



Java básico

JAVA Básico

Sobre o curso

Este curso tem como objetivo preparar o aluno que deseja entrar no ramo da programação, preparando para que você conheça as ferramentas básicas deste programa, que hoje é muito solicitada pelo mercado de trabalho. Sendo assim uma linguagem de programação orientada a objetos.

O que aprender com este curso?

Você aprenderá a utilizar um aplicativo chamado Netbeans, que será a nossa plataforma de desenvolvimento. Noções fundamentais para qualquer linguagem como: Estruturas de controle e tomada de decisão serão conhecidas e ainda, conceitos fundamentais de orientação a objetos, herança, encapsulamento e interface, sendo outros assuntos do curso, importantes para o nosso aprendizado.

Cronograma

- AULA 01** Introdução ao Java
- AULA 02** Interface, componentes e variáveis.
- AULA 03** Operadores matemáticos, relacionais e controle de fluxo
- AULA 04** Estrutura de repetição FOR e WHILE
- AULA 05** Manipulação de Strings
- AULA 06** Variáveis compostas
- AULA 07** Orientação a objetos: Introdução.
- AULA 08** Sem Orientação a objetos
- AULA 09** Orientação a objetos: Classes
- AULA 10** Orientação a objetos: Métodos
- AULA 11** Orientação a objetos: Métodos II
- AULA 12** Encapsulamento
- AULA 13** OOP: Vetor, Laço e Lista
- AULA 14** Herança
- AULA 15** Sobreposição e Interface Gráfica I
- AULA 16** Interface Gráfica II



Java *básico*



Quantidade de Aulas
16 aulas



Carga horária
24 horas



Programas Utilizados
Netbeans

Sumário

1. INTRODUÇÃO AO JAVA

- 1.1. O que é o Java?
- 1.2. Tecnologia Java
- 1.3. Plataforma Java
- 1.4. Java Virtual Machine
- 1.5. Principais IDEs
- 1.6. Exercícios Passo a Passo
- 1.7. Exercícios de Fixação

2. INTERFACE, COMPONENTES E VARIÁVEIS.

- 2.1. O que é o Netbeans?
- 2.2. Programação
- 2.3. Exercícios Passo a Passo
- 2.4. Exercícios de Fixação

3. OPERADORES MATEMÁTICOS, RELACIONAIS E CONTROLE DE FLUXO

- 3.1. Operadores matemáticos
- 3.2. Operadores de igualdade
- 3.3. Operadores relacionais
- 3.4. Controle de fluxo
- 3.5. Exercícios Passo a Passo
- 3.6. Exercícios de Fixação

4. ESTRUTURA DE REPETIÇÃO FOR E WHILE

- 4.1. Estrutura de repetição: FOR()
- 4.2. Estrutura de repetição: WHILE()
- 4.3. Exercícios Passo a Passo
- 4.4. Exercícios de Fixação

5. MANIPULAÇÃO DE STRINGS

- 5.1. Classe Scanner
- 5.2. Exercícios Passo a Passo
- 5.3. Exercícios de Fixação

6. VARIÁVEIS COMPOSTAS

- 6.1. Como criar um vetor:
- 6.2. Procedimentos e funções
- 6.3. Exercícios Passo a Passo
- 6.4. Exercícios de Fixação

7. ORIENTAÇÃO A OBJETOS: INTRODUÇÃO.

- 7.1. Exercícios Passo a Passo

- 7.2. Exercícios de Fixação

8. SEM ORIENTAÇÃO A OBJETOS

- 8.1. Exercícios Passo a Passo
- 8.2. Exercícios de Fixação

9. ORIENTAÇÃO A OBJETOS: CLASSES

- 9.1. Exercícios Passo a Passo
- 9.2. Exercícios de Fixação

10. ORIENTAÇÃO A OBJETOS: MÉTODOS

- 10.1. Métodos:
- 10.2. Exercícios Passo a Passo
- 10.3. Exercícios de Fixação

11. ORIENTAÇÃO A OBJETOS: MÉTODOS II

- 11.1. Variáveis constantes
- 11.2. Exercícios Passo a Passo
- 11.3. Exercícios de Fixação

12. ENCAPSULAMENTO

- 12.1. Private
- 12.2. Exercícios Passo a Passo
- 12.3. Exercícios de Fixação

13. OOP: VETOR, LAÇO E LISTA

- 13.1. Vetor
- 13.2. Laço For each.
- 13.3. ArrayList
- 13.4. Exercícios Passo a Passo
- 13.5. Exercícios de Fixação

14. HERANÇA

- 14.1. Exercícios Passo a Passo
- 14.2. Exercícios de Fixação

15. SOBREPOSIÇÃO E INTERFACE GRÁFICA I

- 15.1. Override
- 15.2. Interface gráfica
- 15.3. Exercícios Passo a Passo
- 15.4. Exercícios de Fixação

16. INTERFACE GRÁFICA II

- 16.1. Exercícios Passo a Passo
- 16.2. Exercícios de Fixação



JAVA Básico

Introdução ao Java

Aula 1

1. INTRODUÇÃO AO JAVA

A apostila tem como objetivo, ser fonte de pesquisa, tanto para as aulas como para os exercícios complementares do curso.

1.1. O que é o Java?

Java é uma linguagem de programação orientada a objetos, desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems.



Existem muitas aplicações e sites que não funcionarão, a menos que você tenha o Java instalado, e mais desses são criados todos os dias. O Java é rápido, seguro e confiável. De laptops a datacenters, consoles de games a supercomputadores científicos, telefones celulares à Internet, o Java está em todos os lugares!

A Sun Microsystems iniciou um projeto para pequenos dispositivos eletrônicos, a ideia era possibilitar a criação de programas portáteis que pudessem ser executados em diversos dispositivos.



Desta forma a equipe teria mais trabalho desenvolvendo programas específicos de acordo com cada sistema dispositivo. Então surgiu a ideia de criar um sistema operacional que permitisse a utilização dos programas. A nova linguagem foi batizada de Oak (carvalho), uma referência ao carvalho que era vista do escritório.



Após ter criado um sistema operacional e uma interface gráfica, a equipe desenvolveu um aparelho chamado Star7, no entanto o projeto foi apresentado para várias empresas e mesmo sendo um produto de alta qualidade o mercado ainda não estava preparado para o Star7. Logo o projeto foi engavetado e a empresa se voltou para a internet que já começava a crescer.

O nome da linguagem desenvolvida pelo projeto Green foi mudada de Oak para Java, que foi uma homenagem à uma ilha da Indonésia de onde os Norte-Americanos importavam o café que era consumido pela equipe de James Gosling. Até 1994, não havia uma aplicação definida para o Java. Foi quando Jonathan Payne e Patrick Naughton criaram um novo navegador para Web que podia executar programas escritos em Java (applets), batizado de Web Runner.

E em 1996, em uma iniciativa inédita, a Sun Microsystems resolveu disponibilizar gratuitamente um kit de desenvolvimento de software para a comunidade, que ficou conhecido como Java Developer's Kit (JDK). Desde então a aceitação da tecnologia Java cresceu rapidamente entre empresas e desenvolvedores. A Sun Microsystems lançou o JDK 1.1 com melhorias significativas para o desenvolvimento de aplicações gráficas e distribuídas. Depois disso, a empresa continuou lançando novas versões gratuitas com novas melhorias e recursos.

Em abril de 2009, a Oracle ofereceu US\$ 7,4 bilhões pela aquisição da Sun Microsystems e a proposta foi aceita. Essa aquisição deu à Oracle a propriedade de vários produtos, incluindo o Java e o sistema operacional Solaris. Em comunicado, a Oracle afirmou que o Java foi o software mais importante adquirido ao longo de sua história. Muitas especulações foram feitas a cerca do futuro do Java depois de passar a ser propriedade da Oracle. Mas com certeza essa aquisição contribuiu muito para que o Java tivesse um salto qualitativo.



1.2. Tecnologia Java

O Java é a base para praticamente todos os tipos de aplicações em rede e é o padrão global para o desenvolvimento e distribuição de aplicações móveis e incorporadas, jogos, conteúdo baseado na Web e softwares corporativos. Com mais de 9 milhões de desenvolvedores em todo o mundo, de forma eficiente, o Java permite que você desenvolva, implante e use aplicações e serviços estimulantes.

De laptops a datacenters, consoles de games a supercomputadores científicos, telefones celulares à Internet, o Java está em todos os lugares!

- 97% dos Desktops Corporativos executam o Java;
- 89% dos Desktops (ou Computadores) nos EUA Executam Java;

- 9 Milhões de Desenvolvedores de Java em Todo o Mundo;
- A Escolha Nº 1 para os Desenvolvedores;
- Plataforma de Desenvolvimento Nº 1;
- 3 Bilhões de Telefones Celulares Executam o Java;
- 100% dos Blu-ray Disc Players Vêm Equipados com o Java;
- 5 bilhões de Placas Java em uso;
- 125 milhões de aparelhos de TV executam o Java;
- 5 dos 5 Principais Fabricantes de Equipamento Original Utilizam o Java ME.

1.3. Plataforma Java

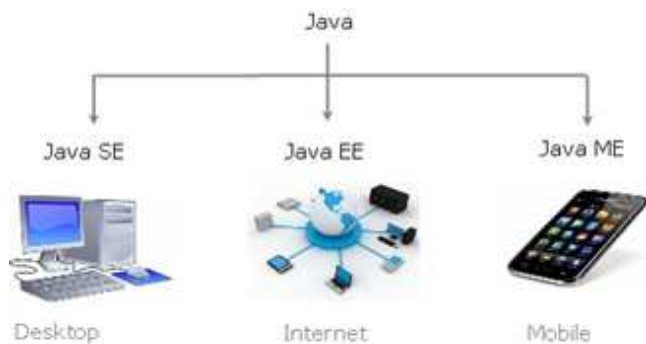
Plataforma Java é o nome dado ao ambiente computacional, ou plataforma, criada pela empresa estadunidense Sun Microsystems, e vendida para a Oracle depois de alguns anos. A plataforma permite desenvolver programas utilizando a linguagem de programação Java. Uma grande vantagem da plataforma é a de não estar presa a um único sistema operacional ou hardware, pois seus programas rodam através de uma máquina virtual que pode ser emulada em qualquer sistema que suporte a linguagem C++.

O universo Java é um vasto conjunto de tecnologias, composto por três plataformas principais que foram criadas para segmentos específicos de aplicações:

Java SE (Java Platform, Standard Edition). É a base da plataforma. Inclui o ambiente de execução e as bibliotecas comuns.

Java EE (Java Platform, Enterprise Edition). A edição voltada para o desenvolvimento de aplicações corporativas e para internet.

Java ME (Java Platform, Micro Edition). A edição para o desenvolvimento de aplicações para dispositivos móveis e embarcados.



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

Java Card. Voltada para dispositivos embarcados com limitações de processamento e armazenamento, como smart cards e o Java Ring.

JavaFX. Plataforma para desenvolvimento de aplicações multimídia em desktop/web (JavaFX Script) e dispositivos móveis (JavaFX Mobile).

Tecnologias Java

A plataforma Java é constituída de um grande número de tecnologias, cada uma provê uma porção distinta de todo o ambiente de desenvolvimento e execução de software. Os usuários finais, tipicamente, interagem com a máquina virtual Java (Java Virtual Machine, ou JVM) e um conjunto padrão de bibliotecas de classe.

Existe um grande número de maneiras de se utilizar uma aplicação Java, incluindo applets embutidas em páginas web, aplicativos de uso geral em desktops, aplicativos em aparelhos celulares e em servidores de aplicações para Internet (Apache Tomcat, Glassfish, JBoss etc).

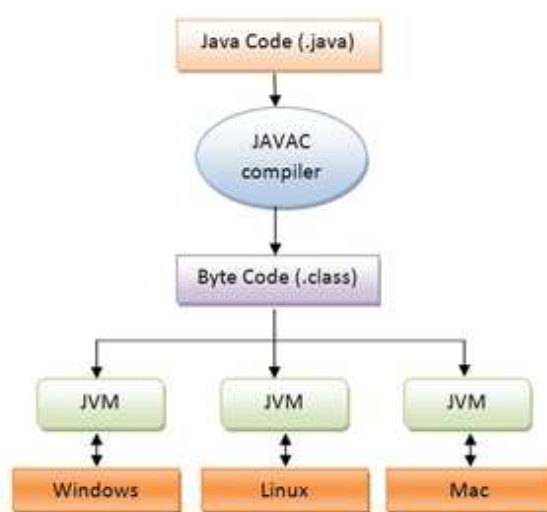
Os desenvolvedores de aplicações em Java utilizam um conjunto de ferramentas de desenvolvimento, o JDK.

Ambiente de execução Java

Um programa escrito para a plataforma Java necessita de dois componentes para ser executado: a máquina virtual Java, e um conjunto de bibliotecas de classe que disponibilizam uma série de serviços para esse programa. O pacote de software que contém a máquina virtual e esta biblioteca de classes é conhecido como JRE (Java Runtime Environment).

1.4. Java Virtual Machine

O coração da plataforma Java é o conceito de um processador "virtual", que executa os programas formados por bytecodes Java. Este bytecode é o mesmo independentemente do hardware ou sistema operacional do sistema em que o programa será executado. A plataforma Java disponibiliza um interpretador, a JVM, que traduz, em tempo de execução, o bytecode para instruções nativas do processador. Isto permite que uma mesma aplicação seja executada em qualquer plataforma computacional que possua uma implementação da máquina virtual.



Desde a versão 1.2 da JRE, a implementação da Sun da JVM inclui um compilador just-in-time (JIT). Com este compilador todo o bytecode de um programa é transformado em instruções nativas e carregado na máquina virtual em uma só operação, permitindo um ganho de desempenho muito grande em comparação com a implementação anterior, onde as instruções em bytecode eram interpretadas uma por vez. O compilador JIT pode ser projetado de acordo com a plataforma ou hardware de destino, e o código que ele gera pode ser otimizado com base na observação de padrões de comportamento dos programas.

Desde a primeira versão, este ambiente de execução vem equipado com gestão automática de memória, realizada por um algoritmo coletor de lixo que liberta o programador das tarefas de alocação e liberação de memória, fonte de muitos erros de programação.

A plataforma Java não é a primeira plataforma baseada em uma máquina virtual, mas é de longe a mais conhecida e a que alcançou maior sucesso. Anteriormente esta tecnologia era utilizada na criação de emuladores para auxílio ao projeto de hardware ou de sistemas operacionais. A plataforma Java foi desenhada para ser implementada inteiramente em software, enquanto permitindo a sua migração de maneira fácil para plataformas de hardware de todos os tipos.

O que é uma IDE?

IDE, ou Ambiente Integral de Desenvolvimento em tradução livre, é um software criado com a finalidade de facilitar a vida dos programadores. Neste tipo de aplicação estão todas as funções necessárias para o desenvolvimento desde programas de computador a aplicativos mobile, assim como alguns recursos que diminuem a ocorrência de erros nas linhas de código.

Se no passado os desenvolvedores precisavam apenas de um editor de texto e de um navegador para criar um software, agora, com os IDEs, eles possuem mais opções para otimizar o tempo gasto com os códigos. Imagine os IDEs como as calculadoras. Logicamente você aprende a fazer as operações matemáticas na escola, mas raramente as faz manualmente quando precisa.

Uma das principais vantagens dos IDEs está na capacidade de compilar bibliotecas completas de linguagem. Outra função bastante comum neste tipo de programa são os debuggers, que apontam os erros que ocasionalmente podem ocorrer ao escrever o código. Alguns IDEs também possuem o autocompletar.

1.5. Principais IDEs

NetBeans – ambiente multiplataforma, tem como principal característica o layout simples e intuitivo. Suporta XML, C, PHP, C++, Ruby e HTML.

Notepad++ – se destaca pelo recurso que permite a identificação da linguagem que está sendo usada. Leve, tem como ponto negativo estar disponível apenas para Windows. Suporta Assembly, Ruby, C, HTML, PHP, JavaScript, ASP, SQL, C++, Java, C#, XML, Objective-C, CSS, Pascal, Perl, Python e Lua.

Sublime Text – um dos mais populares, tem como principal recurso a possibilidade de instalar plugins de acordo com a necessidade do usuário. Suporta C, C++, C#, CSS, HTML, Haskell, Java, Latex, PHP, Ruby, SQL, XML, JavaScript e Groovy.

1.6. Exercícios Passo a Passo

1. Este exercício tem como objetivo configurar o ambiente de sistema. Clique no menu Iniciar e selecione a opção Configurações;
2. Clique na opção Sistema;
3. Clique na categoria Sobre;
4. Clique em Informações do sistema;
5. Clique na categoria "Configurações avançadas do sistema";
6. Caso necessário, clique na aba "Avançado" e, em seguida, clique na opção "Variáveis de Ambiente";
7. Crie uma nova variável de sistema, com o nome JAVA_HOME.

1.7. Exercícios de Fixação

1. Crie um novo projeto chamado "Projeto01". Acesse o menu File, New Project, mantenha selecionado a opção "Java with Ant", avance a próxima etapa para preencher o nome do projeto e finalize a etapa.



JAVA Básico

Interface, componentes e variáveis.

Aula 2

2. INTERFACE, COMPONENTES E VARIÁVEIS.

O NetBeans foi iniciado em 1996 por dois estudantes tchecos na Universidade de Charles, em Praga, quando a linguagem de programação Java ainda não era tão popular como atualmente. Primeiramente o nome do projeto era Xelfi, em alusão ao Delphi, pois a pretensão deste projeto era ter funcionalidades semelhantes aos IDEs então populares do Delphi que eram mais atrativas por serem ferramentas visuais e mais fáceis de usar, porém com o intuito de ser totalmente desenvolvido em Java.

Em 1999 o projeto já havia evoluído para uma IDE proprietário, com o nome de NetBeans DeveloperX2, nome que veio da ideia de reutilização de componentes que era a base do Java. Nessa época a empresa Sun Microsystems havia desistido de sua IDE Java Workshop e, procurando por novas iniciativas, adquiriu o projeto NetBeans DeveloperX2 incorporando-o a sua linha de softwares.

Por alguns meses a Sun mudou o nome do projeto para Forte for Java e o manteve por um bom tempo como software proprietário, porém, em junho de 2000 a Sun disponibilizou o código fonte do IDE NetBeans tornando-o uma plataforma OpenSource. Mais tarde, com a aquisição da Sun Microsystems pela Oracle em 2010, tornou-se parte da Oracle. Ao longo de sua história na Sun Microsystems e Oracle, o NetBeans foi gratuito e de código aberto e foi alavancado pelo seu patrocinador como um mecanismo para impulsionar o ecossistema Java para a frente.

2.1. O que é o Netbeans?

O NetBeans IDE permite o desenvolvimento rápido e fácil de aplicações desktop Java, móveis e Web e também aplicações HTML5 com HTML, JavaScript e CSS. O IDE também fornece um grande conjunto de ferramentas para desenvolvedores de PHP e C/C++. Ela é gratuita e tem código-fonte

aberto, além de uma grande comunidade de usuários e desenvolvedores em todo o mundo.

Instalação

Para instalar o Netbeans, acesse o site <https://netbeans.org>.

Componentes do Netbeans

Na parte superior da janela aparece a barra de menu, logo abaixo aparece alguns atalhos para facilitar alguns comandos.

No lado esquerdo é onde iremos organizar os nossos projetos, na parte superior desta área aparecem três abas, Projects, Files e Services.

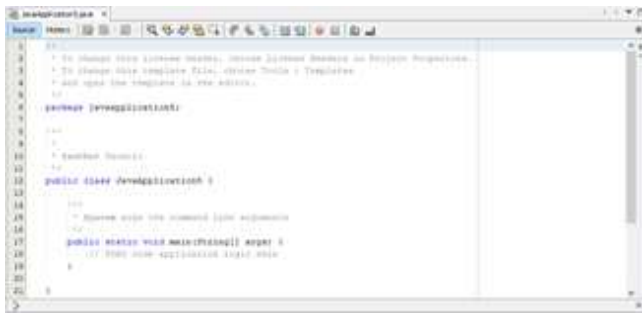
A aba Projects exhibe toda a estrutura do projeto como pastas e classes. Manter uma ampla visão geral de aplicações grandes, com várias pastas e arquivos é uma tarefa árdua. O NetBeans IDE oferece diferentes views de dados, de várias janelas do projeto a ferramentas úteis para configurar suas aplicações e gerenciá-las.

A janela Projetos é o ponto de entrada principal para os códigos-fonte do projeto. Ela mostra uma view lógica do conteúdo importante do projeto.

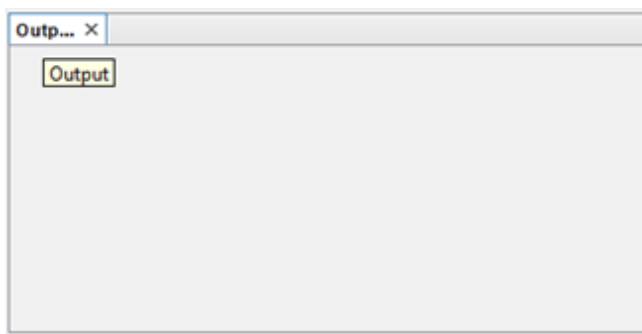
Além da janela Projetos, o IDE oferece a janela Arquivos, em que você pode ver todos os arquivos pertencentes a um projeto, e a janela Favoritos, onde é possível adicionar pastas e arquivos para busca no IDE.

A aba Files exhibe todos os arquivos do projeto e na aba Services temos acesso a serviços como banco de dados, servidores, entre outras aplicações que podemos utilizar no projeto.

No lado direito é onde digitaremos os códigos do projeto.



Abaixo da área de digitação encontramos um espaço para exibição dos resultados chamado de Output.



2.2. Programação

Ao estudar um assunto, é essencial entender o significado dos termos, e com a programação não é diferente, e saber o que cada palavra quer dizer é fundamental para aprender a programar.

Mas antes de iniciarmos é muito importante saber que dentro de um ambiente de programação existem os famosos comentários de código, quando entramos neste mundo da programação é muito ocorrente ouvir “Vou comentar meu código”.

Se você não é programador, ou é novo nesta área, fique tranquilo que vou explicar. Se você já sabe o que é, te convido a ler sobre esse tema mesmo assim.

O que são comentários nos códigos?

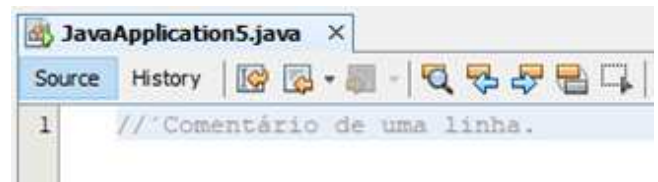
Comentários são textos que você pode inserir no código que não são reconhecidos nem como comandos nem como objetos e, como o próprio nome já diz, não passam de comentários.

Para que servem?

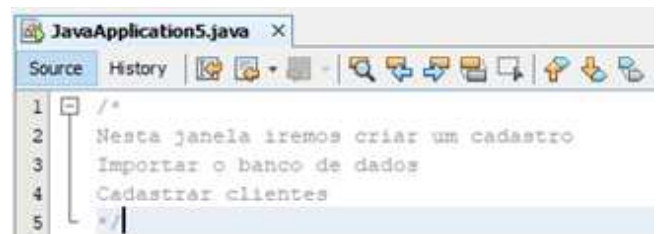
Deixar um breve resumo para que serve aquela parte do código ou onde começa porque se alguém for modificar ou editar seu código acha mais fácil o caminho.

Como usar?

A barra dupla “//” define o comentário de uma linha.



A barra seguida de asterisco “/* */” define um bloco de comentários.



Regras para a definição de palavras.

O Java é totalmente case sensitive, que é uma análise tipográfica da informática.

Observe as palavras "MinhaClasse". Sempre que a primeira letra for maiúscula é uma classe.

Se a primeira letra for minúscula a as outras palavras maiúsculas, ela pode ser um Atributo, uma variável ou um método. Não se preocupe com esta teoria agora, é mais para sabermos como funciona a tipografia.

Se uma palavra estiver em minúscula, pode ter certeza que é o nome de um pacote. Exemplo: aulas ou aulasdeinformatica.

Outra observação é adotar o nome de variáveis sem qualquer caracter especial, ou seja, não usar acentuação, cedilha, espaço em branco, etc. Assim adotamos um padrão e facilitamos na hora de corrigir ou atualizar os códigos em todo o projeto. O único caractere especial e que pode ser utilizado é o underline _. Variáveis não devem começar ou terem

apenas números, exemplo: `int 3num=23` ou `int 100="parcela"`.

Se você quer decorar o que pode e o que não pode, pense da seguinte maneira:

1-Números só se for o segundo caractere;

2-Há 50 palavras reservadas, porém, todas são em Inglês, logo, evite termos em Inglês num primeiro momento.

O que são variáveis?

Uma variável é um objeto capaz de reter e representar um valor ou expressão. Para funcionar corretamente, as variáveis precisam ser definidas por nomes e tipos. Veja os diferentes tipos de dados:

Família	Tipo	Classe Invólucro	Tamanho	Exemplo
Lógico	<code>boolean</code>	<code>Boolean</code>	1 bit	<code>True</code>
Literais	<code>char</code>	<code>Character</code>	1 byte	<code>A</code>
		<code>String</code>	1 byte/cada	<code>Java = 4 bytes</code>
Inteiros	<code>byte</code>	<code>Byte</code>	1 byte	<code>127</code>
	<code>short</code>	<code>Short</code>	2 bytes	<code>32 767</code>
	<code>int</code>	<code>Int</code>	4 bytes	<code>2 147 483</code>
	<code>long</code>	<code>Long</code>	8 bytes	<code>2⁶³</code>
Reais	<code>float</code>	<code>Float</code>	4 bytes	<code>3.4e+38</code>
	<code>double</code>	<code>Double</code>	8 bytes	<code>1.8e+308</code>

O motivo de existir uma variedade de tipos no Java, é que ele foi criado para desenvolver para qualquer dispositivo, como um simples relógio, neste caso, de repente não será necessário usar tanta memória para desenvolver uma pequena aplicação.

Como declarar variáveis:

```
int idade = 20;
```

```
float preco = 420.35f;
```

```
char texto = "B";
```

```
boolean casado = false;
```

Há casos, também, em que até mesmo o compilador não compreende que tipo de dado estamos atribuindo a uma variável.

Em Java, nós podemos fazer uso do que chamamos de indução de tipo ou `typecast`.

O `typecast` dita ao compilador como tal dado deve ser interpretado e manipulado.

```
int idade = (int) 15;
```

```
float preco = (float) 350.45;
```

Saída de dados

Neste momento vamos realizar alguns testes de saída de dados, que nada mais é que pegar algum dados da memória do computador e mostrar em algum lugar para o usuário.

Vamos aprender o primeiro comando de saída de dados, o comando `System.out.print("Olá, Mundo!");`

System

O Java é compatível com diversos sistemas, exemplo, celular, relógio, computador, entre outros, por isso foi o comando `System` que indica sistema, qualquer um destes e de tantos outros que são lançados de tempo em tempo.

Out

O próximo comando é `"out"`, que indica saída, da mesma forma não podemos pensar em somente um tipo de sistema, pode ser saída em um pequeno display como o de um relógio de pulso.

Print

O comando `"print"` indica imprima, mostre, ou seja, cada sistema possui o seu conjunto de saídas e tem uma forma de imprimir.

Criando um exemplo completo:

```
String cliente = "João";
```

```
Int idade = 18;
```

```
String cidade = "São Paulo";
```

```
System.out.print("Cliente:" + cliente);
```

```
System.out.println("Idade:" + idade);
```

```
System.out.print("Cidade:" + cidade);
```

O método print

Imprime na tela e o cursor permanece exatamente após o último caractere impresso, ou seja, qualquer outra coisa que for impressa ou digitada virá imediatamente após (colada) ao text impresso pelo `print`.

O println

Imprime e dá uma quebra de linha (enter), o cursor ficará posicionado na próxima linha, tudo q for impresso ou digitado após o println virá na linha imediatamente abaixo do q foi impresso pelo println.

Para executar qualquer linha de código dentro do Netbeans basta clicar no botão Run Project.

2.3. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado "Aula02". Acesse o disco local;

2. No disco local, abra a pasta "netbeans" e acesse a pasta "bin" para executar o "netbeans64";

3. Crie um novo projeto chamado "Aula02".

2.4. Exercícios de Fixação

1. Este exercício tem como objetivo criar três variáveis e seus respectivos valores, e exibir o resultado, as variáveis e valores devem ser seguidas conforme o indicado.

Código: 105

Produto: Caneca

Preço: 11.50



JAVA Básico

Operadores matemáticos, relacionais e controle de fluxo

Aula 3

3. OPERADORES MATEMÁTICOS, RELACIONAIS E CONTROLE DE FLUXO

3.1. Operadores matemáticos

Os operadores aritméticos funcionam com dois operandos. Por exemplo, a expressão “5 + 1” contém o operador “+” (mais) e os dois operandos “5” e “1”.

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/

Os operadores possuem regras que são aplicadas nas expressões aritméticas do Java, que são as mesmas seguidas em álgebra. Quando dizemos que os operadores são aplicados da esquerda para a direita, estamos nos referindo à sua associatividade.

Operadores de multiplicação, divisão e módulo são aplicadas primeiro. Por exemplo, quando aparecer uma expressão com várias dessas operações, elas serão aplicadas da esquerda para a direita.

As operações de adição e subtração são aplicadas em seguida.

Confira a ordem de avaliação:

* /	Avaliado primeiro. Se houver vários operadores desse tipo serão avaliados da esquerda para a direita
+ -	Avaliado em seguida. Se houver vários operadores desse tipo, serão avaliados da esquerda para a direita.
=	Avaliado por último

Exemplos utilizando o símbolo de adição “+”:

```
System.out.println(10 + 5);
```

Ou

```
Int a = 10;
```

```
Int b = 5;
```

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

Neste caso será impresso o valor 15 que é o resultado da soma de 10 + 5, ou da soma da variável a + variável b.

Exemplos utilizando o símbolo de subtração “-”:

```
System.out.println(12 - 7);
```

Ou

```
Int a = 12;
```

```
Int b = 7;
```

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

Neste caso será impresso o valor 5 que é o resultado da subtração de 12 - 7, ou da subtração da variável a - variável b.

Exemplos utilizando o símbolo de multiplicação “*”:

```
System.out.println(3 * 5);
```

Ou

```
Int a = 3;
```

```
Int b = 5;
```

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

Neste caso será impresso o valor 15 que é o resultado da multiplicação de 3 * 5, ou da multiplicação da variável a * variável b.

Exemplos utilizando o símbolo de divisão “/”:

```
System.out.println(36 / 4);
```

Ou

```
Int a = 36;
```

```
Int b = 4;
```

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

Neste caso será impresso o valor 9 que é o resultado da divisão de 36 / 4, ou da divisão da variável a / variável b.

3.2. Operadores de igualdade

Os operadores de igualdade verificam se o valor ou o resultado da expressão lógica à esquerda é igual (“==”) ou diferente (“!=”) ao da direita, retornando um valor booleano.

Exemplo:

```
Int a = 1;
```

```
Int b = 2;
```

```
Se a == b
```

Imprimir “Valores iguais”

Senão

Imprimir “Valores diferente”

Esse código verifica se duas variáveis contêm o mesmo valor e imprime o resultado. Uma vez que as variáveis A e B possuem valores diferentes, o trecho de código presente no “Senão” será executado.

Opções de operadores de igualdade:

==	Utilizado quando desejamos verificar se uma variável é igual a outra.
!=	Utilizado quando desejamos verificar se uma variável é diferente de outra.

3.3. Operadores relacionais

Os operadores relacionais, assim como os de igualdade, avaliam dois operandos. Neste caso, mais precisamente, definem se o operando à esquerda é menor, menor ou igual, maior ou maior ou igual ao da direita, retornando um valor booleano.

Exemplo:

```
Int a = 1;
```

```
Int b = 2;
```

```
Se a == b
```

Imprimir “Valores iguais”

Senão

Imprimir “Valores diferentes”

Opções de operadores relacionais

>	Utilizado quando desejamos verificar se uma variável é maior que outra.
>=	Utilizado quando desejamos verificar se uma variável é maior ou igual a outra.
<	Utilizado quando desejamos verificar se uma variável é menor que outra.
<=	Utilizado quando desejamos verificar se uma variável é menor ou igual a outra.

```
Int a = 1;
```

```
Int b = 2;
```

```
Se a > b
```

Imprimir “Valor em A é maior”

```
Se a >= b
```

Imprimir “Valor em A é maior ou igual a B”

```
Se a < b
```

Imprimir “Valor em A é menor”

```
Se a <= b
```

Imprimir “Valor em A é menor ou igual a B”

São inúmeras comparações entre duas variáveis para determinar o que será impresso.

Operadores lógicos

Os operadores lógicos representam o recurso que nos permite criar expressões lógicas maiores a partir da junção de duas ou mais expressões. Para isso, aplicamos as operações lógicas E (representado por "&&") e OU (representado por "||").

Opções de operadores lógicos

&& E	Utilizado quando desejamos que as duas expressões sejam verdadeiras.
 OU	Utilizado quando precisamos que pelo menos uma das expressões seja verdadeira.

Exemplo com "OU":

```
String filme;
```

```
filme = "Terror";
```

```
Se filme == "Terror" ou filme == "Suspense"
```

```
Imprimir "Sessão das 22horas";
```

Neste caso a variável filme precisa armazenar a categoria Terror ou Suspense para exibir a mensagem.

Caso nenhuma das categorias seja verdadeira, nenhuma mensagem será exibida.

Exemplo com "E":

```
Int idade;
```

```
Idade = 18;
```

```
Se idade>16 E idade<22
```

```
Imprimir "Vaga para montador"
```

Neste caso, a variável idade é testada, de acordo com uma faixa de valores: deve ser maior que 16 e menor que 22 anos para preencher a vaga de montador na empresa.

3.4. Controle de fluxo

Controle de fluxo é a habilidade de ajustar a maneira como um programa realiza suas tarefas. Por meio de instruções especiais, chamadas comandos, essas tarefas podem ser executadas seletivamente,

repetidamente ou excepcionalmente. Não fosse o controle de fluxo, um programa poderia executar apenas uma única sequência de tarefas, perdendo completamente uma das características mais interessantes da programação: a dinâmica.

O comando IF é comum em muitas linguagens de programação, sua função é verificar se uma condição é verdade ou falsa.

A forma mais simples de controle de fluxo é o comando if-else. Ele é empregado para executar seletivamente ou condicionalmente um outro comando mediante um critério de seleção. Esse critério é dado por uma expressão, cujo valor resultante deve ser um dado do tipo booleano, isto é, true ou false. Se esse valor for true, então o outro comando é executado; se for false, a execução do programa segue adiante. A sintaxe para esse comando é:

```
if ([condição])
```

```
[comando] // Executado se a condição for true
```

Uma variação do comando if-else permite escolher alternadamente entre dois outros comandos a executar. Nesse caso, se o valor da expressão condicional que define o critério de seleção for true, então o primeiro dos outros dois comandos é executado, do contrário, o segundo.

```
if([condição])
```

```
[comando 1] // Executado se a condição for true
```

```
else
```

```
[comando 2] // Executado se a condição for false
```

Exemplo 1:

Vamos classificar uma pessoa como sendo criança.

```
Int idade;
```

```
idade = 10;
```

```
If(idade<=12){
```

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

```
}
```

No exemplo acima, se a idade for maior que 12 nenhuma mensagem será exibida.

Exemplo 2:

Vamos classificar uma pessoa como adolescente ou adulta.

```
Int idade;

idade = 16;

If(idade<18){

    System.out.println("Adolescente");

}else{

    System.out.println("Adulto");

}
```

No exemplo acima, verificamos se a pessoa é adolescente ou adulta.

Exemplo 3:

Vamos classificar uma pessoa como criança, adolescente ou adulta.

```
Int idade;

idade = 16;

If(idade<12){

    System.out.println("Criança");

}else if(idade<18){

    System.out.println("Adolescente");

}

System.out.println("Adulto");

}
```

Exemplo 4:

Vamos verificar se uma pessoa é maior de idade e do sexo masculino, para preencher a vaga de emprego, somente com esses dois critérios ela segue adiante na seleção.

```
int idade;
```

```
String sexo;
```

```
idade = 20;
```

```
sexo = "Masculino";
```

```
if(idade > 18 && sexo == "Masculino"){

    System.out.println("Próxima etapa");

}else{

    System.out.println("Etapa encerrada");

}
```

Exemplo 5:

Vamos verificar se uma pessoa é maior de idade ou do sexo feminino, para preencher a vaga de emprego, com qualquer um destes critérios ela segue adiante na seleção.

```
int idade;

String sexo;

idade = 15;

sexo = "Feminino";

if(idade > 18 || sexo == "Feminino"){

    System.out.println("Próxima etapa");

}else{

    System.out.println("Etapa encerrada");

}
```

Exemplo 6:

Vamos verificar se o um determinado usuário preencheu corretamente seus dados de acesso, login e senha para prosseguir a etapa.

```
String login, senha;

login = "adm";

senha = "a123m";

if(login == "adm" && senha == "a123m"){

    System.out.println("Próxima etapa");

}else{
```

```
        System.out.println("Confira os dados");  
    }  
}
```

3.5. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “Aula03”. Criaremos uma condição para dar R\$ 25,00 de desconto para compras acima de 10 unidades, sendo o preço unitário de R\$ 11,00. Se a quantidade de unidades for abaixo de 10 unidades, o preço total será calculado sem desconto. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Aula03”;

4. Digite o código conforme o indicado.

```
public static void main(String[] args) {  
    // TODO code application logic here  
    double desconto, prUnidade, total;  
    int quantidade;  
    desconto=25;  
    quantidade = 9;  
    prUnidade = 11;  
  
    if(quantidade>10){  
        total = (quantidade * prUnidade) - desconto;  
        System.out.println("total: "+total);  
    }else{  
        total = (quantidade * prUnidade);  
        System.out.println("total: " + total);  
    }  
}
```

5. Para executar, clique no botão Run Project.

6. Informação: A quantidade inicial é 9, por isso deixamos o resultado aparecendo com o valor de R\$ 99,00. Alterando para 14 unidades, o total vai ser de R\$ 129,00, por causa do desconto, caso contrário o total seria de R\$ 154,00.

3.6. Exercícios de Fixação

1. Este exercício tem como objetivo verificar se o usuário é maior de idade, e reside na cidade de Montenegro. Se as informações estiverem corretas, devemos exibir a mensagem “Etapa concluída”, se não, devemos exibir a mensagem, “Tente novamente”.

Crie as variáveis cidade e idade, e para fazer o teste condicional utilize o controle de fluxo if(), para auxiliar faça uso dos operadores relacionais e lógicos.



JAVA Básico

Estrutura de repetição FOR e WHILE

Aula 4

4. ESTRUTURA DE REPETIÇÃO FOR E WHILE

Estudaremos a estrutura de repetição for, a estrutura mais utilizada pelos desenvolvedores na construção de laços de repetição que possuem uma quantidade de ciclos pré-definidos.

Os "Loops", conhecidos como laços, são estruturas de repetição, utilizados para executar, repetidamente, uma instrução ou bloco de instrução enquanto determinada condição for satisfatória.

As estrutura de repetição, possuem quatro elementos fundamentais: inicialização, condição, corpo e iteração. A inicialização compõe-se de todo código que determina a condição inicial da repetição. A condição é uma expressão booleana avaliada após cada leitura do corpo e determina se uma nova leitura deve ser feita ou se a estrutura de repetição deve ser encerrada. O corpo compõe-se de todas as instruções que são executadas repetidamente. A iteração é a instrução que deve ser executada depois do corpo e antes de uma nova repetição.

4.1. Estrutura de repetição: FOR()

Veremos agora o tipo de laço mais importante e usado em Java, o laço for.

A sintaxe do laço for é a seguinte:

```
for(cond. Inicial; teste condicional; após iteração){  
    //código  
}
```

O laço funciona da seguinte maneira:

O laço se inicia pela condição inicial, geralmente se inicia o contador. Esse primeiro estágio SEMPRE acontece.

O segundo estágio é o teste da condição do laço, um simples teste condicional. Caso seja verdade, o código é executado.

Ao término de execução do código, ocorre o fator de mudança, que geralmente é um incremento ou decremento, sempre após cada iteração do looping.

Depois a condição é testada novamente. Caso retorne 'true', o código é executado.

Ao término de execução do código, sempre ocorre o fator de mudança... e assim sucessivamente.

Veja o exemplo prático:

```
int i;  
  
for(i=1;i<=5;i++){  
  
    System.out.println("Número: "+i);  
  
}
```

O exemplo acima tem como propósito criar uma lista numerada de 1 a 5, para isso a variável "i" inicia com o valor 1, em segundo lugar, a condição avalia que o contador deve chegar em 5, em seguida, os valores são incrementados, entenda:

```
i = 1  
  
1 = 1 + 1  
  
2 = 2 + 1  
  
3 = 3 + 1  
  
4 = 4 + 1  
  
5
```

Quando a condição for satisfatória, conforme a condição exige, o programa lista dos e encerra, sai for do laço.

Nota: O FOR() é usado em outras situações que certamente utilizaremos mais tarde.

4.2. Estrutura de repetição: WHILE()

O while é um comando que manda um bloco de código ser executado enquanto uma condição não for satisfeita. Assim, permite que sejam criados loops de execução. O while é um comando muito útil, mas pode ser perigoso, pois, se não tratarmos corretamente o critério de parada, o laço pode não ter fim, e o programa não faz o que deveria e pode entrar em loop infinito, como é chamado.

Frequentemente em nossas aplicações precisamos repetir a execução de um bloco de códigos do programa até que determinada condição seja verdadeira, ou senão até uma quantidade de vezes seja satisfeita. Para que essas repetições sejam possíveis, usamos os laços de repetições do C#.

While – Esta instrução é usada quando não sabemos quantas vezes um determinado bloco de instruções precisa ser repetido. Com ele, a execução das instruções vai continuar até que uma condição seja verdadeira. A condição a ser analisada para a execução do laço de repetição deverá retornar um valor booleano.

Veja a sintaxe:

```
While(teste condicional){  
  
    //comandos;  
  
    serão executados enquanto o teste condicional  
for  
  
    igual a verdadeiro (true)  
  
}
```

Perceba que, somente se a condição for verdadeira o corpo do laço de repetição, com seus respectivos comandos, serão executados. Portanto, o conteúdo será repetido até que esta condição não seja mais verdadeira.

Veja o exemplo:

Será calculado R\$ 10,00 sobre o valor do salário enquanto ele for menor que R\$ 1400,00.

```
double salario;  
  
salario = 1100;
```

```
while(salario<=1400){  
  
    salario +=10;  
  
    System.out.println("Número: "+salario);  
  
}
```

No exemplo acima, o salário é iniciado com o valor de R\$ 1100, quando entra no laço a condição é avaliada, e enquanto for menor ou igual a R\$ 1400, será atribuído R\$ 10,00 sobre o valor, quando chegar em R\$ 1400, será atribuído o valor de R\$ 10,00 e o laço é encerrado.

4.3. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “aula04”. Criaremos uma estrutura de repetição, onde iremos listar oito números em ordem decrescente, começando com o número oito e terminando no número um. Acesse o disco local;

2. No disco local, abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Aula04”;

4. Digite o código conforme o indicado.

```
public static void main(String[] args) {  
    // TODO code application logic here  
    int x;  
    for(x=8; x>=1; x--){  
        System.out.println("Valor de x é igual a "+x);  
    }  
}
```

5. Para executar, clique no botão Run Project.

4.4. Exercícios de Fixação

1. Este exercício tem como objetivo criar uma estrutura de repetição que aumente o bônus em R\$ 20,00 para clientes entre 18 a 26 anos.

Crie a variável bônus, do tipo double, com o valor de R\$ 20,00

Crie a variável idade, do tipo int, com a idade inicial de 18 anos.

Dentro da estrutura while deve ser realizado a condição onde a listagem deve ser até 26 anos. Exibir

a mensagem:

Você tem... anos e ganhou bônus de...



5. MANIPULAÇÃO DE STRINGS

5.1. Classe Scanner

P rincipalmente quando damos início no mundo da programação, com o tempo surge a vontade do desenvolvedor iniciante trabalhar com programas no modo texto (console). Com esse princípio, muitos começam a usar a classe Scanner, pois tem justamente a finalidade de facilitar a entrada de dados no modo Console.

Antes de tudo é necessário saber alguns funcionamentos desta classe.

Primeiramente devemos realizar a seguinte importação:

```
import java.util.Scanner;
```

O próximo a ser usado é o seguinte objeto:

System.in

Que tem como finalidade ler os dados de entrada padrão.

```
import java.util.Scanner;

public class TestaDeclaracaoScanner {

    public static void main(String[] args) {

        //Lê a partir da linha de comando

        Scanner sc1 = new Scanner(System.in);

        String textoString = "João Silva";

        //Lê a partir de uma String

        Scanner sc2 = new Scanner(textoString);

    }

}
```

Confira alguns métodos utilizados:

```
Scanner sc = new Scanner(System.in);
```

```
float numF = sc.nextFloat();
```

```
int num1 = sc.nextInt();
```

```
byte byte1 = sc.nextByte();
```

```
long lg1 = sc.nextLong();
```

```
boolean b1 = sc.nextBoolean();
```

```
double num2 = sc.nextDouble();
```

```
String nome = sc.nextLine();
```

Veja alguns métodos que acompanham a classe:

close()

Fecha o escaneamento de leitura.

next()

Procura e retorna a próxima informação do objeto Scanner que satisfazer uma condição.

nextLine()

Mostra a linha atual do objeto Scanner e avança para a próxima linha.

Veja o exemplo abaixo que permite entrada de dados, o nome:

```
Scanner inserir=new Scanner(System.in);
```

```
System.out.println("Informe o seu nome: ");
```

```
String nome=inserir.next();
```

```
System.out.println("O nome preenchido foi "+nome);
```

Neste momento estudaremos sobre o tratamento de Strings.

Os métodos da classe String são acionados adicionando-se um ponto ao final do nome da String, o da própria string, seguidos pelo nome do método (incluindo os parênteses).

A maioria dos métodos é de RETORNO, ou seja, será necessário declarar uma variável para receber o resultado.

length()

O método retorna o tamanho da String numa variável tipo int. Por exemplo a String "AULAS", o length dela será 5, que é o número de caracteres(letras) dentro da String. Pense que a String é uma coleção de letras.

Veja o exemplo:

```
String texto = "Programação";

int tamanho = texto.length();

System.out.println("Total de: "+tamanho);
```

Resultado:

Podemos usar o método diretamente dentro de um comando de impressão, sem precisar antes guardar em um inteiro, como no exemplo abaixo:

```
System.out.println("Total de: "+texto.length());
```

trim()

O método serve para retirar espaços em branco no início e fim de uma String.

Veja o exemplo:

```
String texto = " Texto Java ";

String remove = texto.trim();

System.out.println("Total de: "+remove+
":espaços");
```

Resultado:

toUpperCase()

O método toUpperCase transforma todas as letras em maiúsculas.

Veja o exemplo:

```
String texto = "Código Java";
```

```
String maiuscula = texto.toUpperCase();
```

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

Resultado:

toLowerCase()

O método toLowerCase converte toda a String para minúsculas.

Veja o exemplo:

```
String texto = "EXECUTAR ROTINA";

String minuscula = texto.toLowerCase();

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

Resultado:

IndexOf()

Esse método é simples: serve para sabermos qual índice determinado pedaço da string corresponde. Ele retorna um valor do tipo int.

```
String site = "www.meusite.com.br";

int indice = site.indexOf(".");

System.out.println("Total de: "+indice+
pontos no endereço");
```

Resultado:

substring()

O método usado para retornar uma String, com as letras indicadas pelos parâmetros início e fim.

Veja o exemplo:

```
String texto = "Estudo Java";

String res = texto.substring(0,6);

System.out.println("Foi extraído: "+res);
```

Resultado:

5.2. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado "Aula05". Utilizaremos a classe Scanner para preencher o nome do aluno e para

exibir, converteremos as letras em minúsculas, mantendo um padrão de saída. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado "Aula05";

4. Digite o código conforme o indicado.

```
package aula05;
import java.util.Scanner;

public class Aula05 {

    public static void main(String[] args) {
        // TODO code application logic here
        Scanner inserir = new Scanner(System.in);
        System.out.println("Informe o Nome do cliente:");
        String nome = inserir.next();
        System.out.println("Cliente: "+nome.toLowerCase());
    }
}
```

5. Para executar, clique no botão Run Project.

5.3. Exercícios de Fixação

1. Este exercício tem como objetivo contabilizar o total de caracteres na primeira ocorrência do ponto "." neste exemplo digitaremos o seguinte e-mail: joao@gmail.com.br. Lembre-se: Abra o netbeans que está no disco local. Crie um novo projeto chamado "Fixacao05". Utilize o método indexOf().



6. VARIÁVEIS COMPOSTAS

Um Array é uma Estrutura de Dados, isto significa que é uma forma de representar, manipular e armazenar dados em um computador. Um Array também é chamado de Variável Composta Homogênea, isto significa que um Array é um tipo de variável que consegue armazenar mais de um dado de um único tipo. Composta: mais de um dado; Homogênea: um único tipo. Exemplos: armazenar 10 salários, armazenar 15 nomes, armazenar 20 notas, etc. Existem vários "tipos" de Arrays, hoje veremos o Array denominado de "Variáveis Compostas Homogêneas Unidimensionais"

Variáveis Compostas Homogêneas Unidimensionais (Array de uma dimensão).

São variáveis compostas que necessitam de apenas um índice para individualizar um elemento do conjunto. Essas variáveis também são chamadas de Vetores.

Índice	i	i+1	i+2	i+3	i+4	i+5
Vetor	Laranja	Mamão	Goiaba	Abacaxi	Morango	Manga
Posição	0	1	2	3	4	5

A Figura acima ilustra um Array de uma dimensão com seis posições. Cada quadrado desse Array é correspondente a um espaço de memória que armazena um dado. Como o Array tem seis posições, então, são seis espaços de memória que serão utilizados para armazenar esses dados.

O índice nos ajuda a percorrer o Array, permitindo que o elemento daquela determinada posição possa ser armazenado, acessado, atualizado, excluído, enfim, manipulado. O índice pode "andar" da esquerda pra direita, por isso vocês veem escrito "i", "i + 1", e assim por diante. O índice "i" é a variável de incremento (contador) e a cada iteração que

fizemos, ela é somada de um, para podermos caminhar no array.

6.1. Como criar um vetor:

O exemplo abaixo mostra 5 variáveis sendo criadas, do tipo inteiro, que é a forma que vimos até agora:

```
int n1, n2, n3, n4, n5;
```

Variáveis compostas:

```
tipo_do_vetor nome_do_vetor[tamanho]
```

Vamos conferir alguns detalhes sobre um vetor. Os colchetes "[]", tem como finalidade definir o número que corresponde à posição do elemento desejado, ou seja, o índice do elemento. O nome com esse índice, pode ser utilizado em seu programa, como se fosse uma variável qualquer.

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

O new int[5] estamos criando um objeto com 5 elementos, ele possui métodos e atributos.

No Java a primeira posição de um vetor é zero. Sendo assim para declarar uma variável com 5 elementos utilizaremos a estrutura conforme mostrado na imagem.

Outra forma de declarar uma variável do tipo vetor é seguindo este segundo formato, definindo o tipo e dentro do bloco o número de elementos. int n[]={7,6,4,2,9}

Exemplo:

```
String  frutas[]    =    {"Laranja","Mamão",  
"Manga","Morango"};
```

```
for(int i=0;i<=3;i++){  
  
System.out.println("total: "+frutas[i]);  
  
}
```

No exemplo acima criamos uma variável armazenando 4 elementos, lembrando que inicia em zero.

Isso significa que frutas[0] terá o valor Laranja, frutas[1] terá o valor Mamão, frutas[2] terá o valor Manga, frutas[3] terá o valor Morango.

Utilizando o laço de repetição for() foi possível criar um contador de elementos e listando as posições dentro da variável composta.

Length

A length é um atributo de um array e representa o comprimento de uma string.

Sintaxe:

```
array.length
```

Veja o exemplo:

Usando o mesmo recurso anterior, veja como é criado a condição.

```
String  frutas[]  =  {"Laranja","Mamão","Manga",  
"Morango","Cereja","Caqui"};
```

```
for(int i=0; i  
  
System.out.println("total: "+frutas[i]);  
  
}
```

A finalidade do length é substituir o uso da numeração, imagine que em um momento o vetor possui 6 elementos, em outro possui 70, então para ser algo dinâmico utilizamos o length, que vai retornar o comprimento do vetor.

foreach

A utilização é bem simples e direta. Vamos usar essa variante do for que percorre sempre, do começo

ao fim, todos os elementos de um Array.

É bem útil, também, em termos de precaução e organização, pois alguns programadores não gostam de usar o índice 0, usam direto o índice 1 do array, ou as vezes nos confundimos e passamos (durante as iterações) do limite do array. Isso pode ser facilmente evitado usando o laço for modificado para percorrer os elementos de um array.

A sintaxe do for each é a seguinte:

```
for ( tipo  variavel_do_tipo_do_seuArray  :  
seuArray){
```

```
//seu código
```

```
}
```

Se o 'seuArray' for de inteiro, ficaria:

```
for (int count : seuArray){
```

```
...
```

```
}
```

Como podemos interpretar esse laço foreach?

Muito simples, o inteiro 'count' vai receber, a cada iteração, todos os valores de 'seuArray'.

Ou seja, ele vai percorrer todo o seu Array e receber seus valores, na ordem (do começo para o fim), um por vez. E com esses valores você vai fazer o que desejar.

Veja o exemplo:

```
String  estados[]={ "rs","sc","sp","rj","mg",  
"mt","ms"};
```

```
for(String itens: estados){
```

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

```
}
```

Ordenando Arrays em Java.

Pode haver uma situação em que seja necessário que o desenvolvedor organize um Array em Java. Nesse caso podemos utilizar a classe java.util.Arrays que é capaz de fornecer vários métodos utilitários, a fim de classificar Arrays de qualquer tipo.

Poderíamos incluir tipos primitivos, na medida em que os objetos estão em arrays, eles estarão no pacote `java.util` que é capaz de expor todos os métodos de classificação relacionados como funções de utilitário estático.

Facilmente pode-se ir em frente e acessar `sort()` como `Arrays.sort()` e só passar o Array e isso resultará em um objeto Array.

Pode ser classificado em ordem crescente, decrescente ou qualquer ordem personalizada definida pelo comparador personalizado em Java.

```
import java.util.Arrays;

public class Fixacao06 {

    public static void main(String[] args) {

        // TODO code application logic here

        String cores[] = {"Vermelho", "Amarelo",
            "Laranja", "Azul", "Verde", "Branco", "Violeta"};

        Arrays.sort(cores);

        for(int i = 0; i < 7; i++){

            System.out.println(cores[i]);

        }

    }

}
```

6.2. Procedimentos e funções

Procedimento: é algo que deve ser feito uma ou várias vezes, sempre que for necessário.

- Pode, ou não, receber parâmetros.
- Não retorna um resultado.

Exemplos:

- Gravar dados em arquivo no disco;
- Enviar documento para a impressora;
- Ordenar que o computador desligue.

Função: é algo que deve ser feito uma ou várias vezes, sempre que for necessário para se obter um

resultado.

- Pode, ou não, receber parâmetros;
- Sempre retorna um resultado;
- Exemplos:
- Mostrar um menu e retornar a opção;
- Solicitar uma entrada ao usuário;

Como declarar uma função:

Toda função deve ter um tipo (`String`, `int`, `float`), o qual indicara o tipo de seu valor de retorno (saída).

```
int soma(int a, int b) {
    int c;
    c = a + b;
    return c;
}
```

Os argumentos (ou parâmetros) indicam o tipo e quais valores são esperados para serem manipulados pela função (entrada).

```
int soma(int a, int b) {
    int c;
    c = a + b;
    return c;
}
```

Veja o exemplo sem parâmetro:

```
static String mensagem(){

    String msg="Bom dia";

    System.out.println(msg);

    return msg;

}

public static void main(String[] args) {

    // TODO code application logic here
```

```

mensagem();
}

```

Veja o exemplo passando parâmetro:

```

static int quantidade(int qt1, int qt2, int qt3){

    int soma=qt1+qt2+qt3;

    return soma;

}

public static void main(String[] args) {

    // TODO code application logic here

    int total=quantidade(5,8,4);

    System.out.println("total de: "+total);

}

```

6.3. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “Aula06”. Criaremos uma lista de nomes e classificaremos em ordem alfabética. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Aula06”;

4. Digite o código conforme o indicado e, em seguida, clique no botão Run Project.

```

import java.util.Arrays;

public class Vetor2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String nomes[] = {"Ana", "João", "Marcos", "Carlos", "Edson", "Camila"};
        Arrays.sort(nomes);
        for(int i=0; i<nomes.length; i++){
            System.out.println(nomes[i]);
        }
    }

}

```

6.4. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado “Fixacao06”. Crie uma lista com as seguintes cores: Vermelho, Amarelo, Laranja, Azul, Verde, Branco e Violeta, lembre-se: utilize o comando for().



JAVA Básico

Orientação a objetos: Introdução.

Aula 7

7. ORIENTAÇÃO A OBJETOS: INTRODUÇÃO.

Uma programação bem conhecida é a estruturada, um programa possui uma sequência de comandos a serem executados, uma outra sequência que só deve ser executada se uma condição for satisfatória, e outra rotina que cria sequência executadas por repetição até que uma condição seja satisfeita.

A programação orientada a objetos é uma forma especial de programar, mais próximo de como expressaríamos as coisas na vida real.

Um objeto é uma coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas características, comportamentos e estado atual.

Daí o nome objeto, algo genérico, que pode representar qualquer coisa tangível.

Podemos usar como exemplo um carro, computador, cliente, conta bancária, etc.

Os elementos básicos da Programação Orientada a Objetos são:

Objetos, Classes e Instâncias.

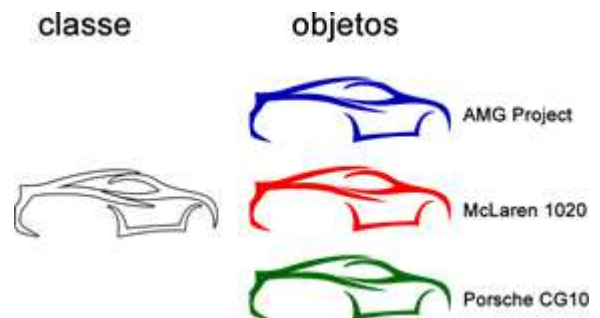
Pode parecer algo tão citado, mas vou utilizar o carro como exemplo de um objeto.

Quando chegamos na loja existem diversos carros, similares, onde podemos observar que possuem quatro rodas, volante, faróis, entre outras partes.

Podemos informar que o nosso carro é único porque ele possui um registro, porém iremos encontrar outros carros com os mesmos atributos, outros totalmente diferentes, mas que são considerados, carros.

Esse é um motivo onde podemos dizer que seu objeto pode ser classificado, ou que pertence à uma

classe, chamada “carro”. E que o nosso carro é uma instância dessa classe.



Assim, uma classe é um conjunto de características e comportamentos que definem o conjunto de objetos pertencentes a ela. Podemos nos referir uma classe a um molde. O molde torna o objeto algo concreto. Quando a partir de um molde surge um objeto, chamamos de instância da classe.

Instanciar

Por exemplo, você cria uma classe Cachorro. Você criou apenas a idéia de um cachorro, mas pra poder usar efetivamente, vai ter que criar uma instância desse cachorro (um objeto).

Tipo:

Cachorro rex = new Cachorro(azul);

Atributos

Os objetos possuem atributos, que são as características de um objeto.

Por exemplo, o carro possui como atributo, cor, marca, modelo, entre outros.

Métodos

Os métodos definem o comportamento dos objetos de uma classe, as ações que um objeto pode realizar.

Exemplo de comportamentos, o carro tem ação de ligar e acelerar.

Abstração

Abstração para a Orientação a Objetos, nada mais é do que você observar comportamentos e estruturas do dia a dia, e transformá-los em uma linguagem computacional.

7.1. Exercícios Passo a Passo

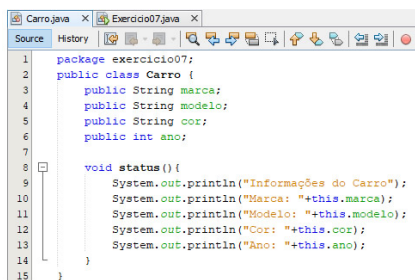
1. Este exercício tem como objetivo criar um projeto chamado “Exercicio07”. Criaremos um objeto chamado carro, com os seguintes atributos: marca, modelo, cor e ano. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Exercicio07”.

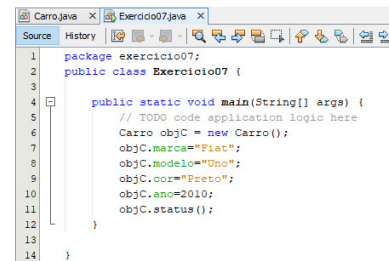
4. A classe principal foi criada. Criaremos uma nova classe chamada Carro. Clique com o botão direito do mouse no pacote exercicio07, escolha New, Java Class. Digite “Carro” e pressione Finish.

5. Digite conforme o indicado, não esqueça de clicar na aba Carro;



```
1 package exercicio07;
2 public class Carro {
3     public String marca;
4     public String modelo;
5     public String cor;
6     public int ano;
7
8     void status() {
9         System.out.println("Informações do Carro");
10        System.out.println("Marca: "+this.marca);
11        System.out.println("Modelo: "+this.modelo);
12        System.out.println("Cor: "+this.cor);
13        System.out.println("Ano: "+this.ano);
14    }
15 }
```

6. Clique na aba Exercicio07.java e digite o conforme o indicado. Em seguida, clique no botão Run Project para executar o exercício.



```
1 package exercicio07;
2 public class Exercicio07 {
3
4     public static void main(String[] args) {
5         // TODO code application logic here
6         Carro objC = new Carro();
7         objC.marca="Fiat";
8         objC.modelo="Uno";
9         objC.cor="Preto";
10        objC.ano=2010;
11        objC.status();
12    }
13
14 }
```

7.2. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado “Fixacao07”.

Criaremos um objeto chamado "rádio", e os seguintes atributos: fabricante, tamanho, ligado.

O atributo "fabricante" recebe o valor “Philips”.

O atributo "tamanho" recebe o valor “médio”

O atributo "ligado", recebe valor “true”

Não esqueça de criar um método chamado ligar, onde iremos avaliar, se "ligado", for igual a true, exibir a mensagem: ligado. Caso contrário, exibir a mensagem: desligado. Por fim, criar o método status, responsável por enviar os atributos fabricante e tamanho.



8. SEM ORIENTAÇÃO A OBJETOS

Sabemos como é importante criar interatividade entre o usuário e o programa, permitindo que ele possa preencher os dados e ficar mais próximo da realidade. Por esse motivo usamos a classe Scanner.

Quando utilizamos a classe Scanner, o compilador pedirá para fazer a seguinte importação:

```
import java.util.Scanner;
```

Essa classe ajuda na leitura dos dados informados. Para fazer essa ação, é necessário criar um objeto do tipo Scanner, que passa como argumento o objeto System.in.

O System.in faz a leitura do que se escreve no teclado.

Vamos relembrar a forma de usar:

```
import java.util.Scanner;  
String nome;  
Scanner sc = new Scanner(System.in);  
System.out.print("Digite o seu nome:");  
nome = sc.nextLine();
```

Para obter os dados de um determinado tipo usamos:

```
float numF = sc.nextFloat();  
int num1 = sc.nextInt();  
byte byte1 = sc.nextByte();  
long lg1 = sc.nextLong();  
boolean b1 = sc.nextBoolean();  
double num2 = sc.nextDouble();  
String nome = sc.nextLine();
```

Usando **next()** só retornará o que vem antes de um espaço. **nextLine()** move automaticamente o scanner depois de retornar a linha atual.

O método **nextInt()** permite capturar as entradas de dados do tipo inteiro, como números do tipo: 1, 20, 140...

O método **nextDouble()** captura um valor com casas decimais, e armazena em uma variável do mesmo tipo. Exemplo: 4,50 ou 100,50.

Além de permitir que o usuário possa preencher os dados, podemos realizar um teste usando fórmulas, calculando valores dentro de variáveis.

Os operadores aritméticos funcionam com dois operandos. Por exemplo, a expressão "5 + 1" contém o operador "+" (mais) e os dois operandos "5" e "1".

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/

Os operadores possuem regras que são aplicadas nas expressões aritméticas do Java, que são as mesmas seguidas em álgebra. Quando dizemos que os operadores são aplicados da esquerda para a direita, estamos nos referindo à sua associatividade.

Operadores de multiplicação, divisão e módulo são aplicadas primeiro. Por exemplo, quando aparecer uma expressão com várias dessas operações, elas serão aplicadas da esquerda para a direita.

As operações de adição e subtração são aplicadas em seguida.

Confira a ordem de avaliação:

* /	Avaliado primeiro. Se houver vários operadores desse tipo serão avaliados da esquerda para a direita
+ -	Avaliado em seguida. Se houver vários operadores desse tipo, serão avaliados da esquerda para a direita.
=	Avaliado por último

Exemplos utilizando o símbolo de adição "+":

System.out.println(10 + 5);

Ou

Int a = 10;

Int b = 5;

System.out.println(a + b);

Como no exemplo da adição, podemos criar outras operações matemáticas, já vistas na **aula 03**.

Símbolo + é chamado de adição, utilizado para somar o valor de dois operandos.

Símbolo - é chamado de subtração, utilizado para subtrair o valor de dois operandos.

Símbolo * é chamado de multiplicação, utilizado para multiplicar o valor de dois operandos.

Símbolo / é chamado de divisão, utilizado para dividir o valor de dois operandos.

A apostila foi apenas uma revisão de conteúdo, é muito importante relembrar o que já foi visto para fortalecer o que foi aprendido.

8.1. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado "Exercicio08". Criaremos um controle de usuários. Precisaremos dos atributos usuário e senha. Acesse o disco local;

2. Abra a pasta netbeans e acesse a pasta bin para executar o programa;

3. Crie um novo projeto chamado "Exercicio08".

4. Digite o código conforme o indicado.

```

1 package exercicio08;
2 import java.util.Scanner;
3 public class Exercicio08 {
4
5     public static void main(String[] args) {
6         // TODO code application logic here
7         Scanner sc = new Scanner(System.in);
8         String usuario;
9         String senha;
10        System.out.print("Digite o usuário:");
11        usuario=sc.next();
12        System.out.print("Digite a senha:");
13        senha=sc.next();
14        if(usuario.equals("admin") && senha.equals("op123")){
15            System.out.println("Acesso permitido");
16        }else{
17            System.out.println("Acesso negado");
18        }
19    }
20 }
21

```

8.2. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao08".

Criaremos um programa que leia o nome de um determinado produto, o preço de custo, o preço de venda, e calcule o lucro. Será avaliado se o lucro for acima de R\$ 100,00 exibir produto com desconto. É necessário importar a classe Scanner().



9. ORIENTAÇÃO A OBJETOS: CLASSES

Podemos dizer que são "receitas" de um objeto, possui características e comportamentos, permite armazenar propriedades e métodos. Geralmente uma classe representa um lugar, pessoa, algo "abstrato".

Sendo assim, é possível ter vários objetos do mesmo tipo, que compartilham características em comum.

Alguns objetos possuem semelhanças, desta forma temos uma vantagem, criar modelos para esses objetos. Esse modelo é chamado de CLASSE. As classes são tipos que podem ser criados.

Por definição: Uma classe é um modelo (protótipo) que define as variáveis (estado) e os métodos (comportamento) comuns a todos os objetos do mesmo tipo.

Preste atenção:

- Toda classe deve possuir um nome;
- Toda classe deve possuir as visibilidades, exemplo: public, private, protected;
- Toda classe deve possuir características e ações.

Podemos dizer que os atributos são variáveis ou campos que armazenam os diferentes valores que as características dos objetos podem conter.

A sintaxe básica para declaração de uma classe:

```
class nomeDaClasse{  
    // atributos ou propriedades  
  
    // métodos  
  
}
```

Iniciaremos um exemplo utilizando um carro, mostrando apenas o que ele tem.

```
class Carro{  
    String marca;  
    String placa;  
    float preco;  
}
```

Lembre-se: os atributos são as características de um determinado objeto.

Utilizamos o exemplo do "Carro", onde foi possível definir os atributos, tais como: marca, placa, preço, rodas entre outros atributos.

Quando utilizamos o comando "**new**" estamos criando uma variável que permite utilizar os atributos e métodos de uma classe.

O **New** trata de reservar memória o suficiente para o objeto e criar automaticamente uma referência a ele. Para **new** conseguir determinar o objeto, precisamos usar o método construtor que será usado como base para instanciar a classe e gerar o objeto.

Declaramos uma variável qualquer como sendo do tipo da classe, depois instanciamos o objeto atribuindo a variável o resultado obtido por **new** mais o método construtor.

Exemplo:

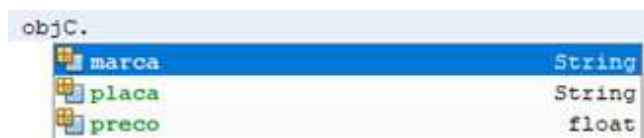
```
class Carro{  
    String marca;  
    String placa;  
    float preco;  
}  
  
Carro objC = new Carro();
```

Essa é a forma de instanciarmos um objeto, neste caso, do tipo **Carro** e amarrá-lo a uma referência também do mesmo tipo, chamado de **objC**.

Em primeiro lugar vem o tipo "Carro" da referência, depois o nome da referência "objC", logo em seguida o operador de atribuição "=", após o operador "new", e por último, a chamada do construtor da classe do objeto, Carro().

Com a referência "objC", faremos a chamada da seguinte forma:

Por exemplo, para informar a marca do carro, devemos proceder da seguinte forma:



Para definir o valor dos atributos que serão armazenados em Carro, precisaremos acessar o objeto que vive na memória. Fazemos isso utilizando o operador ".", informando qual é o atributo que queremos acessar. Podemos, por exemplo, guardar o valor 4 como o número de rodas do carro. Observe o código abaixo:

```
class Carro{
    String marca;
    String placa;
    float preco;
}

Carro objC = new Carro();
objC.placa = "LSN1233";
```

Com esse código, estamos navegando na referência armazenada no "objC", e acessando o atributo placa do objeto Carro que vive na memória. Dentro desse atributo colocamos o valor LSN1233. Podemos fazer o mesmo para os outros atributos do Carro:

```
Carro objC = new Carro();
objC.placa = "LSN1233";
objC.marca = "Ford";
objC.preco = 40600.30f;
```

Depois de executarmos esse código, veja o resultado.

```
run:
Carro
Marca:Ford
Placa:LSN1233
Preço:40600.3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Veja que, quando utilizamos um objeto para guardar informações, todos os atributos ficam agrupados dentro de um único objeto na memória, e não espalhados dentro de diversas variáveis diferentes.

9.1. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado "exercicio09". Criaremos uma classe chamada Produto, e os seguintes atributos: descrição, quantidade, preço e total. Deverá ser exibido o valor total. Acesse o disco local;

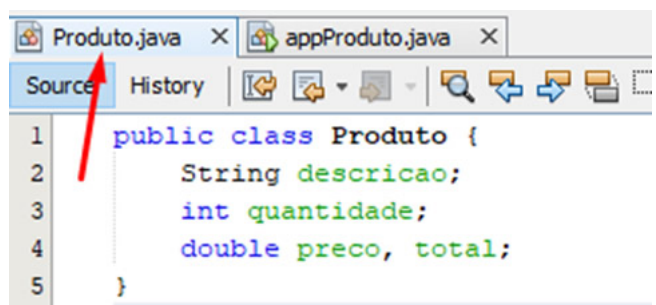
2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado "exercicio09". Caso necessário, desmarque a opção "Create Main Class";

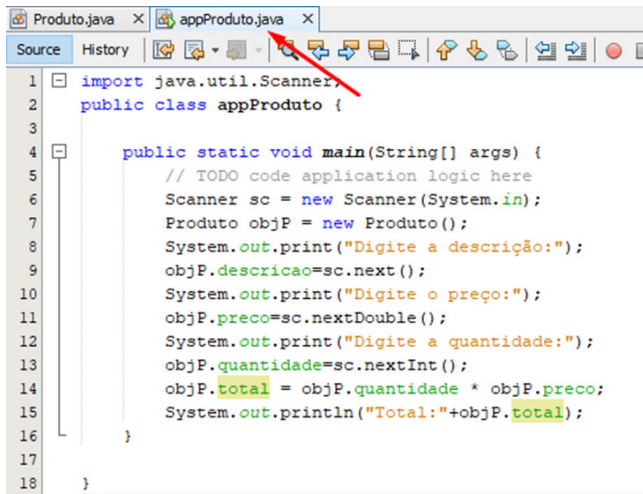
4. Crie uma classe chamada "Produto" clicando com o botão direito em "default package" e, em seguida, escolha New, Java Class;

5. Crie a aplicação com o nome "appProduto", e clique com o botão direito em "default package", após escolha a opção New, Java Main Class;

6. Clique na aba "Produto.java" e digite o código conforme o indicado.



7. Clique na aba "appProduto" e digite o código abaixo, conforme o indicado e, em seguida, execute e faça o teste.



```
1 import java.util.Scanner;
2 public class appProduto {
3
4     public static void main(String[] args) {
5         // TODO code application logic here
6         Scanner sc = new Scanner(System.in);
7         Produto objP = new Produto();
8         System.out.print("Digite a descrição:");
9         objP.descricao=sc.next();
10        System.out.print("Digite o preço:");
11        objP.preco=sc.nextDouble();
12        System.out.print("Digite a quantidade:");
13        objP.quantidade=sc.nextInt();
14        objP.total = objP.quantidade * objP.preco;
15        System.out.println("Total:"+objP.total);
16    }
17
18 }
```

9.2. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao09".

Criaremos uma classe chamada "Escola", com os seguintes atributos: aluno, turma, nota1, nota2, nota3, nota4 e média. No aplicativo "appEscola", iremos preencher os dados e calcular a média.



10. ORIENTAÇÃO A OBJETOS: MÉTODOS

10.1. Métodos:

São funções que realizam tarefas específicas e pode ser chamado por qualquer outro método ou classe, para realizar a referida função num determinado contexto.

Os métodos possuem algumas características como:

- Podem ou não retornar um valor;
- Podem ou não aceitar argumentos;
- Após encerrar sua execução, o método retorna o fluxo de controle do programa para quem o chamou.

Como boas práticas, é indicado sempre usar o nome dos métodos declarados como verbos, para que quando for efetuada alguma manutenção seja de fácil entendimento. Veja algumas nomenclaturas de nomes de métodos:

- correr, voltar, pagar, somar e calcularMedia;

Declaração de um método:

```
void [nomeDoMetodo] () {  
    [corpo do método]  
}
```

Onde o [nomeDoMetodo] é um identificador que define o nome pelo qual o método é conhecido.

O par de chaves delimita uma região para declaração de variáveis e métodos.

[corpo do método], consiste de uma lista ordenada para declaração de variáveis, de expressões e de comandos.

Void, define o valor retornado pelo método. Neste caso, nenhum.

Um exemplo de uso:

```
package testemetodo;  
  
public class TesteMetodo {  
  
    static void soma(int x, int y){  
  
        int total = x+y;  
  
        System.out.println("Total da  
soma:"+total);  
    }  
  
    public static void main(String[] args) {  
  
        // TODO code application logic here  
  
        soma(10,30);  
    }  
}
```

O nosso método, soma, realiza a adição de dois números inteiros, fornecidos pelos argumentos x e y, onde calculamos no total.

O método é chamado dentro do método padrão, o main().

Agora vamos ver as funções, que são rotinas que retornam um valor.

Veja o modelo:

Funcao Soma(x,y: Inteiro) : Inteiro

Var

Total: Inteiro

Inicio

Total = x + y

Retorne total

FimFuncao

Exemplo:

```
package testemetodo;
```

```
public class TesteMetodo {  
  
    static int soma(int y, int x){  
  
        int total = x+y;  
  
        return total;  
  
        //System.out.println("Total da soma:"+total);  
    }  
  
    public static void main(String[] args) {  
  
        // TODO code application logic here  
  
        int sm = soma(10,30);  
  
        System.out.println("O total de:"+sm);  
    }  
}
```

10.2. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “exercicio10”. Criaremos uma classe chamada Estoque, e os seguintes atributos: produto, quantidade, addProd (adicionar produto), delProd (remover produto) e totalProd (total produto). Criaremos um método para calcular o total de produtos. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Exercicio10”. Caso necessário, desmarque a opção “Create Main Class”.

4. Crie uma classe chamada "Estoque", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

5. Crie uma aplicação com o nome "appEstoque", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Main Class;

6. Clique na aba "Estoque.java" e digite o código conforme o indicado.

```
1 public class Estoque {  
2     public String produto;  
3     public int quantidade, addProd, delProd, totalProd;  
4  
5     public int totalProd(){  
6         totalProd = quantidade + addProd - delProd;  
7         return totalProd;  
8     }  
9 }
```

7. Clique na aba "appEstoque" e digite o código conforme o indicado e, em seguida, execute e faça o teste.

```
1 import java.util.Scanner;  
2 public class appEstoque {  
3     public static void main(String[] args) {  
4         // TODO code application logic here  
5         Scanner sc = new Scanner(System.in);  
6         Estoque objE = new Estoque();  
7         System.out.print("Informe o produto:");  
8         objE.produto = sc.next();  
9         System.out.print("Quantidade em estoque");  
10        objE.quantidade = sc.nextInt();  
11        System.out.print("Adicionar produto:");  
12        objE.addProd = sc.nextInt();  
13        System.out.print("Remover produto:");  
14        objE.delProd = sc.nextInt();  
15        System.out.println("Estoque atual:"+objE.totalProd());  
16        sc.close();  
17    }  
18 }
```

10.3. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado “Fixacao10”.

Criaremos uma classe chamada “Folha”, com os seguintes atributos: nome, salBruto (Salário bruto), Comissão, salLiquido (Salário líquido), descRef (desconto refeição), descTransp (desconto transporte) do tipo double.

Já o vRef(valor de desconto vale refeição), e o vTransp (valor de desconto vale transporte) vai ser do tipo Inteiro. No aplicativo, "appFolha" iremos preencher os dados e calcular o desconto do vale refeição, sobre o salário bruto, o desconto do vale transporte, sobre o salário bruto e deverá ser calculado o salário líquido.



JAVA Básico

Orientação a objetos: Métodos II

11

11. ORIENTAÇÃO A OBJETOS: MÉTODOS II

A

ntes de falarmos sobre método estático, vamos relembrar um pouco o conceito de classes.

Classes são sequência a serem seguidas, contém instruções de todas as ações que eles poderão executar, e os atributos que o objeto possuirá.

Desta forma, toda vez que instanciamos uma classe, geramos um novo objeto.

A partir deste momento podemos utilizar os atributos através de métodos.

Exemplo:

```
public class Funcionario {

    public String nome;

    public float salario;

}
```

Através desta classe, podemos criar várias instancias, a cada nova instancia, um novo objeto é gerado.

Exemplo:

```
Funcionario objSetor1 = new Funcionario();

objSetor1.nome = "Paulo";

Funcionario objSetor2 = new Funcionario();

objSetor2.nome = "Marcia";
```

Os métodos estáticos acessam apenas variáveis e métodos do mesmo tipo.

Para acessar um método "static" de outra classe basta utilizar o nome da classe.metodo.

Exemplo = NomeDaClasse.Metodo

O "não-static" precisa primeiramente criar uma instância da classe para só então chamar algum método também não-static.

Exemplo = NomeDaClasse ndc =new
NomeDaClasse()...

... ndc.Metodo.

11.1. Variáveis constantes

Uma constante é declarada quando precisamos lidar com dados que não devem ser alterados durante a execução do programa. Para isso, utilizamos a palavra reservada final para que a variável seja inicializada uma única vez.

Exemplos de declaração de constantes:

Final float VT = 35;

Final float VR = 20;

O VT é a abreviação para "Vale transporte" e VR, a abreviação para "Vale refeição".

Observe a instrução final a frente da declaração. Esta é a responsável por declarar que um determinado membro não poderá ter o seu valor alterado.

Em Java, existe uma peculiaridade quando trabalhamos com constantes. Nós podemos declarar uma constante sem inicializá-la. A partir do momento que um valor for atribuído, este não mais poderá ser alterado. Assim, temos a liberdade de definir o valor

da constante de maneira dinâmica, o que facilita a programação, porém, pode ocasionar problemas.

Os nomes das constantes sempre devem estar em letra maiúscula, assim, conseguimos distinguir facilmente entre uma variável e uma constante. O Java não nos obriga a definir o nome das constantes com letra maiúscula, porém, essa é uma convenção da comunidade e devemos segui-la.

11.2. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “Exercicio11”. Criaremos uma classe chamada Cobrança, e os seguintes atributos: nome, parcela1 e parcela2. Utilizaremos um método estático para somar as parcelas. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Exercicio11”. Caso necessário, desmarque a opção “Create Main Class”;

4. Crie a classe "Cobrança", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

5. Criando a aplicação com o nome "appCobrança". Clique com o botão direito em “default package” e, em seguida, escolha New, Java Main Class;

6. Clique na aba "Cobrança.java" e digite o código conforme o indicado.

```
1 public class Cobrança {
2     public String nome;
3
4     public static double pagto(double parcela1, double parcela2){
5         return parcela1 + parcela2;
6     }
7 }
```

7. Clique na aba "appCobrança" e digite o código conforme o indicado e, em seguida, execute e faça o teste;

```
1 public class appCobrança {
2
3     public static void main(String[] args) {
4         // TODO code application logic here
5         double total;
6         total = Cobrança.pagto(150, 120);
7         System.out.println("Total das parcelas:"+total);
8     }
9 }
```

11.3. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado “Fixacao11”.

Crie um programa para calcular o total à pagar, por um determinado produto, e crie os seguintes atributos: produto e preço, uma variável constante chamada "acrécimo", com valor de 10%. Duas fórmulas devem ser criadas, uma para calcular o valor do acréscimo, e outra para calcular o total à pagar. É necessário criar a classe Cobrança, e uma classe para rodar a aplicação com o nome appCobrança.



12. ENCAPSULAMENTO

Encapsulamento, é a técnica utilizada para esconder uma ideia, ou seja, não expor detalhes internos para o usuário, tornando partes do sistema mais independentes possível. Por exemplo, quando um celular estraga levamos para o conserto, aqui podemos ver o encapsulamento, pois quando o usuário faz uma ligação, por exemplo, não sabemos que programação acontece entre o botão de ligar e a parte interna para efetuar tal ação.

Como um exemplo mais técnico podemos descrever o que acontece em um sistema de vendas, aonde temos cadastros de funcionários, usuários, gerentes, clientes, produtos entre outros. Se por acaso acontecer um problema na parte do usuário é somente nesse setor que será realizada a manutenção não afetando os demais.

12.1. Private

O modificador de acesso `private` é o mais restritivo modificador de acesso. Todo membro de uma classe definido com o modificador `private` só é acessível para a própria classe. Não importa a localização dentro de pacotes ou se a classe foi herdada ou não, um membro `private` só é acessível dentro da mesma classe em que ele foi declarado.

Vamos então mostrar um exemplo que pega o nome de um cliente.

```
public class Clientes {  
  
    private String nome;  
  
}
```

Para acessar o atributo `nome`, já que é do tipo **private**, a prática é criar dois métodos, um que retorna o valor e outro que muda o valor.

A convenção para esses métodos é de colocar a palavra **get** ou **set** antes do nome do atributo.

GET

É utilizado para recuperar alguma informação, geralmente utilizado para trazer informação de algum atributo, sem ter que utilizar o atributo explicitamente.

Os **getters** e **setters** são sempre usados quando queremos encapsular uma classe, ou seja, os atributos (`private`) dessa classe só poderão ser acessados por outras classes através desses métodos. Isso serve para controlar o acesso aos atributos da classe e é uma boa prática.

Sintaxe:

```
public get( )  
  
{  
  
    return ;  
  
}
```

SET

É utilizado para setar um valor dentro de um objeto, de uma variável.

Sintaxe:

```
public void set(tipoVariavel )  
  
{  
  
    this. = ;  
  
}
```

Por exemplo, a nossa classe `Clientes`, possui o atributo `nome`, no caso da gente desejar dar acesso a leitura e escrita a todos os atributos:

```
public class Clientes {  
  
    private String nome;
```

```

private int idade;

public String getNome(){

    return this.nome;

}

public void setNome(String nome){

    this.nome = nome;

}

}

public static void main(String[] args) {

    // TODO code application logic here

    Clientes objC = new Clientes();

    objC.setNome("João Paulo");

    System.out.println("Nome:"+objC.getNome());

}

```

Utilizamos o nosso objeto e através do método “set” adicionamos o nome João Paulo. O método “get”, foi usado para exibir o valor definido em set.

Observação:

THIS

A função da palavra chave this é informar que a variável em questão é local. Um bom exemplo seria ter duas variáveis com nomes iguais, como pode ser visto acima.

CONCLUSÃO

O utilizador de uma classe não tem necessariamente de saber como estão estruturados os dados no objeto, isto significa que um utilizador não tem de conhecer a aplicação.

Assim, proibindo o usuário de alterar diretamente os atributos, e obrigando-o a utilizar as funções definidas para alterá-los (chamadas interfaces), podemos garantir a integridade dos dados (por exemplo garantir que o tipo dos dados fornecido está conforme ao desejado, ou ainda que os dados se encontram no intervalo esperado).

O encapsulamento permite definir os níveis de visibilidade dos elementos da classe. Estes níveis de visibilidade definem os direitos de acesso aos dados conforme acessamos um método da própria classe, de uma classe herdada, ou de uma classe qualquer. Existem três níveis de visibilidade: **pública** onde as funções de todas as classes podem acessar os dados ou os métodos de uma classe definida com o nível público de visibilidade, tratando-se do nível de proteção de dados mais baixo de proteção; **protegida** onde o acesso aos dados está reservado às funções das classes herdadas, ou seja, às funções membros da classe, bem como às classes derivadas; **privada** com o acesso aos dados é limitado aos métodos da própria classe, trata-se do nível de proteção dos dados mais elevado.

De fato, se pensarmos em termos de informática, é possível para um usuário comum usar uma impressora sem nem mesmo entender seu funcionamento interno. Imagine o desastre que seria se todos os usuários resolvessem abrir suas impressoras para investigar o que há dentro delas.

Da mesma forma, ao construir uma classe, devemos fazê-lo de forma que o usuário desta classe tenha acesso apenas aos métodos que permitem ler informações da classe ou fornecer os dados necessários para sua correta operação. O funcionamento interno da classe deve permanecer oculto e acessível somente aos métodos da própria classe.

O encapsulamento deve ser aplicado de forma a permitir que alterações na estrutura interna de uma classe não prejudique o funcionamento do código externo que a usa.

12.2. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “Exercicio12”. Criaremos um programa com os dados de um colaborador, segue os seguintes atributos: nome, cargo e salário. Abra o disco local;

2. Abra a pasta “netbeans” e acesse a pasta “bin” para executar o programa “netbeans64”;

3. Crie um novo projeto chamado “Exercicio12”. Caso necessário desmarque a opção “Create Main

Class". Crie uma classe chamada "Funcionario", clicando com o botão direito em "default package" e, em seguida, escolha New, Java Class;

4. Crie uma aplicação chamada "appFuncionario", clicando com o botão direito em "default package" e, em seguida, escolha New, Java Main Class;

5. Clique na aba "Funcionario.java" e digite o código conforme o indicado.

```
1 public class Funcionario {
2     private String nome, cargo;
3     private double salario;
4
5     public String getNome(){
6         return nome;
7     }
8     public void setNome(String nome){
9         this.nome = nome;
10    }
11    public String getCargo(){
12        return cargo;
13    }
14    public void setCargo(String cargo){
15        this.cargo = cargo;
16    }
17    public double getSalario(){
18        return salario;
19    }
20    public void setSalario(double salario){
21        this.salario = salario;
22    }
23 }
```

6. Clique na aba "appFuncionario.java" e digite o código conforme o indicado.

```
1 public class appFuncionario {
2     public static void main(String[] args) {
3         // TODO code application logic here
4         Funcionario objF = new Funcionario();
5         objF.setNome("Marcos Silva");
6         objF.setCargo("Eletrecista");
7         objF.setSalario(1800);
8         System.out.println("Colaborador: "+objF.getNome());
9         System.out.println("Cargo: "+objF.getCargo());
10        System.out.println("Salário: "+objF.getSalario());
11    }
12 }
```

12.3. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao12".

Criaremos os seguintes atributos: "nome" e "curso", que deve ser do tipo String, o "numeroTurma", que deve ser do tipo Int, "parcela" e "media", que deve ser do tipo double. Definir o valor de cada atributo, com o método "Set", e exibir o valor com o método "Get". É necessário criar uma classe chamada Aluno e uma do tipo aplicativo o nome appAluno.



13. OOP: VETOR, LAÇO E LISTA

A matriz no Java, fornece uma estrutura de dados, de tamanho fixo de elementos do mesmo tipo.

Internamente um vetor armazena diversos valores, cada um associado a um número que se refere à posição de armazenamento, conhecimento como índice.

Os vetores são estruturas indexadas, cada valor pode ser armazenado em uma determinada posição (índice) é chamado de elemento do vetor.

O número de posições de um vetor corresponde ao tamanho que ele tem.

Se um vetor possuir tamanho 8, tem esse número de elementos, isto é, pode armazenar até oito elementos distintos.

Cada posição de um vetor é unicamente identificada por um valor inteiro.

As posições de um vetor iniciam a numeração a partir do valor 0, portanto, um vetor de tamanho 8 teria índices iniciados em 0 prosseguindo até o 7.

13.1. Vetor

Declaração de variáveis do tipo vetor.

Quando declaramos um vetor, devemos fornecer três informações:

- 1) Nome do vetor
- 2) Número de posições do vetor
- 3) Tipo de dados

Declaração do vetor do tipo inteiro

Int vetor[];

Para definir o tamanho do vetor.

Vetor = new int[6];

Definimos o tamanho, e utilizamos o operador new, uma palavra reservada para definir o espaço.

As duas declarações podem ser combinadas da seguinte forma:

Int vetor[] = new int[6];

Declarando um vetor para armazenar 6 números inteiros:

Int num = new int[6];

Declarando um vetor do tipo real.

Double preco = new double[6];

Declarando um vetor para armazenar 5 produtos.

String produtos = new String[5];

Inicialização de vetores ao declarar.

String nome[] = {"Ana", "Carina", "Marcos"};

Isso significa que **nome[0]** terá o valor Ana, **nome[1]** terá o valor Carina e o **nome[2]** terá o valor Marcos.

Estruturas de repetição

As estruturas de repetição também são conhecidas como laços (loops) e são utilizados para executar, repetidamente, uma instrução ou bloco de instrução enquanto determinada condição estiver sendo satisfeita.

Laço FOR.

for(cond inicial; teste condicional ; apos iteração){

//código

```
}
```

Condição inicial. Geralmente se inicia o contador. Esse primeiro estágio SEMPRE acontece.

O segundo estágio é o teste da condição do laço, um simples teste condicional. Caso seja verdade, o código é executado.

Ao término de execução do código, ocorre o fator de mudança, que geralmente é um incremento ou decremento, sempre após cada iteração do looping,

Depois a condição é testada novamente. Caso retorne 'true', o código é executado.

Ao término de execução do código, sempre ocorre o fator de mudança...e assim sucessivamente.

O nosso exemplo é simples, vai listar números de 0 até 6, apenas para mostrar como funciona o for.

Veja:

```
for(int i = 0; i<=6; i++){  
    System.out.println("Lista: "+i);  
}
```

13.2. Laço For each.

A utilização é bem simples, o for percorre sempre, do começo ao fim, todos os elementos de um Array.

A sintaxe do for each é a seguinte:

```
for (tipo variavel_do_tipo_do_seuArray: seuArray)  
{  
    //seu código  
}
```

Se o 'seuArray' for de inteiro, ficaria:

```
for (int count : seuArray){  
    ...  
}
```

Como podemos interpretar esse laço foreach?

Muito simples, o inteiro 'count' vai receber, a cada iteração, todos os valores de 'seuArray'.

Ou seja, ele vai percorrer todo o seu Array e receber seus valores, na ordem (do começo para o fim), um por vez. E com esses valores você vai fazer o que desejar.

```
String alunos[] = new String[6];  
  
alunos[0] = "Ana";  
alunos[1] = "Carlos";  
alunos[2] = "Camila";  
alunos[3] = "Pedro";  
alunos[4] = "Jonas";  
alunos[5] = "Bruna";  
  
for(String lista: alunos){  
    System.out.println("Aluno(a):" + lista);  
}
```

O nosso exemplo, cria um vetor de 6 posições, com nomes de alunos, em seguida, através de um for, listamos.

13.3. ArrayList

É uma classe para coleções. uma coleção é uma estrutura de dados, na realidade um objeto, que pode armazenar ou agrupar referências a outros objetos (um contêiner). As classes e interfaces da estrutura de coleções são membros do pacote java.util.

A sintaxe básica de um ArrayList é:

```
ArrayList nomedoArrayList = new ArrayList();
```

Para adicionar dados ao ArrayList, veja o nosso exemplo:

```
ArrayList<String> dados = new  
ArrayList<String>();  
  
dados.add("João");  
dados.add("Lucas");  
dados.add("Melissa");
```



```

dados.add("Guilherme");

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

    System.out.println("Aluno(a):" +
dados.get(i));

}

```

Como dissemos, um ArrayList declarado herda da classe ArrayList, a qual possui vários métodos. Um deles é o método Add, que permite inserir dados no ArrayList. Note que no ArrayList da figura 5 estamos inserindo dados de vários tipos.

13.4. Exercícios Passo a Passo

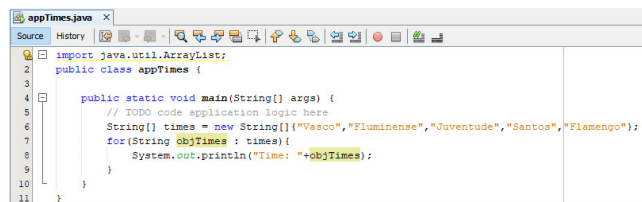
1. Este exercício tem como objetivo criar um projeto chamado “Exercicio13”. Criaremos um programa que leia cinco times de futebol e liste em um vetor. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Exercicio13”, clicando no botão New Project. Caso necessário

desmarque a opção “Create Main Class”;

4. Crie uma aplicação chamada "appTimes", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Main Class e, em seguida, digite o código conforme o indicado.



```

appTimes.java
Source History
1 import java.util.ArrayList;
2 public class appTimes {
3
4     public static void main(String[] args) {
5         // TODO code application logic here
6         String[] times = new String[]{"Vasco", "Fluminense", "Juventude", "Santos", "Flamengo"};
7         for(String objTimes : times){
8             System.out.println("Time: "+objTimes);
9         }
10    }
11 }

```

13.5. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado “Fixacao13”.

Criaremos um ArrayList, listaremos o nome de algumas bandas de pop rock. O nome do ArrayList vai ser “Lista”, dentro do laço for, o nome vai ser “exibir”. Nome das bandas: U2, Pearl Jam, Coldplay, Bon Jovi, Nickelback. É necessário criar uma aplicação com o nome appBandas.



14. HERANÇA

Podemos definir herança como um princípio da programação orientada a objetos, que permite criar uma nova classe a partir de uma já existente.

Também chamada de subclasses o nome herança, provém do fato de que a subclasse contém atributos e métodos da classe da qual deriva.

A vantagem é não ter que partir do zero para especializar uma classe existente. Uma subclasse herda métodos e atributos de sua superclasse.

Para efetuar uma herança de uma classe é utilizada a palavra reservada chamada **extends**.

Para representar, utilizaremos um exemplo de modelagem de uma academia, representando alunos, professores e funcionários. Certamente tem características comuns entre eles.

Para não ter que digitar o mesmo código duas ou mais vezes, podemos criar uma só superclasse chamada **Pessoa**.

```
package heranca01;

public class Pessoa {

    public String nome;

    public String cpf;

    public String getNome(){

        return nome;

    }

    public void setNome(String nome){

        this.nome = nome;

    }

    public String getCpf(){
```

```
        return cpf;

    }

    public void setCpf(String cpf){

        this.cpf = cpf;

    }

}
```

O código acima mostra que a classe "Pessoa" possui Nome e cpf, como atributos.

Criando a classe Aluno.

```
package heranca01;

public class Aluno extends Pessoa {

    public int matricula;

    public String curso;

    public void cancelarMatr(){

        System.out.println("Matricula

cancelada");

    }

    public int getMatricula(){

        return matricula;

    }

    public void setMatricula(int matricula){

        this.matricula = matricula;

    }

    public String getCurso(){

        return curso;
```

```

    }

    public void setCurso(String curso){

        this.curso = curso;

    }

}

```

A classe Aluno possui suas próprias características, mas também possui propriedades comuns, nome e cpf, porque foi estendido da classe Pessoa.

Criando a classe Professor.

```

package heranca01;

public class Professor extends Pessoa {

    public String especialidade;

    public String getEspecialidade(){

        return especialidade;

    }

    public void setEspecialidade(String
especialidade){

        this.especialidade = especialidade;

    }

}

```

Na classe Professor criamos o atributo especialidade, e também a classe recebeu atributos da classe Pessoa, para reaproveitamento de código.

Classe Principal.

```

package heranca01;

public class Heranca01 {

    public static void main(String[] args) {

        // TODO code application logic here

        Pessoa objPes = new Pessoa();

        Aluno objAlu = new Aluno();

        Professor objPro = new Professor();
    }
}

```

```

        objAlu.setNome("Fabiana");

        System.out.println("Nome:"+objAlu.getNome());

        objAlu.setCurso("Informática");

        System.out.println("Curso
de:"+objAlu.getCurso());

        objPro.setNome("Rodrigo");

        objPro.setEspecialidade("Inglês");

        System.out.println("Professor:"+objPro.getNome());

        System.out.println("Especialidade:"+objPro.getEspe

        }

    }
}

```

Na classe principal chamamos todos os atributos, conseguimos assim trabalhar com todas as classes e alimentar com informações, facilitando o nosso trabalho.

14.1. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “Exercicio14”. Criaremos um programa onde iremos definir uma classe chamada “Professor”, com atributos os atributos nome e idade. Duas classes serão criadas, “ProfIntegral e ProfHorista”, essas duas classes vão herdar os atributos da classe “Professor”, porém o professor integral, recebe um salário fixo e o horista recebe um salário por horas trabalhadas. Acesse o disco local;

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado “Exercicio14”, clicando no botão New Project. Caso necessário, desmarque a opção “Create Main Class”;

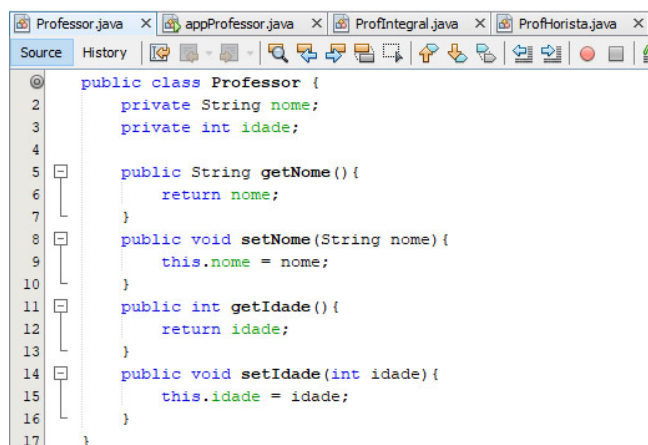
4. Crie uma classe chamada "Professor", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

5. Crie uma classe chamada "ProfIntegral", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

6. Crie uma classe chamada "ProfHorista", clicando com o botão direito em "default package" e, em seguida, escolha New, Java Class;

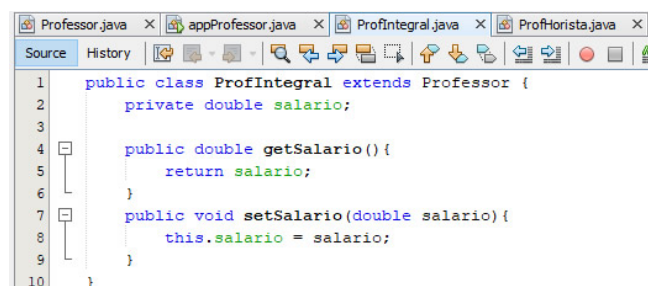
7. Crie uma aplicação chamada "appProfessor", clicando com o botão direito em "default package" e, em seguida, escolha New, Java Main Class;

8. Clique na aba "Professor" e digite o código conforme o indicado.



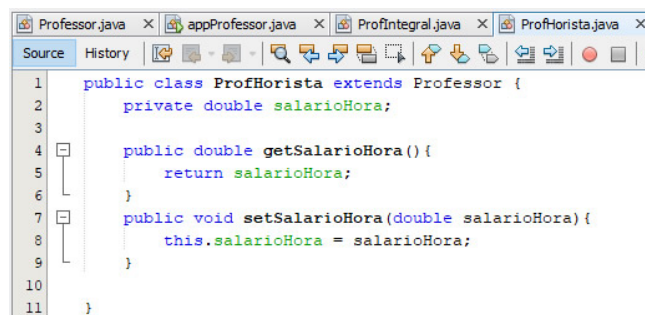
```
1 public class Professor {
2     private String nome;
3     private int idade;
4
5     public String getName(){
6         return nome;
7     }
8     public void setName(String nome){
9         this.nome = nome;
10    }
11    public int getIdade(){
12        return idade;
13    }
14    public void setIdade(int idade){
15        this.idade = idade;
16    }
17 }
```

9. Clique na aba "ProfIntegral" e digite o código conforme o indicado.



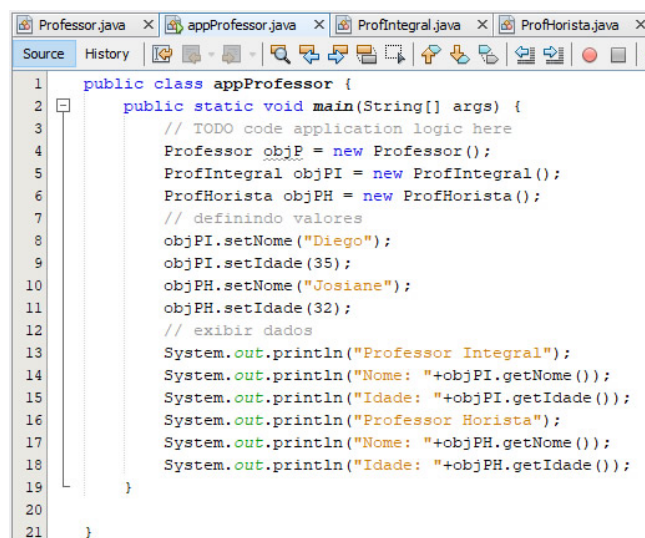
```
1 public class ProfIntegral extends Professor {
2     private double salario;
3
4     public double getSalario(){
5         return salario;
6     }
7     public void setSalario(double salario){
8         this.salario = salario;
9     }
10 }
```

10. Clique na aba "ProfHorista" e digite o código conforme o indicado.



```
1 public class ProfHorista extends Professor {
2     private double salarioHora;
3
4     public double getSalarioHora(){
5         return salarioHora;
6     }
7     public void setSalarioHora(double salarioHora){
8         this.salarioHora = salarioHora;
9     }
10
11 }
```

11. Clique na aba "appProfessor" e digite o código conforme o indicado.



```
1 public class appProfessor {
2     public static void main(String[] args) {
3         // TODO code application logic here
4         Professor objP = new Professor();
5         ProfIntegral objPI = new ProfIntegral();
6         ProfHorista objPH = new ProfHorista();
7         // definindo valores
8         objPI.setName("Diego");
9         objPI.setIdade(35);
10        objPH.setName("Josiane");
11        objPH.setIdade(32);
12        // exibir dados
13        System.out.println("Professor Integral");
14        System.out.println("Nome: "+objPI.getName());
15        System.out.println("Idade: "+objPI.getIdade());
16        System.out.println("Professor Horista");
17        System.out.println("Nome: "+objPH.getName());
18        System.out.println("Idade: "+objPH.getIdade());
19    }
20
21 }
```

14.2. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao14".

Crie uma classe modelo chamada "Funcionário", com os atributos: nome e salário. Já a classe "Gerente", deve ter o atributo "senha" após ela deve herdar atributos da classe "Funcionário", a classe "appEmpresa" vai listar os dados do gerente.



JAVA Básico

Sobreposição e Interface Gráfica I

Aula 15

15. SOBREPOSIÇÃO E INTERFACE GRÁFICA I

15.1. Override

O **override** é uma sobreposição de métodos. Ou seja, o método da classe filha se sobrepõe ao da classe pai, como é uma sobreposição o método da classe pai não deixa de existir apenas está “embaixo”. Se você quiser chamar esse método da classe filho basta usar o `super()`.

Exemplo:

```
public class Pai {  
  
    public int Soma(int valor){  
  
        return valor + 100;  
  
    }  
  
}  
  
public class Filho extends Pai {  
  
    @Override  
  
    public int Soma(int valor){  
  
        return valor + 200;  
  
    }  
  
}
```

Perceba que na class Pai o valor é um e na class Filha é outra, ou seja, como foi reescrito, você pode ter comportamentos diferentes nas classes.

15.2. Interface gráfica

O Java fornece suporte para bibliotecas gráficas, onde estaremos utilizando a Swing.

As bibliotecas gráficas são bem simples no que diz respeito ao uso.

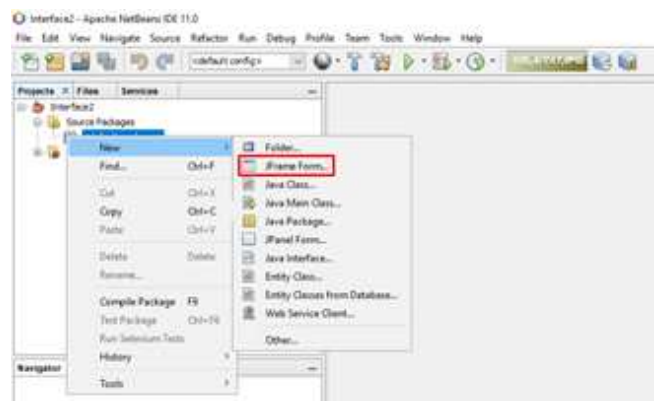
Essa biblioteca Swing é oficial, inclusa em qualquer JRE e JDK.

A biblioteca Swing fornece muitos componentes como: botões, entradas de texto, tabelas, janelas, abas, e muitos outros.

O Swing faz parte do pacote **javax.swing**.

Para fazer uso da interface gráfica precisamos ativar o recurso, criado o projeto, usamos o botão direito do mouse na pasta padrão, selecionando a opção New, JFrame Form.

Depois basta definir o nome do formulário.



Área de Design:

A janela principal que permite criar e editar formulários.

Na parte superior da tela vai aparecer os botões **Source** e **Design**.

Source:

Exibe todo o código, que automaticamente o Java se encarrega de criar, a partir dos componentes que são inseridos na tela, tem componentes padrão, e outros que vamos inserindo conforme necessidade.

```

1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7   /**
8   *
9   */
10
11  public class Cadastro extends javax.swing.JFrame {
12
13      /**
14       * Creates new form Cadastro
15       */
16      public Cadastro() {
17          initComponents();
18      }
19  }

```

Design:

Tudo que é feito no código, aparece de forma visual neste modo de exibição, facilitando o nosso entendimento, é nesse modo que vamos ver os botões, caixas de texto, entre outros componentes. Além de visualizar, podemos inserir, mover e apagar os componentes.

Paleta

Exibe uma lista de componentes disponíveis contendo guias.

Janela Propriedades

Exibe as propriedades do componente atualmente selecionado no navegador, janela e projetos.

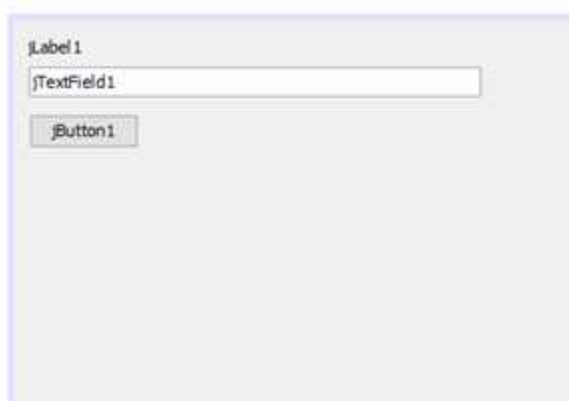
Veja alguns componentes:

Exibe texto não editável.

Insere dados do teclado e serve também para exibição do texto editável ou não editável.

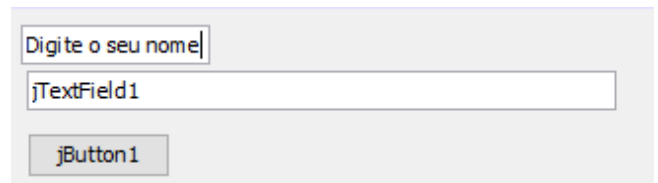
Libera um evento quando o usuário clicar nele com o mouse.

Veja os três componentes no design.



Renomear componentes:

Para realizar o processo de alteração do nome, basta apenas clicar com o botão direito sobre cada um deles, e clicar na opção **Edit Text**.



Guia Properties

Permite alterar as propriedades do componente inserido na tela. Propriedade como, cor, fonte, texto entre outros.

Guia Events

Aqui existe alguns eventos de acordo com a necessidade, como exemplo, temos um evento que reage quando o usuário clicar no componente, entre outros.

Cada componente no Java vira um objeto, possui um nome padrão, para alterar basta clicar com o botão direito do mouse e selecionar a opção Change Variable Name (Alterar o nome da variável).

Para nomear os objetos normalmente usamos uma convenção, mas fica a critério do programador.

Para os objetos do tipo Label nomeamos como **lb_** seguido de um nome.

lb_digiteUmNome

Exemplo:

Nome, Cidade e Telefone

lb_nome, lb_cidade e lb_telefone

Para os objetos do tipo TextField nomeamos como **tf_** seguido de um nome.

Exemplo:

Nome, Cidade e Telefone.

Tf_nome, tf_cidade e tf_telefone.

Para os componentes do tipo Button nomeamos como **bt_** seguido de um nome.

Exemplo:

Fechar e Gravar

Bt_fechar e bt_gravar.

Events (Eventos):

ActionPerformed – este evento dispara uma ação ao clicar.

O **getText()** tem como finalidade buscar o valor digitado no objeto TextField.

15.3. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado “exercicio15”. Criaremos uma classe chamada “Bateria” e dois métodos: afinação e montagem. Para aproveitamento usaremos a sobrescrita nas classes Pearl e RMV. Na classe principal “appBateria” vamos exibir os métodos das duas classes. Acesse o disco local:

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um projeto chamado "Exercicio15";

4. Crie uma classe chamada "Bateria", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

5. Crie uma classe chamada "Pearl", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

6. Crie uma classe chamada "Rmv", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Class;

7. Crie uma aplicação chamada "appBateria", clicando com o botão direito em “default package” e, em seguida, escolha New, Java Main Class;

8. Clique na aba “Bateria” e digite o código conforme o indicado.

```
public class Bateria {  
    public void montagem() {  
        System.out.println("Montagem padrão");  
    }  
    public void afinacao() {  
        System.out.println("Afinação padrão");  
    }  
}
```

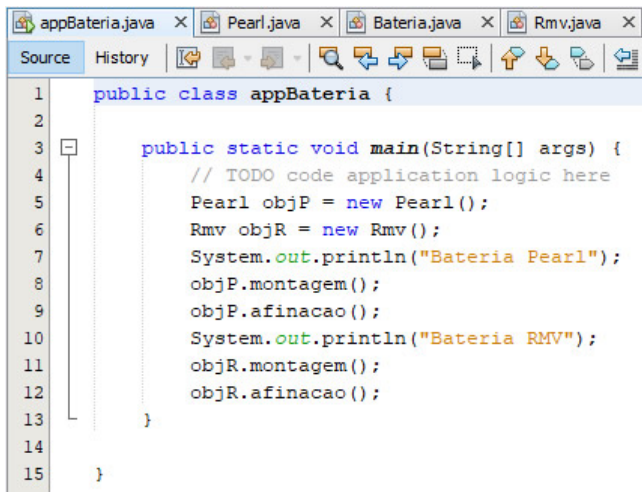
9. Clique na aba “Pearl” e digite o código conforme o indicado.

```
public class Pearl extends Bateria {  
    @Override  
    public void montagem() {  
        System.out.println("Estúdio");  
    }  
    @Override  
    public void afinacao() {  
        System.out.println("Estúdio");  
    }  
}
```

10. Clique na aba “Rmv” e digite o código conforme o indicado.

```
public class Rmv extends Bateria {  
    @Override  
    public void montagem() {  
        System.out.println("Ar livre");  
    }  
    @Override  
    public void afinacao() {  
        System.out.println("Ar livre");  
    }  
}
```

11. Clique na aba “appBateria” e digite o código conforme o indicado. Após execute o código para testar.



```

1 public class appBateria {
2
3     public static void main(String[] args) {
4         // TODO code application logic here
5         Pearl objP = new Pearl();
6         Rmv objR = new Rmv();
7         System.out.println("Bateria Pearl");
8         objP.montagem();
9         objP.afinacao();
10        System.out.println("Bateria RMV");
11        objR.montagem();
12        objR.afinacao();
13    }
14
15 }

```

formulário de cadastro de clientes, com os seguintes componentes: Código, Nome, Endereço, Cidade, E-mail e Telefone. conforme o indicado.



15.4. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao15". Criaremos um



16. INTERFACE GRÁFICA II



showMessageDialog() exibe uma caixa de diálogo com uma mensagem de texto simples.

`JOptionPane.showMessageDialog` (quadro, "Um diálogo básico da mensagem do `JOptionPane`");

Nesse exemplo, meu primeiro argumento para o `showMessageDialog` método `JOptionPane` é um frame objeto que, presumivelmente, é uma instância de a `JFrame`.

Se, por algum motivo, você não tiver uma referência `JFrame` ou `JWindow` instância, poderá criar esse campo nulo ainda exibir a `JOptionPane` caixa de diálogo idêntica.

Tipo de mensagem

Define o estilo da mensagem. O gerenciador de aparência pode exibir a caixa de diálogo de maneira diferente, dependendo desse valor, e geralmente fornece um ícone padrão. Os valores possíveis são:

- `ERROR_MESSAGE`
- `INFORMATION_MESSAGE`
- `WARNING_MESSAGE`
- `QUESTION_MESSAGE`
- `PLAIN_MESSAGE`

`setLocationRelativeTo()`

Esta função define o local da janela, em relação ao componente que foi especificado. Caso o componente não estiver sendo exibido no momento ou for nulo, a janela será centralizada.

Sintaxe:

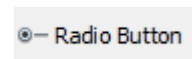
`Objeto.setLocationRelativeTo(null);`

//para centralizar no meio da tela.

Mais componentes:



Este componente representa uma caixa de seleção e permite selecionar uma ou mais opções. Quando um checkbox for selecionado ele retorna um valor verdadeiro, caso contrário, retorna um valor falso.



Este componente permite selecionar uma entre diversas opções.

O componente possui o método `isSelected()`, que verificar se o componente está selecionado.

Exemplo de formulário contendo os componentes `Checkbox` e `RadioButton`.



No exemplo acima, o botão de Rádio vai permitir que apenas umas das opções seja selecionada. Podemos ainda agrupar com um `ButtonGroup`, para que haja melhor integridade dos dados. Já no exemplo das contas bancárias, o componente `CheckBox` permite marcar ou desmarcar várias opções de uma vez.

16.1. Exercícios Passo a Passo

1. Este exercício tem como objetivo criar um projeto chamado "Exercicio16". Criaremos um formulário de acesso, com login e senha. Será exibido em uma caixa de diálogo os dados. Acesse o disco local.

2. Abra a pasta "netbeans" e acesse a pasta "bin" para executar o programa "netbeans64";

3. Crie um novo projeto chamado "Exercicio16", caso necessário desmarque a opção "Create Main Class".

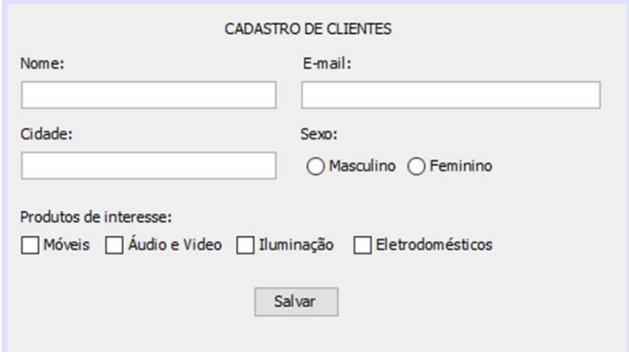
4. Crie um formulário chamada "AcessoRestrito", clicando com o botão direito em "default package" e, em seguida, escolha New, JFrame Form.

5. Arraste da categoria "Swing Controls" os componentes "Label e Text Field". O componente "Button" será usado para realizar um evento, ao clicar, vai exibir uma caixa de diálogo com os dados preenchidos;

6. Veja o código do botão Salvar.
JOptionPane.showConfirmDialog(rootPane, "Seu Login: "+tf_login.getText()+" Senha: "+tf_senha.getText());

16.2. Exercícios de Fixação

1. Este exercício tem como objetivo criar um projeto chamado "Fixacao16". Criaremos um formulário de cadastro de clientes, conforme o indicado.



O formulário, intitulado "CADASTRO DE CLIENTES", contém os seguintes campos e controles:

- Nome:** Campo de texto.
- E-mail:** Campo de texto.
- Cidade:** Campo de texto.
- Sexo:** Botões de opção para "Masculino" e "Feminino".
- Produtos de interesse:** Quatro caixas de seleção para "Móveis", "Áudio e Vídeo", "Iluminação" e "Eletrodomésticos".
- Salvar:** Botão para salvar os dados.