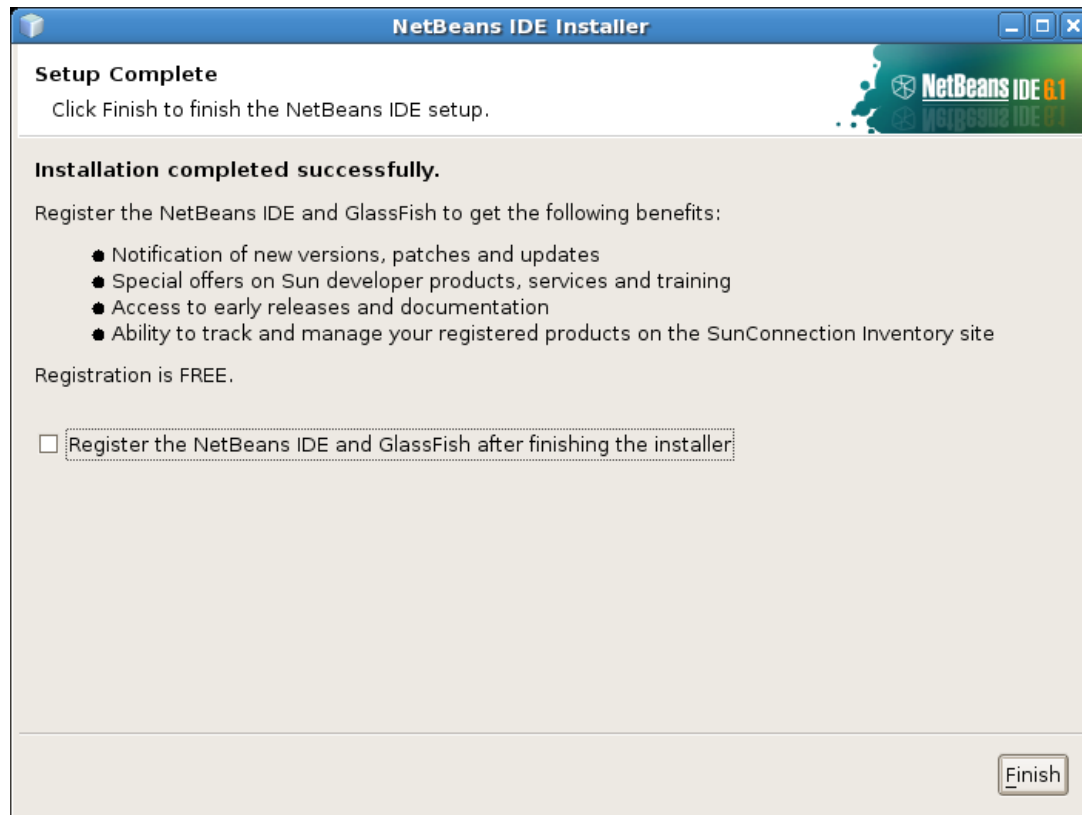
# NetBeans

## ■ Instalando...



# NetBeans

## ■ Fim da instalação



# NetBeans

- O NetBeans foi instalado na pasta:

```
/opt/netbeans-6.1
```

- Executando o NetBeans :

```
[root@aula]# cd /opt/netbeans-6.1/bin/  
[root@aula]# ./netbeans &
```



# IDE NetBeans

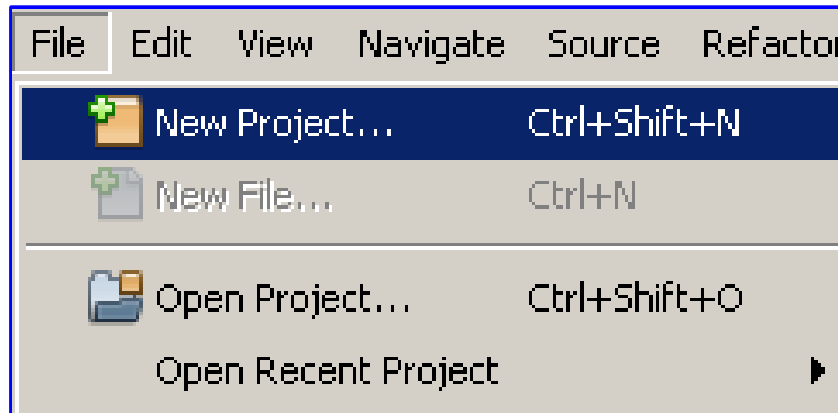
---

IDE - Integrated Development Environment  
NetBeans



# NetBeans

- Criando o primeiro projeto

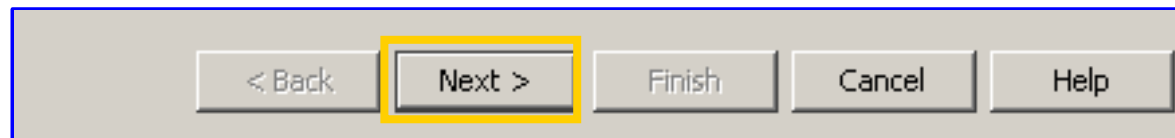
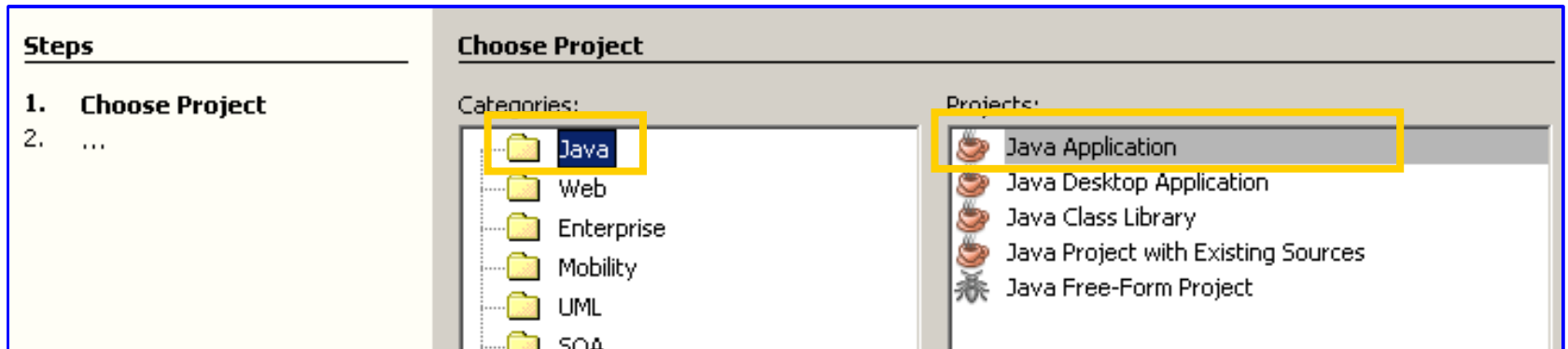






# NetBeans

- Criando o primeiro projeto





# NetBeans

- Criando o primeiro projeto

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

☒ Set as Main Project



# NetBeans

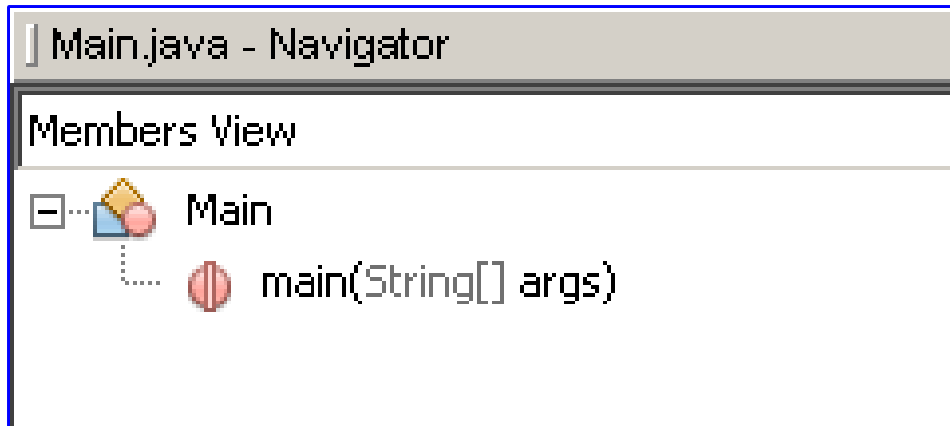
- Janela de estrutura do projeto





# NetBeans

- Janela do navegador de classes





# NetBeans

- Janela de edição



# NetBeans

- Primeiro projeto – Olá Mundo

```
package meuprimeiroprojeto;  
  
public class Main {  
  
    public Main() {  
    }  
  
    public static void main (String[] args) {  
        System.out.println("Ola Mundo!");  
    }  
  
}
```



# NetBeans

- Primeiro projeto – Olá Mundo

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package meuprimeiroprojeto;

/**
 *
 * @author Paulo
 */
public class Main {

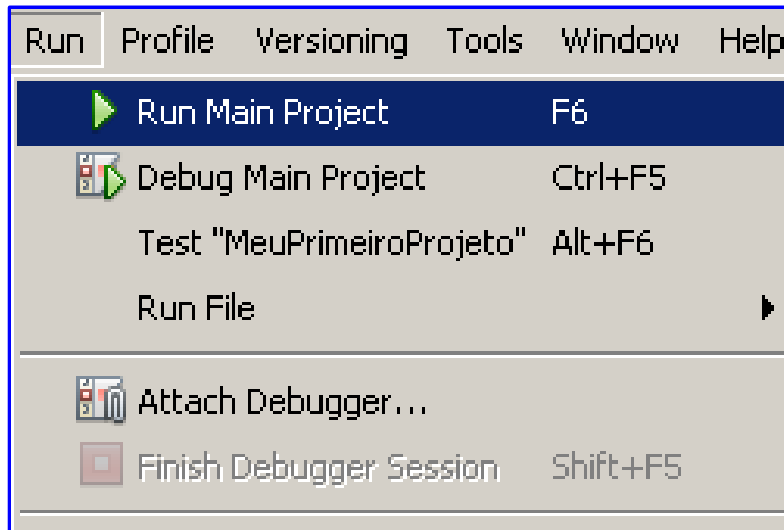
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Ola Mundo!");
    }

}
```



# NetBeans

- Executando o projeto







# NetBeans

- Visualizando a execução do projeto

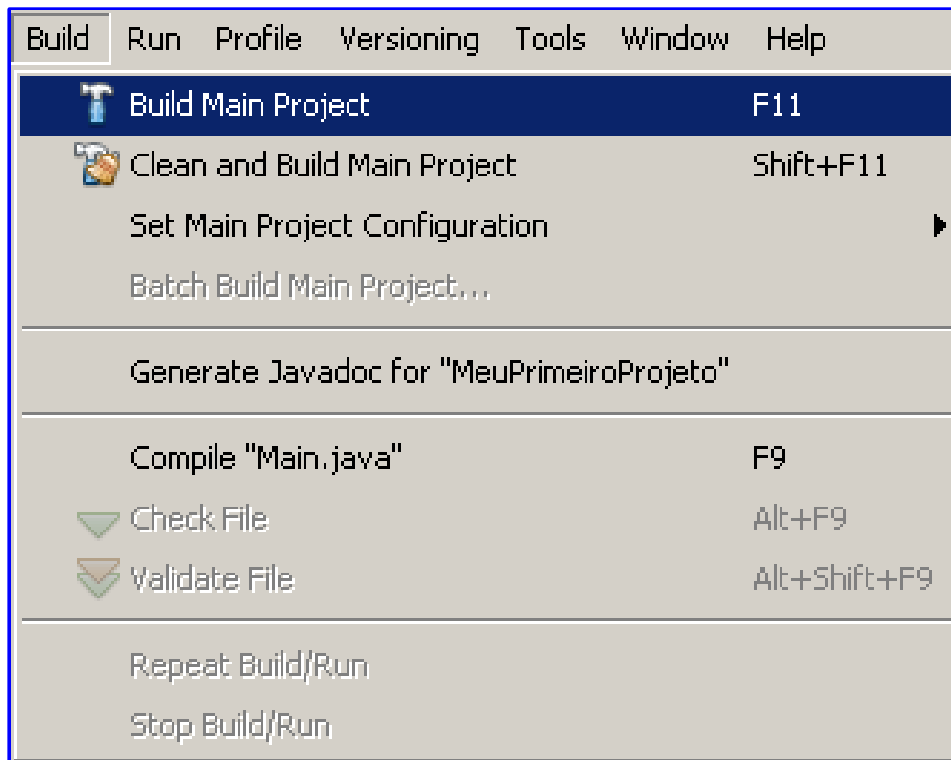
```
Output - MeuPrimeiroProjeto (run)

init:
deps-jar:
Created dir: /root/.NetBeansProjects/MeuPrimeiroProjeto/build/classes
Compiling 1 source file to /root/.NetBeansProjects/MeuPrimeiroProjeto/build/classes
compile:
run:
Ola Mundo!
BUILD SUCCESSFUL (total time: 1 second)
```



# NetBeans

- Construindo o projeto





# NetBeans

- Construindo o projeto

```
Output - MeuPrimeiroProjeto (jar)

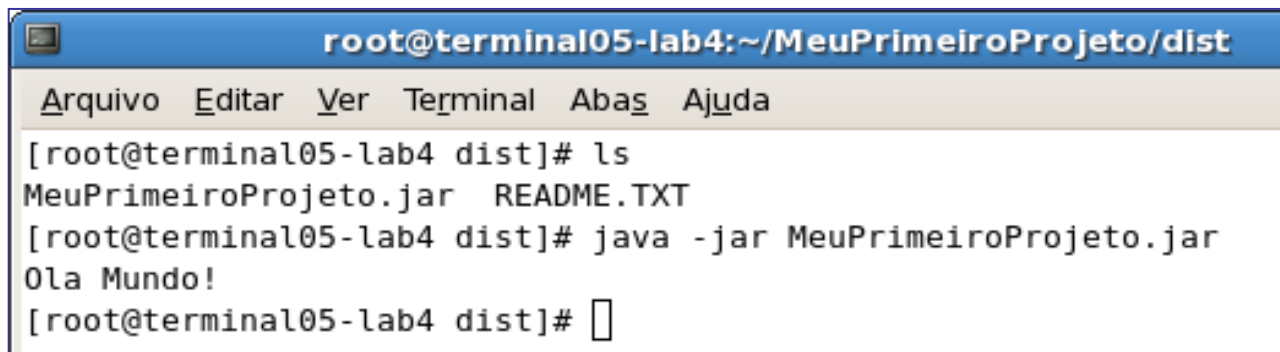
init:
deps-jar:
compile:
Created dir: /root/NetBeansProjects/MeuPrimeiroProjeto/dist
Building jar: /root/NetBeansProjects/MeuPrimeiroProjeto/dist/MeuPrimeiroProjeto.jar
Not copying the libraries.
To run this application from the command line without Ant, try:
java -jar "/root/NetBeansProjects/MeuPrimeiroProjeto/dist/MeuPrimeiroProjeto.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```



# NetBeans

- Para executar o projeto:

```
[root@aula]# cd NetbeansProjects/  
[root@aula]# cd MeuPrimeiroProjeto/  
[root@aula]# cd dist  
[root@aula]# java -jar MeuPrimeiroProjeto.jar
```

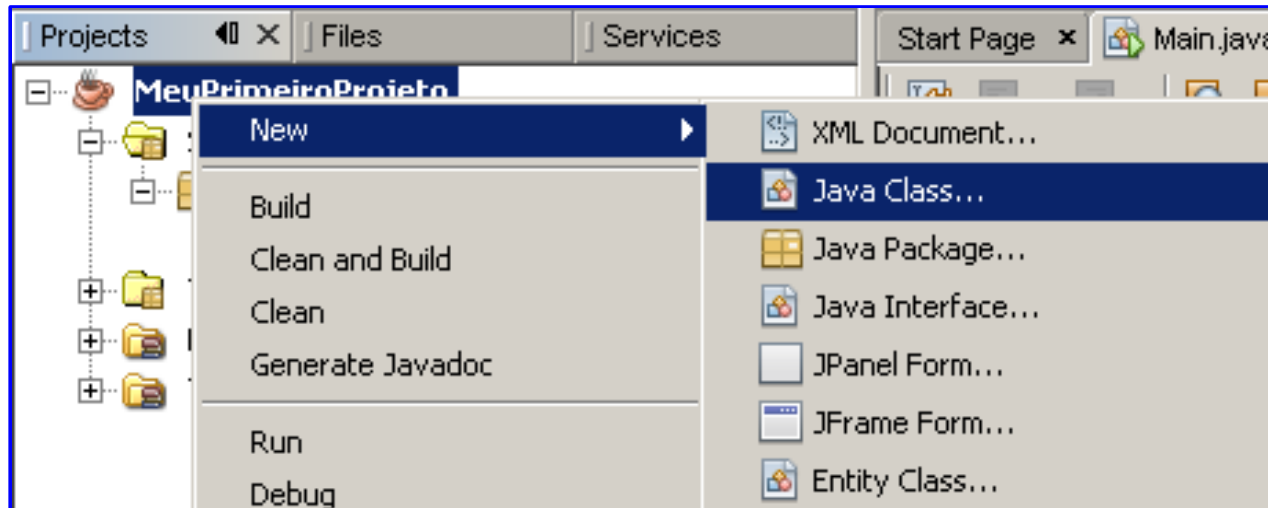


```
root@terminal05-lab4:~/MeuPrimeiroProjeto/dist  
Arquivo  Editar  Ver  Terminal  Abas  Ajuda  
[root@terminal05-lab4 dist]# ls  
MeuPrimeiroProjeto.jar  README.TXT  
[root@terminal05-lab4 dist]# java -jar MeuPrimeiroProjeto.jar  
Ola Mundo!  
[root@terminal05-lab4 dist]#
```



# NetBeans

- Criando uma nova classe





# NetBeans

- Criando uma nova classe

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:



# NetBeans

- Nova classe

```
package meuprimeiroprojeto;  
  
public class OlaBrasil {  
  
    public OlaBrasil() {  
        System.out.println("Ola Brasil!");  
    }  
  
}
```



# NetBeans

- Nova classe

```
package meuprimeiroprojeto;

/**
 *
 * @author root
 */
public class OlaBrasil {

    /** Creates a new instance of OlaBrasil */
    public OlaBrasil() {
        System.out.println("Ola Brasil!");
    }

}
```

20:43 INS





# NetBeans

- Alterando o código da classe **Main.java**

```
package meuprimeiroprojeto;

public class Main {

    public Main() {
    }

    public static void main (String[] args) {
        System.out.println("Ola Mundo!");

        OlaBrasil msg;
        msg = new OlaBrasil();
    }
}
```



# NetBeans

- Alterando o código da classe **Main.java**

```
package meuprimeiroprojeto;

/**
 *
 * @author root
 */
public class Main {

    /** Creates a new instance of Main */
    public Main() {

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Ola Mundo!");

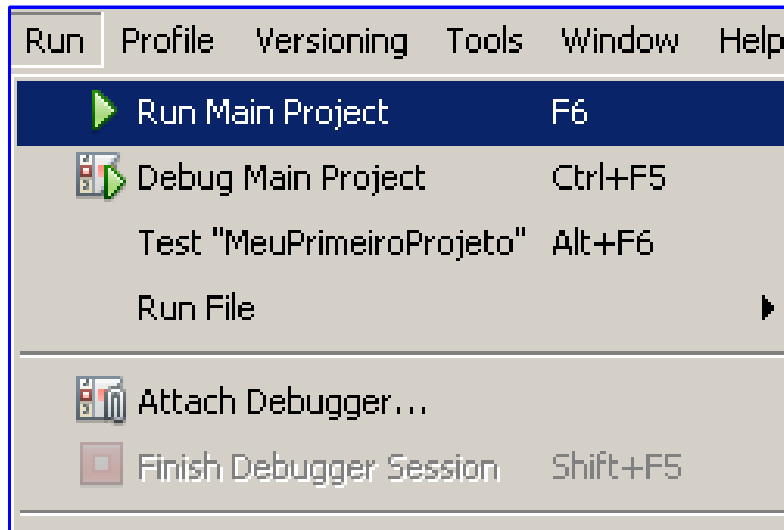
        OlaBrasil msg;
        msg = new OlaBrasil();
    }
}
```

30:31 INS



# NetBeans

- Executando o projeto





# NetBeans

- Visualizando a execução do projeto

```
Output - MeuPrimeiroProjeto (run)
init:
deps-jar:
Compiling 2 source files to /root/NetBeansProjects/MeuPrimeiroPr
compile:
run:
Ola Mundo!
Ola Brasil!
BUILD SUCCESSFUL (total time: 0 seconds)
```



# NetBeans

- Para executar o projeto:

```
[root@aula]# cd NetBeansProjects
[root@aula]# cd MeuPrimeiroProjeto/
[root@aula]# cd dist
[root@aula]# java -jar MeuPrimeiroProjeto.jar
```

```
[root@terminal05-lab4 OlaMundo]# java OlaMundo
Ola Mundo!
Ola Brasil!
[root@terminal05-lab4 OlaMundo]#
```



# Java no terminal

- Crie uma pasta para armazenar os arquivos Java e crie o arquivo **OlaMundo.java**

```
[root@aula]# cd /<pastadousuario>/  
[root@aula]# mkdir meuPrograma  
[root@aula]# cd meuPrograma  
[root@aula]# gedit OlaMundo.java
```



# Java no terminal

- Digite o seguinte código no arquivo **OlaMundo.java** (Exemplo 1)

```
public class OlaMundo {  
  
    public static void main (String[] args) {  
        System.out.println("Ola Mundo!");  
    }  
}
```



# Java no terminal

- Para compilar, digite

```
[root@aula]# javac OlaMundo.java
```

- Para executar, digite

```
[root@aula]# java OlaMundo
```

```
[root@terminal05-lab4 OlaMundo]# java OlaMundo  
Ola Mundo!
```



# Java no terminal

- Crie o arquivo **OlaBrasil.java**

```
[root@aula]# gedit OlaBrasil.java
```



# Java no terminal

- Digite o seguinte código no arquivo **OlaBrasil.java**

```
public class OlaBrasil {  
  
    public OlaBrasil() {  
        System.out.println("Ola Brasil!");  
    }  
}
```



# Java no terminal

- Altere o arquivo **OlaMundo.java**

```
public class OlaMundo {  
  
    public static void main (String[] args) {  
        System.out.println("Ola Mundo!");  
  
        OlaBrasil msg;  
        msg = new OlaBrasil();  
    }  
}
```



# Java no terminal

- Para compilar, digite

```
[root@aula]# javac OlaMundo.java
```

- Como **OlaMundo.java** utiliza uma classe que está em outro arquivo (**OlaBrasil.java**), e como este está no mesmo diretório, é compilado automaticamente pelo **javac**



# Java no terminal

- Para compilar, digite

```
[root@aula]# java OlaMundo
```

```
[root@terminal05-lab4 OlaMundo]# java OlaMundo  
Ola Mundo!  
Ola Brasil!  
[root@terminal05-lab4 OlaMundo]#
```



# Prática 1

---

Modifique o Exemplo 1 - `OlaMundo.java` - para que imprima duas ou mais mensagens.



## Exemplo 2 – soma de dois inteiros

```
import java.util.Scanner;
import java.lang.String;
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner( System.in );
        int numero1, numero2, soma;
        System.out.print( "Entre com o primeiro inteiro:" );
        numero1 = input.nextInt();
        System.out.print( "Entre com o segundo inteiro: " );
        numero2 = input.nextInt();
        soma = numero1 + numero2; // soma os numeros
        System.out.printf( "A soma eh %d\n", soma); // exhibe a soma
        System.out.printf( "Deseja finalizar?\n");
        String c = input.next();
        if (c.equals("sim"))
            System.out.printf( "Finalizacao Correta\n" );
        else
            System.out.printf( "Finalizacao Incorreta\n" );
    }
}
```



## Prática 2

---

Modifique o Exemplo 2 - **Main.java** – para que permita ler dois inteiros e a operação a ser realizada e em seguida apresente o resultado.





# Linguagem Java

## **package - definição e uso ( import )**

- método de organizar grupos de classes relacionadas - pode conter qualquer número de classes que se relacionam pelo objetivo, pelo escopo ou pela herança.
- uso de pacotes já desenvolvidos – importação para a classe que está sendo desenvolvida.

```
package Nome_do_pacote;           // definição
import java.io.*;                 // uso
public class nova-classe {
    ....
}
```

- Não serão importados todas as classes dos pacotes pela cláusula "*import ...;*", mas apenas as classes que os códigos desse pacote fazem referência.



# Linguagem Java

---

## # javac -d . ex1.java

- **d** informa ao compilador criar diretórios apropriados com base na declaração `package` da classe
- . informa que é para ser criado a partir do diretório atual

## Importação de pacotes

```
import java.util.*; // todas as classes do pacote java.util  
import Ex1.ex1; // importa a classe ex1 do pacote Ex1  
public class ex2 {...
```


- Ao compilar `ex2.java` será necessário localizar `ex1` – o compilador usará o *CLASSPATH* (contém uma lista de locais de armazenamento das classes - *default* = .)

## # export CLASSPATH =./usr/Ex1/ex1




Tipo	Descrição do Tipo de Dados
<b>boolean</b>	Pode assumir o valor <b>true</b> ou o valor <b>false</b>
<b>char</b>	Caractere em notação de 16 bits. Serve para a armazenagem de dados alfanuméricos. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535.
<b>byte</b>	Inteiro de 8 bits em notação de complemento de dois, valores entre $-2^7=-128$ e $2^7-1=127$ .
<b>short</b>	Inteiro de 16 bits em notação de complemento de dois, valores de $-2^{15}=-32.768$ a $2^{15}-1=32.767$
<b>int</b>	Inteiro de 32 bits em notação de complemento de dois, valores entre $-2^{31}=2.147.483.648$ e $2^{31}-1=2.147.483.647$ .
<b>long</b>	Inteiro de 64 bits em notação de complemento de dois, valores entre $-2^{63}$ e $2^{63}-1$ .
<b>float</b>	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável por esse tipo é $1.40239846e-46$ e o maior é $3.40282347e+38$
<b>double</b>	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável é $4.94065645841246544e-324$ e o maior é $1.7976931348623157e+308$

# Linguagem Java - Operadores e Precedência



. [ ] ( ) ( <i>tipo</i> )	Máxima precedência: separador, indexação, parâmetros, conversão de tipo
+ - ~ ! ++ --	Operador unário: positivo, negativo, negação (inversão bit a bit), não (lógico), incr, decr
* / %	Multiplicação, divisão e módulo (inteiros)
+ -	Adição, subtração
<< >> >>>	Translação (bit a bit) à esquerda, direita sinalizada, e direita não sinalizada (bit sinal= 0)
< <= >= >	Operador relacional: menor, menor ou igual, maior ou igual, maior
== !=	Igualdade: igual, diferente
&	Operador lógico <b>e</b> bit a bit
^	Ou exclusivo ( <b>xor</b> ) bit a bit
	Operador lógico <b>ou</b> bit a bit
&&	Operador lógico <b>e</b> condicional
	Operador lógico <b>ou</b> condicional
?:	Condicional: <b>if-then-else</b> compacto
= op=	Atribuição, onde op pode ser +, -, *, / ou %



# Linguagem Java – Tipos de Dados

## Conversão entre tipos de dados – implícita e explícita

```
{ double x; int i = 20; x = i; }
```

o valor do inteiro i é convertido automaticamente para um double antes de ser armazenado na variável x.

As regras de conversão implícita empregadas pela linguagem Java convertem tipo byte e short para um int nos operadores unários **+** e **--** e para os operadores binários, em geral, converte-se o operando de menor tipo para o maior e em seguida realizar a operação.

Uso da conversão explícita para garantir o resultado desejado da expressão

```
{ float eventos = 25.7;  
  float dias = 7.2;  
  x = (int) (eventos / dias); }
```



# Linguagem Java – Tipo de Dados Array

```
float nota[] = { 2.1, 3.4, 7.8, 9.0 };
```

```
double b[][] = { {3.1,1.2,8.7}, {2.6,1.5,9.5}, {-15.4,8.7,-2.1} };
```

A utilização de vetores e matrizes em Java envolve três etapas:

1. **Declarar o vetor ou matriz.** acrescentar um par de colchetes depois do nome da variável.

```
int a[ ];          double b[ ][ ],c[ ][ ][ ];
```

2. **Reservar espaço de memória e definir o tamanho.**

```
a = new int[10];
```

```
b = new double[10][20];
```

3. **Armazenar elementos no vetor ou matriz.**

```
A[3] = 5.2;
```

**Os índices começam em zero e vão até o número de posições reservadas, menos um.**



# Linguagem Java – Classes

Uma classe pode ser entendida como sendo um conjunto de dados (variáveis) e métodos (funções) da forma:

```
public class [nome] {  
    [dados e métodos]  
}
```

ou

```
class [nome] {  
    [dados e métodos]  
}
```

- onde **[nome]** é um identificador que dá nome à classe, e o par de chaves delimita uma região para declaração de variáveis e métodos. Uma classe pode ser *privada* ou *pública*, sendo que as classes privadas (default) são conhecidas apenas no escopo do arquivo que a contém. As classes públicas são conhecidas por qualquer arquivo fonte que componha o programa, e devem ser precedidas, na sua declaração, pela palavra-chave **public** .



# Linguagem Java – Definição de Classes

```
class nome-da-classe extends nome-da-superclasse {  
    tipo var-instancia1;  
    tipo var-instancia2;  
  
    tipo var-instanciaN;  
  
    tipo nome-metodo1 ( lista-de-parametros ) {  
        corpo do método1;  
    }  
    tipo nome-metodo2 ( lista-de-parametros ) {  
        corpo do método2;  
    }  
  
    tipo nome-metodoM ( lista-de-parametros ) {  
        corpo do métodoM;  
    }  
}
```

**Não existe herança  
múltipla em Java.**





# Linguagem Java – Declaração de Classe

**[*modificadores*] class [nome classe] *extends* [nome super]  
*implements* [nome interface]**

- **modificadores da classe** determinam como uma classe será manipulada mais tarde no decorrer do desenvolvimento do programa, e tem o mesmo significado dos modificadores de métodos.
- 
- **nome de uma classe** é um identificador em Java (letras maiúsculas e as minúsculas são consideradas diferentes)
- 
- POO possui a capacidade de usar campos e métodos de uma classe previamente construída. Ao derivar uma classe fazemos uma cópia da classe pai, e podemos ainda acrescentar novos campos e métodos à subclasse, além de sobrepor métodos existentes na superclasse, declarando-os exatamente como na superclasse, exceto por dar um corpo diferente.

# Linguagem Java – Classes – Exemplo 3



```
package Pconta;
public class conta {
    String numero;
    double saldo;

    public void depositoinicial
        (double valor, String nro)
    {
        numero = nro;
        saldo =valor;    }

    public void credito ( double valor)
    {
        saldo += valor;    }

    public void debito (double valor)
    {
        saldo -= valor;    }

    public String getNumero ()
    {
        return numero;    }

    public double getsaldo ()
    {
        return saldo;    }
}
```

```
import Pconta;
public class usarconta {
    public static void main(String args[]) {
        String n="12345",n2;
        double p1,p2;
        conta c1;
        System.out.println("Teste de classe e objeto!");
        c1= new conta();
        c1.depositoinicial(12.5,n);
        c1.credito(10.0);
        p1=c1.getsaldo();
        System.out.print("saldo = " );
        System.out.print(p1);
        n2=c1.getNumero();
        c1.debito(p1);
        System.out.print("numero da conta = " );
        System.out.print(n2);
        p2 = c1.getsaldo();
        System.out.print("Novo saldo = " );
        System.out.print(p2);
    }
}
```



## Prática 3

---

Modifique o Exemplo `conta.java` e `usarConta.java`, de forma que as opções de manipulação dos dados nas contas possam ser feitas por solicitação externa.



# Linguagem Java – Construtores de Objetos

---

- **Construtor** é um método especial que inicializa um objeto, imediatamente depois de sua criação.
  - tem o mesmo nome que a classe.
  - são métodos semelhantes aos demais, mas não permitem valores de retorno.
  - pode existir mais de um construtor para uma mesma classe: eles serão distinguidos pelo número de parâmetros e também pelo tipo de cada um dos seus parâmetros.
  - os construtores não são ativados como os demais métodos, mas de forma implícita no momento da instanciação do objeto.



# Linguagem Java – Construtores de Objetos

```
class Teste {  
    public int x;  
    public int y;  
    Teste (int x, int y)  
    {   this.x =x;  
        this.y=y; }  
}
```

```
public class criaTeste {  
    public static void main (String args[])  
    { Teste t = new Teste(10,67);  
      System.out.println ("x="+t.x+"y="+t.y); }  
}
```

**O acesso às variáveis de instância**

**x (t.x) e y (t.y) do objeto t  
poderiam ser feitas por métodos públicos  
da classe Teste (setX e getX)**



# Linguagem Java – Operadores em Objetos

---

- **Operador new:** cria uma instância de classe
  - Cria a área de armazenamento para a instância da classe conta, com c fazendo referência.
  - Quando o objeto c não for mais visível, a área é retornada como disponível ao Coletor de Lixo.

## **CRIAÇÃO DE INSTÂNCIAS DE OBJETOS**

**Declaração:** *Nome-Classe Lista-de-Identificadores;*  
*conta c1, c2;*

**Criação:** *identificador = new Nome-Classe ( argumentos );*  
*c1 = new conta; c2= new conta(120.0);*

**Combinando declaração com a criação da área de armazenamento:**  
*Nome-classe identificador = new Nome-Classe ( argumentos );*  
*conta c3 = new conta (145.20);*



# Linguagem Java – Operadores em Objetos

---

- **Operador Ponto**

- usado para acessar as variáveis de instância e os métodos dentro de um objeto
- referência-a-um-objeto . Nome-da-variável= ...;
- referência-a-um-objeto . nome-do-método( );



# Linguagem Java – Operadores em Objetos

---

- **Operador this:** faz referência ao objeto corrente – `this.x=i;`

O atributo x do objeto corrente que está sendo referenciado é atualizado com o valor da variável x.

útil quando há dúvidas do que significa o identificador, como em `this.x=x;` em que o atributo x do objeto é atualizado com o valor da variável x (possivelmente parâmetro)

- **Método super:** permite o acesso a partes de uma superclasse a partir de uma subclasse.

útil quando estamos sobrepondo um método, fazendo referência como **`super.nome-metodo()`** pra identificar que quero que seja executado o método da classe pai, e não o da classe filha.



# Linguagem Java – Variáveis em Classes / Acesso



Os possíveis moderadores empregados na declaração de campos são os seguintes:

- **public:** idêntico ao moderador de acesso dos métodos. O campo é acessível a partir de qualquer outra classe, independentemente do package.
- 
- **protected:** os campos protected podem ser acessados a partir de qualquer classe derivada da classe atual, e são acessíveis de fora do package.
- 
- **private:** é o maior grau de proteção. Um campo private é acessível unicamente pela classe atual.
- 
- **static:** Um campo static é compartilhado por todas as instâncias de uma classe, isto é, há um único valor para esse campo, independentemente da quantidade de instâncias existentes, mesmo que não haja nenhuma.
- 
- **final:** um modificador **final** precedendo um campo declara esse campo como uma constante. Seu valor não pode mudar durante a execução do programa. Por isso, é necessário que haja uma inicialização de campo.

# Linguagem Java – Variáveis em Classes / Acesso

- A capacidade de acessar uma variável de uma classe depende fundamentalmente de duas coisas: moderadores de acesso e localização da variável dentro da classe. As variáveis locais somente são acessíveis pelo método que as declara, enquanto que os campos dependem dos moderadores.

**Apesar de ser possível deixar todos os campos de uma classe publicamente acessíveis, isto não é recomendável.**



# Linguagem Java – Métodos

---

**[moderadores de acesso] [modificador] [tipo do valor de retorno] [nome] ([parâmetros]) *throws* [lista de exceções]**  
**{ [corpo] }** *(os termos em itálico são opcionais)*

**Moderadores de acesso** usados para restringir o acesso a um método:

- **public:** O método pode ser chamado a partir de métodos contidos em qualquer outra classe.
- **private:** O método é privativo da classe que o contém e seu uso é vedado a qualquer outra classe.
- **protected:** Uso do método é permitido dentro da classe que o contém, assim como dentro de qualquer classe que tenha sido derivada dessa classe, ainda que esteja fora do package.



# Linguagem Java – Modificadores de Métodos

---

- **static:** o método é compartilhado por todos os objetos instanciados a partir da mesma classe.
- **abstract:** método sem especificação de conteúdo do corpo, que deve ser redefinido em classes derivadas.
- **final:** Especifica que nenhuma classe derivada pode alterar ou redefinir este método.
- **synchronized:** é usado para processamento concorrente e não permite que dois objetos executando concorrentemente usem esse método para acessar os dados privados ao mesmo tempo.



# Linguagem Java – Parâmetros

---

- A lista de parâmetros é a lista de valores que o método vai precisar, obedecendo a sintaxe [tipo 1] [nome 1], [tipo 2] [nome 2], ...
  - Vale observar que Java trata o objeto por referência e por isso, se o método modificar o objeto recebido como parâmetro, o objeto será modificado externamente. Se o parâmetro recebido for tipo primitivo (int, double, char, float, etc), a transferência é feita por valor, ou seja, não altera o parâmetro externamente.



# Linguagem Java – Estruturas de Controle

## COMANDO CONDICIONAL

*if ([condição])  
[comando] // Executa o comando se a condição for true*

*If ([condição])  
[comando 1] // Executa o comando1 se a condição for true  
else  
[comando 2] // Executa o comando2 se a condição for false*



# Linguagem Java – Estruturas de Controle

## SELEÇÃO MÚLTIPLA

```
switch ( [expressão] ) {  
    case [constante 1]:    [comando 1] break;  
    case [constante 2]:    [comando 2] break;  
    .  
    .  
    case [constante n]:    [comando n] break;  
    default:               [comando]  
}
```

- A [expressão] pode ser qualquer expressão, que será avaliada e o seu valor resultante será comparado com as constantes distintas [constante 1], [constante 2], ..., [constante n].



# Linguagem Java – Estruturas de Controle

---

## REPETIÇÃO

```
while ( [condição] )  
    [comando subjacente]
```

```
do  
    [comando]  
while ( [condição] );
```

```
for ( [expressão 1] ; [condição] ; [expressão 2] )  
    [comando]
```

- onde [expressão 1] é chamada *expressão de inicialização*, [condição] é uma expressão condicional e [expressão 2] é uma expressão qualquer a ser executado no final de cada iteração.





# Linguagem Java – Estruturas de Controle

---

## COMANDO BREAK

- O comando break é usado para interromper a execução de um dos laços de iteração ou de um comando switch.
- Este comando é utilizado para produzir a parada de um laço quando ocorre alguma condição específica, antes da chegada do final natural do laço.

## COMANDO CONTINUE

- O comando continue tem a função de pular direto para final do laço, mas em vez de interromper o laço como no break, ele continua executando o próximo passo do laço.



## Prática 4

---

Alterar a classe de utilização do jogo **Craps**, de forma que o jogo continue até que o computador ganhe o jogo.



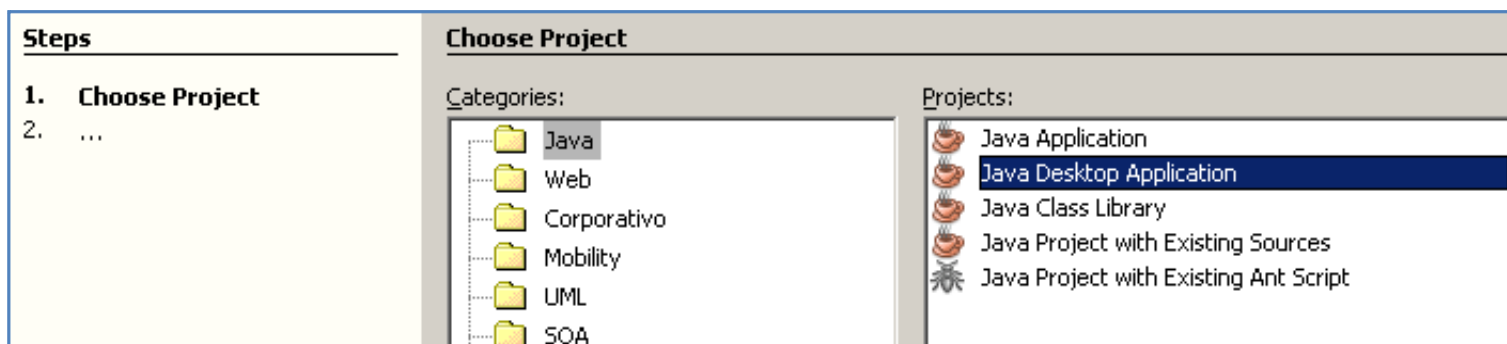
# Criando interfaces gráficas no NetBeans

---

- O NetBeans GUI Builder elimina as dificuldades de criação de interfaces gráficas (GUI)
  - Permite desenhar interfaces simplesmente arrastando componentes
  - Utiliza AWT e Swing
    - **AWT**: Coleção de componentes de interface
    - **Swing**: API semelhante ao AWT, mas manipula os componentes da interface de modo diferente

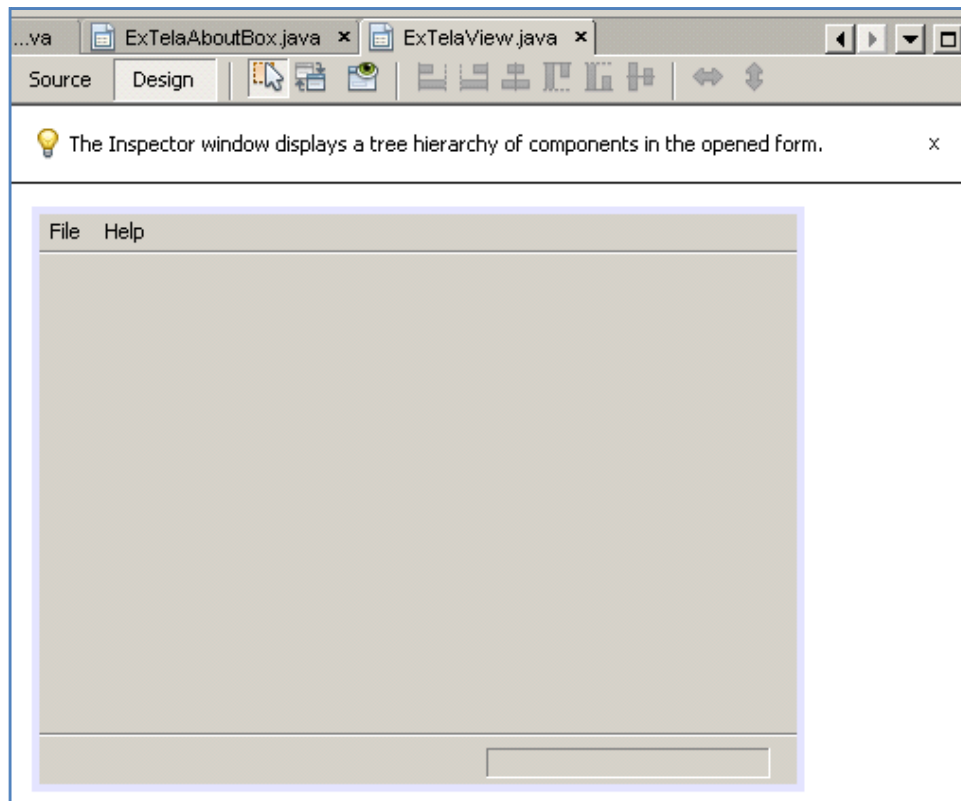
# Criando interfaces gráficas no NetBeans

- Criando um novo projeto



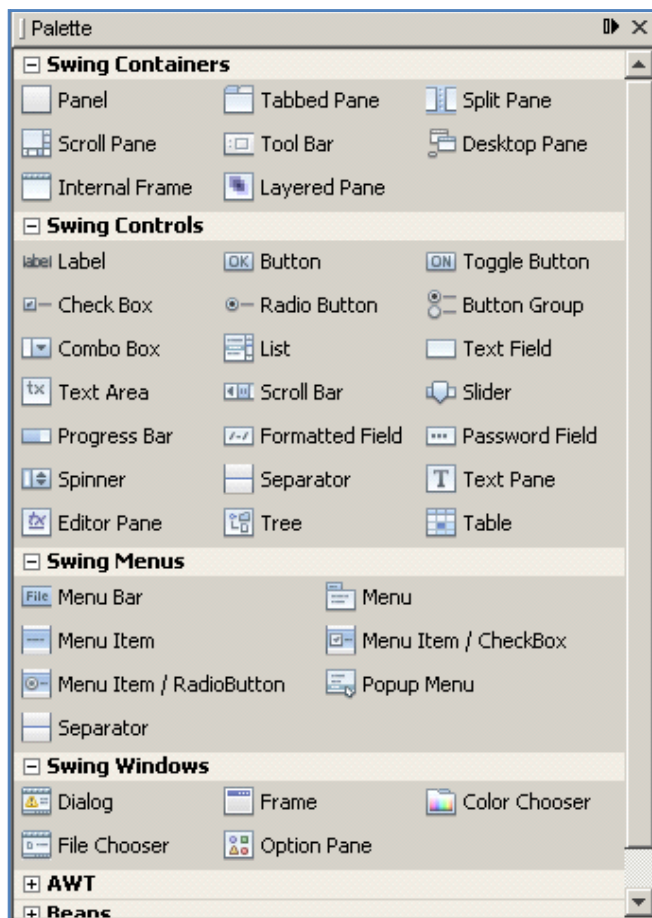
# Criando interfaces gráficas no NetBeans

- Tela do formulário



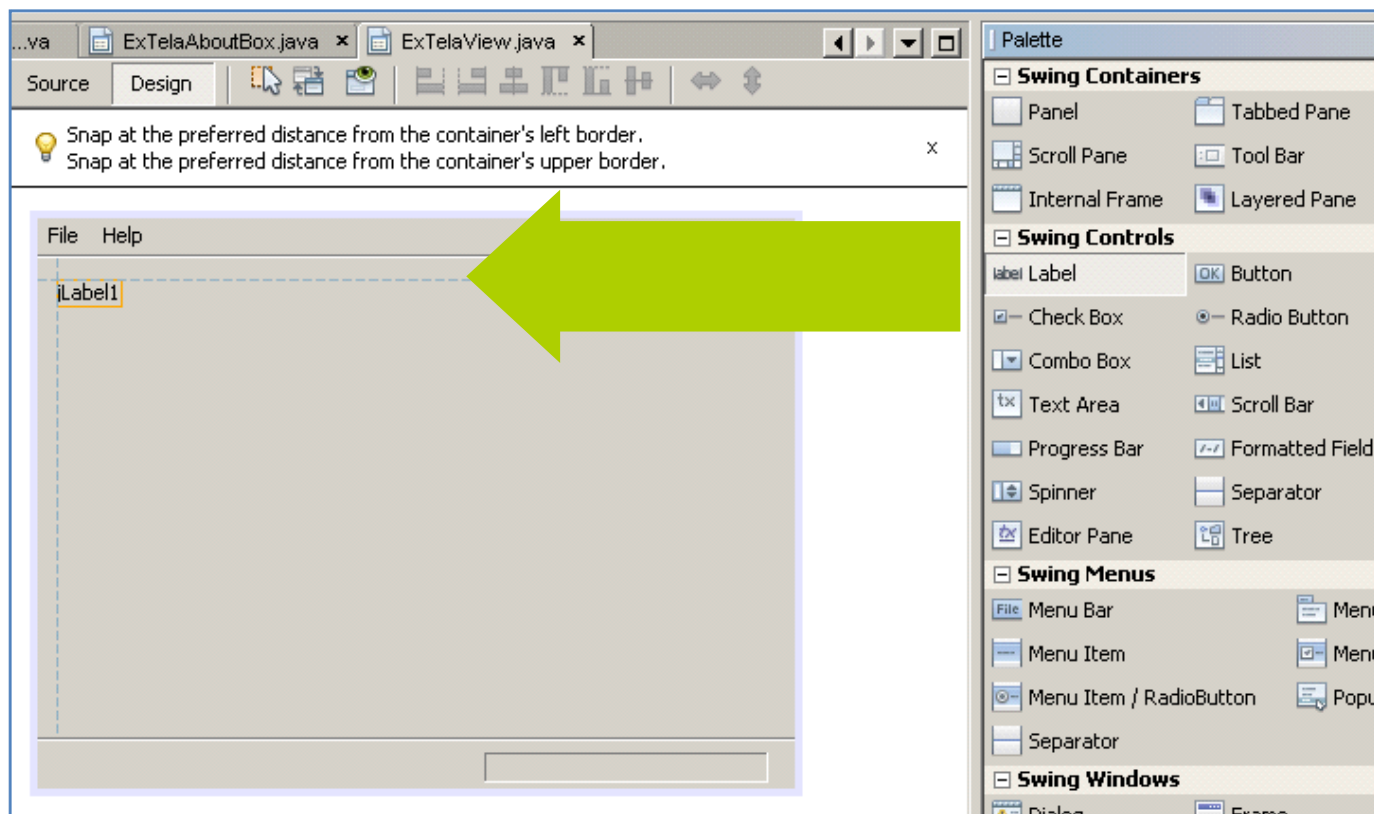
# Criando interfaces gráficas no NetBeans

- Paleta dos componentes



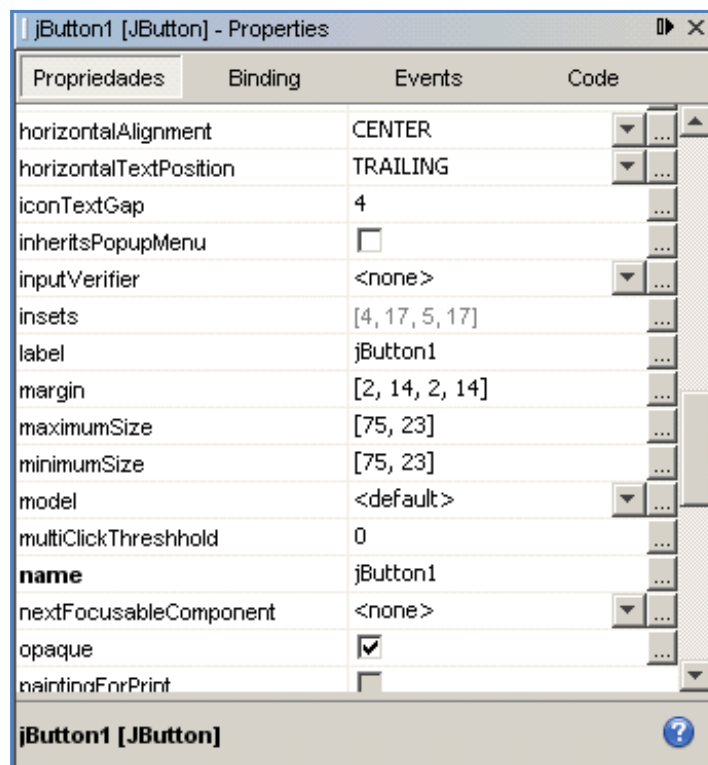
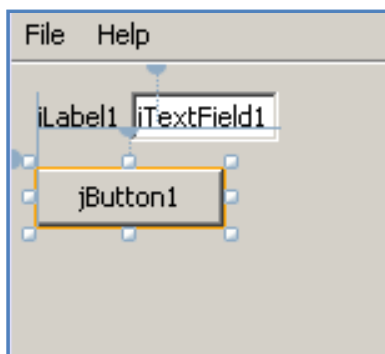
# Criando interfaces gráficas no NetBeans

- Arrastando componentes para a interface gráfica



# Criando interfaces gráficas no NetBeans

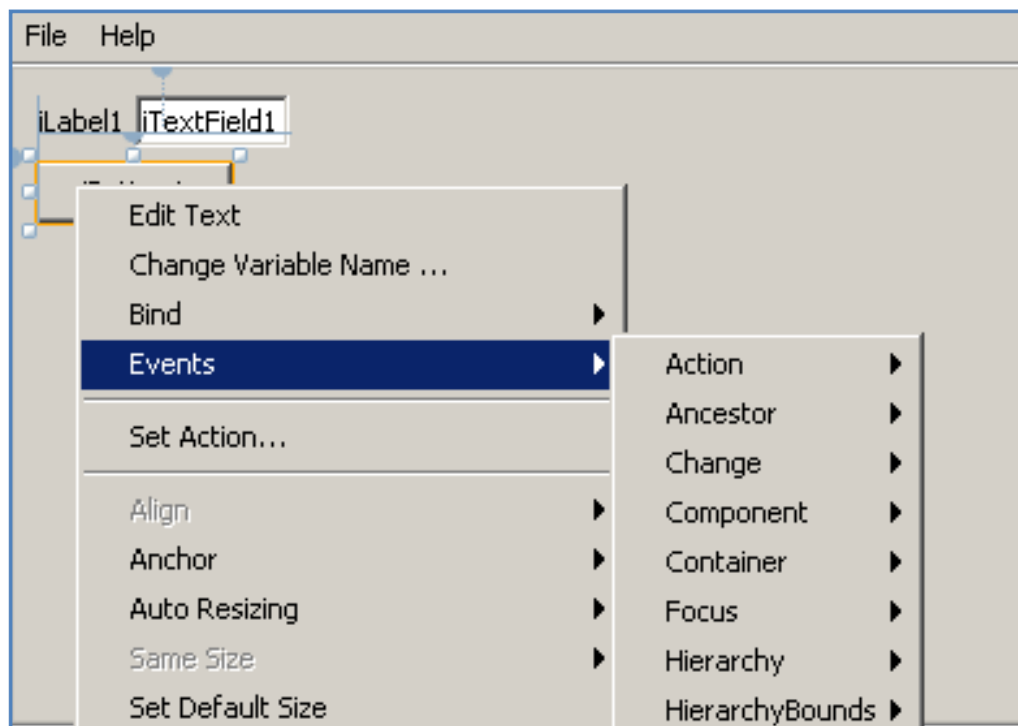
- Propriedades do componente selecionado





# Criando interfaces gráficas no NetBeans

- Definindo eventos para os componentes





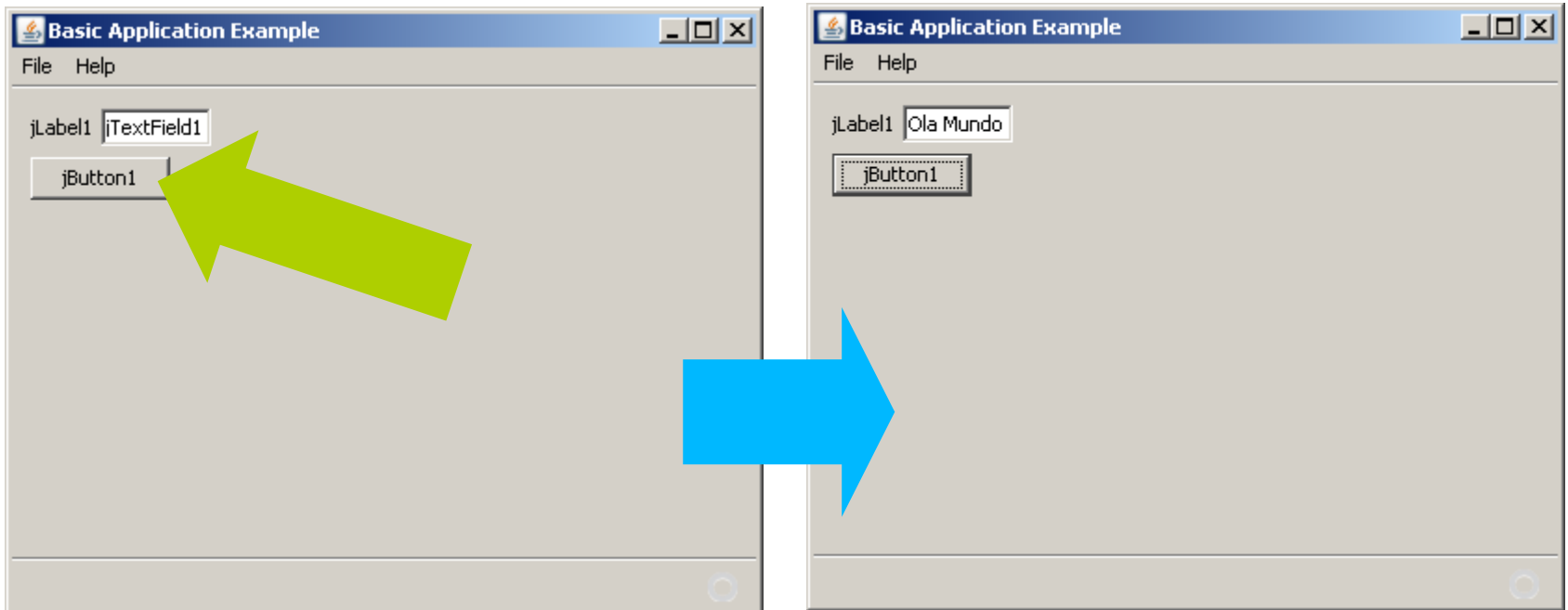
# Criando interfaces gráficas no NetBeans

- Definindo eventos para os componentes

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    jTextField1.setText("Ola Mundo");  
}  
  
// Variables declaration - do not modify  
private javax.swing.JButton jButton1;  
private javax.swing.JLabel jLabel1;  
private javax.swing.JTextField jTextField1;  
private javax.swing.JPanel mainPanel;  
private javax.swing.JMenuBar menuBar;  
private javax.swing.JProgressBar progressBar;  
private javax.swing.JLabel statusAnimationLabel;  
private javax.swing.JLabel statusMessageLabel;  
private javax.swing.JPanel statusPanel;  
// End of variables declaration
```

# Criando interfaces gráficas no NetBeans

- Executando o programa





## Exemplo 5 – Soma de inteiros

```
private void botao2ActionPerformed(java.awt.event.ActionEvent evt) {  
    System.exit(0);  
}
```

```
private void botao3ActionPerformed(java.awt.event.ActionEvent evt) {  
    campo1.setText("0");  
    campo2.setText("0");  
    campo3.setText("0");  
}
```

```
private void botao1ActionPerformed(java.awt.event.ActionEvent evt) {  
    int c1,c2,c3=0;  
    c1= Integer.parseInt(campo1.getText());  
    c2= Integer.parseInt(campo2.getText());  
    c3=c1+c2;  
    campo1.setText(String.valueOf(c1) );  
    campo2.setText(String.valueOf(c2));  
    campo3.setText(String.valueOf(c3));  
}
```



## Exemplo 5 – Soma de inteiros

- Executando o programa



## Prática 5

---

- Alterar o exemplo 5 para incluir a operação de multiplicação de inteiros

# Linguagem Java – Applets



- É um programa em Java, que é executado por um browser, quando é carregada a página que contém o applet.
- É um programa que será transportado pela internet, como os documentos html.



- **São construídos para serem pequenos.**
- **por questões de segurança, obedecem critérios rígidos para que sua execução seja possível pelos browsers.**

```
import java.applet.Applet;
import java.awt.Graphics;
public class alo extends Applet {
    public void paint (Graphics g) {
        g.drawString ("Alo a todos, Exercitanda Java", 25,25);
        g.drawString ("Lembre-se de que o arquivo tem que ter o nome do applet",
25,50);
    }
}
```

```
<html>
  <applet code="alo.class" width=300 height=80>
  </applet>
</html>
```



# Linguagem Java – Applets

Campos do tag `<applet>`

ARCHIVE	nome do arquivo compactado do applet (".jar")
ALT	opcional, texto alternativo se o browser não puder exibir o applet
ALIGN	alinhamento (top,middle,botton,left,right)
CODE	nome do arquivo de classe, incluindo a extensão ".class"
CODEBASE	onde os arquivos estão localizados- se não for especificado será considerado a URL da página que contém o applet.
HEIGHT	altura do applet (em pixels)
HSPACE	margem horizontal do applet (em pixels)
NAME	opcional, nome do applet para comunicação inter-applets
OBJECT	nome do arquivo applet serializado (".ser")
VSPACE	margem vertical (em pixels)
WIDTH	largura do applet (em pixels)





# Funcionamento dos Applets

---

## **public void init ( )**

- esse método é chamado uma vez pelo appletviewer ou pelo browser quando um applet é carregado para execução - Ações Típicas: Inicializações de variáveis, carregamento de sons para reproduzir ou imagens a exibir.

## **public void start ( )**

- esse método é chamado assim que o init termina, e a cada vez que o usuário do navegador retorna para a página HTML em que o applet reside.

## **public void paint (Graphics g)**

- esse método é chamado toda vez que o applet necessita atualizar sua exibição, que ocorre na primeira vez depois da execução do start ou quando a tela foi movimentada ou redimensionada e aí precisa ser redesenhado.


## **public void stop ( )**

- esse método é chamado toda vez que o applet deixa de ser visível, ou seja, quando ocorre um rolamento da tela ou quando o applet fica encoberto por outra tela.  
(start e stop podem ocorrer inúmeras vezes no ciclo de vida dos applets)

## **public void destroy ( )**

- esse método é chamado toda vez que o applet está sendo descarregado da página - para que seja realizada a liberação final de todos os recursos utilizados durante a sua execução. É acionado quando o browser troca de páginas.

# Exemplo de Applet – Contagem do número de execução de um Applet



```
// ex2.java - contagem de numero de execucao de um applet
import java.awt.Graphics; // import class Graphics e Applet
import java.applet.Applet;

public class ex2 extends Applet {
    private int nroinits=0;
    private int nrostarts=0;
    private int nropaints=0;
    private int nrostops=0;

    public void init( )    { nroinits ++; }
    public void start( )   { nrostarts ++; }
    public void stop( )    { nrostops ++; }

    public void paint( Graphics g )
    { nropaints++;
      g.drawString( "Init: "+ nroinits, 5, 15 );
      g.drawString( "Start: "+ nrostarts, 5, 30 );
      g.drawString( "Paint: "+ nropaints, 5, 45 );
      g.drawString( "Stop: "+ nrostops, 5, 60 );
    }
}
```



## Exemplo de Applet – Olá Applet

```
import java.awt.Graphics;           // import class Graphics e Applet
import java.applet.Applet;
import java.awt.Color;
import java.awt.Font;

public class ex3 extends Applet {
    Font f = new Font("TimesRoman", Font.BOLD,36);
    String entrada;

    public void init ( )    {
        this.entrada=getParameter("nomeparam");
        if ( (this.entrada).equals(""))
            this.entrada="Sergio Zorzo";
        this.entrada = "Ola " + this.entrada  + "**";
    }
}
```



## Exemplo de Applet – Olá Applet (cont)

```
public void paint( Graphics g ) {  
    g.setFont(f);  
    g.setColor(Color.red);  
    g.drawString( this.entrada, 5, 50 );  
}  
  
}
```

- Arquivo que chama o applet – note que esse arquivo html pode ser gerado dinamicamente.

```
<html>  
  <applet code="ex3.class" width=400 height=80>  
    <param name ="nomeparam" value="" >  
  </applet>  
</html>
```

# Exemplo de Applet – Soma de dois números inteiros



```
import java.applet.Applet;
import java.awt.*; // Abstract Windows Toolkit - AWT
import java.awt.event.*;
public class Soma extends Applet implements ActionListener {
    int valor1, valor2, parcial ;
    TextField campo1, campo2 , resultado;
    Button botao;
    public void init () {
        campo1= new TextField(5);      add (campo1);
        campo2= new TextField(5);      add (campo2);
        resultado = new TextField(30);  add (resultado);
        botao = new Button("SOMAR");   add (botao);
        botao.addActionListener(this);
    }
}
```

# Exemplo de Applet – Soma de dois números inteiros (cont)



```
public void actionPerformed (ActionEvent e) {  
    int parcial;  
    if (e.getSource() == botao ) {  
        valor1= Integer.parseInt (campo1.getText()) ;  
        valor2 = Integer.parseInt (campo2.getText());  
        parcial = valor1 + valor2;  
        resultado.setText("o valor resultante e': "+ parcial );  
    }  
}  
}
```



## Prática 6 - Applets

---

- Definir dois campos de entrada
- Definir dois botões, com rótulo somar e multiplicar
- Executar a operação solicitada e apresentar o resultado em um terceiro campo de entrada