

Sistema Aberto de Educação



Guia de Estudo

Programação Orientada a Objetos I



Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas



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/MG
Unidade de Gestão da Educação a Distância – GEaD**

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

Varginha/MG

Todos os direitos desta edição reservados ao Sistema Aberto de Educação – SABE.
É proibida a duplicação ou reprodução deste volume, ou parte do mesmo, sob
qualquer meio, sem autorização expressa do SABE.

**Revisão pelo Novo Acordo Ortográfico
Da Língua Portuguesa**

005.13
M188P Magalhães, Demétrio Reno.

Guia de Estudo – Programação Orientada a
Objetos I. Demétrio Renó Magalhães. Varginha:
GEaD-UNIS/MG, 2009.

146p.

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

REITOR

Prof. Ms. Stefano Barra Gazzola

GESTOR

Prof. Ms. Tomás Dias Sant' Ana

Supervisor Técnico

Prof. Ms. Wanderson Gomes de Souza

Coord. do Núcleo de Recursos Tecnológicos

Prof^a. Simone de Paula Teodoro Moreira

Coord. do Núcleo de Desenvolvimento Pedagógico

Prof^a. Vera Lúcia Oliveira Pereira

Coord. do Núcleo de Comunicação Relacionamento

Prof. Ms. Renato de Brito

Revisão ortográfica / gramatical

Prof^a. Ms. Silvana Prado

Design/diagramação

Prof. César dos Santos Pereira

Equipe de Tecnologia Educacional

Prof. Celso Augusto dos Santos Gomes

Prof^a. Débora Cristina Francisco Barbosa













Jacqueline Aparecida da Silva

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 Unileste MG 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.

TABELA DE 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 ideias, partes ou unidades do curso virão precedidas desse ícone.
	IMPORTANTE. Apona 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.....	7
EMENTA.....	8
AVALIAÇÃO	8
INTRODUÇÃO	9
Introdução a Programação Orientada a Objetos e Fundamentos da Linguagem Java.....	10
1 AS CLASSES.....	10
1.1 As partes de uma classe	11
1.2 O modelo conceitual.....	13
1.3 Herança.....	15
1.4 Os objetos	15
1.5 Conceitos de programação	16
1.6 Introdução à programação	17
1.7 Tipos de programação	17
1.7.1 Programação linear	17
1.7.2 Programação estruturada	17
1.7.3 Programação orientada a objetos.....	18
1.8 Introdução à tecnologia Java	18
1.8.1 Linguagem de programação Java	18
1.8.2 A plataforma de desenvolvimento Java	19
1.9 Programação orientada a objetos	20
1.10 Programando em JAVA.....	21
Leia o item 2.8 e veja como instalar o JDK e o Eclipse.....	21
1.10.1 Gravar classes em Java	21
1.10.2 Compilar programas em Java utilizando o eclipse	22
1.11 Estruturas de controle em Java.....	25
1.11.1 Estrutura condicional: if.....	25
1.11.2 Estruturas de seleção múltipla: switch	26
1.11.3 Estruturas de repetição: while.....	27
1.11.4 Estruturas de repetição: do..while	27
1.11.5 Estruturas de repetição: for	28
1.12 Exercícios.....	30
1.13 Um estudo mais profundo sobre classes	32
1.13.1 Definindo uma classe	33
1.13.2 Variáveis de instância ou Atributos	34
Como declarar um atributo?	35
1.13.3 Métodos	36
1.14 Combinando uma classe com um programa.....	39
1.14.1 Definindo uma instância de uma classe.....	39
1.14.2 Como acessar uma variável de instância	40
1.15 Exercícios.....	42

2 GUI - INTERFACE GRÁFICA COM O USUÁRIO	44
2.1 Gerenciadores de Layout	46
2.1.1 Construindo GUI's sem Gerenciadores de Layout	46
2.1.2 GridLayout.....	48
2.1.3 BorderLayout.....	50
2.1.4 FlowLayout.....	52
2.2 Exercícios.....	55
2.3 Painéis, menus e componentes da interface gráfica	55
2.3.1 Painéis	55
2.3.2 JScrollPane	58
2.3.3 Menus em Java	61
2.3.4 Submenus	62
2.3.5 Radio Button	63
2.3.6 Check box	65
2.3.7 Listas.....	66
2.3.8 Password	68
2.3.9 ComboBox	69
2.4 Exercícios.....	71
2.5 Manipuladores de Eventos em Java	72
2.5.1 Evento para JButton	72
2.5.2 Evento para JTextField.....	76
2.5.3 Evento para JMenuitem	78
2.5.4 Eventos para JList.....	80
2.5.5 Eventos para ComboBox.....	82
2.5.6 Eventos para JRadioButton	84
2.5.7 Eventos para JCheckBox	85
2.6 Exercícios.....	88
2.7 Resolução dos Exercícios Propostos	88
2.8 Preparando o ambiente de trabalho	134
REFERÊNCIAS	145

APRESENTAÇÃO

Prezado (a) aluno (a),

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

O guia ajuda o aluno 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, por exemplo, aprenderá muito mais e não terá dificuldades para resolver os exercícios.

Prof. Demétrio Renó Magalhães

*“Há os que se queixam do vento. Os que esperam que
ele mude. E os que procuram ajustar as velas”*
Willian George

EMENTA

Classes e objetos (campos e métodos). Fundamentos da linguagem Java. Construção de interface gráfica com o usuário a partir do uso de componentes da biblioteca gráfica – Swing. Encapsulamento (abstração, ocultamento de informação, divisão de responsabilidade). Herança simples. Herança múltipla (através de Interface, delegação). Classe Abstrata. Interface. Polimorfismo de inclusão e paramétrico (sobreposição e sobrecarga).

AVALIAÇÃO

- 1ª avaliação – 25 pontos (prova escrita)
- 2ª avaliação – 30 pontos (prova escrita)
- 3ª avaliação – 5 pontos (fórum)
- 4ª avaliação – 5 pontos (Exercícios – Portifólio)
- 5ª avaliação – 5 pontos (Exercícios – Portifólio)
- 6ª avaliação – 5 pontos (fórum)
- 7ª avaliação – 5 pontos (Exercícios – Portifólio)
- 8ª avaliação – 5 pontos (chat)
- 9ª avaliação – 5 pontos (fórum)
- 10ª avaliação – 10 pontos (Projeto – portfólio)

INTRODUÇÃO

Vamos iniciar nosso curso dando uma definição para a Programação Orientação a Objetos (POO). A Programação Orientada a Objetos é um paradigma de programação que tem como base a interação entre as diversas unidades de software que vamos chamar de objetos.

Você já estudou programação e aprendeu a programação estruturada que é um conjunto de linhas de programas que são executados de cima para baixo com chamadas de algumas funções e procedimentos.

A programação estruturada resolveu e ainda resolve muitos problemas, mas à medida que os computadores foram evoluindo, custando mais barato e sendo mais utilizados pelas empresas, os programas tornaram-se mais complexos e a Programação Orientada a Objetos é uma ferramenta que irá nos auxiliar na construção de programas mais complexos e principalmente na manutenção e na extensão (implementação de mais módulos) desses programas.

Na programação Orientada a Objetos não utilizamos funções nem procedimentos como os utilizados na programação estruturada, usamos os OBJETOS (daí o nome Programação Orientada a OBJETOS).

A Programação Orientada a Objetos nos faz programar um conjunto de Classes que definem os Objetos que estarão presentes no sistema de software. Cada classe irá determinar o comportamento que o objeto terá no sistema.

Essa é uma pequena introdução do que iremos estudar na Programação Orientada a Objetos que é uma disciplina envolvente e empolgante.

Bem vindo à Programação Orientada a Objetos



DESENVOLVIMENTO: UNIDADE I

Introdução a Programação Orientada a Objetos e Fundamentos da Linguagem Java

Ao final dessa unidade você será capaz de:

- Criar suas próprias classes, utilizar classes da linguagem Java para construir interfaces gráficas e manipular eventos.

1 AS CLASSES

Ao final da aula você será capaz de:

- Conceituar classes, atributos, métodos;
- Identificar os diversos tipos de programação;
- Diferenciar classes de programas;
- Escrever pequenos programas em Java que utilize entrada e saída de dados;
- Utilizar as estruturas de controle (condicionais e de repetição).

Vamos iniciar nossos estudos definindo o que é uma classe. Classe é o **conceito** que temos sobre alguma coisa e coisa é tudo aquilo que tivemos, temos ou queremos ter. Um exemplo seria uma casa. Sabemos que toda casa tem:

- portas
- janelas
- telhado
- piso
- banheiro
- sala
- cozinha
- ...

A classe casa (ou nosso conceito de casa) nos disse que a casa tem as partes citadas acima (que chamaremos de **atributos**), mas não nos disse como são essas partes, não define exatamente quantas portas, que cor é essa porta, qual o material da porta (ferro ou madeira), ela apenas nos diz que a casa possui todas essas partes.

Uma classe é um conceito como já foi dito antes e por ser um conceito ela nunca “morre”.



Você já viu um cavalo. Pode me falar os atributos de um cavalo. Pode me falar também o que um cavalo faz. Sendo assim você tem um conceito sobre o que é um cavalo.

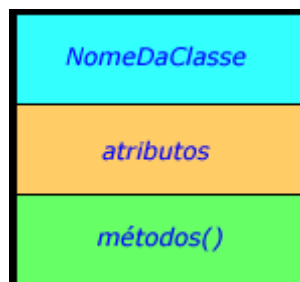
Vamos supor que você foi transportado para um lugar onde nunca existiu um cavalo. Nunca passou um cavalo por ali e eu te perguntar sobre o **conceito** de um cavalo você poderá me dizer correto? Eu não preciso ter ou ver um cavalo para conceituar um cavalo.

1.1 As partes de uma classe

Uma classe é composta pelas partes abaixo:

- NomeDaClasse
- Atributos
- Métodos

A figura abaixo mostra como definimos uma classe:



Nome

Toda classe tem um **nome** que define o que a classe faz.

Atributos

Os **atributos** são as **propriedades** ou **características** da classe.

Métodos

Os métodos que são as funcionalidades da classe, ou seja, o que a classe pode fazer por nós. Os métodos podem ser divididos em dois tipos:

- Construtores
- Operacionais

Métodos Construtores são utilizados no momento da **instanciação** de um objeto e depois não são mais utilizados. Eles servem somente para construir objetos.

Métodos Operacionais são utilizados durante a execução do programa. Tanto os métodos construtores como os métodos operacionais podem ser sobrecarregados.

Professor! Não entendi “o que a classe pode fazer por nós”.

Tudo bem, vamos lá.

Uma classe armazena valores em seus atributos. Como por exemplo, se nós tivermos uma classe **Automóvel**, teremos como atributos cor, ano, quantidadeDePortas, etc., tudo isso são atributos de um automóvel. Veja que não colocamos nenhum valor para eles somente dizemos que ele tem esses atributos. Os métodos são tudo aquilo que um automóvel consegue fazer como ligar(), desligar(), andar(), parar() e assim por diante.

Vamos falar um pouco de orientação a objetos em *software*! Se você tiver uma classe chamada **Pedido** (um formulário de pedido), quais atributos teriam em um formulário de pedido?

Alguns atributos seriam nomeDoCliente, codigoDoProduto, valorUnitario, etc. Tudo bem professor esses são os atributos mas e os métodos (funcionalidades)?

Alguns métodos seriam imprimir(), alterar(), configurarNome(), mostrarNome(), etc.



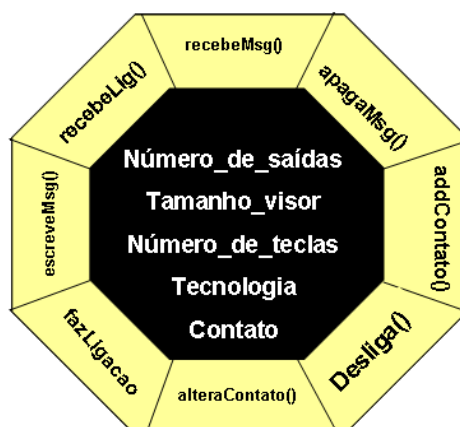
Vamos ver. Quais seriam os atributos e métodos para um aparelho de Celular?

Os métodos são os comportamentos (ações ou funcionalidades) que um objeto pode ter. Se tivermos uma classe chamada **Celular** essa classe terá atributos e métodos. Depois de instanciarmos a classe **Celular** teremos um **Objeto Celular** que terá os mesmos atributos e métodos da classe **Celular**. Um método poderia ser o de receberLigação() um outro método poderia ser fazerChamada() e assim por diante.

Nós sabemos que um celular é envolto em uma carcaça (caixa) e para fazermos uma ligação utilizamos o teclado. Podemos então concluir que os componentes do celular estão encapsulados dentro de uma caixa e que para conseguirmos acessar esses componentes (para fazer uma ligação, por exemplo) devemos utilizar o teclado que é uma interface de comunicação entre o usuário e os componentes internos do celular. Muitas pessoas não têm nem ideia de como é o funcionamento interno de um celular (não sei se você sabe como funciona internamente o celular), mas todas sabem utilizar o celular mesmo sem saber como é seu funcionamento interno.

Isso também acontece na programação orientada a objetos. Eu não preciso saber como uma classe foi codificada eu tenho apenas que saber como enviar dados para ela e como receber os dados dela.


A figura abaixo ilustra o encapsulamento:





1.2 O modelo conceitual

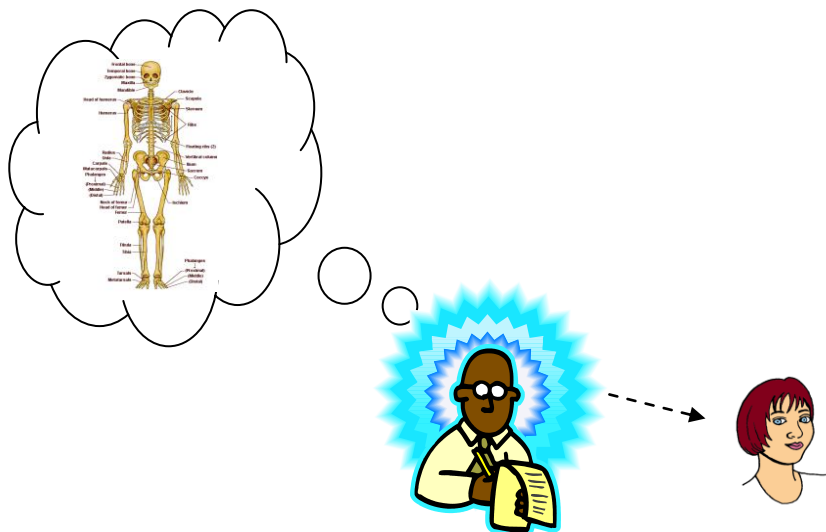
A construção de um modelo conceitual é feita através da análise do **domínio** da aplicação (mundo real) para identificar sua estrutura e **capturar** as entidades, ações, atributos que forem importantes para a descrição do domínio. Observe a figura abaixo e veja como funciona a abstração de dados ou a construção do modelo conceitual:



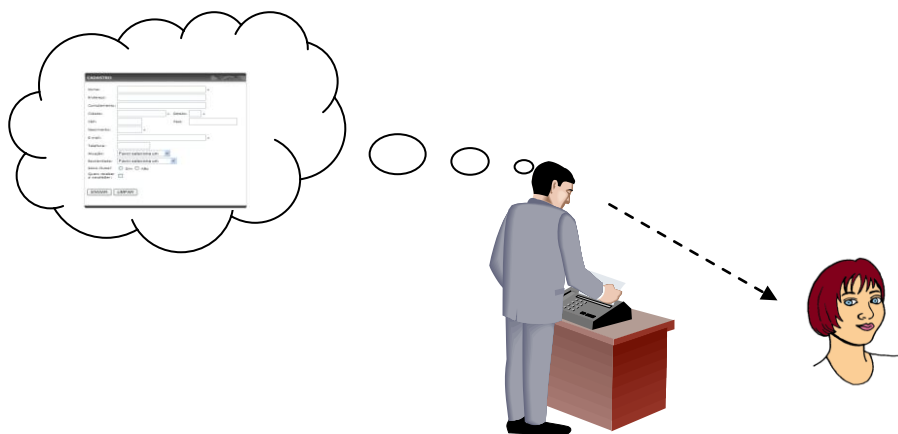
Podemos notar que o observador  analisa o domínio  e retira dele o que realmente interessa para a construção do modelo. Isso é passado

para o programador  que cria o **modelo conceitual** ou a **classe** .

A abstração de dados ou a criação do modelo conceitual depende muito do contexto. Se uma pessoa for analisada por um médico, ele abstrairá ou criará o modelo conceitual dessa pessoa, com dados como peso, idade, altura, tipo de sangue, se toma remédio ou não etc.



Caso o observador que vá examinar a mesma pessoa seja um contador, ele irá criar um modelo conceitual dessa pessoa com nome, endereço, cpf, telefone, número da conta, número da carteira de trabalho, etc.



1.3 Herança

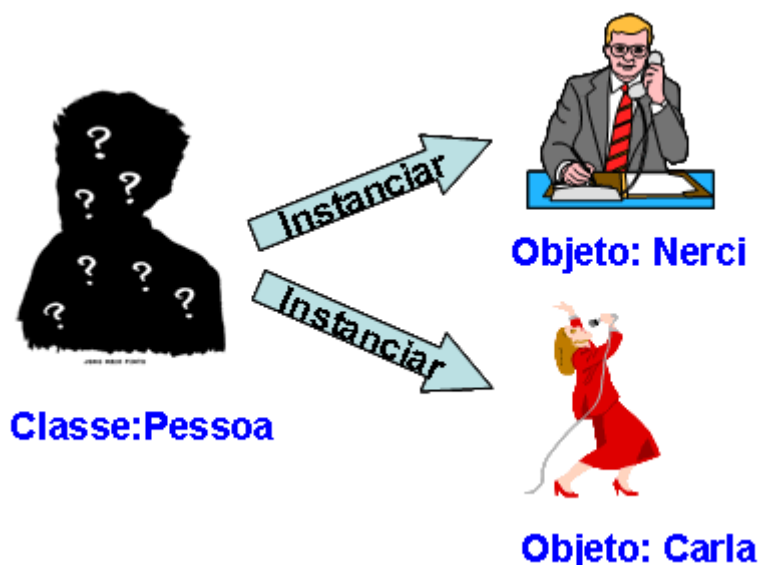
Como veremos em um estudo mais profundo adiante, a herança é uma técnica utilizada na Programação Orientada a Objetos que facilita a criação de novas classes a partir de uma classe já existente (reutilização de código). A classe que fornece os atributos e métodos para serem herdados é chamada de classe mãe ou superclasse e a classe que herda os atributos e métodos é chamada de classe filha ou subclasse. As subclasses têm **mais recursos** que suas superclasses. A herança caracteriza o relacionamento **é um**. Não se preocupe veremos isso mais a frente no nosso curso.

1.4 Os objetos

Um objeto é **concretização** de uma classe. Suponha que tenhamos uma classe Pessoa e nessa classe temos os seguintes atributos: nome, altura, cor dos cabelos, profissão. Essa classe será escrita em Java da seguinte maneira:

```
class Pessoa{  
    String nome;  
    float altura;  
    String corDosCabelos;  
    String profissao;  
}
```

Para **concretizar** uma classe ou criar um objeto devemos **instanciá-la**. A figura abaixo ilustra a instanciação de uma classe Pessoa.



Um objeto (ou classe instanciada) é cada uma das entidades identificáveis num dado domínio de aplicação (mundo real). Existem dois tipos de objetos que são os

objetos concretos (Carla, Nerci...) e objetos abstratos tais como: endereço, estilo da casa (barroco, gótico, romano, colonial...), valor (em unidade monetária), etc.

Os objetos podem

- ser compostos por outros objetos
- herdar atributos e métodos de outros objetos



Quando um objeto tem como parte outro objeto, chamamos esse relacionamento de **tem um** e quando um objeto herda atributos de outro objeto, chamamos esse relacionamento de **é um**.



Suponhamos um gerente de banco. Temos uma classe **Gerente**, o gerente **é uma** pessoa então poderíamos ter uma classe **Pessoa** e a classe **Gerente** que irá herdar os atributos e métodos da classe **Pessoa**. Esse tipo de relacionamento entre **Gerente** e **Pessoa** é chamado de relacionamento **é um**.

Um gerente também pode ter uma conta em um banco. Temos então uma classe **Conta** e o Gerente **tem uma** Conta. Esse tipo de relacionamento é chamado de relacionamento **tem um**. Veremos isso com mais detalhes a frente.

1.5 Conceitos de programação

Como nós já sabemos, um programa de computador é um conjunto de instruções, que dizem quais tarefas e como o computador deve executar essas tarefas. Esse programa foi construído utilizando um ou vários tipos de compiladores. Nós já sabemos também que um compilador executa códigos fontes que são escritos em uma determinada linguagem de programação.

Cada linguagem de programação tem seu propósito definido sendo que algumas delas atuam diretamente no hardware e movem bits de um lado para outro. Essas linguagens são as chamadas “Linguagens de baixo nível”. Existem também as linguagens de alto nível que também movem bits de um lado para outro, mas de forma mais fácil para o programador.

Linguagens como C, C++, Pascal e Java são consideradas linguagens de alto nível e os programas digitados em linguagens de alto nível são chamados de código fonte, o qual é traduzido para um programa de baixo nível em um processo chamado compilação ou interpretação.

1.6 Introdução à programação

As linguagens de programação são formadas por palavras que agrupadas em frases produzem um significado. As palavras utilizadas em uma linguagem de programação são denominadas palavras-chave e as frases construídas com as palavras-chave são denominadas de estruturas de programação.

Logo um programa é constituído de palavras-chave e estruturas de programação que respeita as regras da linguagem, elaboradas de forma que sejam mais facilmente entendidas por um ser humano.

A linguagem que usamos no nosso dia-a-dia e a linguagem de programação possuem uma sintaxe definidas por regras.



Quais são as regras da linguagem que usamos em nosso dia-a-dia....

1.7 Tipos de programação

1.7.1 Programação linear

São programas que quando são executados respeitam uma sequência de passos executados um atrás do outro com início e fim específicos. Um exemplo bem fácil é a linguagem Basic.

A programação linear é fácil de ser utilizada, mas não é indicada para programas muito complexos. Os códigos dos programas lineares são imensos e difíceis de serem compreendidos.

1.7.2 Programação estruturada

Dividir para conquistar. Você já ouviu falar essa frase algumas vezes (ou não!). Bem essa frase se aplica bem na construção de programas de computador. Quando temos um programa muito grande dividimos o programa em pequenas tarefas bem definidas, distribuímos para a equipe e então o tempo gasto para desenvolver o sistema se torna bem menor.

Essa abordagem deu início a programação estruturada. Quando dividimos o código em tarefas temos um processo chamado de modularização. Os pequenos módulos são chamados de procedimentos e/ou funções que executam determinada tarefa. Cada módulo recebe um nome o qual é utilizado como chamada para o procedimento ou função. A linguagem C é o exemplo de programação estruturada.

1.7.3 Programação orientada a objetos

A programação orientada a objetos também divide o problema em partes que são os objetos. Cada objeto tem seus atributos e métodos e podem interagir como outros objetos através do envio de mensagem. Os objetos podem ser reutilizados em outros programas e podem ser extensíveis (se tornarem especializados).

1.8 Introdução à tecnologia Java

Como iremos ver agora a tecnologia Java não é apenas uma linguagem de programação em sim uma plataforma de desenvolvimento.

1.8.1 Linguagem de programação Java

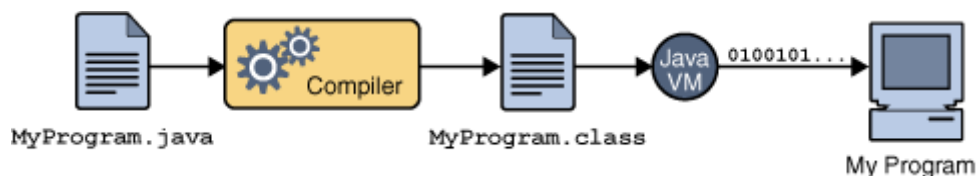
Java é uma linguagem de programação de alto nível e tem como características:

- simples
- orientada a objetos
- distribuída
- multithread
- dinâmica
- arquitetura neutra
- alta performance
- robusta
- segura

Agora que você irá entra no mundo da orientação a objetos utilizando a linguagem Java você verá que o código fonte é escrito em um editor de textos simples (bloco de notas, por exemplo) e salvo com a extensão **.java**.

Depois de compilar esse código em alto nível um arquivo **.class** é criado. O arquivo **.class** possui os *bytecodes* (códigos de baixo nível) que serão interpretados pela Máquina Virtual Java (JVM).

Quando invocamos o interpretador da linguagem o interpretador lê os *bytecodes* e executa a aplicação. Veja a figura abaixo:



Fonte www.sun.com

O desenho acima mostra o mecanismo de compilação e interpretação de um código em Java. Temos o código fonte chamado *MyProgram.java* esse código é compilado

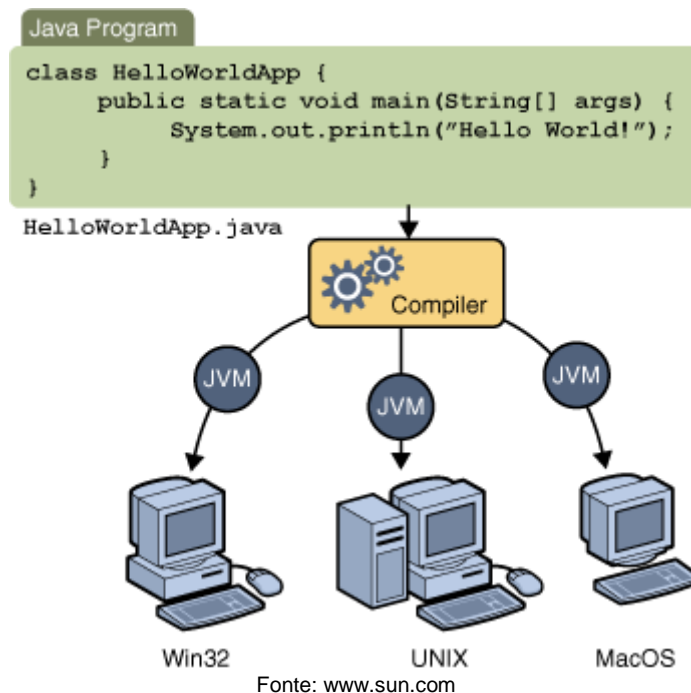
pelo compilador Java (javac) que transforma o conteúdo do arquivo *MyProgram.java* em um arquivo chamado *MyProgram.class* que possui os *bytecodes* e esses *bytecodes* são interpretados pela máquina virtual Java (JVM) quando invocado o interpretador da linguagem (java).

Conclusão

Se o processo de execução de um programa em Java passa por dois processos sendo o primeiro de compilação e o segundo de interpretação então quando queremos compilar um programa em Java e executar um programa escrito em Java podemos ter dois tipos de erro. Um erro de compilação ou um erro de execução.

Para que a linguagem Java se tornasse uma linguagem multiplataforma foi necessário implementar uma máquina virtual Java para cada sistema operacional.

A máquina virtual Java é um programa desenvolvido para cada tipo de sistema operacional e é ela quem faz a interpretação dos *bytecodes*.



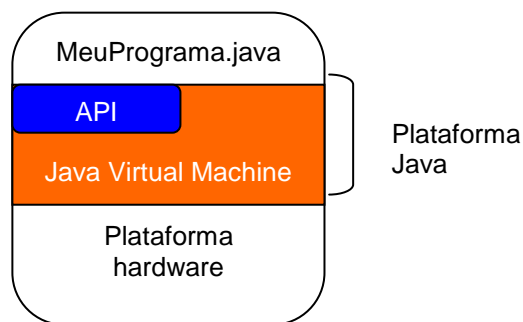
1.8.2 A plataforma de desenvolvimento Java

Uma plataforma é um ambiente de *hardware* ou *software* onde os programas são executados. As plataformas mais conhecidas são Microsoft Windows, Linux, Solaris OS e Mac OS. Algumas plataformas são descritas como a combinação entre sistemas operacionais e *hardware*. A plataforma Java é diferente de todas as plataformas, pois ela é feita somente de *software* e é executada em cima de todas as outras plataformas de *hardware*.

A plataforma Java possui dois importantes componentes:

- Java Virtual Machine - JVM ou Máquina Virtual Java (MVJ);
- a Java *Application Programming Interface*(API)

A Máquina Virtual Java pode ser executada em várias plataformas e a API é uma grande coleção de componentes de *software* prontos que têm grande poder de compatibilidade. Eles são agrupados em bibliotecas de classes relacionadas e interfaces. As bibliotecas são chamadas de pacotes (*package*). Vejamos a figura abaixo:



Fonte: o autor

1.9 Programação orientada a objetos

Vamos agora falar um pouco mais sobre a Programação Orientada a Objetos. A programação orientada a objetos (POO) representa uma mudança no enfoque da programação, na forma como os sistemas eram vistos até então. A POO é um novo paradigma de programação que vem revolucionando todos os conceitos de projeto e desenvolvimento de sistemas existentes anteriormente.

“O enfoque tradicional para o desenvolvimento de sistemas e, por consequência, para a programação, baseia-se no conceito de que um sistema é um **conjunto de programas inter-relacionados** que atuam sobre um determinado conjunto de programas inter-relacionados que **atuam sobre um determinado conjunto de dados que se deseja manipular** de alguma forma para obter os resultados desejados.

O enfoque da modelagem de sistemas por objetos procura enxergar o mundo como um conjunto de objetos que interagem entre si e apresentam características e comportamento próprios representados por seus atributos e suas operações. Os atributos estão relacionados aos dados, e as operações, aos processos que um objeto executa.”[1]

Esse enfoque justifica-se, pelo fato de que objetos já existem na natureza antes de haver qualquer tipo de negócio envolvido ou qualquer tipo de sistema para controlá-los. Equipamentos, pessoas, materiais, produtos, peças, ferramentas, combustíveis etc. existem por si sós e possuem características próprias determinadas pelos

atributos (nome, tamanho, cor, peso...) e um determinado comportamento no mundo real relacionado aos processos que eles sofrem.

1.10 Programando em JAVA

Exemplo da classe chamada Empregado que possui os atributos código e salário em Java.

```
class Empregado{  
    int codigo;  
    float salario;  
}
```

Observe que para criar uma classe em Java, utilizamos a palavra reservada **class** seguido do nome da classe (Empregado). O nome da classe sempre é iniciado com letra maiúscula. Uma chave { é aberta para a criação do corpo da classe. Os atributos código e salário estão no corpo da classe e são do tipo int e float respectivamente. Uma chave } é utilizada para fechar o corpo da classe.

Um programa em Java é definido pela presença de um método principal chamado **main**.

O código abaixo mostra uma classe de programa. Veja que ela possui o método *main*.

```
public class MeuPrograma{  
    public static void main(String[] args){  
        //corpo do método;  
    }  
}
```

O método **main** que está na classe MeuPrograma por definição da linguagem Java tem que ser público (public) static (estático) não retornar nada (void) ser chamado de main (principal) e possuir um vetor de String como argumento. Veremos o que significam as palavras *public* - *static* e *void* mais tarde.



Leia o item 2.8 e veja como instalar o JDK e o Eclipse.

1.10.1 Gravar classes em Java

Para salvar a uma classe em Java, o nome do arquivo tem que ser o mesmo nome da classe - inclusive letras maiúsculas e minúsculas - com a extensão .java. A classe acima deverá se chamar **MeuPrograma.java**.

1.10.2 Compilar programas em Java utilizando o eclipse

Escrevendo no console com Java

Para escrever uma mensagem no console em Java, utilizamos o comando

```
System.out.println("mensagem");  
ou  
System.out.print("mensagem");
```

O primeiro comando imprime uma mensagem no console e salta uma linha, o segundo imprime uma mensagem na tela e continua na mesma linha.

O programa abaixo escreve Java é fácil na tela.

```
class EscreveNaTela{  
    public static void main(String[] args){  
        System.out.println("Java é fácil");  
    }  
}
```

Utilizando o que você aprendeu no vídeo de como compilar um arquivo no Eclipse, digite e compile o código acima.

Lendo dados com Java

Entrada de dados via teclado - com interface gráfica

Para fazer a leitura do teclado, podemos utilizar o pacote gráfico de Java. O programa abaixo mostra como fazer a leitura de dados do teclado e mostra os resultados em uma janela gráfica:

```
import javax.swing.*;  
class EntradaDeDados{  
    public static void main(String[] args){  
        //declara x do tipo inteiro  
        int x;  
        //declara sx do tipo string  
        String sx;  
        //faz a leitura e grava em sx  
        sx = JOptionPane.showInputDialog("Digite o valor de x");  
        //converte sx(string) inteiro  
        x = Integer.parseInt(sx);  
        //mostra o resultado na janela gráfica.  
        JOptionPane.showMessageDialog(null,"Valor de x: " +x);  
    }  
}
```



```
}
```

O código acima é bem interessante e apavorante para quem o vê pela primeira vez, mas é muito fácil, vejamos.

A linha `import javax.swing.*;` como a própria instrução nos diz importará * (todas) as classes do pacote *swing* que está dentro do pacote *javax*. Pacotes são os diretórios onde são organizadas as classes em Java. Usaremos as classes durante o programa.

A linha `class EntradaDeDados{` define que *EntradaDeDados* é uma classe (lembro que o nome desse arquivo tem que ser *EntradaDeDados*).

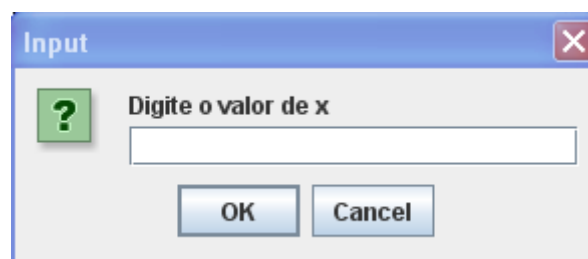
A linha `public static void main(String[] args){` diz que essa classe (*EntradaDeDados*) é uma classe principal e que o código que escrevemos é considerado uma aplicação Java. (Se fosse um Applet não teria essa linha – *Applets* são códigos em Java que são executados em *Browsers*).

As linhas `int x;` e `String sx;` declaram *x* como sendo do tipo primitivo inteiro e *sx* como sendo do tipo classe *String*. *String* é uma classe do pacote *Java.lang* que é um pacote padrão do Java. A classe *String* possui atributos e métodos como veremos mais adiante no curso. Ela serve por hora para armazenar dados do tipo *String* que são letras, caracteres especiais e números (não podemos operar com eles).



Toda vez que fizermos uma leitura de um dado do teclado utilizando a interface gráfica, esse dado é lido como um tipo *String*.

A linha `sx = JOptionPane.showInputDialog("Digite o valor de x");` emite uma caixa gráfica solicitando a um valor para *x*. O valor lido pelo teclado quando preenchido o campo da janela gráfica é atribuído à variável *sx*.



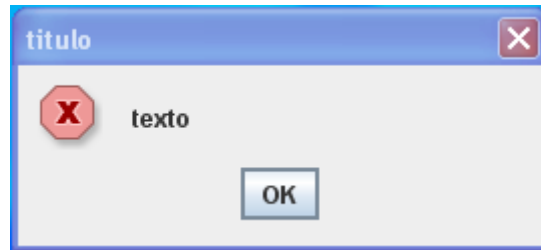
Depois de preenchido o valor e pressionado o botão OK então o valor é enviado como um *String* para a variável *sx*.

A linha `x = Integer.parseInt(sx);` faz a conversão de *sx* que é do tipo *String* para um tipo inteiro. Para isso utilizou o método `parseInt()` da classe empacotadora (Wrapper) *Integer*.

A linha `JOptionPane.showMessageDialog(null, "Valor de x: " + x);` utiliza o método `showMessageDialog` da classe `JOptionPane` para escrever na tela. Os parâmetros do método são: Depois ela utiliza uma string “valor de x: “ que pode ser qualquer texto. O sinal de + nesse caso é para concatenar o texto “Valor de x: “ com o valor atribuído a x.

O método `showMessageDialog()` tem os seguintes parâmetros:

`JOptionPane.showMessageDialog("null", texto, titulo, ícone);`







O primeiro parâmetro é o `null` que irá exibir a caixa de texto no centro da janela. Ela pode ser alterada como iremos ver quando formos programar utilizando a interface gráfica. O terceiro parâmetro é o título que será exibido na janela (ao invés de "Message")

O segundo parâmetro é o texto que será a mensagem a ser enviada para o usuário.

O terceiro parâmetro é o título que será o título da caixa de mensagem.

O quarto parâmetro controla o ícone apresentado à esquerda da janela:

- `JOptionPane.PLAIN_MESSAGE` - nenhum ícone
- `JOptionPane.ERROR_MESSAGE` - ícone de erro 
- `JOptionPane.INFORMATION_MESSAGE` - ícone de informação 
- `JOptionPane.WARNING_MESSAGE` - ícone de aviso 
- `JOptionPane.QUESTION_MESSAGE` - ícone de interrogação 

Veja que o quarto parâmetro é uma constante da classe `JOptionPane`. Quando escrevemos um símbolo todo em maiúsculo em Java isso significa que o valor inserido nele será uma constante.

Observe que no exemplo anterior utilizamos apenas dois parâmetros para o método `showMessageDialog()` e agora foi mostrado que podemos usar até quatro parâmetros para o método `showMessageDialog()` e o nome do método não foi alterado. Isso é o *polimorfismo* que temos em Java. Bacana não? Você ter apenas

um nome de método com parâmetros diferentes (os parâmetros são também chamados de assinatura do método).



Pesquise o que é uma classe Wrapper e quais tipos de tipos primitivos possuem seu equivalente na classe Wrapper.

JOptionPane é uma classe do pacote javax.swing e showDialog("Digite o valor de x") é o método da classe JOptionPane.

Cada classe tem seu conjunto de métodos. Como saber quais são os métodos de uma classe?

Devemos usar a documentação da API (*Application Programming Interface* ou Interface de Programação de Aplicativos) .



Você poderá encontrar a documentação de todas as classes da linguagem Java em <http://java.sun.com/javase/6/docs/api/>



O vídeo “Como usar a API do Java” mostra como utilizar a API do Java

1.11 Estruturas de controle em Java

1.11.1 Estrutura condicional: if

```
if(condição) {  
    //comandos;  
}  
ou  
  
if(condição) {  
    //comandos1;  
}  
else {  
    //comandos2;  
}
```

A estrutura condicional simples *if* é uma estrutura que toma uma decisão dependendo da condição apresentada. Quando essa verificação de condição é verdadeira, o primeiro bloco de comandos é executado (Bloco é o código que está entre as chaves).

A estrutura condicional composta *if else* é uma estrutura que também toma uma decisão dependendo da condição apresentada e quando a verificação de condição é verdadeira ela executa o bloco logo abaixo do comando *if* caso a condição seja falsa, é então executado o bloco que está logo abaixo da instrução *then*. Lembrando que *if* significa **se** e *then* significa **então**.



Escreva um programa em Java utilizando a IDE Eclipse para escrever um programa que declare uma variável tamanho1 do tipo inteiro e atribua a ela o valor de 203 e declare também uma variável chamada tamanho2 do tipo inteiro e atribua a ela o valor 832. Compare as variáveis e envie a mensagem de qual variável possui o maior valor.

1.11.2 Estruturas de seleção múltipla: switch

```
switch(opção) {  
    //opção pode ser do tipo byte, short, int ou char  
    case opcao1:  
        comandos1;  
        break;  
    case opcao2:  
        comandos2;  
        break;  
    .  
    .  
    case opcao:  
        comandosn;  
        break;  
    default:  
        comandos;  
}
```

A estrutura de seleção múltipla *switch* é utilizada para verificar se uma opção é verdadeira. Ela compara a entrada (no exemplo acima – opção) com o que está escrito na instrução *case* caso seja igual então o bloco de comando logo abaixo é executado. Vi que você viu uma instrução *break* (parar) em todos os casos. Essa instrução *break* é utilizada para que o programa não leia todas as opções possíveis escritas no código. O *break* irá interromper o fluxo de dados e voltar para onde ele foi chamado. Quando uma opção não é válida então o bloco de comandos que está dentro de *default* será executado.



Escreva um programa em Java utilizando a IDE Eclipse para ler o código de um produto e escrever o nome de acordo com o código do produto seguindo a lista abaixo:

	Código do produto	Nome do produto
	PenDrive	
	DVD	
	CD	
	Outro qualquer	opção inválida

1.11.3 Estruturas de repetição: while

```
while(condição) {
    comandos;
}
```

A estrutura de repetição *while* ou enquanto é utilizada quando queremos que um determinado código seja executado várias vezes. Com essa estrutura podemos definir quantas vezes um bloco de código pode ser executado. Observe que o bloco de código só será executado caso a condição seja verdadeira.



Escreva um programa em Java utilizando a IDE Eclipse para escrever um texto lido do teclado 450 vezes na tela. Utilize a saída do console.

1.11.4 Estruturas de repetição: do..while

```
do{
    comandos;
}while(condição);
```

A estrutura *do..while* também é uma estrutura de repetição que faz a validação da condição no final, isto é, primeiro o bloco de comandos é executado depois a condição é validada. Caso a condição seja falsa o programa para de executar o *looping* e vai para a próxima linha caso a condição seja verdadeira ele continua a executar o *looping*.



Escreva um programa que lê um valor do teclado se esse valor for par então ele solicita novamente que um outro valor seja lido se ele for par solicita novamente que um outro valor seja lido e assim sucessivamente. Caso você digite um valor ímpar então ele sai do programa e emite a mensagem "Tchau".

1.11.5 Estruturas de repetição: for

```
for(inicializa; condição; incremento){  
    comandos;  
}
```

A estrutura de repetição *for* também executa um bloco de comandos enquanto sua condição é verdadeira. A diferença entre a estrutura de repetição *for* e as estruturas *do..while* e *while* é que ela já tem um contador interno e as outras duas esse contador deve ser colocado dentro do bloco de comandos.



Escreva um programa para mostrar os números ímpares de 1 a 1000. Utilize a saída do console.



Consulte o livro "Java: Como Programar" - Deitel nos capítulos 4 e 5 para revisar as estruturas de controle.

Conclusão

- Uma classe possui um nome, atributos e métodos.
- Os atributos são as propriedades ou características da classe.
- Métodos são as funcionalidades das classes e são divididos em métodos construtores e métodos operacionais.
- Podemos utilizar os métodos para acessar os atributos de uma classe.
- Uma classe é um conceito e para construirmos um modelo conceitual fazemos a abstração de dados de um determinado domínio.
- A construção do modelo conceitual depende do contexto.
- Uma característica importante da orientação a objetos é a herança.
- Um objeto é construído sempre a partir de uma classe e uma classe pode construir vários objetos.
- Quando construímos um objeto dizemos que estamos instanciando uma classe.
- A linguagem Java é utilizada para escrever programas orientados a objetos onde o elemento principal do sistema é sempre o objeto.
- Objetos trocam mensagens entre si.
- Java é multiplataforma.
- Java utiliza a máquina virtual Java para se tornar uma linguagem multiplataforma.
- Java é uma linguagem compilada e interpretada.
- Depois de compilado Java gera um código, class chamado de bytecodes.

1.12 Exercícios



Agora que você já conhece basicamente o funcionamento de Java, vamos aos exercícios.

1. O que é uma classe?
2. Quais as partes de uma classe?
3. Quando utilizamos um método construtor?
4. Quando utilizamos um método operacional? Dê exemplo.
5. Qual a diferença entre um método construtor e um método operacional?
6. Crie o modelo conceitual de um automóvel.
7. Como criamos um objeto?
8. O que é “modularização” na programação estruturada?
9. O que é necessário para que uma classe seja um programa na linguagem Java?
10. Qual o comando utilizado para escrever uma mensagem em uma janela gráfica utilizando Java?
11. O que é um pacote?
12. Qual pacote devemos importar para utilizarmos a impressão de dados em uma janela gráfica?
13. Qual método da classe *JOptionPane* devemos utilizar para ler os dados do teclado?

No computador

Agora que você já conhece basicamente o funcionamento de Java, vamos aos exercícios práticos.

1. Escrever um programa em Java que mostre seu nome na tela utilizando a interface gráfica.
2. Escrever um programa em Java que mostre a soma de dois números quaisquer. Os números devem ser lidos do teclado utilizando a interface gráfica.
3. Escreva um programa que mostre a figura abaixo na tela usando o console - não é necessário utilizar comandos de repetição para este exercício.

```
*  
***  
*****  
*****  
*****  
*****  
*****
```

4. Escreva um programa para calcular o n-ésimo termo da série de Fibonacci. A entrada deverá ser utilizando a janela gráfica. Exemplo: Se o usuário digitar 8 deverá ser apresentado 1 1 2 3 5 8 13 21 e assim por diante.
5. Escreva um programa para calcular a divisão de dois números reais (*float*) quaisquer, utilizando a entrada gráfica.
6. A empresa QSD Ltda. concedeu um bônus de 20 por cento do valor do salário a todos os funcionários com tempo de trabalho na empresa igual ou superior a cinco anos e dez por cento aos demais. Ler o salário, o tempo de trabalho, calcular e exibir o valor do bônus para 100 funcionários.
7. Ler 20 números fornecidos pelo usuário calcular e exibir a média. (utilize do-while)

Bons estudos e qualquer dúvida não deixe de entrar em contato!

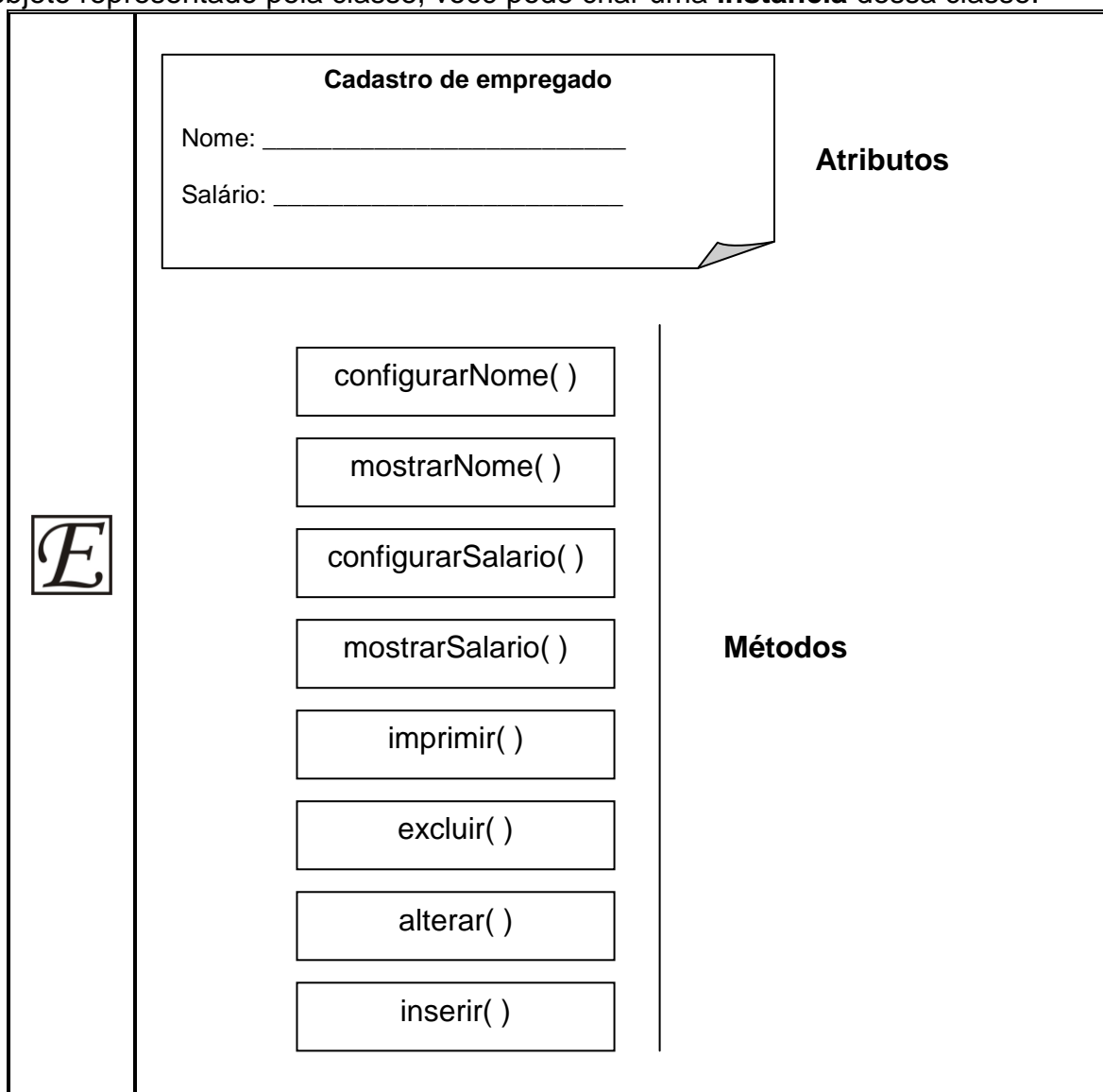
Estou a disposição para esclarecimentos.

1.13 Um estudo mais profundo sobre classes

Ao final da aula você será capaz de:

- Instanciar uma classe em Java;
- Entender o conceito de construtores;
- Entender o conceito de métodos operacionais;
- Construir métodos construtores e operacionais;
- Utilizar métodos construtores;
- Utilizar métodos operacionais;
- Fazer acesso direto e indireto a atributos e métodos
- Acessar membros de uma classe instanciada;

No capítulo anterior de POO vimos que uma **classe** é um gabarito que define **atributos** e **métodos** de um **objeto** do mundo real. Sempre que precisar de um objeto representado pela classe, você pode criar uma **instância** dessa classe.



A **instância de uma classe** ou um **objeto** possui os mesmos atributos e métodos de uma classe, embora cada **objeto** contenha a sua própria cópia desses atributos.

1.13.1 Definindo uma classe

A criação de uma classe define quais são os atributos e os métodos que fazem parte de uma classe específica. Vejamos o código abaixo:

```
class Empregado{  
    String nome;  
    double salario;  
}
```

Uma definição de classe possui três partes a saber:

- Modificador - **será visto mais tarde**;
- A palavra chave **class**;
- O nome da classe sempre iniciando com letra maiúscula;
- O corpo da classe.

A palavra chave **class** informa o compilador que aquilo que vem a seguir é uma classe.

O **nome** da classe é um símbolo único atribuído pelo programador para identificar uma classe específica e diferenciá-las das outras classes. O nome da classe deve sempre ser relacionado com o objeto do mundo real que está sendo emulado. No exemplo anterior Empregado emula um funcionário com os seus atributos e métodos.

O corpo da classe é a parte da definição de classe identificada por chaves de abertura e de fechamento **{ }**. Os atributos e os métodos são definidos dentro da área delimitada por essas chaves.

```
class NomeDaClasse {  
    // Aqui colocamos os atributos.  
    // Aqui colocamos os métodos.  
    // Barras duplas são para comentário em Java.  
    /* Os atributos e métodos ficam entre as chaves  
       que delimitam o corpo da classe.*/  
    /* Significa comentário  
       também */  
}
```

1.13.2 Variáveis de instância ou Atributos

Uma variável é uma referência a uma posição de memória que pode ser utilizada para armazenar dados.

Um atributo de uma classe é uma variável conhecida também como **variável de instância**. O atributo referencia um endereço específico dentro do bloco de memória reservado para a instância da classe. O atributo é utilizado para armazenar dados na memória. Ela é também chamada de **variável de instância** porque pertence ao grupo de atributos de uma instância da classe. O atributo só irá alocar memória depois que a classe é instanciada, isto é, depois que criamos um objeto. Por isso o atributo recebe também o nome de variável de instância.

```
class Empregado{  
    String nome;  
    double salario;  
}
```

Uma variável e um atributo são semelhantes com algumas exceções:

- A variável é declarada através de uma instrução incluída no programa. A memória é reservada quando essa instrução de declaração é executada.
- O atributo é declarado dentro de uma **definição de classe**. A memória só é reservada quando uma instância específica da classe é declarada, porque a definição de classe é apenas um gabarito, ao passo que a instância da classe é uma versão digital de um objeto real.

No exemplo acima temos dois atributos um do tipo **double** e outro do tipo **String**. Observe que String é uma classe e está escrita com a letra inicial em maiúscula e **double** é um tipo primitivo.

Como declarar um atributo?

Um atributo é declarado dentro de uma classe. Em Java usamos:

- Modificador - **será visto mais tarde**;
- Tipo de dados;
- Nome da variável de instância;
- Ponto-e-vírgula;

Tipos de dados

É a palavra chave que informa ao computador qual é o tipo de dados que você quer armazenar. A tabela abaixo mostra os tipos de dados primitivos utilizados em Java.

Tipo de dados	Faixa de valores
byte	-128 a 127
short (curto)	-32.768 a 32.767
int (inteiro)	-2.147.483.648 a 2.147.483.647
long (longo)	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
char (caractere)	65000
float (vírgula flutuante)	3.4e-038 a 3.4e+038
double (duplo)	1.7e-308 a 1.7e+308
boolean (booleano)	true (verdadeiro) ou false (falso)

Nome do atributo

É um símbolo atribuído pelo programador a uma variável de acordo com dados armazenados na posição de memória correspondente. O nome do atributo deve sempre começar com letras **minúsculas**. Se o nome for composto por mais de um nome, o primeiro começa com letras minúsculas e o próximo nome começa com letra maiúscula **sem espaços** entre eles.

1.13.3 Métodos

Os métodos simulam o comportamento de um objeto dentro do programa. Eles são chamados de métodos membro. Toda vez que queremos que um determinado objeto retorne ou atribua um valor a um atributo, por exemplo, utilizamos o método membro para fazer isso.

Existem dois tipos de método:

1. Métodos construtores
2. Métodos operacionais

O método membro é um conjunto de instruções que implementam um comportamento específico, definido através dos itens a seguir:

- Modificador - **será visto mais adiante**;
- Tipo de retorno;
- Nome do método;
- Lista de argumentos do método (opcional);
- Corpo do método delimitado por chaves;

Tipo de retorno

Alguns métodos não retornam nenhum valor para o programa (instrução que os acionou), outros métodos devem retornar dados para o programa (instrução que os acionou) para que o resultado seja conhecido ou manipulado. Os dados retornados por um método são conhecidos como **valor de retorno**.

A sintaxe para a construção de um método é:

```
modificador tipo_de_retorno nomeDoMetodo([argumentos]);
```

onde:

modificador - **será visto mais adiante**;

tipo_de_retorno - pode ser qualquer valor primitivo, um objeto ou *void*. Quando o tipo de retorno é **void** significa que o método não irá retornar nada. Quando utilizamos um tipo de retorno primitivo ou um objeto devemos inserir na última linha

do corpo do método a instrução **return** seguida pela variável do "mesmo tipo" do valor de retorno.

nomeDoMetodo - O nome do método reflete o tipo de comportamento que ele executa, ele é um símbolo utilizado para referenciar uma determinada posição de memória. O nome de um método sempre inicia com letras minúsculas e se for um nome composto as iniciais dos próximos nomes sempre serão maiúsculas.

argumentos - alguns métodos necessitam de dados externos para executar um comportamento. Uma lista de argumentos é uma relação de dados externos ao método que são necessários para que ele execute um comportamento. Esses dados devem ser declarados entre parênteses que ficam logo após o nome do método. Cada argumento deve ser declarado individualmente e separados por vírgulas.

```
void setSalario(double salario){  
    this.salario = salario;  
}
```

Observe que no exemplo acima o tipo de retorno do método chamado setSalario é **void** e que no corpo do método **não** existe a palavra **return** pois **void** significa que o método não irá retornar nada.

Alguns métodos não necessitam de dados externos para executar um comportamento.

```
double getSalario(){  
    return this.salario;  
}
```

Observe que no exemplo acima o tipo de retorno do método é um **double** e que no corpo do método existe a palavra **return** que indica que a variável salario que será retornada é uma variável do tipo **double**.

Corpo do método

O corpo do método é a parte que contém as instruções que serão executadas quando o método for disparado. O corpo do método é delimitado por chaves de abertura { e fechamento }. Isso é chamado de **bloco de código**. As instruções são executadas sequencialmente dentro do corpo do método. As instruções terminam quando encontram uma instrução de retorno (*return*) ou o final do corpo do método marcado com }.

Definindo um método dentro da classe

```
class Empregado{  
    String nome;  
    double salario;  
  
    void setSalario(double salario){  
        //faz a atribuição do nome ao atributo nome  
        this.salario = salario;  
    }  
}
```

Existe uma convenção para nomear métodos que é a chamada JavaBeans. Ela diz que os métodos que irão “passar” valores para um atributo, ou seja, que irão configurar um valor para um atributo devem começar com `setSeguidoPeloNomeDoMétodo`. Quando queremos ler um valor de um atributo, ou seja, que o método “pegue”, ou melhor, retorne um valor devemos escrever o nome do método começando com `getSeguidoPeloNomeDoMétodo`.



Os prefixos `set` e `get` não têm poder especial de configuração ou leitura de atributos. Eles são somente um nome e se você não escrever o código no corpo do método de acordo com o que você quer eles não farão nada automaticamente.

Na linha:

```
f.setSalario(5000);
```

podemos ver que o nome do método possui em seu nome o prefixo `set` isso quer dizer que ele irá atribuir o valor 5000 a um atributo (porque está escrito no corpo do método para ele fazer isso).

Na linha:

```
f.getSalario();
```

Podemos ver que não existe parâmetro e que o método começa com o `get` dizendo para nós que esse método irá retornar um valor para quem o chamou. Observe que nos dois casos o nome do método começa com uma letra minúscula `setSalario` ou `getSalario` e a próxima letra é sempre maiúscula. Isso é a convenção de nomes de métodos do JavaBeans.

1.14 Combinando uma classe com um programa

Uma classe deve ser inserida fora do corpo principal do programa (fora do método `main()`).

```
class Empregado{
    String nome;
    double salario;

    void setSalario(double salario){
        this.salario = salario;
    }
}
```

Aqui teremos uma outra classe em um outro arquivo

```
public class MinhaAplicacao{
    public static void main(String[] args)
    {
        //instancia classe Empregado
        Empregado f = new Empregado();

        //utiliza o método da classe Empregado
        f.setSalario(5000);

    }
}
```

1.14.1 Definindo uma instância de uma classe

Uma instância de uma classe deve ser declarada antes que os atributos e métodos dessa classe possam ser usados num programa. Quando instanciamos uma classe, estamos criando o **objeto**. Para instanciarmos uma classe usamos o operador **new** que reserva espaço na memória para o objeto (nesse caso `Empregado`). Essa reserva de memória é dinâmica, pois só será reservada quando o computador executar o programa (runtime).

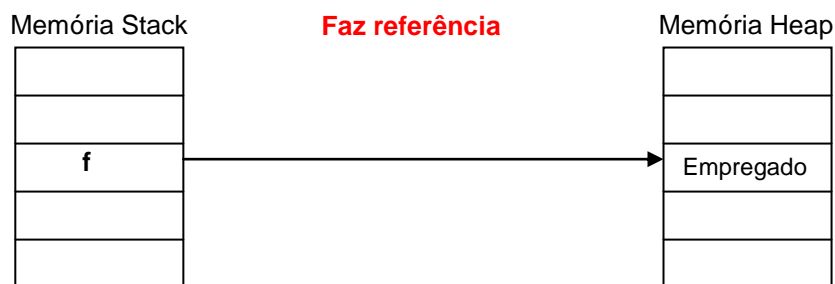
A reserva dinâmica de memória é feita no momento em que o programa é executado e não quando o programa é compilado. Para instanciarmos uma classe precisamos de uma **variável de referência** que é utilizada para nos referenciarmos a classe instanciada (**objeto**). Através da variável de referência, teremos acesso aos atributos e métodos desse objeto.

```
Empregado f = new Empregado();
```

Vamos analisar.

No exemplo acima temos **f** do tipo **Empregado** que é uma **variável de referência** ao objeto **Empregado** que está na memória. Uma referência **não** é uma instância de uma classe ela é apenas um símbolo que se refere a essa instância.

Observe que o **construtor `Empregado()`** foi utilizado na instanciação da classe. O **construtor** é um método membro especial e é acionado automaticamente sempre que uma instância de uma classe (**objeto**) é declarada. O construtor possui o **mesmo nome da classe**, não possui nenhum tipo de retorno (nem o *void*). No exemplo anterior **f** é a variável de referência. Veja a figura abaixo:



Na figura acima temos uma variável de referência **f** fazendo uma referência à classe **Empregado**. Sendo assim, através da variável de referência **f** podemos acessar os atributos e métodos não privados da classe **Empregado**.

1.14.2 Como acessar uma variável de instância

A variável de instância ou atributo pode ser acessado no programa usando-se o **nome** da variável, um operador **ponto** e o nome do atributo ou **método** que se pretende acessar.

Exemplo:

```
//acessando um método da classe Empregado.  
//Chamado de acesso indireto.  
f.setSalario(5000);  
  
//acessando um atributo da classe Empregado.  
//Chamado de acesso direto.  
f.salario = 5000
```

Podemos utilizar ou acionar métodos da mesma forma que utilizamos ou acionamos um atributo usando o **nome da variável de referência** o operador **ponto** e o **nome do método**. Sendo assim para utilizarmos o método **setSalario(Double salario)** da classe **Empregado** que foi instanciada (tornando-se um objeto) e referenciada pela variável **f** fazemos:

```
f.setSalario(5000);
```

Depois de executado o comando acima, o valor 5000 será atribuído ao atributo salário da classe empregado. (Essa atribuição está no corpo do método setSalario.)

Alguns métodos possuem uma lista de parâmetros (argumentos). Esses dados devem ser inseridos entre parênteses logo após a declaração do método. Alguns programadores fazem distinção entre parâmetros e argumentos, vejamos:

```
void setSalario(double salario){  
    this.salario = salario;  
}
```

O método acima possui um **argumento** *salario* do tipo *double* e no seu corpo esse valor é atribuído ao atributo *salario* que pertence a esta classe (*this*). Para acionar esse método utilizamos a variável de **referência**, nesse caso *f*, o operador ponto e o valor do argumento que agora é chamado de parâmetro. Vejamos:

```
f.setSalario(5000);
```

O valor que é passa do argumento é chamado de **parâmetro** (nesse caso 5000).

Caso o **parâmetro** seja um tipo String ele deve ser colocado entre aspas duplas.


Os métodos também podem ter um tipo de valor de retorno na sua declaração. Esse tipo deve ser colocado à esquerda do nome do método. Vejamos:

```
double getBonificacao(){  
    return salario 0.10;  
}
```


A parte do programa que aciona um método pode acessar o valor retornado pelo método. No exemplo acima, o tipo de retorno do método getBonificacao() é **double** isso quer dizer que a parte do programa que acionou esse método receberá um número do tipo double. O tipo de retorno pode ser qualquer tipo primitivo, um objeto ou void. O tipo void quer dizer vazio (não retorna nada). Em Java o uso de um tipo de retorno é obrigatório caso o método **não** seja *void*.



CONSTRUTORES NÃO TÊM TIPO DE RETORNO.

 <p>Conclusão</p>	<ul style="list-style-type: none"> • Vimos que para instanciar uma classe utilizamos o operador new • Aprendemos que os construtores são métodos de uma classe e não têm tipo de retorno e servem para inicializar os atributos no momento em que um objeto é criado (utilizando o new); • Construtores não são chamados durante o programa; • Os métodos operacionais têm tipo de retorno e caso não retornem nada utilizam a palavra void como sendo o tipo de retorno; • void indica que o método não irá retornar nada; • Os tipos de retorno podem ser primitivos, objetos ou void; • Nomes de atributos são inicializados com letra minúsculas; • Nomes de classe devem começar com letra maiúscula; • Utilizamos set para nomear métodos que irão configurar algum atributo; • Utilizamos get para nomear métodos que irão retornar algum valor; • O comando return é usado toda vez que temos um tipo de retorno em um método diferente de void.
---	---

1.15 Exercícios

	<p>Vamos agora fazer alguns exercícios</p> <ol style="list-style-type: none"> 14. O que é uma variável de referência? 15. O que é um atributo? 16. Quais os passos necessários para se criar instância de uma classe utilizando a linguagem Java? 17. Um método é definido por palavras chaves da linguagem respeitando uma sequência. Escreva quais são essas palavras. 18. Qual o comando para acionar um método membro. Sabendo que a variável de referência foi declarada com x e o método se chama imprimir()? 19. O que é um construtor? Quando ele é utilizado na Programação Orientada a Objetos?
---	---

	<p>20. Como acessamos um atributo a partir de um programa?</p> <p>21. De acordo com o JavaBeans qual a regra para escrever o nome de um método</p> <p>No computador</p> <p>8. Escreva uma classe chamada Aluno_UIEx8 que possua os seguintes atributos</p> <ol style="list-style-type: none">1. nome do tipo String;2. idade do tipo int; <p>9. Em um outro arquivo, crie um programa que instancie a classe Aluno.</p> <p>10. Mostre, utilizando a saída gráfica:</p> <ul style="list-style-type: none">• o valor do atributo nome;• o valor do atributo idade; <p>11. Adicione em seu programa o código para fazer a leitura:</p> <ul style="list-style-type: none">• Do nome e atribuir ao atributo nome;• Da idade e atribuir ao atributo idade; <p>12. Adicione em seu programa o código para mostrar os atributos nome e idade (na mesma janela gráfica);</p> <p>13. Inclua em seu programa um método chamado:</p> <ul style="list-style-type: none">• setNome que configura o nome;• setIdade que configura o idade; <p>14. Utilize os métodos acima para atribuir valores para os atributos. Utilize a interface gráfica;</p> <p>15. Inclua em seu programa um método chamado:</p> <ul style="list-style-type: none">• getNome que retorna o valor do atributo nome;• getIdade que retorna o valor do atributo idade;
--	---

UNIDADE II

Ao final da aula o aluno será capaz de:

- Criar interfaces gráficas com o usuário;
- Definir e utilizar os gerenciadores de leiaute;
- Construir interfaces gráficas.

2 GUI - INTERFACE GRÁFICA COM O USUÁRIO

Agora nós vamos construir nossas interfaces gráficas utilizando os pacotes gráficos da linguagem Java. É importante sabermos que quando formos criar nossas janelas gráficas e nossos componentes dentro da janela gráfica, todos eles são objetos.

Instanciaremos os objetos das classes gráficas que serão importadas utilizando o comando *import* e criaremos os objetos.

Veremos também a utilização da palavra *implements* que será utilizada para implementar uma *interface*. Interfaces nesse caso não é a interface com o usuário e sim uma interface com uma classe. Não se preocupe com *interface* agora ela será estudada mais tarde.

Quando criamos uma interface gráfica com o usuário criamos uma janela, nessa janela utilizamos um gerenciador de leiaute e no gerenciador de leiaute inserimos os objetos como botões, caixas de texto, barras de rolagem etc. Depois de inserido os objetos de controle devemos então escrever o que esses objetos irão fazer depois de preenchido (como uma caixa de texto), escolhido (como um *radio button* ou um *checkBox*) ou pressionado como um botão por exemplo.

Java possui dois pacotes importantes na construção de interfaces com o usuário que são:

- `java.awt` - Abstract Window Toolkit que é considerado um pacote **peso pesado** pois tem uma dependência com o sistema operacional na construção da interface gráfica. Eles utilizam os arquivos do sistema operacional para construir uma janela.
- `javax.swing` - que está no pacote `javax`. O pacote `javax` é uma extensão do pacote `java` e o **x** indica uma **extensão** do pacote `java`. Ele é considerado um pacote **peso leve**, pois não tem dependência com o sistema operacional local. O pacote `javax.swing` pode criar janelas com aparência do Unix em ambientes Windows por exemplo.

Para iniciarmos nossos estudos nos pacotes gráficos de Java vamos desenhar uma janela simples.

```
//importando os pacotes de java (bibliotecas)
import javax.swing.*;
public class Janela{
    //construtor
    Janela(){
        JFrame janela = new JFrame();
        //método que configura o tamanho da janela
        janela.setSize(300,200);
        //método que configura a localização da janela
        janela.setLocation(10,20);
        //método que configura o título da janela
        janela.setTitle("Cadastro");
        //método que mostra a janela
        janela.setVisible(true);
    }
}
```

Podemos observar que todas as classes (*) do pacote `javax.swing` foram importadas.

Na linha `JFrame janela = new JFrame();` foi instanciada ou seja criado o objeto `JFrame` referenciado pela variável de referência `janela`.

Na linha `janela.setSize(300,200);` estamos utilizando o método `setSize(300,200);` do objeto `JFrame` e assim sucessivamente.

Quero lembrar que:

Quando temos o arquivo com extensão **.class** de **JFrame** por exemplo no disco, temos uma **classe JFrame** (que um gabarito para construirmos objetos) e quando utilizamos o operador **new**, uma **cópia da classe JFrame** é alocada na memória e então temos agora o **Objeto JFrame**. Sabemos que um objeto tem todos os atributos e métodos que uma classe tem afinal ele é uma cópia da classe. Para acessarmos os atributos e métodos do objeto **JFrame** utilizamos a variável de referência `janela` com o operador ponto e o nome do atributo ou método desejado.

Podemos observar que a classe acima não é classificada como uma aplicação Java, pois não possui o método principal **main**.

O código abaixo cria uma aplicação Java que irá executar a classe acima. Observe que a classe acima tem um construtor chamado `Cadastro` (construtores têm o mesmo nome da classe) e tudo (criação do objeto, posicionamento da janela, dimensionamento da janela etc) acontece dentro dele. Logo toda vez que um objeto for instanciado (criado) todos os comandos dentro do construtor serão executados.

```
public class Cadastro{
    public static void main(String[] args){
        Janela cadastro = new Janela();
    }
}
```

A saída do programa acima será:



Temos então nossa primeira janela criada. Depois de desenhada a janela, temos que adicionar os componentes (botões, campos para texto etc.) à janela. Para isso temos que aprender sobre os **Gerenciadores de layout**

2.1 Gerenciadores de Layout

2.1.1 Construindo GUI's sem Gerenciadores de Layout

Java nos permite também construir GUI's sem utilizarmos um gerenciador de layout pré-definido. Quando configuramos o parâmetro do método **setLayout** para **null** estamos dizendo que não queremos usar nenhum gerenciador de layout e que os componentes serão inseridos na janela utilizando coordenadas **x, y, altura, largura**. Esses parâmetros são utilizados no método **setBounds()**. Vejamos o código:

```
import javax.swing.*;
import java.awt.*;
public class SemLayout{
    public JButton b1, b2, b3;
    public SemLayout(){
        JFrame janela = new JFrame("Sem layout");
        b1 = new JButton("Gravar");
        b2 = new JButton("Cancelar");
        b3 = new JButton("Sair");
        //configura o layout para null
        janela.setLayout(null);
    }
}
```

```
//configura a posição dos botões
b1.setBounds(10,20, 80, 25);
b2.setBounds(120, 20, 100, 25);
b3.setBounds(250, 20, 80, 45);
//adiciona os botões na janela
janela.add(b1);
janela.add(b2);
janela.add(b3);
//configura o tamanho e mostra a janela
janela.setSize(400, 300);
janela.setVisible(true);
}
```

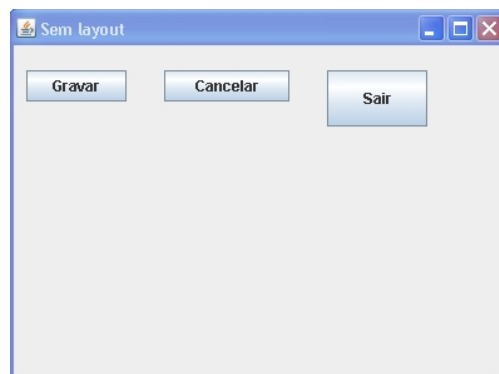
Podemos observar no código acima que no momento de escolhermos qual gerenciador de layout iríamos usar nós colocamos *null* sendo assim quando formos inserir os botões na tela temos que antes configurar as posições e tamanhos dos botões (método *setBounds()*). O método *setBounds()* possui quatro argumentos a saber:

1. posição na coordenada x do componente.
2. posição na coordenada y do componente.
3. Largura do componente.
4. Altura do componente.

Para executar o código usamos o código abaixo:

```
import javax.swing.*;
public class ExecutaSemLayout{
    public static void main(String[] args){
        SemLayout sl = new SemLayout();
    }
}
```

A saída será:



Os gerenciadores de leiaute são classes que definem como os objetos serão dispostos em um *Frame* ou janela.

2.1.2 GridLayout.

A classe `GridLayout` define uma matriz de componentes com linhas e colunas sendo tamanho de cada linha e coluna ser o mesmo, cada componente terá as mesmas dimensões. Cada componente adicionado ao container com o uso de `GridLayout` fica posicionado no índice seguinte da grade. Se a linha não estiver completa, ele é acrescentado à próxima coluna, vejamos:

```
import java.awt.*;
import javax.swing.*;
public class ExemploGrid{
    public JLabel r1, r2, r3, r4, r5, r6;
    public ExemploGrid() {
        JFrame janela = new JFrame("Gerenciadores de leiaute -
GridLayout");
        janela.setLayout(new GridLayout(3,2));
        r1 = new JLabel("rotulo 1");
        r2 = new JLabel("rotulo 2");
        r3 = new JLabel("rotulo 3");
        r4 = new JLabel("rotulo 4");
        r5 = new JLabel("rotulo 5");
        r6 = new JLabel("rotulo 6");
        janela.add(r1);
        janela.add(r2);
        janela.add(r3);
        janela.add(r4);
        janela.add(r5);
        janela.add(r6);
        janela.setLocation(200,300);
        janela.setSize(500,100);
        janela.setVisible(true);
    }
}
```

O código acima importa as classes do pacote **java.awt.*** e as classes do pacote **javax.swing.***. A classe do pacote *awt* que foi importado é a **GridLayout()** que é um tipo de gerenciador de leiaute. As classes do pacote *swing* que foram importadas são todas aquelas que começam com **J** como caractere inicial do nome.

A linha `public JLabel r1, r2, r3, r4, r5, r6;` declara que `r1, r2, r3, r4, r5` e `r6` são do tipo *JLabel* que é um rótulo que escrevemos na tela. Como eu posso declarar que `x` é do tipo *int* eu declarei que de `r1` a `r6` todos são do tipo *JLabel*. Depois iremos utilizar os métodos da classe *JLabel*.

A linha `public ExemploGrid() {` é o início do método construtor. Ele será utilizado quando formos criar um objeto da classe *ExemploGrid*.

A linha `JFrame janela = new JFrame("Gerenciadores de leiaute - GridLayout");` instância o objeto *JFrame* (observe que agora existe um parâmetro `String` nele) e utiliza como variável de referência *janela*. O parâmetro passado será escrito no título da janela.

A linha `janela.setLayout(new GridLayout(3,2));` chama o método `setLayout` da classe *JFrame* através da variável de referência *janela* e diz para o método que a grade a ser montada para comportar os componentes terá 3 linhas e 2 colunas.

As linhas

```
r1 = new JLabel("rotulo 1");  
r2 = new JLabel("rotulo 2");  
r3 = new JLabel("rotulo 3");  
r4 = new JLabel("rotulo 4");  
r5 = new JLabel("rotulo 5");  
r6 = new JLabel("rotulo 6");
```

criam os objetos *JLabel* e utilizam como variáveis de referência de *r1*, *r2*, *r3*, *r4*, *r5* e *r6*. Veja o método construtor em cada variável de referência.

Depois de instanciados os objetos de *r1* a *r6* devemos adicioná-los ao gerenciador de leiaute. Para fazermos isso utilizaremos o método `add()` da classe *JFrame* que está sendo referenciada por *janela*. Vejamos como irá ficar:

```
janela.add(r1);  
janela.add(r2);  
janela.add(r3);  
janela.add(r4);  
janela.add(r5);  
janela.add(r6);
```

Agora temos os objetos já adicionados a janela.

A linha `janela.setLocation(200,300);` utilize o método `setLocation` do objeto *JFrame* referenciado por *janela* e determina a posição que a janela terá na tela do computador nesse caso com 200px para a direita e 300px para baixo.

A linha `janela.setSize(500,100);` determina o tamanho da janela em pixels.

A linha `janela.setVisible(true);` exibirá a janela na tela.

O código para executar a classe acima está abaixo:

```
import javax.swing.*;
public class MostraExemploGrid{
    public static void main(String args[]) {
        ExemploGrid grade = new ExemploGrid();
    }
}
```

Esses métodos são básicos para a criação de uma janela gráfica em Java. Vejamos a saída do código acima.



2.1.3 BorderLayout

O gerenciador de leiaute **BorderLayout** divide a janela em cinco partes, sendo elas norte, sul, leste, oeste e centro. Vejamos o código:

```
import javax.swing.*;
import java.awt.*;
public class ExemploBorder{
    public JButton bn, bs, bl, bo, bc;
    public ExemploBorder(){
        JFrame janela = new JFrame("Exemplo BorderLayout");
        janela.setLayout(new BorderLayout());
        bn = new JButton("Norte");
        bs = new JButton("Sul");
        bl = new JButton("Leste");
        bo = new JButton("Oeste");
        bc = new JButton("Centro");
        janela.add(bn, BorderLayout.NORTH);
        janela.add(bs, BorderLayout.SOUTH);
        janela.add(bl, BorderLayout.EAST);
        janela.add(bo, BorderLayout.WEST);
        janela.add(bc, BorderLayout.CENTER);
        janela.setLocation(200,300);
        janela.setSize(400,200);
        janela.setVisible(true);
    }
}
```

Observe que no código acima você deverá declarar que o leiaute (`janela.setLayout(new BorderLayout())`) é do tipo *BorderLayout* e quando for instanciar os objetos de botão `bn`, `bs`, `bl`, `bo` e `bc` eles utilizam o construtor de *JButton* e o texto entre os parênteses ou parâmetros será escrito no rótulo do botão.

Veja também que para adicionar os botões na tela você utilizará o “mesmo” método `add()` só que agora não mais com a variável de referência do objeto que será inserido e sim com a variável de referência do objeto que será inserido e em qual a posição que ele será inserido (norte, sul, leste, oeste ou centro).

O programa abaixo executa a classe `ExemploBorderLayout`, vejamos:

```
import javax.swing.*;
public class MostraBorder{
    public static void main(String args[]) {
        ExemploBorder cadastro = new ExemploBorder();
    }
}
```

A saída do programa será:



Não é necessário que você coloque componentes em todas as partes do gerenciador `BorderLayout`. Você pode, por exemplo, somente utilizar o centro, sul e o leste. Vejamos o código:

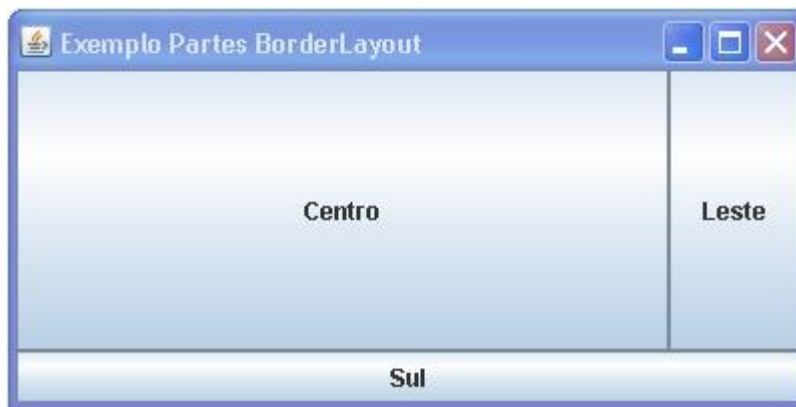
```
import javax.swing.*;
import java.awt.*;
public class ExemploPartesBorder{
    public JButton bn, bs, bl, bo, bc;
    public ExemploPartesBorder(){
        JFrame j = new JFrame("Exemplo Partes BorderLayout");
        j.setLayout(new BorderLayout());
        bs = new JButton("Sul");
        bl = new JButton("Leste");
        bc = new JButton("Centro");
    }
}
```

```
j.add(bs, BorderLayout.SOUTH);  
j.add(bl, BorderLayout.EAST);  
j.add(bc, BorderLayout.CENTER);  
j.setLocation(200,300);  
j.setSize(400,200);  
j.setVisible(true);  
}  
}
```

O código para executar o arquivo acima é:

```
public class MostraExemploPartesBorder{  
    public static void main (String[] args){  
        ExemploPartesBorder epb = new ExemploPartesBorder();  
    }  
}
```

Vejamos a interface que foi criada:



No exemplo acima foi mostrado o gerenciador BorderLayout utilizando apenas botões, mas podemos adicionar qualquer componente a qualquer gerenciador de layout. Observe também que não foram utilizadas as posições norte e oeste do gerenciador de layout.

2.1.4 FlowLayout

O gerenciador de layout **FlowLayout** organiza os componentes da janela da esquerda para a direita. **FlowLayout** são usados para organizar os botões em uma janela. Vejamos o código:

```
import javax.swing.*;  
import java.awt.*;  
public class MostraFlowLayout{  
    public JButton b1, b2, b3;  
    public MostraFlowLayout(){
```

```
JFrame cadastro = new JFrame();  
b1 = new JButton("Gravar");  
b2 = new JButton("Cancelar");  
b3 = new JButton("Sair");  
cadastro.setLayout(new FlowLayout());  
cadastro.add(b1);  
cadastro.add(b2);  
cadastro.add(b3);  
cadastro.setTitle("FlowLayout");  
cadastro.setSize(300,200);  
cadastro.setVisible(true);  
}  
}
```

Observe que no código acima utilizamos o construtor *JFrame()* sem parâmetros, mas o nome é o mesmo. Bacana isso não? Chamamos de polimorfismo e estudaremos isso mais a frente. Como foi colocado o título na janela então? Usamos o método *setTitle()* do objeto *JFrame* que foi referenciado por *cadastro*.

Para executarmos o código acima fazemos:

```
import javax.swing.*;  
public class ExecutaMostraFlowLayout{  
    public static void main(String[] args){  
        MostraFlowLayout mfl = new MostraFlowLayout();  
    }  
}
```

A saída será:




Para saber mais sobre interface gráfica em:
<http://www.exampledepot.com/egs/javax.swing/pkg.html>
Outro lugar interessante também é o:
<http://java.sun.com/docs/books/tutorial/uiswing/index.html>

Conclusão

- As classes que compõem a interface gráfica com o usuário GUI estão no pacote **javax.swing** e no pacote **java.awt**.
- **javax.swing** é considerado um pacote peso leve pois não depende do sistema operacional para construir suas janelas.
- **java.awt** é considerado um pacote peso pesado pois precisa dos arquivos do sistema operacional para construir suas janelas.
- As janelas em Java são chamadas de Frame.
- **JFrame** é uma classe do pacote **javax.swing** utilizada para construir janelas gráficas.
- Os gerenciadores de leiaute auxiliam na construção de janelas gráficas.
- Gerenciadores de leiaute são utilizados para que a aplicação seja independente de plataforma (não depende de coordenadas x e y).
- Métodos **add()** são utilizados para adicionar objetos em uma janela.
- O gerenciador de leiaute **GridLayout** divide a janela em linhas e colunas e cada componente inserido irá ocupar toda a célula reservada para ele.
- O gerenciador de leiaute **BorderLayout** divide a janela em regiões Norte, Sul, Leste, Oeste e Centro.
- O gerenciador de leiaute **FlowLayout** adiciona os componentes em uma janela um atrás do outro.
- O método de **JFrame** chamado **setLayout** é quem decide qual leiaute será usado.
- Quando passamos como parâmetro o valor **null** para o método **setLayout** isso significa que os elementos a serem inseridos estarão dispostos em coordenadas **x** e **y**.
- Quando o gerenciador de layout não é utilizado é preciso configurar a posição e tamanho dos elementos a serem inseridos na janela através do método **setBounds** onde os argumentos posicionam o elemento e definem também o tamanho do elemento.

2.2 Exercícios

	<p>Vamos fazer alguns exercícios</p> <ol style="list-style-type: none">1. Para que serve o gerenciador de leiaute?2. Qual a diferença entre java.awt e javax.swing? <p>No computador</p> <ol style="list-style-type: none">1. Escreva um programa para montar uma agenda com o nome e o telefone. Você deverá inserir botões para mostrar contatos e sair. Não é necessário fazer com que os botões funcionem ainda. Use o gerenciador de leiaute GridLayout.
---	---

2.3 Painéis, menus e componentes da interface gráfica

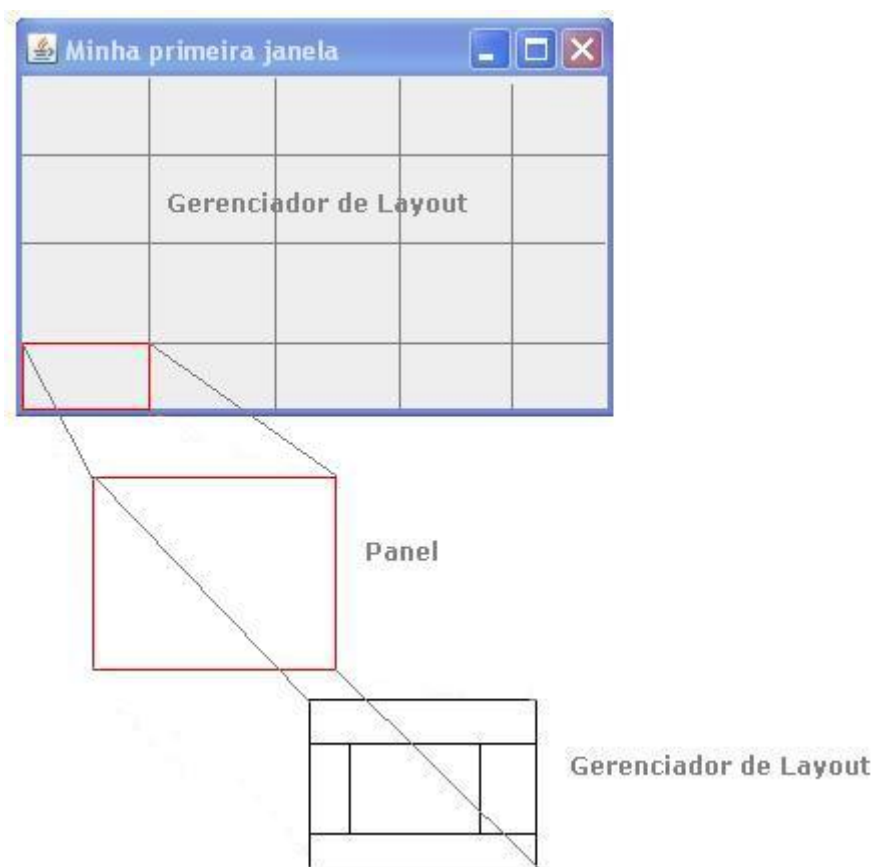
Ao final da aula você será capaz de:

- Utilizar painéis nas interfaces gráficas ;
- Utilizar a classe JScrollPane ;
- Programar menus em Java;
- Programar submenus em Java;
- Utilizar a classe JRadioButton;
- Utilizar a classe JCheckBox;
- Utilizar a classe JList;
- Utilizar a classe JPasswordField;

2.3.1 Painéis

Os painéis são componentes importantes na construção de interfaces gráficas em Java. Os painéis podem conter gerenciadores de leiaute e são adicionados dentro de gerenciadores de leiaute. Eles auxiliam muito na inserção dos componentes dentro dos gerenciadores de leiaute.

A figura abaixo mostra como gerenciar painéis:



O código comentado abaixo mostra a utilização de painéis e gerenciadores de leiaute.

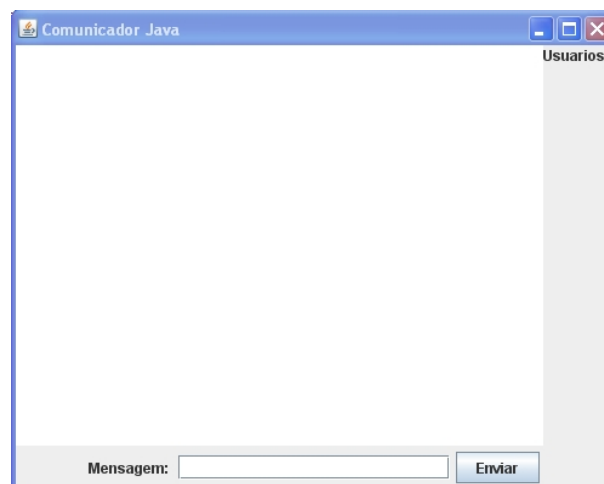
```
import javax.swing.*;
import java.awt.*;
public class Painel{
    public JPanel pnDireito = new JPanel();
    public JPanel pnInferior = new JPanel();
    public JFrame janela = new JFrame();
    public JLabel lblUsuario = new JLabel("Usuarios: ");
    public JLabel lblMens = new JLabel("Mensagem: ");
    public JTextField tfUsuario = new JTextField(50);
    public JTextField tfMens = new JTextField(20);
    public JTextArea taTexto = new JTextArea(20,40);
    public JButton botoEnviar = new JButton("Enviar");
    public Painel(){
        //configura o leiaute da janela
        janela.setLayout(new BorderLayout());
        //configura o leiaute do painel direito
        pnDireito.setLayout(new GridLayout(20,1));
        //configura o leiaute do painel inferior
```

```
pnInferior.setLayout(new FlowLayout());  
//adiciona os componentes ao leiaute do painel direito  
pnDireito.add(lblUsuario);  
//adiciona os componentes ao leiaute do painel inferior  
pnInferior.add(lblMens);  
pnInferior.add(tfMens);  
pnInferior.add(btoEnviar);  
//adiciona o painel direito a janela  
janela.add(pnDireito, BorderLayout.EAST);  
//adiciona o painel inferior a janela  
janela.add(pnInferior, BorderLayout.SOUTH);  
//adiciona a área de texto  
janela.add(taTexto, BorderLayout.CENTER);  
//configura o tamanho da janela  
janela.setSize(500,400);  
//configura o título da janela  
janela.setTitle("Comunicador Java");  
//exibe a janela  
janela.setVisible(true);  
}  
}
```

O código abaixo mostra como executar a classe acima:

```
public class ExibePainel{  
    public static void main(String[] args) {  
        Painel chat = new Painel();  
    }  
}
```

A saída do programa será:



2.3.2 JScrollPane

É necessário adicionar na janela acima uma barra de rolagem no objeto **TextArea** e para isso usamos a classe **JScrollPane** que é um painel com barra de rolagem. Vejamos como ficará o código:

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

public class PaineiRolagem {
    public JPanel pnDireito = new JPanel();
    public JPanel pnInferior = new JPanel();
    public JFrame janela = new JFrame();
    public JLabel lblUsuario = new JLabel("Usuarios: ");
    public JLabel lblMens = new JLabel("Mensagem: ");
    public JTextField tfUsuario = new JTextField(50);
    public JTextField tfMens = new JTextField(20);
    public JTextArea taTexto = new JTextArea(20,40);
    public JButton botoEnviar = new JButton("Enviar");
    public JScrollPane barraRolagem = new JScrollPane(taTexto);
    public PaineiRolagem() {
        //configura o leiaute da janela
        janela.setLayout(new BorderLayout());
        //configura o leiaute do painel direito
        pnDireito.setLayout(new GridLayout(20,1));
        //configura o leiaute do painel inferior
        pnInferior.setLayout(new FlowLayout());
        //adiciona os componentes ao leiaute do painel direito
        pnDireito.add(lblUsuario);
        //adiciona os componentes ao leiaute do painel inferior
        pnInferior.add(lblMens);
        pnInferior.add(tfMens);
        pnInferior.add(botoEnviar);
        //adiciona o painel direito a janela
        janela.add(pnDireito, BorderLayout.EAST);
        //adiciona o painel inferior a janela
        janela.add(pnInferior, BorderLayout.SOUTH);
        //adiciona o painel JScrollPane na posição center
        janela.add(barraRolagem, BorderLayout.CENTER);
        //configura o tamanho da janela
        janela.setSize(500,400);
        //configura o título da janela
        janela.setTitle("Comunicador Java com barra de rolagem");
        //exibe a janela
        janela.setVisible(true);
    }
}
```

Para executar o código fazemos:

```
public class ExibePainelRolagem{
    public static void main(String[]args) {
        PainelRolagem chat = new PainelRolagem();
    }
}
```

Vamos ver agora como escrevemos o código para alinharmos à esquerda a parte inferior da janela.

```
import javax.swing.*;
import java.awt.*;
public class PainelRolagem {
    public JPanel pnDireito = new JPanel();
    public JPanel pnInferior = new JPanel();
    public FlowLayout leiautePnInferior = new FlowLayout();
    public JFrame janela = new JFrame();
    public JLabel lblUsuario = new JLabel("Usuarios: ");
    public JLabel lblMens = new JLabel("Mensagem: ");
    public JTextField tfMens = new JTextField(20);
    public JTextArea taTexto = new JTextArea(20,40);
    public JButton botoEnviar = new JButton("Enviar");
    public JScrollPane barraRolagem = new JScrollPane(taTexto);
    public PainelRolagem(){
        //configura o leiaute da janela
        janela.setLayout(new BorderLayout());
        //configura o leiaute do painel direito
        pnDireito.setLayout(new GridLayout(20,1));
        //configura o leiaute do painel inferior
        pnInferior.setLayout(leiautePnInferior);
        //configura o alinhamento do FlowLayout
        leiautePnInferior.setAlignment(FlowLayout.TRAILING);
        //configura o alinhamento do pnInferior

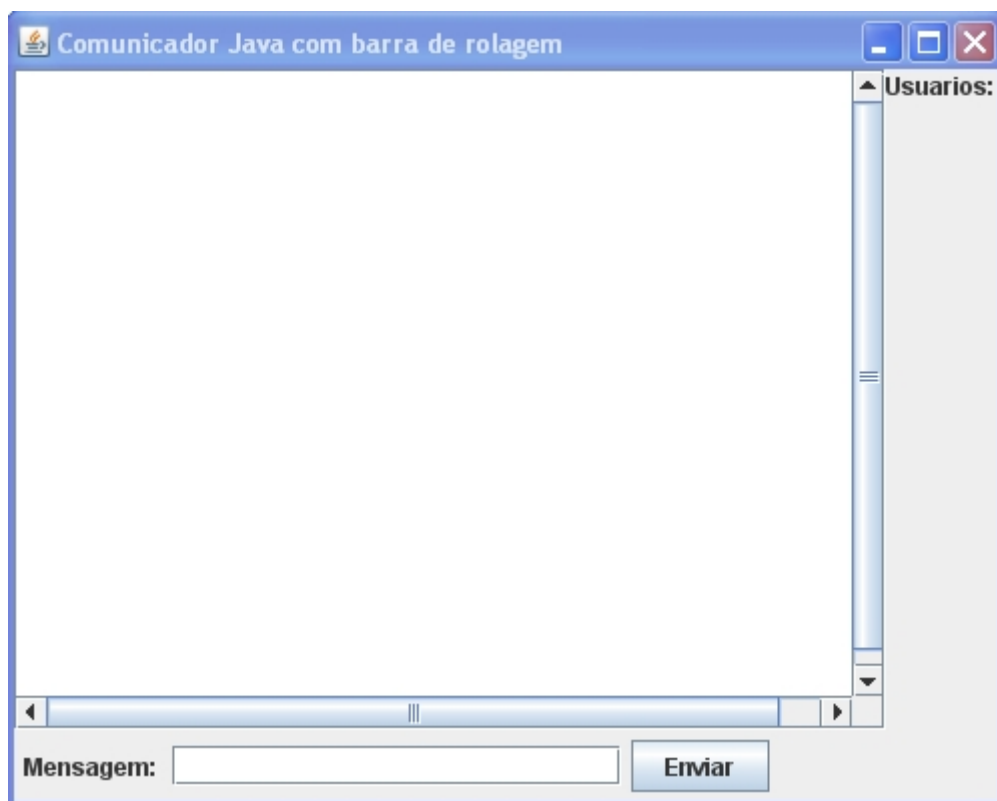
pnInferior.setComponentOrientation(ComponentOrientation.RIGHT_
TO_LEFT);
        //adiciona os componentes ao leiaute do painel direito
        pnDireito.add(lblUsuario);
        //adiciona os componentes ao leiaute do painel inferior
        //a sequência é importante.
        pnInferior.add(botoEnviar);
        pnInferior.add(tfMens);
        pnInferior.add(lblMens);
        //adiciona o painel direito a janela
        janela.add(pnDireito, BorderLayout.EAST);
        //adiciona o painel inferior a janela
    }
}
```

```
janela.add(pnInferior, BorderLayout.SOUTH);  
//adiciona o painel JScrollPane na posição center  
janela.add(barraRolagem, BorderLayout.CENTER);  
//configura o tamanho da janela  
janela.setSize(500,400);  
//configura o título da janela  
janela.setTitle("Comunicador Java com barra de rolagem");  
//exibe a janela janela.setVisible(true);  
}  
}
```

Para executar o arquivo acima fazemos:

```
public class ExibePainelRolagem{  
    public static void main(String[] args) {  
        PainelRolagem chat = new PainelRolagem();  
    }  
}
```

A saída do programa será:



2.3.3 Menus em Java

Escreveremos agora um programa que mostra como construir um menu em Java.

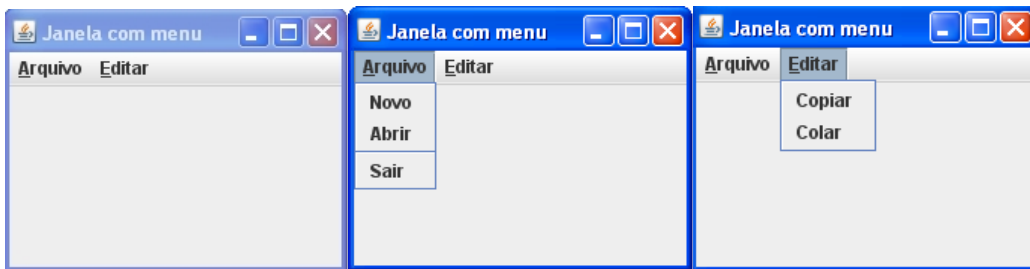
```
import javax.swing.*;
public class Menu{
    //define os objetos
    public JFrame janela;
    public JMenuBar barraMenu;
    public JMenu menuArquivo;
    public JMenu menuEditar;
    public JMenuItem itemNovo;
    public JMenuItem itemAbrir;
    public JMenuItem itemSair;
    public JMenuItem itemCopiar;
    public JMenuItem itemColar;
    public Menu(){
        //instancia os objetos
        janela = new JFrame("Janela com menu");
        barraMenu = new JMenuBar();
        menuArquivo = new JMenu("Arquivo");
        menuEditar = new JMenu("Editar");
        //adiciona um atalho aos menus
        menuArquivo.setMnemonic('A');
        menuEditar.setMnemonic('e');
        itemNovo = new JMenuItem("Novo");
        itemAbrir = new JMenuItem("Abrir");
        itemSair = new JMenuItem("Sair");
        itemCopiar = new JMenuItem("Copiar");
        itemColar = new JMenuItem("Colar");
        //adiciona os itens do menu Arquivo
        menuArquivo.add(itemNovo);
        menuArquivo.add(itemAbrir);
        menuArquivo.addSeparator();
        menuArquivo.add(itemSair);
        //adiciona os itens do menu Editar
        menuEditar.add(itemCopiar);
        menuEditar.add(itemColar);
        //adiciona os menus a barra de menus
        barraMenu.add(menuArquivo);
        barraMenu.add(menuEditar);
        //adiciona a barra de menu a janela
        janela.setJMenuBar(barraMenu);
        //configurao tamanho da janela
        janela.setSize(300,400);
        //mostra a janela
        janela.setVisible(true);
    }
}
```

```
}
```

Para executar o programa acima fazemos:

```
public class ExibeMenu {
    public static void main(String[] args) {
        Menu m = new Menu();
    }
}
```

As janelas abaixo mostram como ficarão os menus:



Podemos inserir botões de opção **ou** (da classe `RadioButtonMenuItem`), botões de opção **e** e ícones nos menus.

2.3.4 Submenus

Um menu em Java pode conter submenus que são menus dentro de menus. O programa abaixo exemplifica essa técnica.

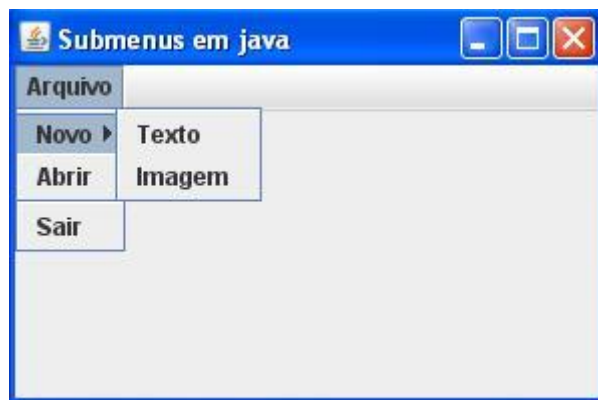
```
import javax.swing.*;
public class SubMenu {
    //declara os componentes
    public JFrame janela;
    public JMenuBar barraMenu;
    public JMenu menu, subMenu;
    public JMenuItem itemAbrir, itemSair, itemTexto, itemImagem;
    public SubMenu(){
        //instancia os componentes
        janela = new JFrame("Submenus em java");
        barraMenu = new JMenuBar();
        menu = new JMenu("Arquivo");
        subMenu = new JMenu("Novo");
        itemAbrir = new JMenuItem("Abrir");
        itemSair = new JMenuItem("Sair");
        itemTexto = new JMenuItem("Texto");
        itemImagem = new JMenuItem("Imagem");
        //adiciona os componentes
    }
}
```

```
subMenu.add(itemTexto);  
subMenu.add(itemImagem);  
menu.add(subMenu);  
menu.add(itemAbrir);  
menu.addSeparator();  
menu.add(itemSair);  
barraMenu.add(menu);  
janela.setJMenuBar(barraMenu);  
//configura o tamanho da janela  
janela.setSize(400,400);  
//mostra a janela  
janela.setVisible(true);  
}  
}
```

Para executarmos o programa usamos o seguinte código:

```
public class ExecutaSubMenu {  
    public static void main(String[] args){  
        SubMenu sm = new SubMenu();  
    }  
}
```

A saída do programa será:



2.3.5 Radio Button

Radio Buttons são grupos de botões que, por convenção, apenas um botão pode ser selecionado por vez. O Swing dá suporte a Radio Buttons com a classe `JRadioButton` e `ButtonGroup`. Vejamos:


```
import javax.swing.*;
import java.awt.*;
public class RadioButton {
    public JFrame janela;
    public JPanel painel;
    public JRadioButton rbMasc, rbFem;
    public ButtonGroup grupo;
    public RadioButton() {
        janela = new JFrame("Janela Radio Button");
        painel = new JPanel();
        rbMasc = new JRadioButton("Masculino");
        rbFem = new JRadioButton("Feminino");
        grupo = new ButtonGroup();
        grupo.add(rbMasc);
        grupo.add(rbFem);
        janela.setLayout(new FlowLayout());
        painel.setLayout(new FlowLayout());
        painel.add(rbMasc);
        painel.add(rbFem);
        janela.add(painel);
        //configura o tamanho da janela
        janela.setSize(400,200);
        //mostra a janela
        janela.setVisible(true);
    }
}
```



Observe que o componente **Radio Button** é quem foi adicionado ao painel e não o **Button Group**

Para executar o arquivo usamos o código:

```
public class ExecutaRadioButton {
    public static void main(String[] args) {
        RadioButton rb = new RadioButton();
    }
}
```

A saída do programa será:



2.3.6 Check box

A classe `JCheckBox` dá suporte aos botões de check box. Podemos também usar os check boxes em menus usando a classe `JCheckBoxMenuItem`.

Check boxes são parecidos com os radio buttons, mas eles trabalham de forma diferente. Os radio buttons permitem que apenas uma opção seja selecionada (**ou**) e o check box permite que mais de uma opção seja selecionada (**e**). Vejamos um exemplo:

```
import javax.swing.*;
import java.awt.*;
public class CheckBox {
    public JFrame janela;
    public JPanel painel;
    public JCheckBox chkArroz, chkFeijao, chkBife, chkBatata;
    public CheckBox() {
        janela = new JFrame("Janela Check Box");
        painel = new JPanel();
        chkArroz = new JCheckBox("Arroz");
        chkFeijao = new JCheckBox("Feijão");
        chkBife = new JCheckBox("Bife");
        chkBatata = new JCheckBox("Batata frita");
        janela.setLayout(new FlowLayout());
        painel.setLayout(new GridLayout(4,1));
        painel.add(chkArroz);
        painel.add(chkFeijao);
        painel.add(chkBife);
        painel.add(chkBatata);
        janela.add(painel);
        //configura o tamanho da janela
        janela.setSize(400,200);
    }
}
```

```
//mostra a janela  
janela.setVisible(true);  
}  
}
```

O programa para executar a classe acima:

```
public class ExecutaCheckBox {  
    public static void main(String[] args){  
        CheckBox cb = new CheckBox();  
    }  
}
```

A saída do programa será:



2.3.7 Listas

A JList mostra como trabalhar com um grupo de itens mostrados em uma ou mais colunas para serem selecionados. As Lists que têm muitos itens devem ser colocadas em um JScrollPane. Vejamos o exemplo abaixo:

```
import javax.swing.*;  
import java.awt.*;  
public class Listas{  
    public JFrame janela;  
    public JPanel painel;  
    public JList minhaLista;  
    public DefaultListModel modelo;  
    public JScrollPane rolagem;  
    public Listas(){  
        janela = new JFrame("Janela com lista");  
        painel = new JPanel();  
        modelo = new DefaultListModel();  
        modelo.addElement("Mouse");  
        modelo.addElement("Teclado");  
        modelo.addElement("Monitor");  
        modelo.addElement("Gabinete");  
    }  
}
```

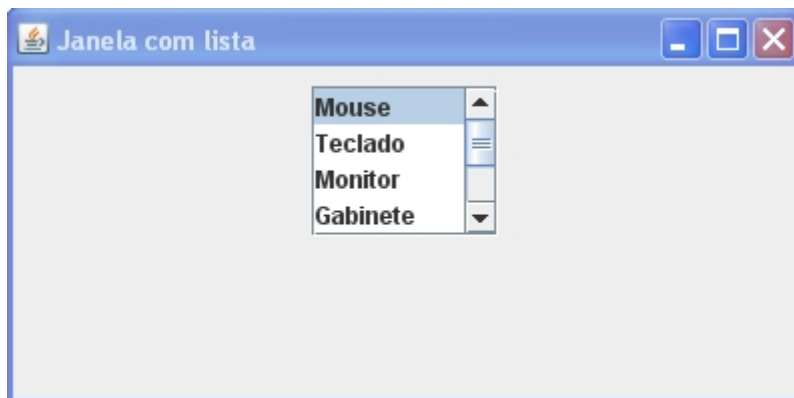
```
modelo.addElement("Web Cam");
modelo.addElement("Estabilizador");
modelo.addElement("Microfone");
//cria a lista e adiciona o modelo
minhaLista = new JList(modelo);
//configura a lista

minhaLista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
minhaLista.setSelectedIndex(0);
minhaLista.setVisibleRowCount(4);
//adiciona barras de rolagem
rolagem = new JScrollPane(minhaLista);
janela.setLayout(new FlowLayout());
painel.setLayout(new FlowLayout());
painel.add(rolagem);
janela.add(painel);
//configura o tamanho da janela
janela.setSize(400,200);
//mostra a janela
janela.setVisible(true);
}
```

Para executar o arquivo fazemos:

```
public class ExecutaListas {
    public static void main(String[] args){
        Listas l = new Listas();
    }
}
```

A saída será:



2.3.8 Password

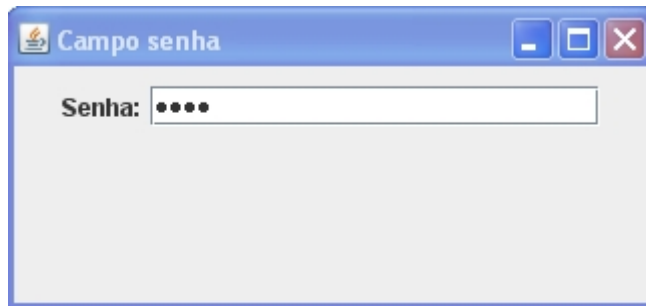
A classe JPasswordField que é uma subclasse de JTextField disponibiliza campos textos especializados para entrada de senhas. Por questões de segurança o campo de senha não exibe o que foi digitado e sim símbolos que representam os caracteres que realmente foram digitados. Vejamos:

```
import javax.swing.*;
import java.awt.*;
public class Password {
    public JFrame janela;
    public JPanel painel;
    public JPasswordField pswSenha;
    public JLabel lblSenha;
    public Password() {
        janela = new JFrame("Campo senha");
        painel = new JPanel();
        pswSenha = new JPasswordField(20);
        lblSenha = new JLabel("Senha:");
        //configura a janela
        janela.setLayout(new FlowLayout());
        painel.setLayout(new FlowLayout());
        painel.add(lblSenha);
        painel.add(pswSenha);
        janela.add(painel);
        //configura o tamanho da janela
        janela.setSize(400,200);
        //mostra a janela
        janela.setVisible(true);
    }
}
```

Para executar o código:

```
public class ExecutaPassword {
    public static void main(String[] args) {
        Password p = new Password();
    }
}
```

A saída será:



2.3.9 ComboBox

A classe **JComboBox** permite que o usuário escolha uma das várias opções listadas. Isso pode ser feito de duas formas diferentes. A forma padrão é escolher uma opção em um campo não editável que é formada por uma lista suspensa de valores e um botão. A segunda forma, chamada de **combo box editável** é formada por um campo texto com um pequeno botão em que o usuário pode escrever o texto ou pressionar o botão para que uma lista suspensa seja mostrada. Vejamos um exemplo:

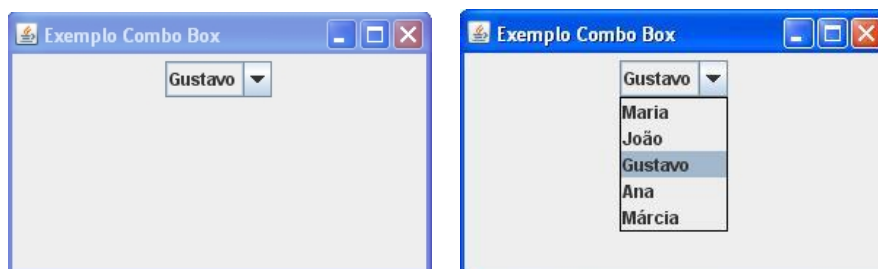
```
import javax.swing.*;
import java.awt.*;
public class ComboBox{
    //declara a janela
    public JFrame janela;
    //declara o painel
    JPanel painel;
    //declara o comboBox
    JComboBox listaNomes;
    public ComboBox(){
        //instancia a janela
        janela = new JFrame("Exemplo Combo Box");
        //instancia o painel
        painel = new JPanel();
        //Carrega um vetor de Strings
        String[] nome = {"Maria", "João", "Gustavo", "Ana",
" Márcia"};
        //Instancia o combo box e atribui os nomes
        listaNomes = new JComboBox(nome);
        //Deixa como selecionado o 2º item da lista
        listaNomes.setSelectedIndex(2);
        //adiciona o combo box ao painel
        painel.add(listaNomes);
        //adiciona o painel a janla
        janela.add(painel);
    }
}
```

```
//configura o tamanho da janela
janela.setSize(300,200);
//mostra a janela
janela.setVisible(true);
}
}
```

O código para executar o a classe acima é:

```
public class ExecutaComboBox{
    public static void main(String[] args){
        ComboBox cb = new ComboBox();
    }
}
```

A saída do programa será:



2.4 Exercícios

No computador

- Escreva um programa para fazer um cadastro de matrícula de um aluno em um dos cursos que a escola DeltaCursos oferece. Veja o formulário atual abaixo.

Delta Cursos – Cursos profissionalizantes

Cadastro de matrícula

Nome: _____
 Rua: _____
 Número: _____
 Bairro: _____
 Cidade: _____
 Estado: (lista com 4 estados MG, SP, BA, RJ)
 Curso: (lista com 5 cursos que são Java, PHP, C, C++, J2ME)
 E-mail: _____
 Senha: _____
 Confirma senha: _____

Enviar

Sair

Quero lembrar que se você clicar no botão enviar ou botão sair nada acontecerá, pois apenas desenhamos a janela (interface com o usuário). Para que os botões funcionem, teremos que implementar ações ao botão. Isso você verá no tópico, quatro logo abaixo.

- Construa o código para mostrar a interface abaixo:

Observe que no exercício acima foram usados dois painéis sendo o superior o GridLayout com duas linhas e duas colunas e o inferior um FlowLayout.

2.5 Manipuladores de Eventos em Java

Ao final da aula você será capaz de:

- Utilizar classes internas
- Programar manipuladores de eventos para JButton
- Programar manipuladores de eventos para JTextField
- Programar manipuladores de eventos para JMenuitem
- Programar manipuladores de eventos para JList
- Programar manipuladores de eventos para JComboBox
- Programar manipuladores de eventos para JRadioButton
- Programar manipuladores de eventos para JCheckBox
- Programar manipuladores de eventos para JTable
- Alterar dados em um JTable
- Inserir e excluir linhas da tabela

2.5.1 Evento para JButton

As interfaces gráficas de Java são baseadas em eventos, isto é, geram eventos (ações) quando o usuário interage com o programa. Uma interação comum pode ser mover o mouse, pressionar um botão, clicar com o mouse, digitar em um campo de texto, selecionar um item no menu etc..Quando o usuário executa qualquer ação, esta pode ser enviada ao programa. As informações de eventos de GUI são armazenadas em um objeto de uma classe que está no pacote **awt.event** . Os eventos do pacote `java.awt.event.*` podem ser usados tanto para componentes AWT quanto componentes SWING. Os eventos específicos de componentes SWING estão no pacote **javax.swing.event**.

O tratamento de eventos consiste em três partes:

- Origem do evento - Objeto com o qual o usuário interage.
- Objeto evento - Encapsula as informações sobre o evento.
- Ouvinte do evento (listener) - Objeto que é notificado pela origem do evento.

Para processar um evento da interface gráfica é preciso que você execute duas tarefas:

1. Registrar um ouvinte de evento para o componente
2. Implementar um método de tratamento de evento (ou conjunto de métodos)

O objeto "ouvidor" listener de eventos aguarda por tipos específicos de eventos gerados por origens de eventos específicas (um botão, por exemplo). O tratador de eventos é um método chamado em resposta a um tipo de evento em particular. Cada **interface listener** de eventos especifica um ou mais métodos de tratamentos

de eventos que **devem** ser definidos na classe que implementa a interface listener de eventos.

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

public class AcaoBotao extends JFrame implements
ActionListener{
    private JButton b1, b2, b3;

    public AcaoBotao() {
        b1 = new JButton("Gravar");
        b2 = new JButton("Cancelar");
        b3 = new JButton("Sair");
        //Registra os tratadores de eventos
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        //configura o leiaute e adiciona os botões
        setLayout(new FlowLayout());
        add(b1);
        add(b2);
        add(b3);
        setTitle("FlowLayout");
        setSize(300,200);
        setVisible(true);
    }

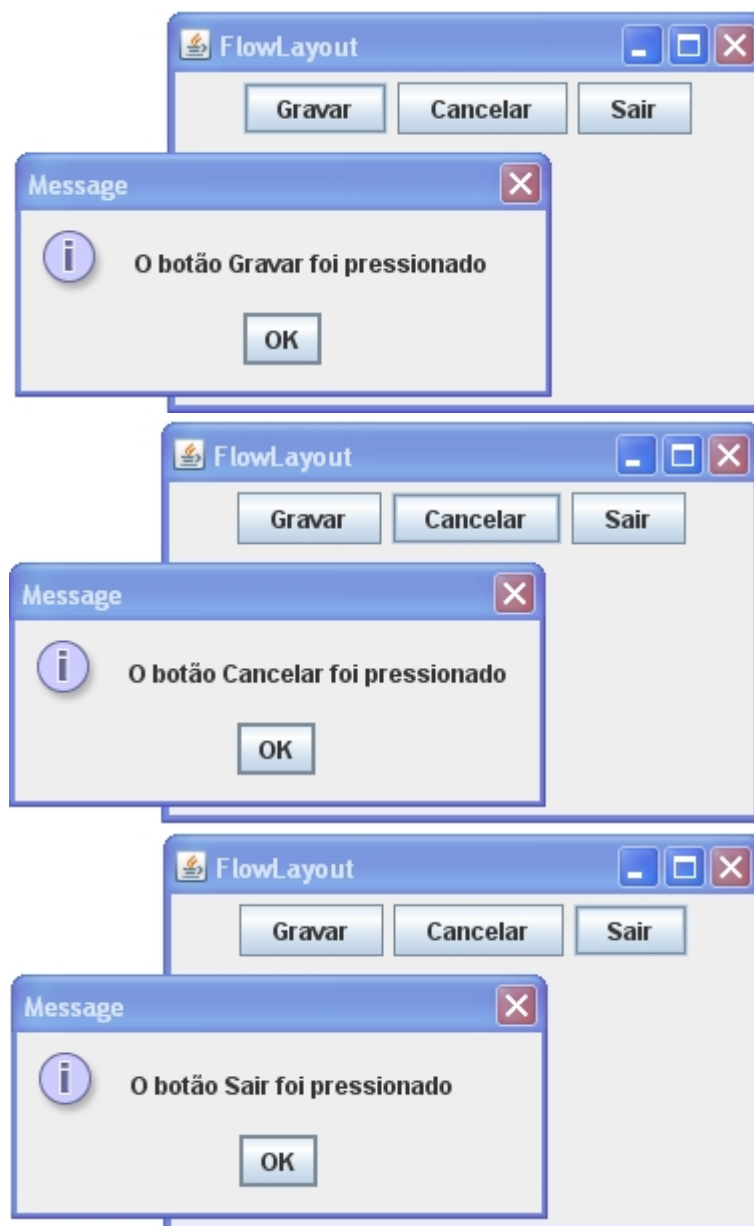
    //implementa o método actionPerformed da interface
    //processa eventos de botões
    public void actionPerformed(ActionEvent e) {
        //verifica qual botão foi pressionado
        if(e.getSource()== b1){
            JOptionPane.showMessageDialog(null, "O botão Gravar foi
pressionado");
        }
        else if (e.getSource()==b2){
            JOptionPane.showMessageDialog(null, "O botão Cancelar
foi pressionado");
        }
        else if (e.getSource()==b3){
            JOptionPane.showMessageDialog(null, "O botão Sair foi
pressionado");
            JOptionPane.showMessageDialog(this, "Tchau");
            System.exit(0);
        }
    }
}
```

```
}  
}
```

Para executar o programa acima fazemos:

```
public class ExecutaAcaoBotao {  
    public static void main(String[] args) {  
        AcaoBotao ab = new AcaoBotao();  
    }  
}
```

A saída será:





Uma outra forma de utilizarmos os tratamentos de eventos é utilizando uma classe interna que é uma classe que declaramos dentro de uma outra classe:

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

public class AcaoBotaoClasseInterna {
    private JButton b1, b2, b3;
    private JFrame janela;

    public AcaoBotaoClasseInterna() {
        b1 = new JButton("Gravar");
        b2 = new JButton("Cancelar");
        b3 = new JButton("Sair");
        //instancia a classe interna ClasseEventos
        ClasseEventos evento = new ClasseEventos();
        //adiciona um "ouvidor" ao botao
        b1.addActionListener(evento);
        b2.addActionListener(evento);
        b3.addActionListener(evento);
        //configura o leiaute e adiciona os botões
        janela.setLayout(new FlowLayout());
        janela.add(b1);
        janela.add(b2);
        janela.add(b3);
        janela.setTitle("FlowLayout");
        janela.setSize(300,200);
        janela.setVisible(true);
    }

    private class ClasseEventos implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource()== b1){
                JOptionPane.showMessageDialog(null,"O botão Gravar foi
pressionado");
            }
        }
    }
}
```

```
        } else if (e.getSource()==b2){
            JOptionPane.showMessageDialog(null, "O botão Cancelar
foi pressionado");
        } else if (e.getSource()==b3){
            JOptionPane.showMessageDialog(null, "O botão Sair foi
pressionado");
            JOptionPane.showMessageDialog(janela, "Tchau");
            System.exit(0);
        }
    }
}
```

Para executar a classe acima fazemos:

```
public class ExecutaAcaoBotaoClasseInterna {
    public static void main(String[] args){
        AcaoBotao ab = new AcaoBotao();
    }
}
```

Se você executar o programa acima verá que a saída é exatamente igual ao do programa anterior. A diferença está na escrita do código. No primeiro exemplo utilizamos uma **herança** de JFrame para a criação da janela e implementamos a **interface** ActionListener (pode ver o this em algumas partes do arquivo). No segundo programa a classe JFrame foi **instanciada** dentro do programa e a interface ActionListener foi implementada dentro de uma **classe interna** chamada ClasseEventos em que os eventos foram manipulados.

2.5.2 Evento para JTextField

O exemplo abaixo mostra um evento de teclado. Ao pressionar a tecla **enter** o conteúdo do campo JTextField será enviado para uma janela de mensagem.

Vejamos o código abaixo:

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

public class AcaoJtextField {
    JFrame janela;
    JTextField tfNome;

    public AcaoJtextField() {
        //instancia os objetos
        janela = new JFrame("Ação de campo texto");
        tfNome = new JTextField(20);
        //instancia a classe interna ClasseEvento
        ClasseEvento evento = new ClasseEvento();

        //registra o evento para o campo texto
        tfNome.addActionListener(evento);
        //configura o leiaute da janela
        janela.setLayout(new FlowLayout());

        //adiciona o campo texto a janela
        janela.add(tfNome);

        //configura o tamanho da janela
        janela.setSize(300,200);

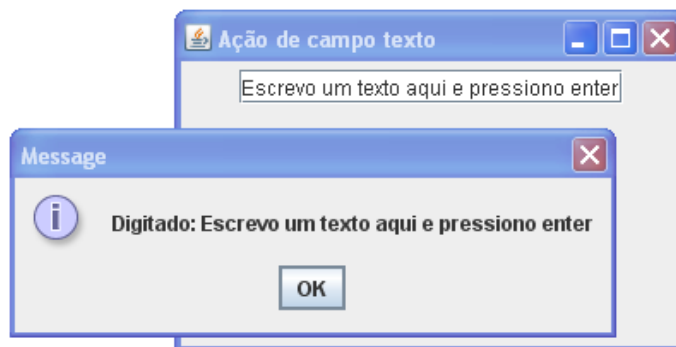
        //mostra a janela
        janela.setVisible(true);
    }

    private class ClasseEvento implements ActionListener{
        public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog(null, "Digitado:
"+tfNome.getText());
        }
    }
}
```

Para executar o programa fazemos:

```
public class ExecutaAcaoJtextField {  
    public static void main(String[] args) {  
        AcaoJtextField atf = new AcaoJtextField();  
    }  
}
```

A saída do programa será:



2.5.3 Evento para JMenuItem

O código abaixo mostra como adicionar eventos às opções de menus. Vejamos o código:

```
import javax.swing.*;  
import java.awt.event.*;  
  
public class EventoSubMenu {  
    //declara os componentes  
    private JFrame janela;  
    private JMenuBar barraMenu;  
    private JMenu menu, subMenu;  
    private JMenuItem itemAbrir, itemSair, itemTexto,  
    itemImagem;  
    public EventoSubMenu() {  
        //instancia os componentes  
        janela = new JFrame("Submenus com eventos em java");  
        barraMenu = new JMenuBar();  
        menu = new JMenu("Arquivo");  
        subMenu = new JMenu("Novo");  
        itemAbrir = new JMenuItem("Abrir");  
        itemSair = new JMenuItem("Sair");  
        itemTexto = new JMenuItem("Texto");
```

```
itemImagem = new JMenuItem("Imagem");

//instancia um objeto da classe interna
TrataEvento te = new TrataEvento();

//registra os eventos para os menus
itemSair.addActionListener(te);
itemTexto.addActionListener(te);
itemAbrir.addActionListener(te);

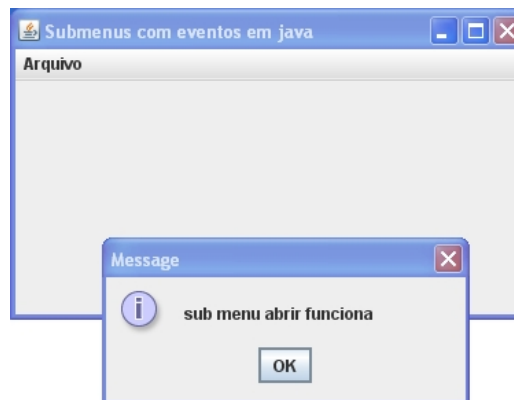
//adiciona os componentes
subMenu.add(itemTexto);
subMenu.add(itemImagem);
menu.add(subMenu);
menu.add(itemAbrir);
menu.addSeparator();
menu.add(itemSair);
barraMenu.add(menu);
janela.setJMenuBar(barraMenu);
//configura o tamanho da janela
janela.setSize(400,400);
//mostra a janela
janela.setVisible(true);
}

private class TrataEvento implements ActionListener{
    public void actionPerformed(ActionEvent e){
        if(e.getSource() == itemSair){
            JOptionPane.showMessageDialog(null, "menu sair
funcionou");
            System.exit(0);
        }
        if(e.getSource() == itemTexto){
            JOptionPane.showMessageDialog(null,"sub menu item
texto funcionou");
        }
        if(e.getSource() == itemAbrir){
            JOptionPane.showMessageDialog(null,"submenu abrir
funciona");
        }
    }
}
}
```


Para executar o código fazemos:

```
public class ExecutaEventoSubMenu {  
    public static void main(String[] args){  
        EventoSubMenu sm = new EventoSubMenu();  
    }  
}
```

A saída será:



2.5.4 Eventos para JList

O código abaixo mostra como adicionar eventos a um componente JList. Vejamos:

```
import javax.swing.event.*;  
import javax.swing.*;  
import java.awt.*;  
  
public class AcaoListas {  
    private JFrame janela;  
    private JPanel painel;  
    private JList minhaLista;  
    private DefaultListModel modelo;  
    private JScrollPane rolagem;  
    public AcaoListas() {  
        janela = new JFrame("Janela com lista");  
        painel = new JPanel();  
        modelo = new DefaultListModel();  
        modelo.addElement("Mouse");  
        modelo.addElement("Teclado");  
        modelo.addElement("Monitor");  
        modelo.addElement("Gabinete");  
        modelo.addElement("Web Cam");  
        modelo.addElement("Estabilizador");  
    }  
}
```

```
modelo.addElement("Microfone");

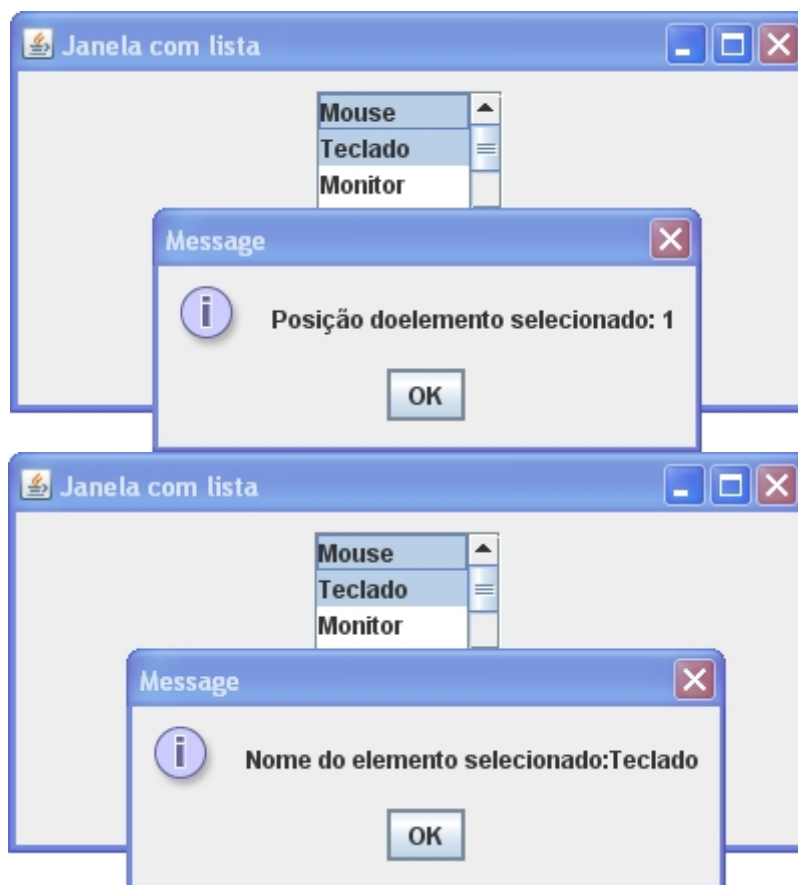
//cria a lista e adiciona o modelo
minhaLista = new JList(modelo);
//configura a lista
minhaLista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
minhaLista.setSelectedIndex(0);
minhaLista.setVisibleRowCount(4);
//instancia a classe interna
TrataEvento te = new TrataEvento();

//registra o evento para a lista
minhaLista.addListSelectionListener(te);
//adiciona barras de rolagem
rolagem = new JScrollPane(minhaLista);
janela.setLayout(new FlowLayout());
painel.setLayout(new FlowLayout());
painel.add(rolagem);
janela.add(painel);
//configura o tamanho da janela
janela.setSize(400,200);
//mostra a janela
janela.setVisible(true);
}
private class TrataEvento implements ListSelectionListener{
    public void valueChanged(ListSelectionEvent e){
        JOptionPane.showMessageDialog(null, "Posição:
"+minhaLista.getSelectedIndex());
        JOptionPane.showMessageDialog(null, "Nome:"
minhaLista.getSelectedValue());
    }
}
}
```

Para executar a classe acima temos:

```
public class ExecutaAcaoListas {
    public static void main(String[] args){
        AcaoListas al = new AcaoListas();
    }
}
```

A saída do programa será:



2.5.5 Eventos para ComboBox

O programa abaixo mostra como construir eventos para componentes ComboBox. Vejamos:

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

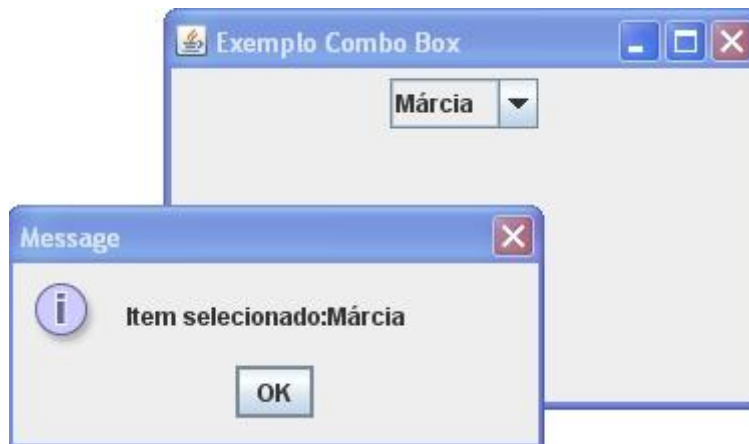
public class AcaoComboBox{
    //declara a janela
    public JFrame janela;
    //declara o painel
    JPanel painel;
    //declara o comboBox
    JComboBox listaNomes;
    public AcaoComboBox(){
        //instancia a janela
```

```
janela = new JFrame("Exemplo Combo Box");
//instancia o painel
painel = new JPanel();
//Carrega um vetor de Strings
String[] nome = {"Maria", "João", "Gustavo", "Ana",
" Márcia"};
//Instancia o combo box e atribui os nomes
listaNomes = new JComboBox(nome);
//Deixa como selecionado o 2º item da lista
listaNomes.setSelectedIndex(2);
//instancia classe interna
TrataEvento te = new TrataEvento();
//registra o evento para listaNomes
listaNomes.addItemListener(te);
//adiciona o combo box ao painel
painel.add(listaNomes);
//adiciona o painel a janela
janela.add(painel);
//configura o tamanho da janela
janela.setSize(300,200);
//mostra a janela
janela.setVisible(true);
}
//criação da classe interna para tratamento de eventos
private class TrataEvento implements ItemListener{
    public void itemStateChanged(ItemEvent e) {
        //verifica qual item está selecionado
        if(e.getStateChange() == ItemEvent.SELECTED) {
            JOptionPane.showMessageDialog(null, "Item:"
+listaNomes.getSelectedIndex());
        }
    }
}
```

Para executar o programa acima fazemos:

```
public class ExecutaAcaoComboBox{
    public static void main(String[] args){
        AcaoComboBox cb = new AcaoComboBox();
    }
}
```

A saída do programa será:



2.5.6 Eventos para JRadioButton

Radio Buttons são grupos de botões (opções) que por convenção apenas um botão (opção) deve ser selecionado. O pacote Swing dá suporte a Radio Buttons através das classes JRadioButton e ButtonGroup. Vejamos o código:

```
import java.awt.event.*;
import javax.swing.*;
public class AcaoRadioButton {

    //define as variáveis de instância
    private JRadioButton sxMasculino, sxFeminino;
    private ButtonGroup grupo;
    private JFrame janela;
    //construtor
    public AcaoRadioButton() {
        //instancia os objetos
        janela = new JFrame("Radio buttons exemplo");
        sxMasculino = new JRadioButton("Masculino");
        sxFeminino = new JRadioButton("Feminino");
        //agrupa os botões
        grupo = new ButtonGroup();
        grupo.add(sxMasculino);
        grupo.add(sxFeminino);
        //instancia a classe interna de eventos
        ClasseEventos evento = new ClasseEventos();
        //registra os eventos
        sxMasculino.addActionListener(evento);
        sxFeminino.addActionListener(evento);
        //coloca os radio buttons na janela
        janela.setLayout(new FlowLayout());
    }
}
```

```
janela.add(sxMasculino);
janela.add(sxFeminino);
//configura a janela
janela.setSize(200,300);
janela.setLocation(300, 300);
janela.setVisible(true);
}
//classe interna para tratamento de eventos
public class ClasseEventos implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        if(sxMasculino.isSelected()){
            JOptionPane.showMessageDialog(null, "Escolhido:
Masculino");
            System.exit(0);
        }
        if(sxFeminino.isSelected()){
            JOptionPane.showMessageDialog(null, "Escolhido:
Feminino");
            System.exit(0);
        }
    }
}
```

A saída será:



2.5.7 Eventos para JCheckBox

JCheckBox são parecidos com os JRadioButton, mas eles trabalham de forma diferente. Os JRadioButton's permitem que apenas uma opção seja selecionada (ou) e o JCheckBox permite que mais de uma opção seja selecionada (e). Vejamos o código:

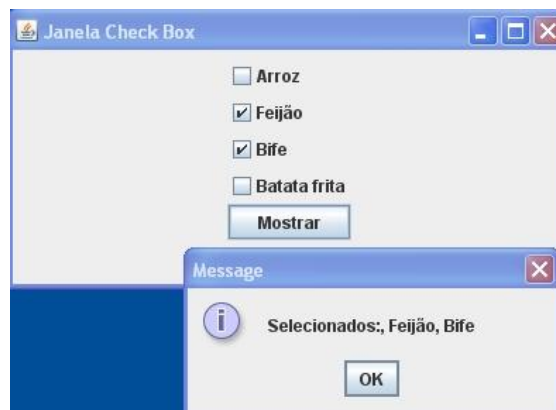
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class AcaoCheckBox {
    private JFrame janela;
    private JPanel painel;
    private JCheckBox chkArroz, chkFeijao, chkBife, chkBatata;
    private JButton botao;
    public AcaoCheckBox(){
        janela = new JFrame("Janela Check Box");
        painel = new JPanel();
        chkArroz = new JCheckBox("Arroz");
        chkFeijao = new JCheckBox("Feijão");
        chkBife = new JCheckBox("Bife");
        chkBatata = new JCheckBox("Batata frita");
        botao = new JButton("Mostrar");
        janela.setLayout(new FlowLayout());
        painel.setLayout(new GridLayout(5,1));
        painel.add(chkArroz);
        painel.add(chkFeijao);
        painel.add(chkBife);
        painel.add(chkBatata);
        painel.add(botao);
        //instancia a classe interna
        TrataEvento evento = new TrataEvento();
        //registra os eventos para o botao
        botao.addActionListener(evento);
        //adiciona o painel a janela
        janela.add(painel);
        //configura o tamanho da janela
        janela.setSize(400,200);
        //mostra a janela
        janela.setVisible(true);
    }
    //classe interna de tratamento de eventos
    public class TrataEvento implements ActionListener{
        String selecionados = "";
        public void actionPerformed(ActionEvent e){
            if(chkArroz.isSelected()){
                selecionados = selecionados + " " +chkArroz.getText();
            }
            if(chkFeijao.isSelected()){
                selecionados = selecionados + " "
+chkFeijao.getText();
            }
            if(chkBife.isSelected()){
                selecionados = selecionados + " " +chkBife.getText();
            }
        }
    }
}
```

```
        if(chkBatata.isSelected()){
            selecionados = selecionados + " "
+chkBatata.getText();
        }
        if(e.getSource() == botao){
            JOptionPane.showMessageDialog(janela, "Selecionados:"+
selecionados);
        }
    }
}
```


O código abaixo instancia e dispara um objeto da classe acima.

```
public class ExecutaAcaoCheckBox {
    public static void main(String[] args) {
        //instanciando o objeto AccaoCheckBox
        AcaoCheckBox acb = new AcaoCheckBox();
    }
}
```

A saída do programa será:



2.6 Exercícios

	<p>Vamos fazer alguns exercícios</p> <ol style="list-style-type: none">3. Qual pacote podemos utilizar para manipularmos eventos?4. O tratamento de eventos é composto por três partes. Quais são elas? <p>No Computador</p> <ol style="list-style-type: none">4. Faça um programa que mostra uma janela com um menu chamado Telefones. O menu Telefones tem um submenu Cadastro um separador e outro submenu chamado Sair.5. Faça com que quando o usuário pressionar o menu Cadastro uma janela chamada cadastro com um campo Nome do tipo texto e um campo chamado telefone do tipo texto sejam mostrados. A janela tem um botão chamado enviar que envia os dados para uma outra janela e um botão sair que fecha a janela atual.6. No exercício 2 da unidade 2, faça com que quando o usuário pressionar o botão enviar os dados preenchidos sejam mostrados em uma outra janela e ao pressionar o botão sair, o programa seja fechado.
---	--

2.7 Resolução dos Exercícios Propostos

Unidade I

Teóricos

1. O que é uma classe?

Uma classe é um conceito que temos do mundo real.

2. Quais as partes de uma classe?

Nome, atributos e métodos.

3. Quando utilizamos um método construtor?

Para instanciar um objeto.

4. Quando utilizamos um método operacional? Dê exemplo.

Quanto queremos fazer com que o objeto realize alguma ação para o programador. Exemplo: Escrever ou ler de um atributo, imprimir etc.

5. Qual a diferença entre um método construtor e um método operacional?

Na criação os métodos construtores têm o mesmo nome da classe e não têm tipo de retorno e os métodos operacionais têm um tipo de retorno (void para não retornar nada) e podem ter qualquer nome.

Na utilização, os construtores são utilizados para instanciar objetos e os métodos operacionais são usados durante a execução do programa.

6. Crie o modelo conceitual de um automóvel.

Para criar um modelo conceitual, temos que pensar em o que um automóvel tem (atributos) e o que um automóvel faz (métodos).

Atributos:

cor, ano, nRodas, nPortas, nMarchas, largura, comprimento

Métodos:

setCor (String cor)

setAno (int ano)

setNRodas (int nRodas)

setNPortas (int nPortas)

setNMarchas(int nM)

setLargura(float larg)

setComprimento(float comp)

getCor()

getAno()

getNRodas()

getNPortas()
getNMarchas()
getLargura()
getComprimento()
ligar()
trocarMarcha()
desligar()

"Observe que foram escritos apenas alguns atributos e métodos. Sabemos que existem muitos outros, mas para o exercício esses (ou os que você colocar) já estão bons. Veja também que alguns argumentos eu coloquei nome diferente do atributo. Isso não gera problema nenhum"

7. Como criamos um objeto?

Criamos um objeto depois que instanciamos uma classe.

8. O que é “modularização” na programação estruturada?

Divisão do problema em pequenos módulos que são pequenas rotinas de programa.

9. O que é necessário para que uma classe seja um programa na linguagem Java?

É preciso que ela tenha o método main em seu corpo.

10. Qual o comando utilizado para escrever uma mensagem em uma janela gráfica utilizando Java?

JOptionPane.showMessageDialog(null, "mensagem");

11. O que é um pacote?

É um diretório onde ficam armazenadas as classes de Java

12. Qual pacote devemos importar para utilizarmos a impressão de dados em uma janela gráfica?

`javax.swing`

13. Qual método da classe `JOptionPane` devemos utilizar para ler os dados do teclado?

`showInputDialog("texto para entrada de dados");`

"Observe que não coloquei a classe, pois foi pedido somente o método. Se colocar a classe não saberei se você consegue diferenciar método de classe".

ESTUDOS DE CLASSES

14. O que é uma variável de referência?

Variável de referência é uma variável que se "refere" a um objeto na memória. Ela referencia um objeto na memória.

15. O que é um atributo?

Um atributo é uma variável da classe. Ele somente alocará memória quando um objeto dessa classe for instanciado.

16. Quais os passos necessários para se criar instância de uma classe utilizando a linguagem Java?

Primeiramente devemos ter uma classe e depois utilizar o operador `new`.

17. Um método é definido por palavras chaves da linguagem respeitando uma sequência. Escreva quais são essas palavras.

A sequência é:

- 1- modificador que é opcional
- 2- tipo de retorno
- 3- nome do método
- 4- argumentos que são opcionais

18. Qual o comando para acionar um método membro. Sabendo que a variável de referência foi declarada com x e o método se chama imprimir()?

Usar a variável de referência o operador ponto e o nome do método.

Exemplo: x.imprimir()

19. O que é um construtor? Quando ele é utilizado na Programação Orientada a Objetos?

O construtor é um método membro de uma classe e é utilizado na instanciação de objetos.

20. Como acessamos um atributo a partir de um programa?

Usamos a variável de referência o operador ponto e o nome do atributo.

21. De acordo com o JavaBeans qual a regra para escrever o nome de um método?

A regra é que o nome do método deve sempre começar com a primeira letra em minúscula e se for composto por mais de um nome o restante deve iniciar em Maiúsculo. Métodos que irão configurar algum atributo devem começar com set e métodos que irão fazer leitura deverão começar com get.

Exercícios práticos da Unidade I

1)

```
import javax.swing.*;

/*Escrever um programa em Java que mostre seu nome na tela
utilizando a interface gráfica.*/

//importa o pacote gráfico

public class UIEx1 {

    public static void main(String[] args) {

        //instancia o objeto da classe JOptionPane

        JOptionPane jop = new JOptionPane();

        jop.showMessageDialog(null,"Seu nome escrito aqui");

    }

}
```

2)

```
import javax.swing.JOptionPane;

/* Escrever um programa em Java que mostre a soma de dois
números quaisquer. Os números devem ser

* lidos do teclado utilizando a interface gráfica.

*/

public class UIEx2 {

    public static void main(String[] args){

        //declara as variáveis

        int x, y, soma;

        String sx, sy;

        //instancia um objeto da classe JOptionPane
```

```
JOptionPane jop = new JOptionPane();

//faz a leitura do teclado

sx = jop.showInputDialog("Digite o primeiro valor");
sy = jop.showInputDialog("Digite o segundo valor");

//converte os valores para inteiro

x = Integer.parseInt(sx);
y = Integer.parseInt(sy);

//faz a soma

soma = x + y;

//escreve o resultado

jop.showMessageDialog(null, "A soma é: "+soma);

}

}

/* A sequencia não precisa ser exatamente essa, você pode
fazer a leitura e imediatamente fazer

* a conversão de String para inteiro.

*/
```

3)

```
/*Escreva um programa que mostre a figura abaixo na tela
usando o console

* - não é necessário utilizar comandos de repetição para este
exercício.

          *

        ***

      *****
```

```
        *****

        *****

        *****

*/

public class UIEx3 {

    public static void main(String[] args){

        System.out.println("    *");

        System.out.println("    ***");

        System.out.println("    *****");

        System.out.println("    *****");

        System.out.println("    *****");

        System.out.println("*****");

    }

}
```

4)

```
/* Escreva um programa para calcular o n-ésimo termo da série
de Fibonacci.

* A entrada deverá ser utilizando a janela gráfica.

* Exemplo: Se o usuário digitar 8 deverá ser apresentado 1 1
2 3 5 8 13 21 e assim por diante.*/

import javax.swing.*;

public class UIEx4 {

    public static void main(String[] args){

        int ant, post, prox, nTermo;

        String snTermo;
```



```
//instancia um objeto de JOptionPane
JOptionPane jop = new JOptionPane();

//inicializa variáveis locais
ant = 1;
post = 1;

//atribui um valor ao atributo
snTermo = jop.showInputDialog("Digite o n-ésimo
termo");

nTermo = Integer.parseInt(snTermo);

System.out.print(+ant);

System.out.print(" "+post);

for(int i = 0; i < nTermo-2; i++){

    prox = ant + post;

    System.out.print(" "+prox);

    ant = post;

    post = prox;

}

}

/* O objetivo do exercício é desenvolver o raciocínio para
desenvolver lógica e utilizar as estruturas de controle de
repetição */
```

5)

```
/* Escreva um programa para calcular a divisão de dois números
reais (float) quaisquer,

* utilizando a entrada gráfica. */
```

```
import javax.swing.*;

public class UIEx5 {

    public static void main (String[] args){

        //declaração de variáveis

        float n1, n2, resultado;

        String sn1, sn2;

        //instancia um objeto de JOptionPane

        JOptionPane jop = new JOptionPane();

        //entrada de dados

        sn1 = jop.showInputDialog(null, "Digite o primeiro
valor");

        sn2 = jop.showInputDialog(null, "Digite o segundo
valor");

        //conversão de dados

        n1 = Float.parseFloat(sn1);

        n2 = Float.parseFloat(sn2);

        //verifica se o denominador não é igual a zero

        if(n2 == 0){

            jop.showMessageDialog(null, "Não é possível
resolver - denominador igual a zero");

        }

        else{

            resultado = n1/n2;

            jop.showMessageDialog(null, "Resultado:
"+resultado);

        }

    }

}
```

```
}  
  
}  
  
/* Podemos ver que nesse programa dividimos dois números reais  
(float). Já sabemos que quando lemos um valor este é do tipo  
String e devemos então convertê-lo para float. Utilizamos a  
classe Wrapper Float e seu método de conversão parseFloat()  
para converter a String em Float. */
```

6)

```
/* A empresa QSD Ltda. concedeu um bônus de 20 por cento do  
valor do salário a todos os funcionários com tempo de trabalho  
na empresa igual ou superior a cinco anos e dez por cento aos  
demais. Ler o salário, o tempo de trabalho, calcular e exibir  
o valor do bônus para 100 funcionários. */  
  
import javax.swing.*;  
  
public class UIEx6 {  
  
    public static void main(String[] args){  
  
        float salario, novoSalario;  
  
        String ssalario, sTempoTrabalho;  
  
        int tempoTrabalho, contador;  
  
        //instancia um objeto de JOptionPane  
  
        JOptionPane jop = new JOptionPane();  
  
        //inicializa a variável contadora  
  
        contador = 1;  
  
        //looping para entrada de dados  
  
        while(contador <= 5){  
  
            //Le o valor do salario  
  
            ssalario = jop.showInputDialog(null, "Digite o valor do  
salário");
```

```
//Le o tempo de trabalho

sTempoTrabalho = jop.showInputDialog("Digite o tempo de
trabalho");

//converte o valor do salario de String para float
tempoTrabalho = Integer.parseInt(sTempoTrabalho);

//converte o valor do salario de String para float
salario = Float.parseFloat(ssalario);

//verifica o tempo de trabalho
if(tempoTrabalho >= 5){

    novoSalario = salario + (salario * 20)/100;

    jop.showMessageDialog(null, "Novo salário:
"+novoSalario);

}

else{

    novoSalario = salario + (salario * 10)/100;

    jop.showMessageDialog(null, "Novo salário:
"+novoSalario);

}

contador++;

}

}

}

/* Nesse programa você pode usar qualquer estrutura de
repetição. Dentro da estrutura de repetição é usada a
estrutura condicional para verificar o tempo de trabalho e o
salário para adicionar os aumentos.
```

* Uma dica para fazer exercícios com looping faça o exercício com uns 5 passos para testar e depois aumente para o total solicitado.* /

7)

```
/*Ler 20 números fornecidos pelo usuário calcular e exibir a
média. (utilize do-while) */

import javax.swing.*;

public class UIEx7 {

    public static void main(String[] args){

        int numero, cont, total, soma;

        float media;

        String sNumero;

        //instancia um objeto de JOptionPane

        JOptionPane jop = new JOptionPane();

        //inicia variáveis

        total = 20;

        cont = 0;

        media = 0;

        soma = 0;

        //inicializa o looping

        do{

            //faz a leitura do número

            sNumero = jop.showInputDialog("Digite o número");

            //converte o valor de String para inteiro

            numero = Integer.parseInt(sNumero);
```

```
//faz a soma dos números cadastrados

soma = soma + numero;

//atualiza o contador

cont++;

}while(cont < total);

//calcula a média

media = soma/total;

//mostra a mensagem para o usuário

jop.showMessageDialog(null, "Média de salario: "+media);

}

}

/*Observe que inicializei as variáveis e utilizei apenas
variáveis dentro do código. Isso facilita a programação

* pois uso a variável total em dois momentos um para
controlar o looping e outro para calcular a média. Caso

* eu queira calcular a média para 100 pessoas posso somente
inicializar a variável com 100 e todo programa estará
atualizado. */
```

8)

```
/* Escreva uma classe chamada Aluno_UIEx8 que possua os
seguintes atributos

* nome do tipo String;

* idade do tipo int;

*/

public class Aluno_UIEx8 {

    //respondendo a questão 8
```

```
String nome;  
  
int idade;  
  
}
```

9)

```
/*Em um outro arquivo, crie um programa que instancie a classe  
Aluno.*/  
  
public class UIEx9 {  
  
    public static void main(String[] args) {  
  
        Aluno_UIEx8 aluno = new Aluno_UIEx8();  
  
    }  
  
}
```

/*Observe que não preciso importar nenhum pacote porque não solicitado nada que necessitasse de importar um pacote

* Observe que a classe usa o método construtor Aluno_UIEx8() e o mesmo não declarado explicitamente no exercício 8.

* Isso é perfeitamente norma porque se você não criar um construtor, a linguagem Java cria um para você.

* Agora temos aluno como variável de referência ao objeto Aluno_UIEx8 que foi colocado na memória com o operador new

* UIEx8 significa é uma sigla para te orientar nos exercícios e significa UI - Unidade I Ex - Exercício e 8 que

* é o número do exercício*/

10)

```
/*Mostre, utilizando a saída gráfica:
*    o valor do atributo nome;
*    o valor do atributo idade;
*/

//para utilizar o pacote gráfico devemos importar javax.swing
import javax.swing.*;

public class UIEx10 {

    public static void main(String[] args) {

        //instancia um objeto Aluno_UIEx8

        Aluno_UIEx8 aluno = new Aluno_UIEx8();

        //instancia um objeto de JOptionPane

        JOptionPane jop = new JOptionPane();

        //imprimindo o conteúdo da variável nome

        jop.showMessageDialog(null, "Nome: "+aluno.nome);

        jop.showMessageDialog(null, "Idade: "+aluno.idade);

    }

}

/* Observe que não foi adicionado nenhum valor para Nome e
idade e que a saída do programa mostrou o valor de nome

* que é do tipo String como null e idade que é do tipo
primitivo int como 0 isso porque os ATRIBUTOS da linguagem

* java quando declarado são inicializados automaticamente e se
forem do tipo String serão inicializados com null,

* se forem numéricos serão inicializados com 0 (zero) e se
forem booleandos serão inicializados com false
```



```
*/
```

11)

```
/*  Adicione em seu programa o código para fazer a leitura:
*   Do nome e atribuir ao atributo nome;
*   Da idade e atribuir ao atributo idade;
*/

import javax.swing.*;

public class UIEx11 {

    public static void main(String[] args){

        //instancia um objeto de Aluno_UIEx8.
        Aluno_UIEx8 aluno = new Aluno_UIEx8();

        //instancia um objeto da classe JOptionPane
        JOptionPane jop = new JOptionPane();

        //lendo o nome da interface gráfica

        //note que a leitura é feita e atribuída ao atributo nome
do objeto Aluno_UIEx8 através da variável de

        //referência aluno.

        aluno.nome = jop.showInputDialog("Digite o nome");

        //leitura da idade. Como a idade é um atributo do tipo
inteiro e a interface gráfica de Java só lê Strings

        //devemos fazer a conversão. Poderíamos ter declarado uma
variável local do tipo String, armazenar nela o

        //valor lido e depois atribuir ao atributo idade. Mas
vamos fazer um pouco diferente e converter diretamente.

        aluno.idade = Integer.parseInt(jop.showInputDialog("Digite
a idade: "));
```

```
}  
  
}  
  
/*Observe que estamos utilizando JOptionPane.showMessageDialog  
ou JOptionPane.showInputDialog e estamos vendo que  
  
* funciona perfeitamente. Mas o que estamos fazendo? Estamos  
acessando os métodos (showInputDialog() e  
  
* showMessageDialog()) da classe JOptionPane. Veja que  
importamos o pacote javax.swing com todas as suas classes  
  
* que está representada por * (asterisco). Quando importamos  
uma classe podemos acessar seus membros (atributos  
  
* e métodos) utilizando o nome da classe, um ponto separador  
e o nome do atributo ou método.  
  
*/
```

12)

```
/*Adicione em seu programa o código para mostrar os atributos  
nome e idade (na mesma janela gráfica)*/  
  
import javax.swing.JOptionPane;  
  
public class UIEx12 {  
  
    public static void main(String[] args){  
  
        //instancia um objeto de Aluno_UIEx8.  
  
        Aluno_UIEx8 aluno = new Aluno_UIEx8();  
  
        //instancia um objeto de JOptionPane  
  
        JOptionPane jop = new JOptionPane();  
  
        //lendo o nome da interface gráfica  
  
        //note que a leitura é feita e atribuída ao atributo nome  
do objeto Aluno_UIEx8 através da variável de  
  
        //referência aluno.
```

```
aluno.nome = jop.showInputDialog("Digite o nome");

//leitura da idade. Como a idade é um atributo do tipo
inteiro e a interface gráfica de Java só lê Strings

//devemos fazer a conversão. Poderíamos ter declarado uma
variável local do tipo String, armazenar nela o

//valor lido e depois atribuir ao atributo idade. Mas
vamos fazer um pouco diferente e converter diretamente.

aluno.idade = Integer.parseInt(jop.showInputDialog("Digite
a idade: "));

jop.showMessageDialog(null, "Nome:" +aluno.nome+ " Idade:
"+aluno.idade);

}

}
```

13)

```
/*Inclua em seu programa um método chamado:

* setNome que configura o nome;
* setIdade que configura o idade;
*/

public class UIEx13 {

    //declarando atributos

    String nome;

    int idade;

    //declarando o método que irá atribuir um valor ao atributo
nome

    void setNome(String n){

        this.nome = n;

    }

}
```

```
//declarando o método que irá atribuir um valor ao atributo
idade

void setIdade(int i){

    //verifica se a idade é negativa

    if(i<0){

        this.idade = 0;

    }

    else{

        this.idade = i;

    }

}

}

/*Veja que quando usarmos o método setName e passarmos um
parâmetro do tipo String para ele, esse parâmetro irá

* ser atribuído ao atributo nome. No corpo do método
poderemos validar o valor do parâmetro

* No método setIdade foi feita uma validação no método. Essa
validação não permite que a idade seja negativa

* Caso ela seja negativa então zero é atribuído ao atributo.
Mais tarde aprenderemos como melhorar esse tratamento de
erros*/
```

14)

```
/*Utilize os métodos acima para atribuir valores para os
atributos. Utilize a interface gráfica */

import javax.swing.*;

public class UIEx14 {

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

```
//declara uma variável String para receber a idade

String sidade;

//declara uma variável inteira i para receber o valor
convertido de idade e atribui a variável i

int i;

//instanciando um objeto de JOptionPane

JOptionPane jop = new JOptionPane();

//instanciando um objeto de Aluno_UIEx8

UIEx13 aluno = new UIEx13();

//faz a leitura do nome e passa por parâmetro para o
método setNome

aluno.setNome(jop.showInputDialog("Digite seu nome
aqui"));

//faz a leitura da idade e atribui a variável sidade

sidade = JOptionPane.showInputDialog("Digite a idade");

//converte a string sidade para inteiro

i = Integer.parseInt(sidade);

//passa a idade como parâmetro para o atributo idade
através do método setIdade

aluno.setIdade(i);

}

}

/*Veja que para acessar o método da classe UIEx13 você teve
que usar a variável de referência, um ponto e o nome

* do método que queria usar. */
```

15)

```
/*Inclua em seu programa um método chamado:

*   getNome que retorna o valor do atributo nome;
*   getIdade que retorna o valor do atributo idade;
*/

public class UIEx15 {

    //declarando atributos

    String nome;

    int idade;

    //declarando o método que irá atribuir um valor ao atributo
    nome

    void setNome(String n){

        this.nome = n;

    }

    //declarando o método que irá atribuir um valor ao atributo
    idade

    void setIdade(int i){

        //verifica se a idade é negativa

        if(i<0){

            this.idade = 0;

        }

        else{

            this.idade = i;

        }

    }

    //declarando o método que irá ler o conteúdo de nome
```

```
String getNome() {  
    return this.nome;  
}  
  
//declarando o método que irá ler o conteúdo de idade  
int getIdade() {  
    return this.idade;  
}  
}
```

Unidade II

Teóricos

1. Para que serve o gerenciador de leiaute?

Gerenciadores de leiaute são utilizados para melhor acomodar os componentes em uma janela (Frame). Por Java ser uma linguagem multiplataforma a utilização de gerenciadores de leiaute não compromete a distribuição dos componentes na tela. Se usarmos coordenadas x e y podemos ter janelas desconfiguradas dependendo da plataforma em que o sistema irá ser executado.

2. Qual a diferença entre java.awt e javax.swing?

java.awt é uma pacote gráfico considerado pesado pois necessita de arquivos do sistema operacional para construir suas interfaces gráficas.

javax.swing é um pacote considerado leve pois não necessita de arquivos do sistema operacional para construir suas janelas. Com ele é possível construir uma interface Linux e executar em uma plataforma Windows.

3. Qual pacote podemos utilizar para manipularmos eventos?

java.awt.event

4. O tratamento de eventos é composto por três partes. Quais são elas?

Origem do evento.

Objeto evento.

Ouvinte do evento (listener).

Exercícios práticos da Unidade II

```
1) package UIEx1;

/*Escreva um programa para montar uma agenda com o nome e o
telefone.

* Você deverá inserir botões para mostrar contatos e sair.
* Não é necessário fazer com que os botões funcionem ainda.
* Use o gerenciador de leiaute GridLayout.*/
```



```
//importa o pacote gráfico
import javax.swing.*;
import java.awt.*;

public class AgendaTelefonica {

    //declara janela como sendo do tipo JFrame

    public JFrame janela;

    //declara os rótulos (labels)

    public JLabel lblNome, lblTelefone;

    //declara as caixas de texto (JTextField)

    public JTextField txtNome, txtTelefone;

    //declara os botões

    public JButton btoMostrar, btoSair;

    //método para disparar a janela

    public void mostraJanela(){

        //instancia o objeto janela

        janela = new JFrame("Agenda telefônica");

        //Vamos definir o gerenciador de leiaute da janela
como foi pedido no exercício

        janela.setLayout(new GridLayout(3,2));

        //vamos determinar a localização da janela na tela

        janela.setLocation(400, 200);
```

```
//vamos determinar o tamanho da janela

janela.setSize(300,100);

//não deixa o usuário alterar o tamanho da janela

janela.setResizable(false);

//instancia o objeto JLabel com variável de
referência lblNome

lblNome = new JLabel("Nome: ");

//instancia o objeto JLabel com variável de
referência lblTelefone

lblTelefone = new JLabel("Telefone: ");

//instancia o objeto JTextField com variável de
referência txtNome com tamanho 40

txtNome = new JTextField(40);

//instancia o objeto JTextField com a variável de
referência txtTelefone com tamanho 10

txtTelefone = new JTextField(10);

//instancia o objeto botão JButton com variável de
referência botoMostrar

botoMostrar = new JButton("Enviar");

//instancia o objeto botão JButton com variável de
referência botoSair

botoSair = new JButton("Sair");

//agora temos todos os objetos já prontos.

//Vamos adicioná-los à janela

janela.add(lblNome);

janela.add(txtNome);

janela.add(lblTelefone);
```

```
janela.add(txtTelefone);

janela.add(btoMostrar);

janela.add(btoSair);

//Agora vamos mostrar a janela

janela.setVisible(true);

}

}

/* Você pode notar que a janela não pode ser maximizada ou
redimensionada devido a

* configuração do método setResizable(false). Se você alterar
o parâmetro para true

* verá que a janela poderá ser redimensionada e seus
componentes também serão automaticamente redimensionados.*/
```

```
package UIEx1;

//programa que executa a classe UIIEx1_AgendaTelefonica
public class AppAgendaTelefonica {

public static void main(String[] args) {
    //instancia um objeto da classe UIIEx1_AgendaTelefonica
    AgendaTelefonica agenda = new AgendaTelefonica();

    //temos agora um objeto UIIEx1_AgendaTelefonica que é
referenciado pela variável de
    //referência agenda. Com a variável de referência agenda
podemos acessar os métodos
    //do objeto UIIEx1_AgendaTelefonica.
    //Veja que na classe UIIEx1_AgendaTelefonica não
colocamos a construção da janela no
    //construtor e sim em um método chamado mostraJanela.
Esse método não tem argumentos
    //mas poderia ter dependendo da situação.
    //Vamos acessar o método mostraJanela do objeto
UIIEx1_AgendaTelefonica
```

```
        agenda.mostraJanela();  
    }  
  
}
```

2)

```
package UIEx2;  
  
//importa o pacote gráfico  
import javax.swing.*;  
import java.awt.*;  
  
public class Agenda {  
  
    //declara janela como sendo do tipo JFrame  
    public JFrame janela;  
  
    //declara os rótulos (labels)  
    public JLabel lblNome, lblTelefone;  
  
    //declara as caixas de texto (JTextField)  
    public JTextField txtNome, txtTelefone;  
  
    //declara os botões  
    public JButton botoMostrar, botoSair, botoAlterar, botoIncluir,  
    botoExcluir;  
  
    //declara um painel  
    JPanel pSuperior, pInferior;  
  
    //método para disparar a janela  
    public void mostraJanela(){  
  
        //instancia o objeto janela  
  
        janela = new JFrame("Agenda telefônica");
```

```
//Vamos definir o gerenciador de leiaute com duas
linhas e uma coluna

janela.setLayout(new GridLayout(2,1));

//vamos determinar a localização da janela na tela
janela.setLocation(400, 200);

//vamos determinar o tamanho da janela
janela.setSize(380,120);

//não deixa o usuário alterar o tamanho da janela
janela.setResizable(false);

//instancia o objeto painel pSuperior
pSuperior = new JPanel();

//instancia o objeto painel pInferior
pInferior = new JPanel();

//configurando o leiaute do painel pSuperior
pSuperior.setLayout(new GridLayout(2,2));

//configurando o leiaute do painel pInferior
pInferior.setLayout(new FlowLayout());

//instancia o objeto JLabel com variável de
referência lblNome

lblNome = new JLabel("Nome: ");

//instancia o objeto JLabel com variável de
referência lblTelefone

lblTelefone = new JLabel("Telefone: ");

//instancia o objeto JTextField com variável de
referência txtNome com tamanho 40

txtNome = new JTextField(40);
```

```
//instancia o objeto JTextField com a variável de
referência txtTelefone com tamanho 10

txtTelefone = new JTextField(10);

//instancia o objeto botão JButton com variável de
referência btoMostrar

btoMostrar = new JButton("Enviar");

//instancia o objeto botão JButton com variável de
referência btoSair

btoSair = new JButton("Sair");

//agora temos todos os objetos já prontos vamos
adicioná-los ao objeto Panel referenciado por pSuperior

//instancia o objeto botão JButton com variável de
referência btoIncluir

btoIncluir = new JButton("Incluir");

//instancia o objeto botão JButton com variável de
referência btoAlterar

btoAlterar = new JButton("Alterar");

//instancia o objeto botão JButton com variável de
referência btoExcluir

btoExcluir = new JButton("Excluir");

pSuperior.add(lblNome);

pSuperior.add(txtNome);

pSuperior.add(lblTelefone);

pSuperior.add(txtTelefone);

//vamos adicionar agora os objetos ao objeto Panel
referenciado por pInferior

pInferior.add(btoMostrar);

pInferior.add(btoIncluir);
```

```
pInferior.add(btoAlterar);

pInferior.add(btoExcluir);

pInferior.add(btoSair);

//agora que temos os dois paineis já carregados com
os componentes que queremos, basta adicioná-lo (o painel)

//à janela

janela.add(pSuperior);

janela.add(pInferior);

//Agora vamos mostrar a janela

janela.setVisible(true);

}

}

/* Podemos ver nesse programa utilizamos o JPanel que é um
painel onde podemos colocar componentes.

* Os passos para criar uma janela em java são:

* 1. Declarar os objetos que serão usados (botões, janelas,
labels...)

* 2. Instanciar os objetos declarados

* 3. Dimensionar a janela

* 4. Definir o gerenciador de leiaute da janela (JFrame)

* 5. Se estiver usando painéis (JPanel), definir o
gerenciador de leiaute do painel (JPanel), se estiver usando

* somente a janela, definir o gerenciador de leiaute da
janela.

* 6. Adicionar os componentes instanciados no painel se
estiver usando JPanel ou na janela se estiver usando o

* gerenciador de leiaute diretamente na janela.
```

* 7. Mostrar a janela.

* A sequência acima mostra um pequeno algoritmo para construção de interfaces gráficas, mas quero deixar claro

* que isso não é uma regra rígida pois existe outras maneiras de programar a interface gráfica.* /

```
package UIIEx2;
//programa que executa a classe UIIEx2

public class AppAgenda {
    public static void main(String[] args) {
        //instancia um objeto da classe UIIEx1_AgendaTelefonica
        Agenda agenda = new Agenda();
        agenda.mostraJanela();
    }
}
```

3)

```
package UIIEx3;

//importa o pacote gráfico
import javax.swing.*;
import java.awt.*;

public class Agenda {

    //declara janela como sendo do tipo JFrame

    public JFrame janela;

    //declara os rótulos (labels)

    public JLabel lblNome, lblTelefone;

    //declara as caixas de texto (JTextField)

    public JTextField txtNome, txtTelefone;

    //declara os botões
```



```
public JButton botoMostrar,btoSair, botoAlterar, botoIncluir,
botoExcluir;

//declara um painel

JPanel pSuperior, pInferior;

//método para disparar a janela

public void mostraJanela(){

    //instancia o objeto janela

    janela = new JFrame("Agenda telefônica");

    //Vamos definir o gerenciador de leiaute com duas
linhas e uma coluna

    janela.setLayout(new GridLayout(2,1));

    //vamos determinar a localização da janela na tela

    janela.setLocation(400, 200);

    //vamos determinar o tamanho da janela

    janela.setSize(380,120);

    //não deixa o usuário alterar o tamanho da janela

    janela.setResizable(false);

    //instancia o objeto painel pSuperior

    pSuperior = new JPanel();

    //instancia o objeto painel pInferior

    pInferior = new JPanel();

    //configurando o leiaute do painel pSuperior

    pSuperior.setLayout(new GridLayout(2,2));

    //configurando o leiaute do painel pInferior

    pInferior.setLayout(new FlowLayout());
```

```
//instancia o objeto JLabel com variável de
referência lblNome

lblNome = new JLabel("Nome: ");

//instancia o objeto JLabel com variável de
referência lblTelefone

lblTelefone = new JLabel("Telefone: ");

//instancia o objeto JTextField com variável de
referência txtNome com tamanho 40

txtNome = new JTextField(40);

//instancia o objeto JTextField com a variável de
referência txtTelefone com tamanho 10

txtTelefone = new JTextField(10);

//instancia o objeto botão JButton com variável de
referência botoMostrar

botoMostrar = new JButton("Enviar");

//instancia o objeto botão JButton com variável de
referência botoSair

botoSair = new JButton("Sair");

//agora temos todos os objetos já prontos vamos
adicioná-los ao objeto Panel referenciado por pSuperior

//instancia o objeto botão JButton com variável de
referência botoIncluir

botoIncluir = new JButton("Incluir");

//instancia o objeto botão JButton com variável de
referência botoAlterar

botoAlterar = new JButton("Alterar");

//instancia o objeto botão JButton com variável de
referência botoExcluir

botoExcluir = new JButton("Excluir");
```

```
pSuperior.add(lblNome);

pSuperior.add(txtNome);

pSuperior.add(lblTelefone);

pSuperior.add(txtTelefone);

//vamos adicionar agora os objetos ao objeto Panel
referenciado por pInferior

pInferior.add(btoMostrar);

pInferior.add(btoIncluir);

pInferior.add(btoAlterar);

pInferior.add(btoExcluir);

pInferior.add(btoSair);

//agora que temos os dois paineis já carregados com
os componentes que queremos, basta adicioná-lo (o painel)

//à janela

janela.add(pSuperior);

janela.add(pInferior);

//Agora vamos mostrar a janela

janela.setVisible(true);

}

}

/* Podemos ver nesse programa utilizamos o JPanel que é um
painel onde podemos colocar componentes.

* Os passos para criar uma janela em java são:

* 1. Declarar os objetos que serão usados (botões, janelas,
labels...)

* 2. Instanciar os objetos declarados
```

- * 3. Dimensionar a janela
- * 4. Definir o gerenciador de leiaute da janela (JFrame)
- * 5. Se estiver usando painéis (JPanel), definir o gerenciador de leiaute do painel (JPanel), se estiver usando somente a janela, definir o gerenciador de leiaute da janela.
- * 6. Adicionar os componentes instanciados no painel se estiver usando JPanel ou na janela se estiver usando o gerenciador de leiaute diretamente na janela.
- * 7. Mostrar a janela.
- * A seqüência acima mostra um pequeno algoritmo para construção de interfaces gráficas, mas quero deixar claro que isso não é uma regra rígida pois existe outras maneiras de programar a interface gráfica.* /

```
package UIIEx3;

import UIIEx2.Agenda;

//programa que executa a classe UIIEx2

public class AppAgenda {
    public static void main(String[] args) {
        //instancia um objeto da classe UIIEx2
        Agenda agenda = new Agenda();
        agenda.mostraJanela();
    }
}
```

4)

```
package UIIEx4;

//importa os pacotes gráficos

import javax.swing.*;
```

```
import java.awt.*;

public class CadTelefone {

    public JFrame jCadTelefone;

    public JMenuBar barraMenu;

    public JMenu Telefones;

    public JMenuItem iCadastro, iSair;

    //construtor

    CadTelefone() {

        //instancia o objeto JFrame

        jCadTelefone = new JFrame("Cadastro de Telefones");

        //instancia o objeto JMenuBar

        barraMenu = new JMenuBar();

        //instancia o objeto JMenu

        Telefones = new JMenu("Telefones");

        //instancia o objeto JMenuItem

        iCadastro = new JMenuItem("Cadastro");

        iSair = new JMenuItem("Sair");

        //adiciona os itens de menu ao menu

        Telefones.add(iCadastro);

        //separador

        Telefones.addSeparator();

        Telefones.add(iSair);

        //adiciona o menu à barra de menus

        barraMenu.add(Telefones);
    }
}
```

```
//adiciona a barra de menus à janela  
jCadTelefone.setJMenuBar(barraMenu);  
  
//configura a posição da janela  
jCadTelefone.setLocation(200,300);  
  
//configura o tamanho da janela  
jCadTelefone.setSize(300,400);  
  
//mostra a janela  
jCadTelefone.setVisible(true);  
  
}  
  
}
```

```
package UIIEx4;  
  
public class AppCadastro {  
    public static void main(String[] args){  
        CadTelefone ct = new CadTelefone();  
    }  
}
```

5)

```
package UIIEx5;  
  
//importa os pacotes gráficos  
import javax.swing.*;  
  
import java.awt.*;  
  
//importa os pacotes de eventos  
import java.awt.event.*;  
  
//declara a classe  
public class CadTelefone implements ActionListener{
```

```
public JFrame jCadTelefone;

public JMenuBar barraMenu;

public JMenu Telefones;

public JMenuItem iCadastro, iSair;

public Cadastro cTelefone;

//construtor

CadTelefone() {

    //instancia o objeto JFrame

    jCadTelefone = new JFrame("Cadastro de Telefones");

    //instancia o objeto JMenuBar

    barraMenu = new JMenuBar();

    //instancia o objeto JMenu

    Telefones = new JMenu("Telefones");

    //instancia o objeto JMenuItem

    iCadastro = new JMenuItem("Cadastro");

    iSair = new JMenuItem("Sair");

    //registra o evento ao menu cadastro

    iCadastro.addActionListener(this);

    iSair.addActionListener(this);

    //adiciona os itens de menu ao menu

    Telefones.add(iCadastro);

    //separador

    Telefones.addSeparator();
```

```
Telefones.add(iSair);

//adiciona o menu à barra de menus
barraMenu.add(Telefones);

//adiciona a barra de menus à janela
jCadTelefone.setJMenuBar(barraMenu);

//configura a posição da janela
jCadTelefone.setLocation(200,300);

//configura o tamanho da janela
jCadTelefone.setSize(300,400);

//mostra a janela
jCadTelefone.setVisible(true);
}

//implementa a interface(ActionEvent)
public void actionPerformed(ActionEvent e){

    if(e.getSource() == iCadastro){

        //instancia o objeto janela
        cTelefone = new Cadastro();

        //dispara a janela
        cTelefone.mostraJanela(true);

    }

    if(e.getSource() == iSair){

        System.exit(0);

    }

}
```



```
}  
  
}  
  
/* Observe que quando o menu cadastro é selecionado, o  
objeto da classe Cadastro é instanciado e é enviada uma  
mensagem (true) para o método mostraJanela para que a janela O  
seja mostrada.  
  
A linha System.exit(0) faz com que a aplicação seja encerrada,  
isto é, retirada da memória  
  
*/
```

```
package UIEx5;  
  
import javax.swing.*;  
  
import java.awt.*;  
  
import java.awt.event.*;  
  
public class Cadastro implements ActionListener{  
  
    public JFrame janela;  
  
    public JLabel lblNome, lblTelefone;  
  
    public JTextField tfNome, tfTelefone;  
  
    public JButton botoEnviar, botoSair;  
  
    public MostraCadastro mc;  
  
    //construtor Cadastro com controle de visibilidade  
    Cadastro(){  
  
        //instancia o objeto JFrame  
  
        janela = new JFrame("Cadastro");  
  
        //configura o gerenciador de leiaute para livre  
  
        janela.setLayout(null);  
  

```

```
//configura a localização da janela
janela.setLocation(100,100);

//configura o tamanho da janela
janela.setSize(450,200);

//instancia os objetos JLabel
lblNome = new JLabel("Nome: ");
lblTelefone = new JLabel("Telefone: ");

//instancia os objetos JTextField
tfNome = new JTextField(40);
tfTelefone = new JTextField(11);

//instancia os objetos JButton
btoEnviar = new JButton("Enviar");
btoSair = new JButton("Sair");

//configura a posição dos objetos JLabel
lblNome.setBounds(10, 10, 50, 20);
lblTelefone.setBounds(10, 50, 70, 20);

//configura a posição dos objetos JTextField
tfNome.setBounds(70, 10, 250, 20);
tfTelefone.setBounds(70, 50, 130, 20);

//configura a posição dos objetos JButton
btoEnviar.setBounds(100, 90, 80, 25);
btoSair.setBounds(190, 90, 80, 25 );

//adiciona evento aos botões
```

```
btoEnviar.addActionListener(this);

btoSair.addActionListener(this);

//adiciona os campos à janela

janela.add(lblNome);

janela.add(tfNome);

janela.add(lblTelefone);

janela.add(tfTelefone);

janela.add(btoEnviar);

janela.add(btoSair);

}

public void mostraJanela(boolean mostra){

    janela.setVisible(mostra);

}

//implementa a interface ActionListener

public void actionPerformed(ActionEvent e){

    if(e.getSource() == btoSair){

        janela.dispose();

    }

    if(e.getSource() == btoEnviar){

        mc = new MostraCadastro();

        mc.lblValorNome.setText(tfNome.getText());

        mc.lblValorTelefone.setText(tfTelefone.getText());

    }

}

}
```

```
}

/* Veja que a linha mc.lblValorNome.setText(tfNome.getText());

Através da variável de referência mc do objeto MostraCadastro
ele acessa o método setText do objeto Label através da
variável de referência lblValorNome e é passado como parâmetro
o valor do que está escrito no objeto JTextField referenciado
pela variável de referência tfNome através do método
getText().

O método dispose() é utilizado para fechar uma janela. */
```

```
package UIIEx5;

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

public class MostraCadastro implements ActionListener{

    public JFrame jMostraCadastro;

    public JLabel lblNome, lblValorNome, lblTelefone,
    lblValorTelefone;

    public Cadastro cadastro;

    public JButton btoSair;

    //construtor

    MostraCadastro(){

        //instancia o objeto JFrame

        jMostraCadastro = new JFrame("Mostra dados do cadastr");

        //instancia os objetos de JLabel
```

```
lblNome = new JLabel("Nome: ");  
lblValorNome = new JLabel();  
lblTelefone = new JLabel("Telefone: ");  
lblValorTelefone = new JLabel();  
//instancia o objeto de JButton  
btoSair = new JButton("Sair");  
//registra o evento para o botão  
btoSair.addActionListener(this);  
//posiciona os objetos JLabel  
lblNome.setBounds(10,10, 80, 20);  
lblValorNome.setBounds(50,10,220,20);  
lblTelefone.setBounds(10, 40, 130, 20);  
lblValorTelefone.setBounds(10, 10, 80, 20);  
//adiciona os elementos na janela  
jMostraCadastro.add(lblNome);  
jMostraCadastro.add(lblValorNome);  
jMostraCadastro.add(lblTelefone);  
jMostraCadastro.add(lblValorTelefone);  
//posiciona a janela  
jMostraCadastro.setLocation(100,100);  
//dimensiona a janela  
jMostraCadastro.setSize(600,400);  
//mostra a janela  
jMostraCadastro.setVisible(true);
```

```
}  
  
//implementa a interface ActionListener  
  
public void actionPerformed(ActionEvent e){  
  
    if(e.getSource()==btoSair){  
  
        jMostraCadastro.dispose();  
  
    }  
  
}  
  
}
```

```
package UIIEx5;  
  
public class AppCadTelefone {  
    public static void main(String[] args) {  
        CadTelefone ct = new CadTelefone();  
    }  
}
```

2.8 Preparando o ambiente de trabalho

Instalando o JDK

Vamos ver aqui como baixar e instalar o **JDK** (Kit de Desenvolvimento Java). Primeiro devemos saber qual a versão iremos trabalhar e eu sugiro a 1.6.

No endereço <http://java.sun.com/javase/downloads/index.jsp> clique em



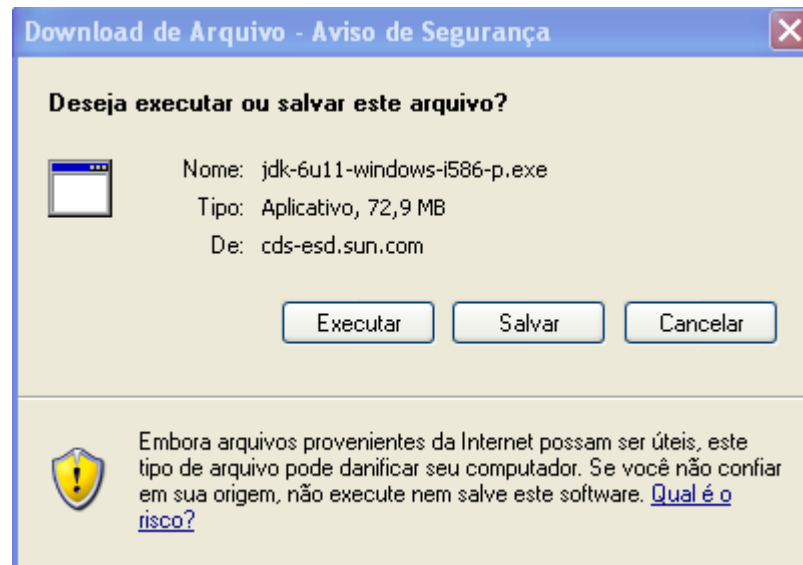
Veja que escolhemos o **JDK 6 update 11** para baixarmos. Esse pacote possui todos os arquivos que precisamos para compilarmos e executarmos aplicações em Java. Clique em **Download**.

Na próxima janela você deve escolher o sistema operacional onde será instalado o Java, selecionar que aceita o contrato e **continue**.



Clique em [jdk-6u11-windows-i586-p.exe](#) para começar a baixar o arquivo.

A janela abaixo será mostrada, clique em Salvar.



Depois de instalado, vá onde você decidiu baixar o arquivo executável e dê um duplo clique em **jdk-6u11-windows-i586-p.exe**.

A janela abaixo será mostrada:

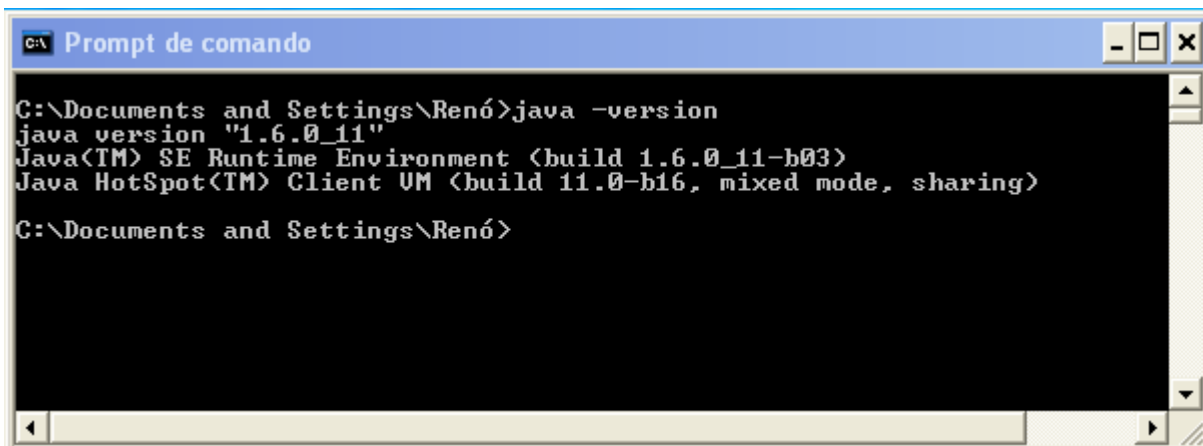


Clique em **Executar**.

Como nossa primeira instação podemos pressionar **next** até o final para termos uma instalação padrão.

Observe que o **JRE** também é instalado quando instalamos o **JDK**.

Para ver se a instalação está correta, vá ao prompt de comando e digite **Java -version**. A janela abaixo deverá ser mostrada.



```
C:\Documents and Settings\Renó>java -version
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode, sharing)
C:\Documents and Settings\Renó>
```

Veja que o comando **Java -version** mostra a versão que estamos utilizando que é a 1.6.0_11.

Depois de instalado o Java corretamente agora vamos instalar o Eclipse que é uma **IDE** (*Integrated Development Enviroment* ou Ambiente de Desenvolvimento Integrado) para programação no Java. Ela também pode ser utilizada para outros programas.

Para baixar o Eclipse vá em <http://www.eclipse.org/downloads/> e selecione



Eclipse IDE for Java Developers (85 MB)

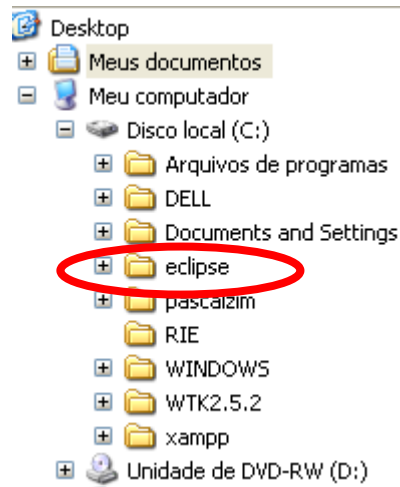
The essential tools for any Java developer, including a Java IDE, a CVS client, XML Editor and Mylyn. [More...](#)

Downloads: 725,082

Veja que a plataforma **Windows** é a padrão.

Escolha o país de onde você quer baixar o Eclipse.

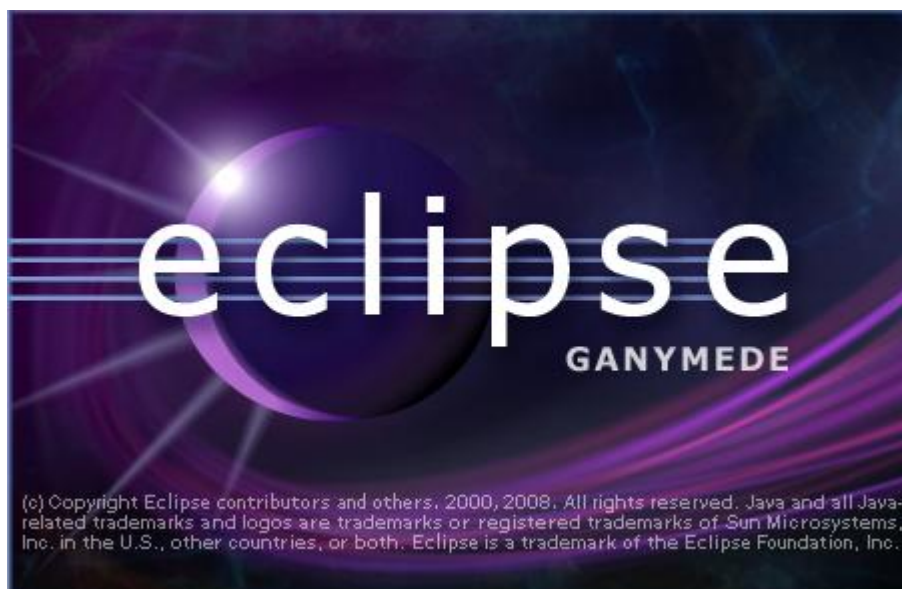
O arquivo é compactado (.zip) e dentro do arquivo tem uma pasta chamada Eclipse. Depois de baixado o arquivo, decompacte-o na raiz da unidade C:\ (por exemplo).



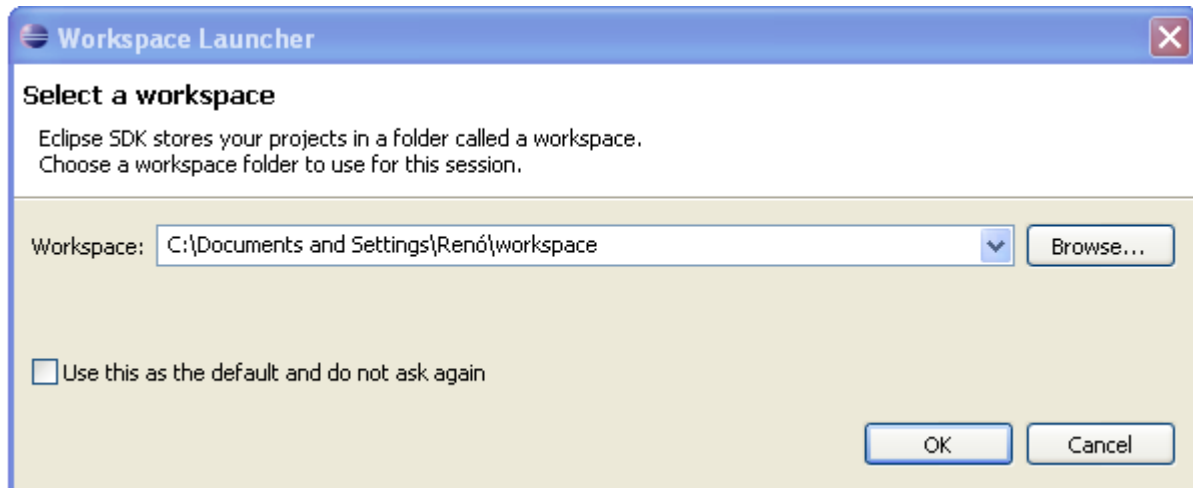
Para executar o Eclipse, abra a pasta que se encontra descompactada na raiz de C:\ (por exemplo) e execute o arquivo eclipse.exe.



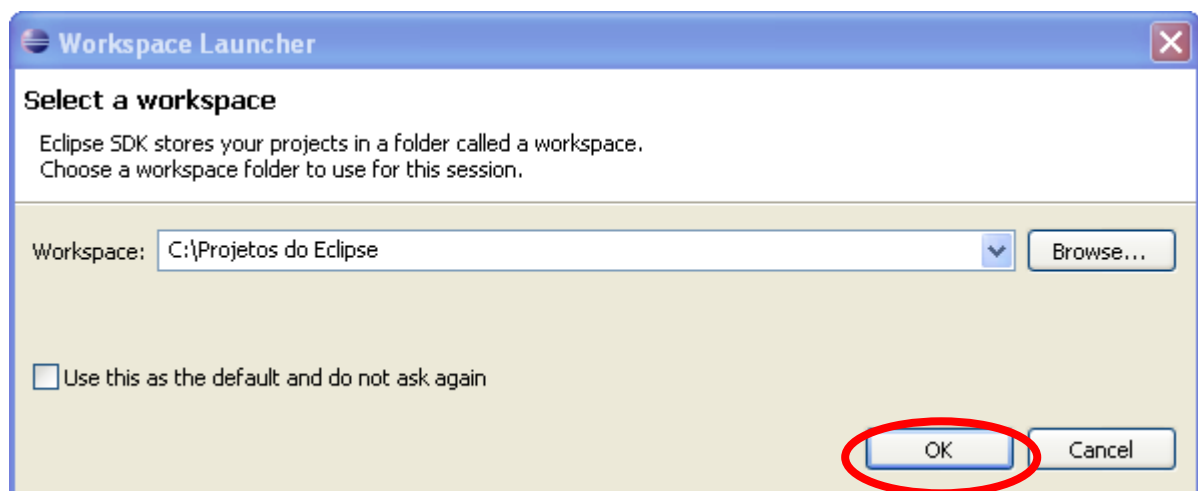
Aparecerá então a janela abaixo:



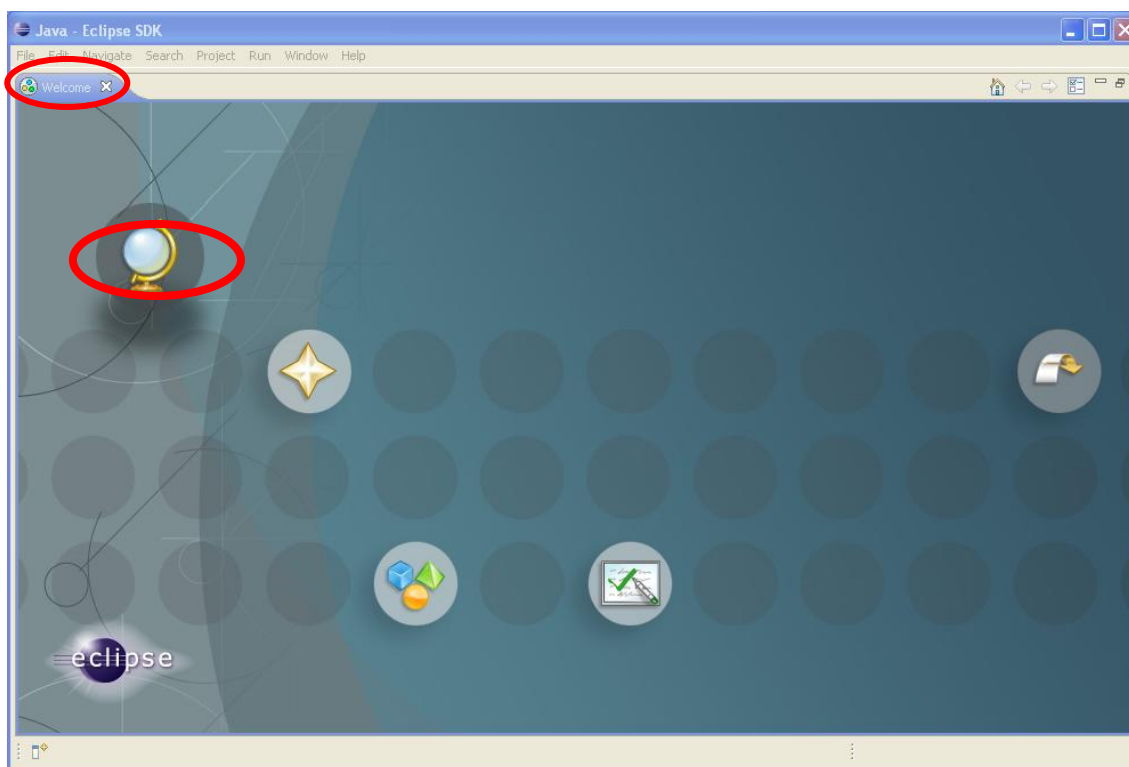
E logo após a janela que solicitará onde você quer fazer sua área de trabalho (**workspace**). Esse diretório será onde seus arquivos serão gravados. Você pode alterar esse caminho. Toda vez que você iniciar o eclipse essa janela irá se abrir. Isso é interessante porque podemos organizar melhor nossos arquivos.



Por padrão o caminho será o definido no campo Workspace, mas você pode alterar. Eu defini o seguinte caminho para meus arquivos e você pode mudar os seus também caso queira.

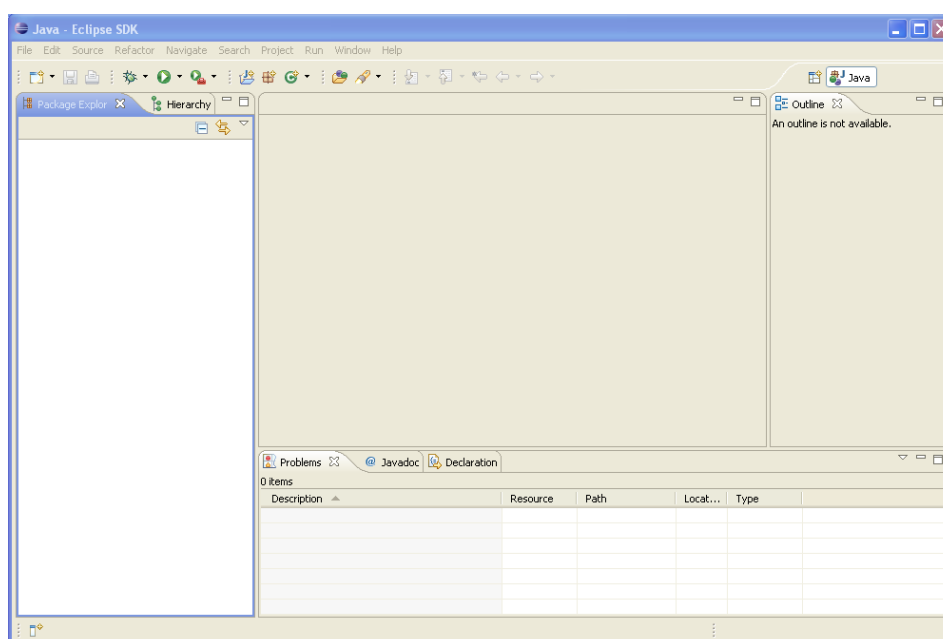


Depois de definido o caminho de onde ficarão seus arquivos, clique em Ok.
Então a janela inicial do Eclipse será mostrada.



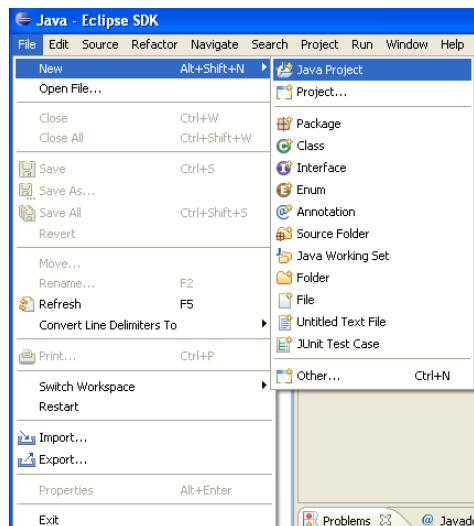
Ao passar o mouse sobre os ícones mostrados uma pequena descrição do ícone é mostrada.

Clique no **x** que está na aba **welcome** então a janela de programação será mostrada.

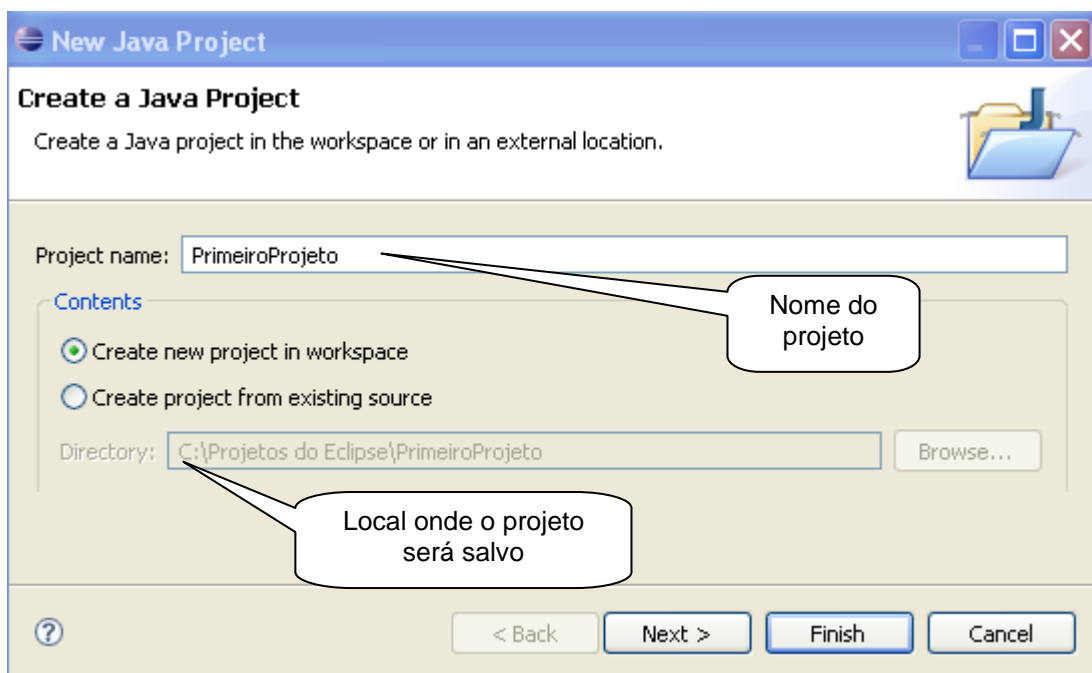


Para iniciar um novo programa devemos primeiro criar um projeto e dentro do projeto iremos colocar os arquivos.

Vamos ver como criar um projeto:



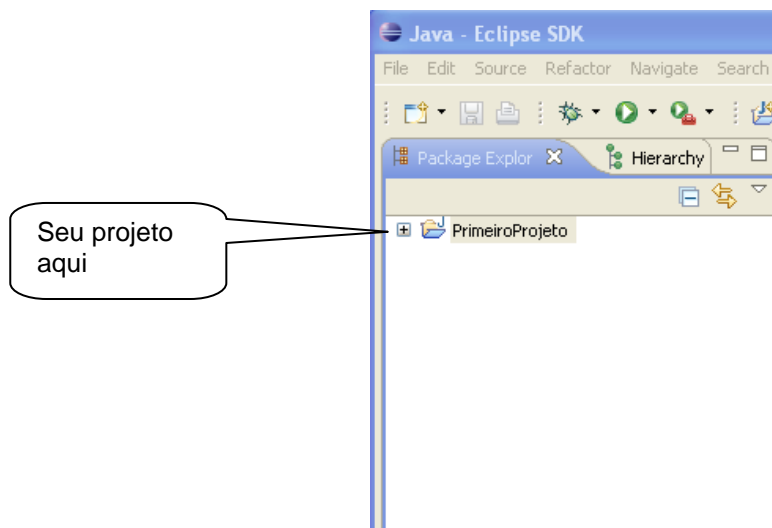
Digite o nome do projeto e veja que ele será criado na workspace definida anteriormente.



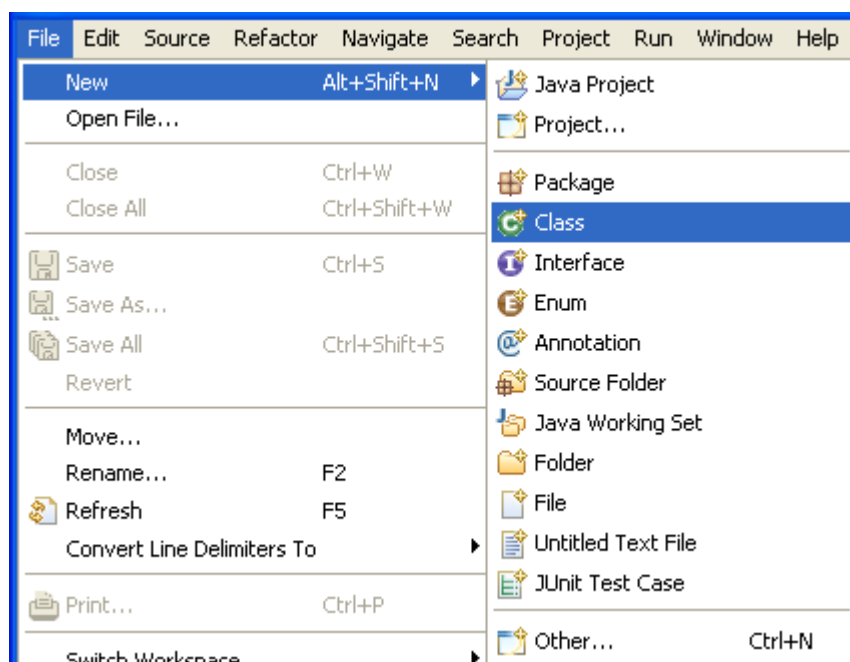
Pressione Finish para encerrar o processo.

Agora que temos um projeto aberto devemos criar o (s) arquivo (s) que será (ão) gravado (s) dentro do projeto.

Seu projeto ficará no lado esquerdo da janela conforme a figura abaixo.



Para criar um arquivo (classe) para seu projeto faça:



Então aparecerá a janela abaixo para ser preenchida.

The screenshot shows the 'New Java Class' dialog box with the following fields and callouts:

- Source folder:** PrimeiroProjeto/src (Callout: Pasta onde ficarão armazenados seus arquivos)
- Package:** (default) (Callout: Nome da classe que será o nome do arquivo também.)
- Enclosing type:** (Callout: Marque aqui para que a classe seja uma aplicação.)
- Name:** AloMundo
- Modifiers:** public (selected), default, private, protected, abstract, final, static
- Superclass:** java.lang.Object
- Interfaces:** (empty)
- Which method stubs would you like to create?**
 - ☒ public static void main(String[] args)
 - ☐ Constructors from superclass
 - ☒ Inherited abstract methods
- Do you want to add comments?** (Configure templates and default value [here](#))
 - ☐ Generate comments

Buttons: Finish, Cancel

Depois de configurada a janela acima pressione Finish para que o “esqueleto” da classe seja montada.

```

public class AloMundo {

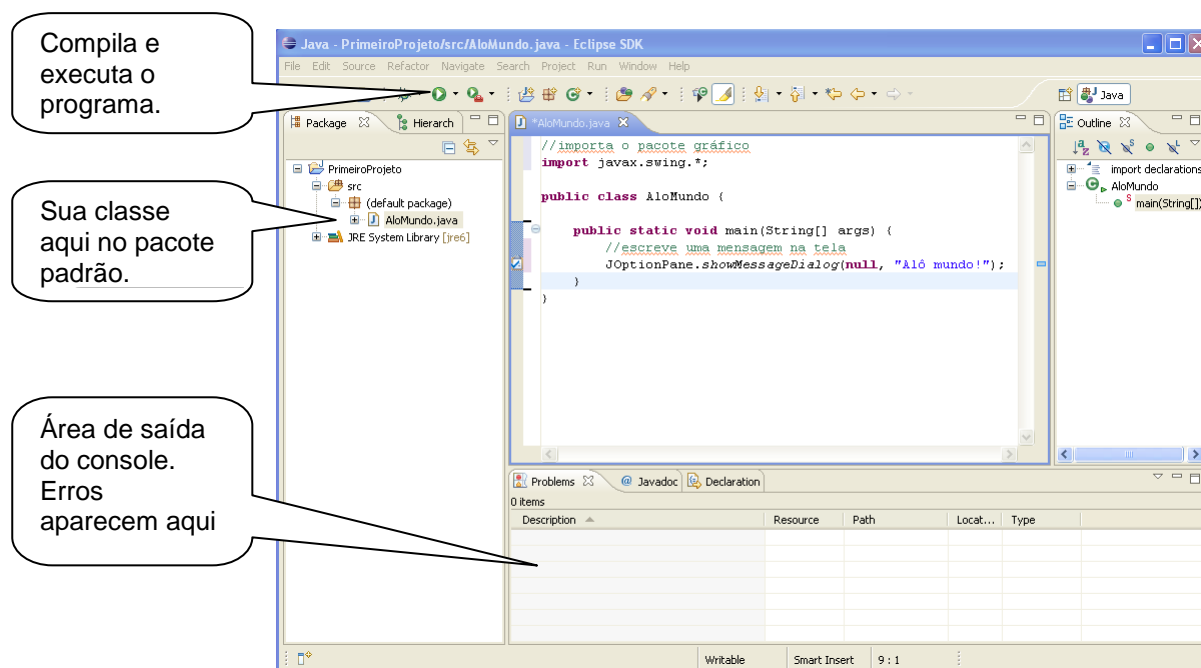
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }

}

```

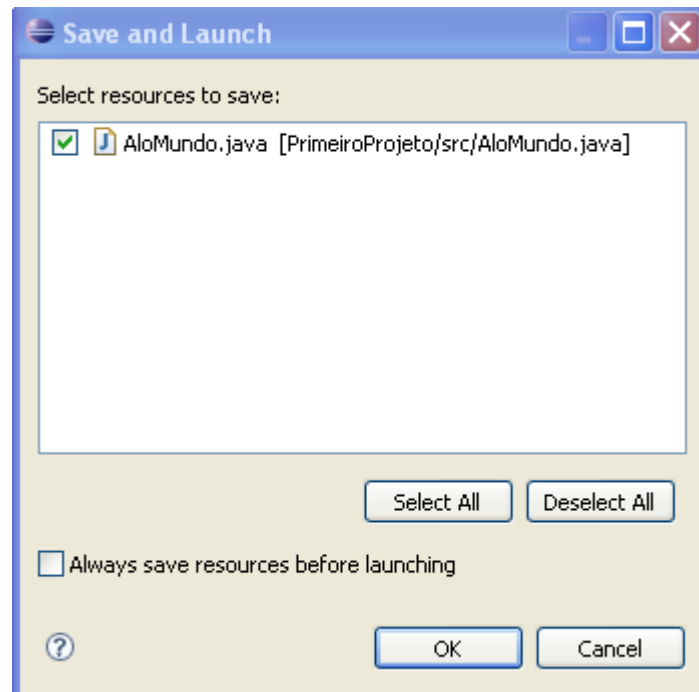
Os comentários podem ser apagados para que possamos escrever nosso programa. Os comentários serão vistos mais tarde nesse curso, eles são utilizados para montar a documentação da classe (quando usamos o aplicativo javadoc.exe).

Agora vamos escrever o famoso “Alo mundo!” na tela usando a interface gráfica. Vejamos.



Depois de escrito o código, pressione **run** para que a aplicação seja compilada e executada.

Antes de executar ele exibe uma janela que pergunta quais arquivos você quer salvar.



Geralmente todos os arquivos ficam já selecionados então você pressiona Ok.

Agora já podemos começar nossos programas em Java.

REFERÊNCIAS

- [1] – Deitel, H.M., Java, como programar, 4.ed. – Porto Alegre: Bookman, 2003.
- [2] – Sierra, K., Bates, B., Certificação Sun para programador 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., São Paulo: Pearson Prentice Hall, 2004.
- [5] – Thompson, M.A., Java 2 & banco de dados: aprenda na prática 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
- [9] - Caelum: Ensino e Soluções em Java, site na internet no endereço: www.caelum.com.br, acessado em 19 de Dezembro de 2007.