

**Pratique,
aprenda,
conquiste.**

 **Proz**
Viva sua profissão!

Disciplina | Desenvolvimento de Aplicações

Tec. Desenvolvimento de Sistemas



**Aqui
começa
a sua
jornada**

Vamos nessa?



Disciplina | Desenvolvimento de Aplicações

Tec. Desenvolvimento de Sistemas

SUMÁRIO

DESENVOLVIMENTO DE APLICAÇÕES.....	4
INTRODUÇÃO.....	4
TEMA 01.....	5
INTRODUÇÃO E HISTÓRIA DO JAVA.....	5
TEMA 02.....	26
INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO.....	26
TEMA 03.....	41
CLASSES JAVA: STRINGS.....	41
TEMA 04.....	48
ESTRUTURAS DE CONTROLE - CONDICIONAIS.....	48
TEMA 05.....	55
ESTRUTURAS DE CONTROLE – REPETIÇÃO.....	55
TEMA 06.....	62
VETORES E MATRIZES EM JAVA.....	62
TEMA 07.....	68
POO – CONCEITOS INICIAIS.....	68
TEMA 08.....	79
CONSTRutoRES.....	79
TEMA 09.....	85
ENCAPSULAMENTO E OS MÉTODOS MODIFICADORES E ACESSORES.....	85
TEMA 10.....	93
MODIFICADORES DE ACESSO EM JAVA.....	93

DESENVOLVIMENTO DE APLICAÇÕES



INTRODUÇÃO

A programação orientada a objetos (Programação Orientada a Objeto-POO) é um paradigma fundamental na engenharia de software que revolucionou a forma como desenvolvemos e organizamos programas de computador. Na POO, os programas são construídos em torno de objetos, que são entidades que combinam dados (atributos) e comportamento (métodos). Essa abordagem oferece uma maneira mais intuitiva e estruturada de modelar o mundo real em código, permitindo aos desenvolvedores criar sistemas mais modularizados, flexíveis e de fácil manutenção.

Uma das principais vantagens da POO é a reutilização de código, onde objetos podem ser criados a partir de classes existentes, economizando tempo e esforço de desenvolvimento. Além disso, a encapsulação, herança e polimorfismo são conceitos-chave que permitem uma maior abstração e flexibilidade no design de software. Com a POO, os desenvolvedores podem criar aplicações mais robustas, escaláveis e compreensíveis, tornando-a uma abordagem crucial na criação de sistemas complexos e de alta qualidade.

Java é uma linguagem de programação de alto nível desenvolvida pela Sun Microsystems. Ela foi originalmente projetada para desenvolver programas para decodificadores e dispositivos portáteis, mas mais tarde se tornou uma escolha popular para a criação de aplicativos da web.

A sintaxe Java é semelhante a C++, porém é estritamente uma linguagem de programação orientada a objetos, por exemplo, a maioria dos programas Java contém classes, que são usadas para definir objetos e métodos, que são atribuídos a classes individuais. Ao contrário dos executáveis do Windows (arquivos .EXE), os programas Java não são executados diretamente pelo sistema operacional.

Os programas Java são interpretados pela Java Virtual Machine, ou JVM, que é executada em várias plataformas. Vamos conhecer cada uma das suas definições, conceitos e representações.

TEMA 01

INTRODUÇÃO E HISTÓRIA DO JAVA



Habilidades

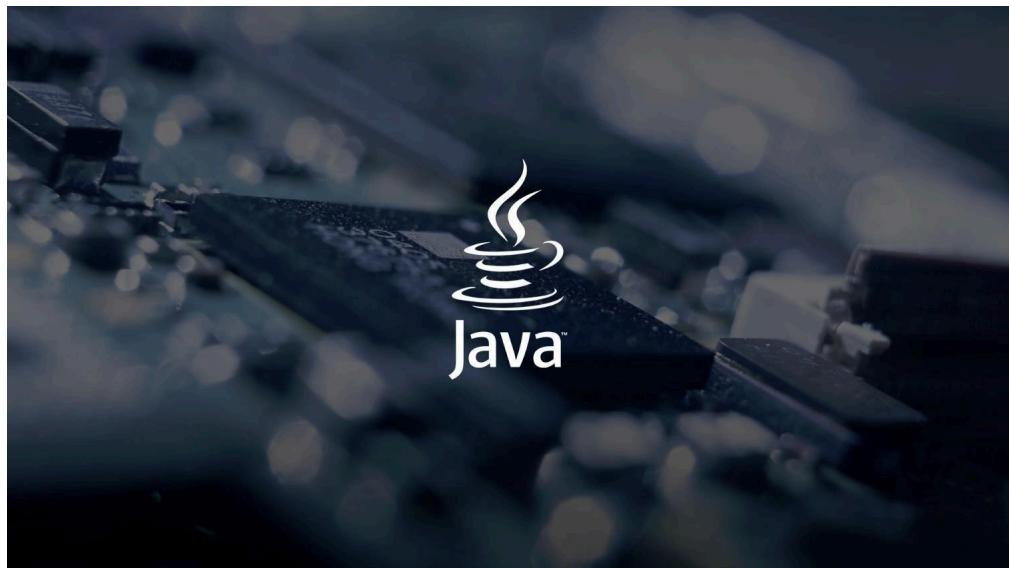
- Projetar sistemas de informação, selecionando linguagens de programação e ambientes de desenvolvimento de acordo com as especificidades do projeto.
- Utilizar ambientes de desenvolvimento para desenvolvimento desktop.
- Aplicar técnicas de orientação a objetos.

Introdução

Java surgiu em um projeto chamado "Oak" por James Gosling em junho de 1991. O objetivo de Gosling era implementar uma máquina virtual e uma linguagem que tivesse uma notação semelhante ao C, mas com maior uniformidade e simplicidade do que C/C++. A primeira implementação pública foi Java 1.0 em 1995. Ele fez uma promessa: "Escreva uma vez, execute em qualquer lugar", com tempos de execução gratuitos em plataformas populares. Era bastante seguro e configurável, permitindo que o acesso à rede e aos arquivos fosse limitado. Os principais navegadores da web logo o incorporaram em suas configurações padrão em uma configuração de "mini aplicativo" segura.



Acesse o QR Code do vídeo Introdução a JAVA - Fonte do vídeo: <<https://youtu.be/9d0Br36UDoc>>



Disponível em: <<https://wallpapersafari.com/w/lvH8Qg>>. Acesso em 11 set. 2023.

Popularização

Novas versões surgiram para grandes e pequenas plataformas (J2EE e J2ME) logo foram projetadas com o advento do "Java 2". A Sun não anunciou nenhum plano

para um "Java 3". Em 1997, a Sun abordou os padrões ISO/ IEC JTC1 e mais tarde a Ecma International para formalizar o Java, mas logo se retirou do processo. Java continua sendo um padrão proprietário de fato controlado por meio do Java Community Process. A Sun disponibilizava a maioria de suas implementações Java gratuitamente.



Acesse o QR Code do vídeo Java, a origem | Hampliature - Fonte do vídeo:
<https://youtu.be/9d0Br36UDochttps://youtu.be/JFhAI8c_zQU>

A Sun distinguiu entre seu Software Development Kit (SDK) e Runtime Environment (JRE), que é um subconjunto do SDK, sendo a principal diferença que no JRE o compilador não está presente. Em 2010, a Oracle comprou a Sun Microsystems – criadora do Java – por US\$ 7,4 bilhões e se tornou proprietária dessa linguagem de programação.



Disponível em: <<https://1000logos.net/java-logo/>>. Acesso em 11 set. 2023.

Filosofia

Havia cinco objetivos principais na criação da linguagem Java:

1. Ela deve usar a metodologia de programação orientada a objetos .
2. Deve permitir que o mesmo programa seja executado em vários sistemas operacionais.
3. Deve conter suporte embutido para o uso de redes de computadores.
4. Deve ser projetado para executar código de fontes remotas com segurança.
5. Deve ser fácil de usar, selecionando o que foi considerado as partes boas de outras linguagens orientadas a objetos.

Orientação do objeto

A primeira característica, orientação a objetos ("OO"), refere-se a um método de programação e design de linguagem. Embora existam muitas interpretações de OO, uma ideia principal distintiva é projetar software de forma que os vários tipos de dados que ele manipula sejam combinados com suas operações relevantes.

Assim, os dados e o código são combinados em entidades chamadas de objetos. Um objeto pode ser considerado um pacote autocontido de comportamento (código) e estado (dados). O princípio é separar as coisas que mudam das coisas que permanecem as mesmas; frequentemente, uma mudança em alguma estrutura de dados requer uma mudança correspondente no código que opera nesses dados, ou vice-versa. Essa separação em objetos coerentes fornece uma base mais estável para o design de um sistema de software.

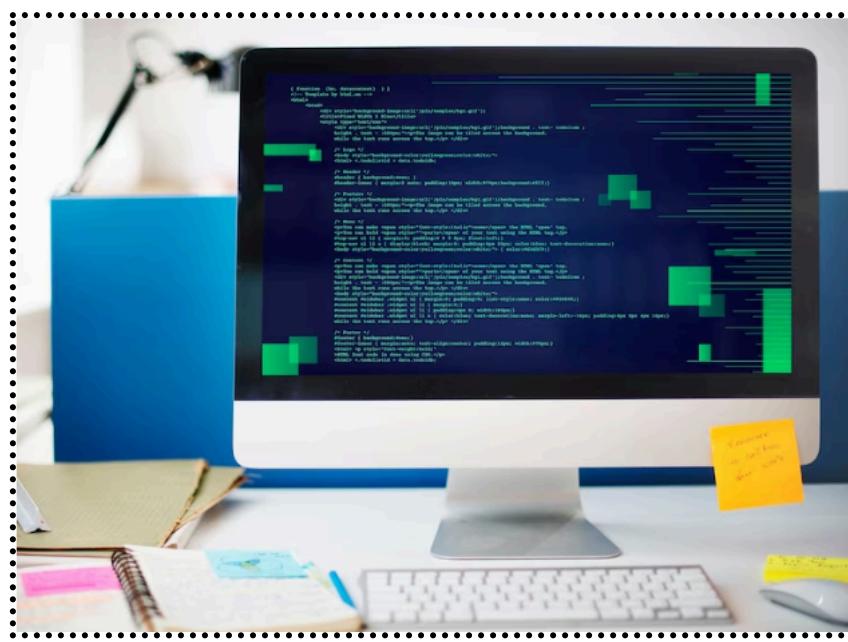
Outro objetivo principal da programação OO é desenvolver objetos mais genéricos para que o software possa se tornar mais reutilizável entre os projetos. Um objeto "cliente" genérico, por exemplo, deve ter aproximadamente o mesmo conjunto básico de comportamentos entre diferentes projetos de software, especialmente quando esses projetos se sobreponem em algum nível fundamental, como costumam acontecer em grandes organizações. Nesse sentido, os objetos de software podem ser vistos mais como componentes plugáveis, ajudando a indústria de software a construir projetos em grande parte a partir de peças existentes e bem testadas, levando assim a uma redução massiva nos tempos de desenvolvimento.

A reutilização de software encontrou resultados práticos mistos, com duas dificuldades principais: o design de objetos verdadeiramente genéricos é mal compreendido e falta uma metodologia para comunicação ampla de oportunidades de reutilização.

Independência de plataforma

A segunda característica, independência de plataforma, significa que os programas escritos na linguagem Java devem ser executados de forma semelhante em diversos hardwares. Deve-se ser capaz de escrever um programa uma vez e executá-lo em qualquer lugar.

Isso é obtido pela maioria dos compiladores Java compilando o código da linguagem Java "pela metade" para bytecode (especificamente bytecode Java) - instruções de máquina simplificadas específicas para a plataforma Java. O código é então executado em uma máquina virtual (VM), um programa escrito em código nativo no hardware host que interpreta e executa bytecode Java genérico. Além disso, bibliotecas padronizadas são fornecidas para permitir o acesso aos recursos das máquinas host (como gráficos, threading e rede) de maneiras unificadas. Observe que, embora haja um estágio de compilação explícito, em algum ponto, o bytecode Java é interpretado ou convertido em instruções de máquina nativa pelo compilador JIT.



Disponível em: <[rawpixel-https://tinyurl.com/4yf5wmcm](https://tinyurl.com/4yf5wmcm)>. Acesso em 11 set. 2023.

Existem também implementações de compiladores Java que compilam para código de objeto nativo, como GCJ, removendo o estágio de bytecode intermediário, mas a saída desses compiladores só pode ser executada em uma única arquitetura.

A licença da Sun para Java insiste em que todas as implementações sejam "compatíveis". Isso resultou em uma disputa legal com a Microsoft depois que a Sun alegou que a implementação da Microsoft não suportava as interfaces RMI e JNI e tinha adicionado recursos específicos de plataforma próprios. Em resposta, a Microsoft não fornece mais o Java com o Windows e, nas versões recentes do Windows, o Internet Explorer não oferece suporte a mini aplicativos Java sem um plug-in

de terceiros. No entanto, a Sun e outros disponibilizaram sistemas de tempo de execução Java sem nenhum custo para essas e outras versões do Windows.

As primeiras implementações da linguagem usaram uma máquina virtual interpretada para conseguir portabilidade. Essas implementações produziram programas que rodavam mais lentamente do que programas compilados em executáveis nativos, por exemplo escritos em C ou C++, então a linguagem sofreu uma reputação de baixo desempenho. Implementações de JVM mais recentes produzem programas que são executados significativamente mais rápido do que antes, usando várias técnicas.

A primeira técnica é simplesmente compilar diretamente no código nativo como um compilador mais tradicional, pulando os bytecodes completamente. Isso proporciona um bom desempenho, mas às custas da portabilidade.

Outra técnica, conhecida como compilação just-in-time (JIT), converte os bytecodes Java em código nativo no momento em que o programa é executado, o que resulta em um programa que executa mais rápido do que o código interpretado, mas também incorre em sobrecarga de compilação durante a execução. VMs mais sofisticadas usam recompilação dinâmica, na qual a VM pode analisar o comportamento do programa em execução e recompilar e otimizar seletivamente partes críticas do programa. A recompilação dinâmica pode alcançar otimizações superiores à compilação estática porque o compilador dinâmico pode basear as otimizações no conhecimento sobre o ambiente de tempo de execução e o conjunto de classes carregadas.

A portabilidade é uma meta tecnicamente difícil de atingir, e o sucesso do Java nessa meta foi misto. Embora seja realmente possível escrever programas para a plataforma Java que se comportem de forma consistente em muitas plataformas host, o grande número de plataformas disponíveis com pequenos erros ou inconsistências levou alguns a parodiar o slogan "Grave uma vez, execute em qualquer lugar" da Sun como "Grave uma vez, depure em todos os lugares".

Java independente de plataforma é, no entanto, muito bem-sucedida com aplicativos do lado do servidor, como serviços da Web, servlets e Enterprise JavaBeans, bem como com sistemas incorporados baseados em OSGi, usando ambientes Java incorporados.

Coleta de lixo automática (Garbage Collector)

Uma ideia por trás do modelo de gerenciamento automático de memória do Java é que os programadores devem ser poupadados do fardo de ter que executar o gerenciamento manual de memória.

Em algumas linguagens, o programador aloca memória para criar qualquer objeto armazenado no heap e é responsável por desalocar manualmente essa memória posteriormente para excluir tais objetos. Se um programador se esquece de desalocar a memória ou escreve um código que não o faz em tempo hábil, pode ocorrer um vazamento de memória: o programa consumirá uma quantidade de memória potencialmente arbitrariamente grande. Além disso, se uma região da memória for desalocada duas vezes, o programa pode se tornar instável e travar. Finalmente, em ambientes sem coleta de lixo, há um certo grau de sobrecarga e complexidade do código do usuário para rastrear e finalizar as alocações.

Em Java, esse problema potencial é evitado pela coleta de lixo automática. O programador

determina quando os objetos são criados, e o Java runtime é responsável por gerenciar o ciclo de vida do objeto. O programa ou outros objetos podem fazer referência a um objeto mantendo uma referência a ele (que, de um ponto de vista de baixo nível, é seu endereço no heap). Quando nenhuma referência a um objeto permanece, o Garbage Colector Java exclui automaticamente o objeto inacessível, liberando memória e evitando um vazamento de memória.

Vazamentos de memória ainda podem ocorrer se o código de um programador contém uma referência a um objeto que não é mais necessário - em outras palavras, eles ainda podem ocorrer, mas em níveis conceituais mais elevados.

O uso do Garbage Colector em uma linguagem também pode afetar os paradigmas de programação. Se, por exemplo, o desenvolvedor assume que o custo de alocação / coleta de memória é baixo, ele pode escolher construir objetos mais livremente em vez de pré inicializá-los, mantê-los e reutilizá-los.

Com o pequeno custo de possíveis penalidades de desempenho (construção de loop interno de objetos grandes / complexos), isso facilita o isolamento de thread (não há necessidade de sincronizar, pois diferentes threads trabalham em diferentes instâncias de objeto) e ocultação de dados. O uso de objetos de valor imutáveis transitórios minimiza a programação de efeitos colaterais. Comparando Java e C++, é possível em C++ implementar funcionalidade semelhante (por exemplo, um modelo de gerenciamento de memória para classes específicas pode ser projetado em C++ para melhorar a velocidade e reduzir consideravelmente a fragmentação de memória), com o possível custo de tempo de desenvolvimento extra e alguns complexidade do aplicativo. Em Java, a coleta de lixo é integrada e virtualmente invisível para o desenvolvedor. Ou seja, os desenvolvedores podem não ter noção de quando a coleta de lixo ocorrerá, pois ela pode não se correlacionar necessariamente com quaisquer ações sendo explicitamente executadas pelo código que eles escrevem. Dependendo da aplicação pretendida, isso pode ser benéfico ou desvantajoso: o programador fica livre de realizar tarefas de baixo nível, mas ao mesmo tempo perde a opção de escrever código de baixo nível.

Sintaxe

A sintaxe do Java é amplamente derivada do C++. No entanto, ao contrário do C++, que combina a sintaxe para programação estruturada, genérica e orientada a objetos, Java foi construído desde o início para ser virtualmente totalmente orientado a objetos: tudo em Java é um objeto com as exceções dos tipos de dados atômicos (ordinal e números reais, valores booleanos e caracteres) e tudo em Java é escrito dentro de uma classe.

Applet

Java applets são programas que estão embutidos em outros aplicativos, normalmente em uma página da Web exibido em um navegador da Web.

```
//Hello.java  
import java.applet.Applet; import java.awt.Graphics;
```

```
public class Hello extends Applet { public void paint (Graphics gc) {  
    gc.drawString ("Olá, mundo!", 65, 95);  
}  
}
```

Este mini aplicativo irá simplesmente desenhar a string "Olá, mundo!" no retângulo dentro do qual o mini aplicativo será executado. Este é um exemplo um pouco melhor do uso de recursos OO do Java, pois a classe estende explicitamente a classe "Applet" básica, substitui o método "pintar" e usa instruções de importação.

```
<! - Hello.html ->  
<html>  
<head>  
<title> Mini aplicativo Hello World </title>  
</head>  
<body>  
<applet code = "Hello" width = "200" height = "200" >  
</applet>  
</body>  
</html>
```

Este documento está usando o elemento HTML <applet>. A tag do mini aplicativo tem três conjuntos de atributos: code = "Hello" especifica o nome da classe do mini aplicativo e largura = "200" altura = "200" define a largura e altura em pixels do mini aplicativo. (Applets também podem ser incorporados em HTML usando o elemento object ou embed, embora o suporte para esses elementos por navegadores da Web seja inconsistente.

Servlet

Java servlets são componentes Java EE do lado do servidor que geram respostas a solicitações de clientes.

```
//Hello.java import java.io.*;  
import javax.servlet.*;
```

```
public class Hello extends GenericServlet {  
    public void service(ServletRequest request, ServletResponse response) throws  
        ServletException, IOException{  
        response.setContentType("text/html");    PrintWriter pw = response.getWriter();
```

```
pw.println("Hello, world!"); pw.close();
}
}
```

As instruções de importação direcionam o compilador Java para incluir todas as classes e interfaces públicas dos pacotes `java.io` e `javax.servlet` na compilação. A classe `Hello` estende a classe `GenericServlet`; a classe `GenericServlet` fornece a interface para o servidor encaminhar solicitações ao servlet e controlar o ciclo de vida do servlet.

A classe `Hello` substitui o método de serviço (`ServletRequest`, `ServletResponse`) definido pela interface `Servlet` para fornecer o código para o manipulador de solicitação de serviço. O método `service ()` recebe um objeto `ServletRequest` que contém a solicitação do cliente e um objeto `ServletResponse` usado para criar a resposta retornada ao cliente. O método `service ()` declara que lança as exceções `ServletException` e `IOException` se um problema o impedir de responder à solicitação.

Aplicativo

Swing é a biblioteca de interface gráfica com o usuário avançada para a plataforma Java SE.

```
// Hello.java
import javax.swing. *;

public class Hello estende JFrame { Hello () {
    setDefaultCloseOperation (WindowConstants.DISPOSE_ON_CLOSE); add (new JLabel ("Olá,
mundo!"));
    pacote();
}
public static void main (String [] args) { new Hello ().setVisible (true);
}
}
```

A instrução `import` direciona o compilador Java para incluir todas as classes e interfaces públicas do pacote `javax.swing` na compilação. A classe `Hello` estende a classe `JFrame`; a classe `JFrame` implementa uma janela com uma barra de título com um controle de fechamento.

O construtor `Hello ()` inicializa o quadro chamando primeiro o método `setDefaultCloseOperation (int)` herdado de `JFrame` para definir a operação padrão quando o controle de fechamento na barra de título é selecionado para `WindowConstants.DISPOSE_ON_CLOSE` - isso faz com que o `JFrame` seja descartado quando o quadro é fechado (em vez de simplesmente oculto), o que permite que a JVM saia e o programa seja encerrado. Em seguida, um novo `JLabel` é criado para a string "Hello, world!" e o método `add (Component)` herdado da superclasse `Container` é chamado para adicionar o rótulo ao quadro. O método `pack ()` herdado da superclasse `Window` é chamado para dimensionar a janela e fazer o layout de seu conteúdo.

O método main () é chamado pela JVM quando o programa é iniciado. Ele instancia um novo quadro Hello e faz com que ele seja exibido chamando o método setVisible (booleano) herdado da superclasse Component com o parâmetro boolean true. Observe que, uma vez que o quadro é exibido, sair do método principal não faz com que o programa seja encerrado porque o encadeamento de despacho de evento AWT permanece ativo até que todas as janelas de nível superior do Swing tenham sido descartadas.

Aparência e comportamento

A aparência e comportamento padrão dos aplicativos GUI escritos em Java usando o kit de ferramentas Swing é muito diferente dos aplicativos nativos. É possível especificar uma aparência e sensação diferentes por meio do sistema plugável de aparência e toque do Swing. Clones do Windows, GTK e Motif são fornecidos pela Sun.

A Apple também oferece uma aparência Aqua para o Mac OS X. Embora as implementações anteriores dessa aparência tenham sido consideradas insuficientes, o Swing no Java SE 6 resolve esse problema usando mais rotinas de desenho de widget nativas das plataformas subjacentes. Como alternativa, kits de ferramentas de terceiros, como wx4j ou SWT, podem ser usados para aumentar a integração com o sistema de janelas nativo.

Java Runtime Environment

O Java Runtime Environment ou JRE é o software necessário para executar qualquer aplicativo implementado na plataforma Java. Os usuários finais geralmente usam um JRE em pacotes de software e plug-ins de navegador da Web. A Sun também distribui um superconjunto do JRE chamado Java 2 SDK (mais comumente conhecido como JDK), que inclui ferramentas de desenvolvimento como compilador Java, Javadoc e depurador.

Importância do JAVA

Atualmente, Java é uma das cinco linguagens de programação mais usadas do mundo. Alguns desenvolvedores tendem a usar Java para o design de aplicativos GUI, enquanto outros usam Java para construir uma variedade de aplicativos da web. Java também é amplamente utilizado na fabricação de jogos móveis e Android, a plataforma móvel mais instalada.

RAZÕES PELAS QUAIS JAVA É E CONTINUARÁ SENDO IMPORTANTE A LONGO PRAZO:



Disponível em: <<https://1000logos.net/java-logo/>>. Acesso em 11 set. 2023.

Amadurece e continua evoluindo

Java é uma linguagem sofisticada e estável para programação. No entanto, a Oracle Corporation atualiza a linguagem de programação regularmente com a ajuda de uma comunidade dinâmica. Cada nova versão do Java possui muitos novos recursos e melhorias de desempenho.

Plataforma independente

Os programadores precisam escrever aplicativos com o uso de vários dispositivos e plataformas. Portanto, eles estão procurando uma linguagem de programação que lhes permita escrever o código do aplicativo uma vez e usar o código do aplicativo em várias plataformas sem nenhum esforço adicional.

Os programadores podem simplesmente compilar o código Java uma vez e implantá-lo em várias plataformas sem recompilar o código. O bytecode habilita o código do aplicativo implantado por programadores em qualquer plataforma de suporte Java. Eles também podem transportar rapidamente o aplicativo de uma plataforma para outra sem compilar constantemente o código.

Suporta Paradigmas de Programação Comuns

As regras de sintaxe do Java são semelhantes à sintaxe C e C++. Assim, é mais fácil para iniciantes aprender e usar Java em menos tempo. Simultaneamente, Java é uma linguagem de programação rival baseada em classe que é orientada a objetos.

Como o Java adota princípios populares de programação orientada a objetos, como herança, polimorfismo, abstração e invólucro, os aplicativos são modulares, extensíveis e escaláveis, mais facilmente acessíveis aos programadores. Os desenvolvedores também podem se beneficiar dessas bibliotecas Java para incorporar de forma mais eficaz os conceitos de design orientado a objetos.

O Google recomenda para desenvolvimento de aplicativos Android

Durante o desenvolvimento de aplicativos móveis, o dispositivo móvel com a base de instalação mais extensa não pode ser esquecido por nenhum desenvolvedor. Os programadores podem escrever aplicativos Android em C, C ++ e Java. Mas o Google sugere que os desenvolvedores de aplicativos móveis escrevam aplicativos Android apenas em Java.

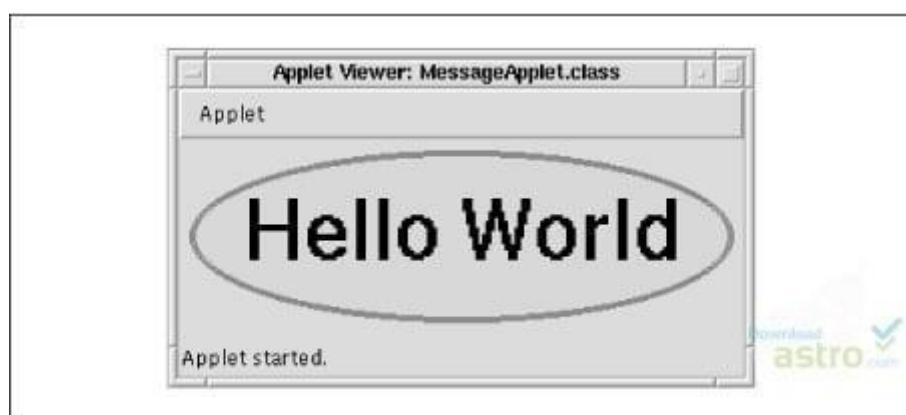
Ao digitá-lo em Java, os desenvolvedores podem melhorar ainda mais o desempenho e a compatibilidade dos aplicativos Android. Os desenvolvedores também têm a opção de escrever aplicativos Android robustos em Java em um tempo mais curto, usando várias ferramentas e bibliotecas.

APIs ROBUSTAS

Java domina outras linguagens na categoria de interfaces de programação de interfaces de programação de aplicativos (APIs). Os programadores têm a opção de criar projetos de desenvolvimento populares usando uma variedade de APIs Java sem adicionar nenhum código extra. Algumas dessas APIs são compartilhadas por empresas importantes e outras são baixadas por membros da comunidade.

Os desenvolvedores podem usar APIs para vincular bancos de dados, entradas e saídas, redes, utilitários, proteção e análise XML de acordo com suas necessidades. Eles podem combinar essas APIs com outras bibliotecas Java Open Source para melhorar a funcionalidade e a eficiência do aplicativo sem adicionar tempo e esforço.

Hello World



Fonte: Elaborado pelo autor (2023).

O primeiro programa sempre se chama Hello World. Este é um exemplo em programação na

linguagem JAVA (você pode executá-lo para testar em algum compilador online, como por exemplo, clicando em Executar no link https://www.tutorialspoint.com/compile_java8_online.php).

The screenshot shows a dark-themed Java code editor interface. On the left, the code for a 'Hello World' program is displayed:

```
1  /* Compilador e Editor Java Online */
2  classe Pública Olá Mundo {
3
4      public static void main ( String [ ] args )
5          Sistema . fora . println ( "Olá, mundo!" ) ;
6
7 }
```

On the right, a terminal window shows the output: "Olá Mundo!". The top bar of the interface includes buttons for 'Executar' (Execute), 'Embelezar' (Beautify), 'Compartilhar' (Share), and 'Código fonte' (Source code).

Disponível em: <https://www.tutorialspoint.com/compile_java8_online.php>. Acesso em 11 set. 2023.

```
public class MeuPrimeiroPrograma {
/* Este é meu primeiro programa em JAVA. Isto é um comentário longo */

    public static void main(String []args) { System.out.println("Hello World"); // imprime Hello
World
}
}
```

Em seguida será apresentada a instalação do ambiente de programação NetBeans, mas pode ser desenvolvido todo o curso com ferramentas online.

Aplicações de programação Java

A versão mais recente do Java Standard Edition é o Java SE 8. Com o avanço do Java e sua ampla popularidade, várias configurações foram criadas para se adequar a vários tipos de plataformas. Por exemplo: J2EE para aplicativos corporativos, J2ME para aplicativos móveis.

As novas versões do J2 foram renomeadas como Java SE, Java EE e Java ME, respectivamente. Java tem algumas características importantes:

- Multithreaded - Com o recurso multithread do Java, é possível escrever programas que podem executar várias tarefas simultaneamente. Este recurso de design permite que os desenvolvedores construam aplicativos interativos que podem ser executados sem problemas.
- Interpretado - o código de bytes Java é traduzido em tempo real para instruções da máquina nativa e não é armazenado em nenhum lugar. O processo de desenvolvimento é mais rápido e analítico, pois a ligação é um processo incremental e leve.

- Alto desempenho - com o uso de compiladores Just-In-Time, o Java permite alto desempenho.
- Distribuído - Java é projetado para o ambiente distribuído da Internet.
- Dinâmico - Java é considerado mais dinâmico do que C ou C++, pois é projetado para se adaptar a um ambiente em evolução. Os programas Java podem transportar uma grande quantidade de informações de tempo de execução que podem ser usadas para verificar e resolver acessos a objetos em tempo de execução.

Ambiente de Desenvolvimento

O NetBeans IDE é um ambiente de desenvolvimento integrado gratuito e de código aberto para desenvolvedores de software nas linguagens Java, JavaScript, HTML5, PHP, C/C++, Groovy, Ruby (sem suporte oficial a partir da versão 7), entre outras. O IDE é executado em muitas plataformas, como Windows, Linux, Solaris e MacOS.

```

AirAlliance1 - NetBeans IDE Dev 200808120201
Projects: AirAlliance1 Services: <default>
File: guest.php Services: flightinfo.php
Edit: flightinfo.php
Search: flightinfo.php
HTML: flightinfo.php
Table: flightinfo.php
Ordered List: flightinfo.php
Unordered List: flightinfo.php
Image: flightinfo.php
Link: flightinfo.php
Meta data: flightinfo.php
HTML Forms: flightinfo.php

```

```

function processReservation($fname,$lname,$sourceList,$destList,$flight,$sdate,$eDate)
{
    $connection = initDB();
    $query1 = "SELECT * FROM Guest WHERE FirstName='".$fname."' AND LastName='".$lname."'";
    $result1 = mysql_query($query1);
    if(mysql_num_rows($result1) == 0)
    {
        $registeredGuest = false;
        $guestID = null;
    }
    else
    {
        $registeredGuest = true;
        $guestID = mysql_result($result1,0,"GuestID");
    }
    $query2 = "UPDATE Guest SET LastFlightID='".$flight."'";
    $result2 = mysql_query($query2);
    if($result2)
    {
        $query3 = "SELECT MAX(GuestID) FROM Guest";
        $result3 = mysql_query($query3);
        $row3 = mysql_fetch_array($result3);
        $SHGID = $row3[0];
    }
    $query4 = "INSERT INTO Guest Values('".$guestID."','".$fname."','".$lname."','".$sdate."','".$eDate."','".$flight."','".$sourceList."','".$destList."','".$SHGID."')";
    $result4 = mysql_query($query4);
    if($result4)
    {
        $query5 = "SELECT * FROM Flights WHERE Flight='".$flight."'";
        $result5 = mysql_query($query5);
        $row5 = mysql_fetch_array($result5);
        $FID = $row5["FID"];
    }
    $query6 = "UPDATE Schedule SET MaxGuestID='".$SHGID."' WHERE FlightID='".$FID."'";
    $result6 = mysql_query($query6);
    if($result6)
    {
        echo "Success";
    }
    else
    {
        echo "Error";
    }
}

```

Disponível em: <<https://tinyurl.com/mnz3t9xr>>. Acesso em 11 set. 2023.

Ele oferece aos desenvolvedores ferramentas para criar aplicativos desktop, Web, corporativos e móveis. O NetBeans foi criado em 1996 por acadêmicos da Universidade de Charles, em Praga, quando a linguagem de programação Java ainda não era tão abrangente no mundo. O nome inicial do projeto era Xelfi, em alusão ao Delphi, pois queriam que as funcionalidades fossem parecidas as IDE's (ambiente de desenvolvimento integrado) desta linguagem, pois eram mais atrativas e visuais, assim como, mais naturais ao uso, desenvolvida em Java.

Em 1999 evoluiu para uma IDE proprietária, chamada NetBeans DeveloperX2. 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.

Hoje em dia, está sendo distribuída em diversos idiomas e a cada dia se torna mais popular, facilitando o acesso a iniciantes em programação e possibilitado o desenvolvimento de aplicativos multilíngue.

Os principais recursos do ambiente são:

- editor de código fonte integrado, rico em recursos para aplicações Web (Servlets e JSP, JSTL, EJBs) e aplicações visuais com Swing que é uma API (Interface de Programação de Aplicativos) Java para interfaces gráficas, a API Swing procura desenhar por conta própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras APIs de interface gráfica trabalham;
- visualizador de classes integrado ao de interfaces, que gera automaticamente o código dos componentes de forma bem organizada, facilitando assim o entendimento de programadores iniciante;
- Suporte ao Java Enterprise Edition, plataforma de programação de computadores que faz parte da plataforma Java voltada para aplicações multicamadas, baseadas em componentes que são executados em um servidor de aplicações;
- plugins para UML, Unified Modeling Language, linguagem de modelagem não proprietária de terceira geração, e desenvolvimento remoto em equipes; interface amigável com CVS ou Concurrent Version System (Sistema de Versões Concorrentes) é um sistema de controle de versão que permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os logs de quem e quando manipulou os arquivos;
- CSS, algumas funcionalidades para editar folhas de estilos como destaque, recursos de auto-completar, análise de código;
- help local e on-line; debug apurado de aplicações e componentes;
- auto-completar avançado; total suporte ao ANT, ferramenta de automatização da construção de programas e TOMCAT, servidor de aplicações Java para web;
- integração de módulos;
- suporte a Database (banco de dados), Data view e Connection wizard que são os módulos embutidos na IDE; geração de Javadoc, a ferramente permite a geração automática de arquivos javadoc em HTML a partir dos comentários inseridos no código, além de recursos que facilitam a inclusão de comentários no código.

Download do ambiente

No momento da construção deste material, a ferramenta de ambiente mais atual é a IDE NetBeans 8.2 com JDK 8u111. Porém é sempre fundamental verificar qual a versão mais estável atual para a preparação do ambiente.

- JDK + Netbeans

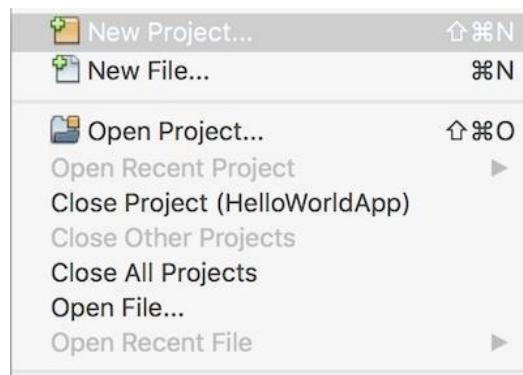


Acesse o QR Code: <<https://www.oracle.com/java/technologies/downloads/#java20>>

Instalação do ambiente

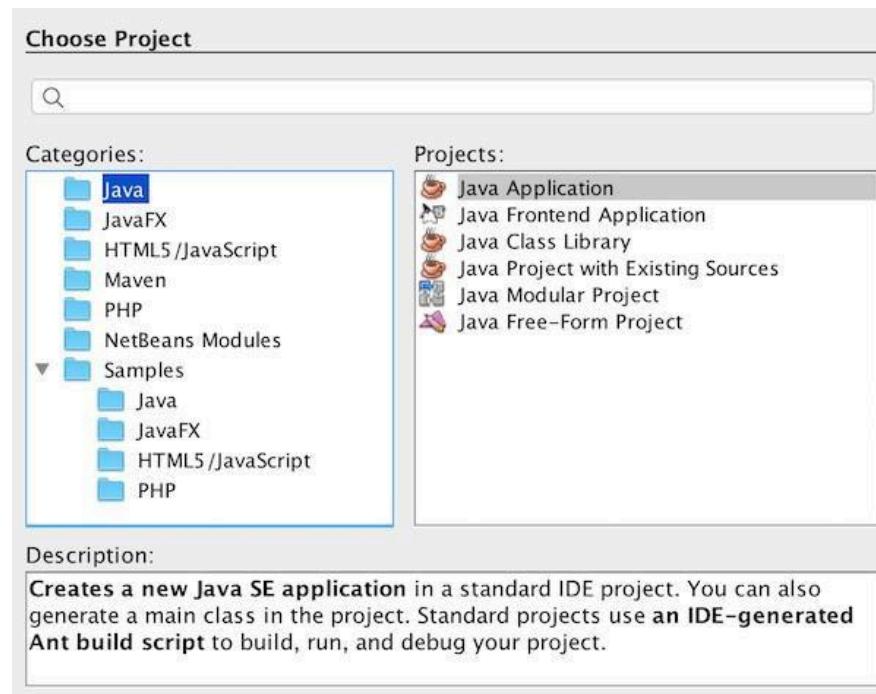
Para criar um projeto na IDE:

1. Inicie o NetBeans IDE.
2. Escolha o menu Arquivo > Novo Projeto, como mostrado na figura abaixo.



Fonte: Elaborado pelo autor (2023).

No assistente, clique em Java e selecione Aplicação Java, em seguida, clique em Próximo.

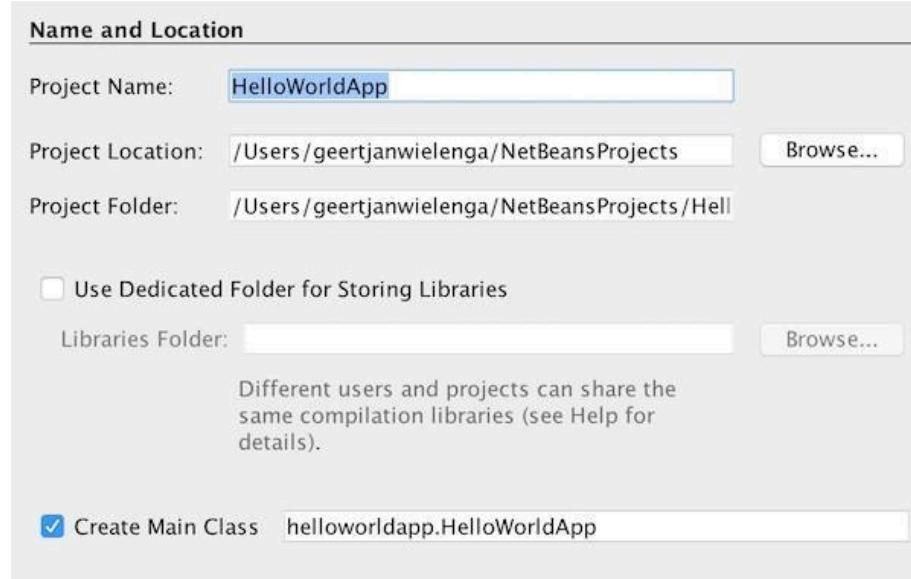


Fonte: Elaborado pelo autor (2023).

Em Nome e Localização, adote o procedimento a seguir (como mostrado na figura abaixo):

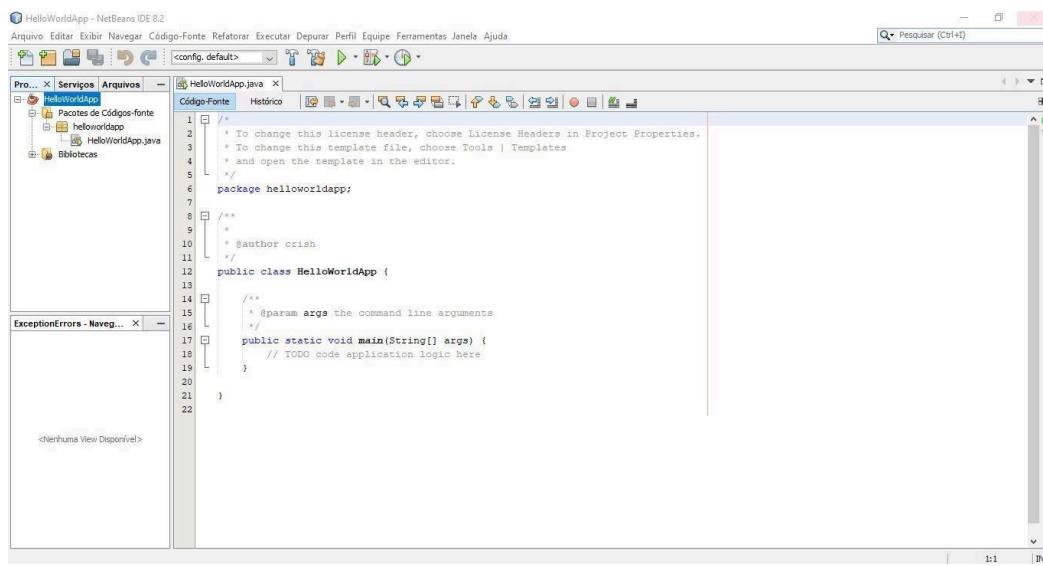
No campo Nome do Projeto, digite HelloWorldApp. Deixe desmarcada a caixa de seleção Utilizar Pasta Dedicada para Armazenar Bibliotecas.

No campo Criar Classe Principal, digite helloworldapp.HelloWorldApp. Clique em Finalizar.



Fonte: Elaborado pelo autor (2023).

O projeto é criado e aberto na IDE.



Fonte: Elaborado pelo autor (2023).

- A janela Projetos, contém uma visualização, em formato de árvore de componentes do projeto, incluindo arquivos de código-fonte, bibliotecas, entre outros.
- A janela Editor de Código-fonte com um arquivo chamado HelloWorldApp é aberta.
- A janela Navegador serve para navegar rapidamente entre os elementos da classe.

O código fonte aberto já inicializa na classe principal, ou seja, a classe main já fica pré-pronta para a construção do código a ser desenvolvido nela.

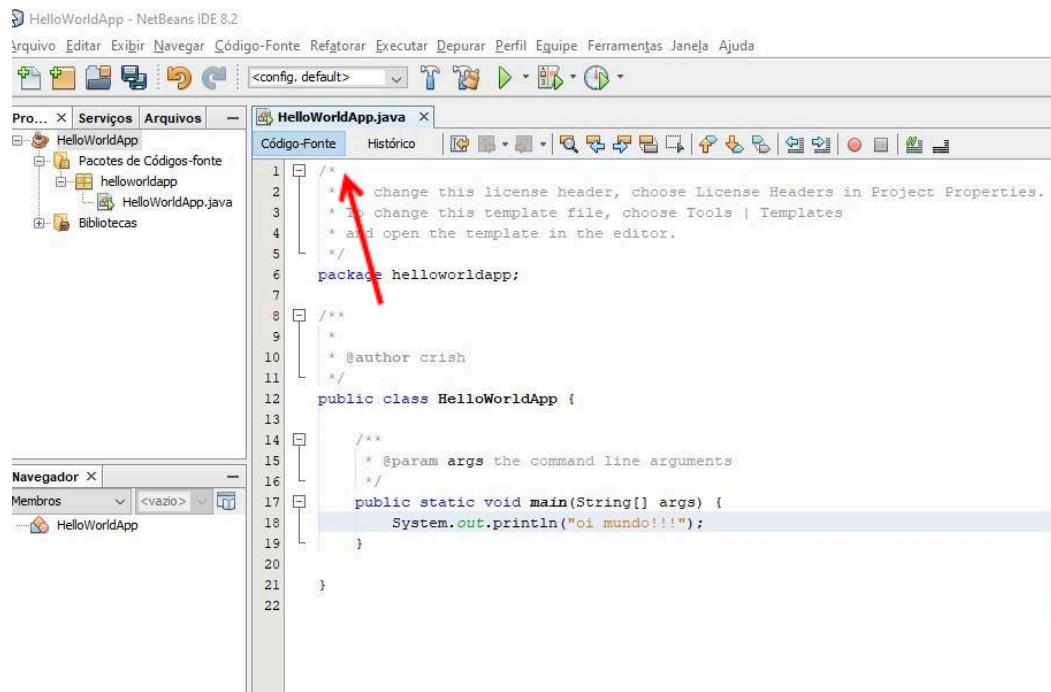
Exemplo do código-fonte inicial de abertura.

```
/*
 *      To change this license header, choose License Headers in Project Properties.
 *      To change this template file, choose Tools | Templates
 *      and open the template in the editor.
 */
package helloworldapp;
/**
 *
 *      @author crish
 */
public class HelloWorldApp {
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
// TODO code application logic here
}
```

}

Compilando e Executando o Programa

Clique em salvar seu projeto e todas suas classes alteradas de uma só vez:



Fonte:

Elaborado pelo autor (2023).

Para executar o programa:

O comando executar o projeto ou, igualmente F6, irá compilar e executar o código.

Escolha Executar > Executar Projeto ou o ícone .





```
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  package helloworldapp;
7
8  /**
9   *
10  * @author crish
11  */
12 public class HelloWorldApp {
13
14     /**
15      * @param args the command line arguments
16     */
17     public static void main(String[] args) {
18         System.out.println("oi mundo!!!");
19     }
20 }
21
22 
```

Fonte: Elaborado pelo autor (2023).

A saída aparecerá abaixo.

```
20
Saída - HelloWorldApp (run) ×
run:
oi mundo!!!
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Fonte: Elaborado pelo autor (2023).



ATIVIDADE DE FIXAÇÃO

1- Quem é frequentemente creditado como o criador do Java?

- a) James Gosling
- b) Bill Gates
- c) Tim Berners-Lee
- d) Linus Torvalds

2- Em que ano o Java foi oficialmente lançado ao público?

- a) 1993
- b) 1995
- c) 2000

d) 1989

3- Qual foi o objetivo principal ao desenvolver o Java?

- a) Criar um novo sistema operacional
- b) Construir uma linguagem para desenvolvimento web
- c) Desenvolver uma linguagem para programação de jogos
- d) Criar uma linguagem portável e robusta para programação

4- Qual das seguintes afirmações é verdadeira sobre a slogan "Write Once, Run Anywhere" (Escreva uma vez, execute em qualquer lugar) relacionada ao Java?

- a) É um termo de marketing sem significado real
- b) Significa que o código Java não precisa ser escrito, apenas copiado
- c) Refere-se à capacidade do código Java ser executado em diferentes plataformas sem modificações
- d) Indica que o Java só pode ser executado em sistemas operacionais específicos

5- Qual é o componente central da plataforma Java que permite a execução de código Java em diferentes ambientes?

- a) JVM (Java Virtual Machine)
- b) JRE (Java Runtime Environment)
- c) JDK (Java Development Kit)
- d) IDE (Integrated Development Environment)

6- O Java é uma linguagem de programação orientada a quais princípios fundamentais?

- a) Programação procedural
- b) Programação estruturada
- c) Programação orientada a objetos
- d) Programação funcional

7- Qual foi a versão inicial do Java que introduziu as classes e conceitos de orientação a objetos?

- a) Java 1.0
- b) Java 1.1
- c) Java 2.0
- d) Java 1.5

8- O que é um "applet" Java?

- a) Um programa Java que é executado apenas no terminal de comando
- b) Uma ferramenta de depuração para código Java
- c) Um tipo de gráfico usado para representar estruturas de controle
- d) Um pequeno programa Java projetado para ser executado em navegadores web

9- Qual empresa foi responsável por adquirir a Sun Microsystems, a criadora original do Java?

- a) Microsoft
- b) IBM

- c) Oracle
- d) Apple

10- O Java é uma linguagem compilada ou interpretada?

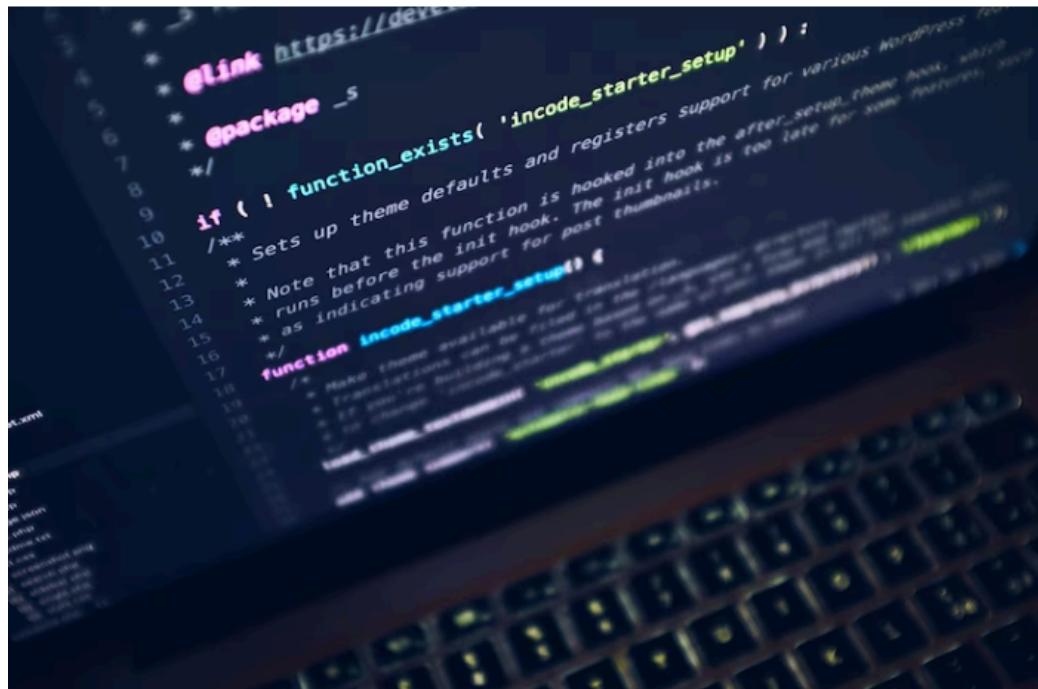
- a) Apenas compilada
- b) Apenas interpretada
- c) Tanto compilada quanto interpretada
- d) Nem compilada nem interpretada

TEMA 02

INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

Habilidades

- Codificar programas orientados a objetos.
- Conectar aplicações com banco de dados.



Disponível em: <[lucabravo-https://tinyurl.com/mnz3t9xr](https://tinyurl.com/mnz3t9xr)>. Acesso em 11 set. 2023.



Acesse o QR Code do vídeo Lógica de Programação e JAVA - Fonte do vídeo: <<https://youtu.be/tekzW3nZLk>>

Conceitos Básicos

Estrutura básica de um programa: Um programa em Java deve ser criado sempre em uma classe. Dentro dessa classe, tem o método main, e é neste método que você deve escrever seu código.

```
public class <nomeprog>
<rotinas-e-variaveis opcionais>
public static void main (String [] args) {
<comandos>
}
<rotinas-e-variaveis opcionais>
}
```

Introdução

A lógica de programação é a linguagem universal dos computadores. É a maneira pela qual os programadores criam instruções lógicas para que as máquinas executem tarefas. Dominar a lógica de programação é fundamental para desenvolver habilidades de resolução de problemas e criar software eficiente. Isso envolve entender sequências lógicas, tomada de decisões e repetição. A lógica de programação é o alicerce sobre o qual a programação de computadores é construída, independentemente da linguagem utilizada. É o primeiro passo para entrar no mundo da codificação e da criação de soluções tecnológicas.

Regras para formação de nomes:

- Devem começar com letra ou _
- Podem ter letras, dígitos ou _
- Nomes de classes devem começar com letra maiúscula
- Nomes de variáveis e subrotinas devem começar com letra minúscula e, se forem nomes compostos, estes devem iniciar com letra maiúscula (Exemplos: verSaldo, nomeAluno, calculaMedia etc.)
- A linguagem JAVA é “Case Sensitive”. Isso significa que letras maiúsculas e minúsculas fazem diferença. Se uma variável com o nome total for declarada, ela será diferente de Total, TOTAL, ToTaL ou tOtAl.

Comentários:

```
// comentários em uma linha
/* comentários */
```

bloco de comentário

Saída De Dados – Classe System

O objeto System.out é a saída padrão em Java, ou seja, a forma de mostrar na tela todos os comandos definidos no console ou na Interface Gráfica (GUI), quando o código Java é executado.

Dentro deste objeto, os métodos de saídas de dados são: println, print e printf.

System.out.println()

A instrução `System.out.println()` gera uma saída de texto entre aspas duplas, seguida da execução de uma nova linha, em que posiciona o cursor na linha abaixo - instrução determinada pelo “In”.

Por exemplo:

Saída com `System.out.println`

```
public class Texto_println {  
    public static void main(String[] args) {  
        System.out.println("Seu texto entre aspas duplas");  
    }  
}
```

System.out.print()

A instrução `System.out.print()`, sem o final “In”, exibe uma String, definida entre aspas duplas, sem criar uma linha, ou seja, o cursor ficará na mesma linha do texto.

Por exemplo:

Saída com `System.out.print`

```
public class Texto_print {  
    public static void main(String[] args) { System.out.print("Crishna "); System.out.print("Irion");  
    }  
}
```

Caractere de escape

O caractere de escape permite inserir uma nova linha dentro dos métodos `print` e `println` do objeto `System.out`.

```
public class Texto_sequencia_caractere { public static void main(String[] args) {  
    System.out.print(" Sou \n uma \n boa \n aluna \n ");  
}
```

Neste exemplo, o `\n` não aparece na saída do console, porém irá quebrar as linhas para cada palavra que segue sua sequência.

Existem vários caracteres de escape:

Caractere de escape	Descrição
<code>\n</code>	Nova linha. Posiciona o cursor de tela no início da próxima linha
<code>\t</code>	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.

\r	Posiciona o cursor da tela no início da linha atual - não avança para a próxima linha. Qualquer saída de caracteres gerada depois de algum retorno já gerado é sobreescrito os caracteres anteriores gerados na linha atual.
\\"	Barras invertidas. Utilizada para imprimir um caractere de barra invertida.
\\"	Aspas duplas. Utilizada para imprimir um caractere de aspas duplas. Exemplo: <code>System.out.println("\\"aspas\\\"");</code> exibe "aspas"

DEFINIÇÃO E CRIAÇÃO DE VARIÁVEIS E CONSTANTES

Variáveis

A linguagem JAVA exige a declaração de tipos de dados para todas as suas variáveis.

Em Java, uma variável é declarada de maneira que é definido o tipo de dados que ela poderá receber, assim como o seu nome (identificador).

Por exemplo:

```
int numero;  
float numerodecimal; String nome;
```

No código exemplo, definimos a variável numero como sendo do tipo inteiro int e, portanto, só poderá receber valores do tipo inteiro.

O mesmo ocorre com a variável numerodecimal, que somente receberá números decimais (reais), definida como float.

A variável nome foi definida como do tipo String, que receberá dados alfanuméricos.

O tipo de dado pode ser definido como dado primitivo int, float, char ou boolean assim como de classe nativa do Java (String ou ArrayList) ou ainda desenvolvida manualmente por fontes externas.

Tipo	Tamanho em bits	Valores	Padrão
<code>boolean</code>		<code>true</code> ou <code>false</code>	
<i>[Observação: a representação de um boolean é específica à Java Virtual Machine em cada plataforma.]</i>			
<code>char</code>	16	'\u0000' a '\uFFFF' (0 a 65535)	(conjunto de caracteres Unicode ISO)
<code>byte</code>	8	-128 a +127 (-2 ⁷ a 2 ⁷ - 1)	
<code>short</code>	16	-32.768 a +32.767 (-2 ¹⁵ a 2 ¹⁵ - 1)	
<code>int</code>	32	-2.147.483.648 a +2.147.483.647 (-2 ³¹ a 2 ³¹ - 1)	
<code>long</code>	64	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807 (-2 ⁶³ a 2 ⁶³ - 1)	
<code>float</code>	32	<i>Intervalo negativo:</i> -3,4028234663852886E+38 a -1,40129846432481707e-45 <i>Intervalo positivo:</i> 1,40129846432481707e-45 a 3,4028234663852886E+38	(IEEE 754, ponto flutuante)
<code>double</code>	64	<i>Intervalo negativo:</i> -1,7976931348623157E+308 a -4,94065645841246544e-324 <i>Intervalo positivo:</i> 4,94065645841246544e-324 a 1,7976931348623157E+308	(IEEE 754, ponto flutuante)

Disponível em: <<https://tinyurl.com/47anxkkf>>. Acesso em 11 set. 2023.

Identificadores de variáveis

Os identificadores (nomes) de variáveis precisam ser definidos respeitando regras e convenções:

1. Pode ser composta por letras, números e underline (_), porém não pode iniciar com um número;
2. Existe uma técnica de organização de nomes de acordo com sua utilização(Camel Case), por exemplo, Classes começam com maiúscula, métodos, atributos e variáveis com minúsculas, seguidas da próxima palavra iniciada com maiúscula;
3. Java tem como característica ser case sensitive. Assim, um identificador nomeado numeroUm não é o mesmo que numero um.

Exemplos de declaração de variáveis:

```
int numeroCasa; float _salario;
String 1variavel; // //Erro no nome dA IDEntificador, por iniciar com caractere numérico.
```

Variáveis de classe

A declaração dentro de uma classe segue o formato:

```
class Aluno {
    private int matricula; public String ra; protected String nome;
}
```

Pode-se definir os modificadores das variáveis antes da declaração. A palavra `private` é um modificador de acesso, que também pode ser `public` e `protected`.

Os modificadores definem se o acesso à variável é público, privado ou protegido.

Constantes

As constantes são declaradas quando é necessário trabalhar com dados que não podem ser alterados durante a execução do programa. Elas são representadas com letras maiúsculas.

Em Java há uma palavra-chave `, const`, para este fim, mas pode-se usar `final` para o mesmo fim. A diferença entre as formas de uso é que utilizando `final` a variável será inicializada uma só vez, porém o valor pode ser definido após a sua declaração.

Por exemplo:

```
final float PI = 3.1416F;  
final String NOME_PAGINA = "home";
```

Operadores Aritméticos, Relacionais e Lógicos

Antes de apresentar os operadores de comparação, é fundamental apresentar a forma de atribuição dos dados em Java.

Para que uma variável “receba” um valor, o operador de atribuição `=` representa que uma variável receberá o valor definido.

Por exemplo:

```
int x = 2;  
final float pi = 3.1415f; String texto = "Aula Android"; int lado = 3;
```

Operadores Aritméticos

Os operadores aritméticos são responsáveis pelas operações matemáticas entre as variáveis, retornando o resultado dessas operações.

No caso de existirem operações mais complexas, pode-se combinar os operadores ou criar expressões que permitam executar todo tipo de cálculo.

Por exemplo:

```
int area = 2 * 2;
```

Neste exemplo o cálculo de um quadrado de lados 2m. Ou no exemplo abaixo, o cálculo da área de uma circunferência de raio 3cm:

```
float raio= 3;  
final float pi = 3.1415f; float area = raio * raio * pi;
```

+	operador de adição
-	operador subtração
*	operador de multiplicação
/	operador de divisão
%	operador de módulo (ou resto da divisão)

Fonte: Elaborado pelo autor (2023).

Operadores de incremento e decremento

Os operadores de incremento e decremento (`++` e `--`) são operadores que podem ser utilizados para incrementar, de um em um ou decrementar, também de um em um, uma variável numérica.

Por exemplo:

```
int num = 2; num++;
num--; //a variável num continuará valendo 2.
```

Observação:

Quando utilizamos esse operador antes da variável, o incremento/decremento é realizado antes do valor da variável ser processado, mas quando utilizado após, o valor da variável é primeiro processado e só então o valor será incrementado/decrementado.

Por exemplo:

```
int num = 5; num++;
++num;
```

Operadores de igualdade

Os operadores de igualdade operam no resultado da expressão lógica entre duas expressões: se são iguais (`==`) ou diferentes (`!=`), e retornam um valor booleano (`true` ou `false`).

Por exemplo:

```
int A = 1; int B = 2; if(A == B){
System.out.println("São iguais");
} else {
System.out.println("São diferentes");
}
```

Operadores relacionais

Os operadores relacionais avaliam dois operandos, comparando as relações de operações entre eles. Neste exemplo, definem se o operando à esquerda é maior ou igual, menor ou igual ao da direita, retornando um valor booleano.

Por exemplo:

```
int A = 1; int B = 2; if(A >= B){  
    System.out.println("A maior ou igual a B");  
}  
if(A <= B){  
    System.out.println("A menor ou igual a B");  
}
```

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.

Fonte: Elaborado pelo autor (2023).

Operadores lógicos

Os operadores lógicos são operadores que comparam expressões, sejam associadas ou não. As operações lógicas possíveis de serem verificadas são: E (representada por `&&`) e OU (representada por `||`).

Por exemplo:

```
if((5 == (10-5)) && (10 == (5 + 5))){  
    System.out.println("Estas expressões são ambas verdadeiras");  
}
```

Precedência de operadores

Os operadores aritméticos reproduzem operações matemáticas no código, o que mantém as regras de precedência, podendo ser manipuladas ao longo do código através do uso de parênteses. Por exemplo, o cálculo da média de 2 notas, n1 e n2, deve ter em sua expressão o uso de parênteses na soma das notas, antes da divisão.

Por exemplo:

```
float media, n1, n2; media = (n1 + n2)/2;
```

CLASSES JAVA: SCANNER E MATH

Classe Scanner

A classe Scanner tem como finalidade de executar o processo de entrada de dados no modo Console. Antes da versão do Java 5 não havia essa alternativa de entrada de dados, o que dificultava a interação do usuário com o código, principalmente nas primeiras interações de teste ao código.

A classe Scanner é uma classe do pacote java.util usada para obter a entrada dos tipos primitivos como int, double, etc. e strings. É a maneira mais fácil de ler a entrada em um programa Java.

Algumas características:

- Para criar um objeto da classe Scanner, normalmente passamos o objeto predefinido System.in, que representa o fluxo de entrada padrão. Podemos passar um objeto da classe File se quisermos ler a entrada de um arquivo (veremos no TEMA 08).
- Para ler valores numéricos de um determinado tipo de dados XYZ, a função a ser usada é nextXYZ ().
- Para ler strings, usamos nextLine () .
- Para ler um único caractere, usamos next(). CharAt (0). A função next () retorna o próximo token / palavra na entrada como uma string e a função charAt (0) retorna o primeiro caractere nessa string.

Por exemplo:

```
// Programa Java para ler vários dados diferentes usando a Scanner.  
import java.util.Scanner; public class ScannerDemo1  
{  
    public static void main(String[] args)  
    {  
        // Declare o objeto da classe Scanner (sc) Scanner sc = new Scanner(System.in);  
        // entrada de uma String String nome = sc.nextLine();  
  
        // entrada de um Caracter  
        char genero = sc.next().charAt(0);  
        // entrada de um número byte, short ou float int idade = sc.nextInt();  
        long fone = sc.nextLong(); double cod = sc.nextDouble();  
    }  
}
```

Entrada :

```
// Imprimindo os valores para checar as entradas: System.out.println("Nome: "+nome);
System.out.println("Gênero: "+genero); System.out.println(Idade: "+idade);
System.out.println("Número do celular: "+fone); System.out.println("COD: "+cod);
```

José M 40

9876543210

9.9

Resultado :

Nome: José Gênero: M Idade: 40

Número do celular: 9876543210 COD: 9.9Métodos da Classe

Na tabela estão apresentados os principais métodos da classe Scanner.

Método	Descrição
close()	Fechá o escaneamento de leitura.
findInLine()	Encontra a próxima ocorrência de um padrão ignorando máscaras ou strings ignorando delimitadores.
hasNext()	Retorna um valor booleano verdadeiro (true) se o objeto Scanner tem mais dados de entrada.
hasNextXyz()	Retorna um valor booleano como verdadeiro (true) se a próxima entrada a qual Xyz pode ser interceptada como Boolean, Byte, Short, Int, Long, Float ou Double.
match()	Retorna o resultado da pesquisa do último objeto Scanner atual.
next()	Procura e retorna a próxima informação do objeto Scanner que satisfazer uma condição.
nextBigDecimal(), nextBigInteger()	Varre a próxima entrada como BigDecimal ou BigInteger.
nextXyz()	Varre a próxima entrada a qual Xyz pode ser interceptado como boolean, byte, short, int, long, float ou double.
nextLine()	Mostra a linha atual do objeto Scanner e avança para a próxima linha.
radix()	Retorna o índice atual do objeto Scanner.

<code>remove()</code>	Essa operação não é suportada pela implementação de um Iterator.
<code>skip()</code>	Salta para a próxima pesquisa de um padrão especificado ignorando delimitadores.
<code>string()</code>	Retorna uma string que é uma representação do objeto Scanner.

Classe Math

A classe Java Math possui muitos métodos que permitem realizar tarefas matemáticas em números.

Essa classe já está na package (pacote) `java.lang`. Ou seja, não é necessário fazer nenhuma importação. Cada método da classe Math é fundamental para as operações matemáticas necessária ao longo do código.

Para a compreensão geral da Classe Math é fundamental conhecer os principais métodos utilizados por ela, entendendo seu funcionamento, seus parâmetros e retornos.

1. `Math.abs(...);`

Descrição: Retorna o valor absoluto (módulo) do numero passado por parâmetro.

Parâmetro: Pode ser um int, um double, um float ou um long
Retorno: Mesmo tipo primitivo do valor de entrada (parâmetro)

2. `Math.acos(...);`

Descrição: Retorna ao usuário o arco-cosseno do ângulo passado por parâmetro (retorno entre 0 e PI [metade superior de uma circunferência trigonométrica])

Parâmetro: double Retorno: double

3. `Math.asin(...);`

Descrição: Retorna ao arco-seno do ângulo passado para o método (retorno entre -PI/2 [3/4 de circunferência trigonométrica] e PI/2 (1/4 da circunferência))

Parâmetro: double Retorno: double

4. `Math.atan(...);`

Descrição: Retorna o arco-tangente do ângulo que o usuário passou (mesmo "range" (intervalo) do Math.asin(...))

Parâmetro: double Retorno: double

5. `Math.ceil(...);`

Descrição: Este método retorna o maior numero inteiro (menor que o passado como parâmetro) - (ATENCAO PARA NUMEROS NEGATIVOS)

Parâmetro: double Retorno: double

6. `Math.cos(...);` Descrição: Retorna o cosseno do ângulo passado

Parâmetro: double Retorno: double

7. `Math.exp(...);`

Descrição: Retorna o valor da Constante de Euller "e" elevada ao numero passado

Parâmetro: double Retorno: double

8. `Math.floor(...);`

Descrição: Retorna o maior numero inteiro (não menor que o passado) - (ATENCAO PARA NUMEROS NEGATIVOS)

Parâmetro: double Retorno: double

9. Math.log(...);

Descrição: Retorna o logaritmo natural do numero passado.

Parâmetro: double Retorno: double

10. Math.max(... , ...);

Descrição: Retorna o maior entre os números passados

Parâmetro: pode ser um par de int, de double, de float ou de long (desde que os 2 parâmetros sejam do mesmo tipo)

Retorno: depende do tipo de entrada

11. Math.min(... , ...);

Descrição: Retorna o menor entre os números passados

Parâmetro: pode ser um par de int, de double, de float ou de long (desde que os 2 parâmetros sejam do mesmo tipo)

Retorno: depende do tipo de entrada

12. Math.pow(... , ...);

Descrição: Para uma estrutura de potenciação a^b este método retorna o primeiro parâmetro como 'a' e o segundo como 'b' Parâmetro: double Retorno: double

13. Math.random();

Descrição: um numero aleatório que vai de zero até 1 (0 incluído, 1 nunca será gerado)

Parâmetro: nenhum Retorno: double

14. Math.round(...);

Descrição: Retorna o long mais próximo do parâmetro passado

Parâmetro: double Retorno: long

15. Math.sin(...);

Descrição: Retorna o seno do parâmetro

Parâmetro: double Retorno: double

16. Math.tan(...);

Descrição: Retorna a tangente do ângulo

Parâmetro: double Retorno: double

17. Math.sqrt(...);

Descrição: Retorna a raiz quadrada do numero passado

Parâmetro: double Retorno: double

18. Math.toDegrees(...);

Descrição: Retorna o angula passado (em radianos) em graus

Parâmetro: double Retorno: double

19. Math.toRadians(...);

Descrição: Retorna o angula passado (em graus) em radianos

Parâmetro: double Retorno: double

Cada método aparece no exemplo abaixo, associado à uma saída no console:

```
class ClasseMath {
```

```
public static void main(String args[]) { System.out.println("Metodo abs(-30): " + Math.abs(-30)); System.out.println("Metodo acos(0.5): " + Math.acos(0.5) ); System.out.println("Metodo asin(0.5): " + Math.asin(0.5) ); System.out.println("Metodo atan(60): " + Math.atan(60) ); System.out.println("Metodo ceil(5.215): " + Math.ceil(5.215) ); System.out.println("Metodo ceil(-5.215): " + Math.ceil(-5.215) ); System.out.println("Metodo cos(60): " + Math.cos(60) ); System.out.println("Metodo exp(10): " + Math.exp(10) ); System.out.println("Metodo floor(54.687): " + Math.floor(54.687) ); System.out.println("Metodo floor(-54.687): " + Math.floor(-54.687) ); System.out.println("Metodo log(2): " + Math.log(2) ); System.out.println("Metodo max(5,7): " + Math.max(5,7) ); System.out.println("Metodo min(-3,2): " + Math.min(-3,2) ); System.out.println("Metodo pow(2,3): " + Math.pow(2,3) ); System.out.println("Metodo random(): " + Math.random() ); System.out.println("Metodo round(13.124): " + Math.round(13.124) ); System.out.println("Metodo sin(30): " + Math.sin(30) ); System.out.println("Metodo sqrt(16): " + Math.sqrt(16) ); System.out.println("Metodo tan(45): " + Math.tan(45) ); System.out.println("Metodo toDegrees(2): " + Math.toDegrees(2) ); System.out.println("Metodo toRadians(90): " + Math.toRadians(90) );  
}  
}  
}
```

O resultado na saída será:

```
Metodo abs(-30): 30  
Metodo acos(0.5): 1.0471975511965979  
Metodo asin(0.5): 0.5235987755982989  
Metodo atan(60): 1.554131203080956  
Metodo ceil(5.215): 6.0  
Metodo ceil(-5.215): -5.0  
Metodo cos(60): -0.9524129804151563  
Metodo exp(10): 22026.465794806718  
Metodo floor(54.687): 54.0  
Metodo floor(-54.687): -55.0  
Metodo log(2): 0.6931471805599453  
Metodo max(5,7): 7  
Metodo min(-3,2): -3  
Metodo pow(2,3): 8.0  
Metodo random(): 0.6100207813062897  
Metodo round(13.124): 13  
Metodo sin(30): -0.9880316240928618  
Metodo sqrt(16): 4.0  
Metodo tan(45): 1.6197751905438615  
Metodo toDegrees(2): 114.59155902616465  
Metodo toRadians(90): 1.5707963267948966
```

A classe math possui, ainda, 2 campos finais estáticos (constantes), que podem ser usados ao longo do código, apenas respeitando o tipo dos dados deles, são eles:

- Math.E -> que é a constante para bases naturais de logs
- Math.PI -> constante do valor PI

Por exemplo, o uso da constante em um cálculo dentro de um método:

```
public double calculaArea(double raio) {  
    ....  
    return raio* raio* Math.PI;  
}
```

Fonte: Elaborado pelo autor (2023).

Obs.: O Math.PI é uma variável do tipo double.



ATIVIDADE DE FIXAÇÃO

1- O que é a lógica de programação?

- a) Uma linguagem natural usada para comunicação entre programadores.
- b) Um conjunto de regras para a criação de algoritmos.
- c) Uma linguagem de programação popular.
- d) Um software usado para depurar código.

2- Qual é o objetivo principal da lógica de programação?

- a) Ensinar gramática e ortografia.
- b) Criar designs gráficos para programas.
- c) Desenvolver interfaces de usuário atraentes.
- d) Resolver problemas computacionais de maneira estruturada.

3- O que são algoritmos?

- a) Linguagens de programação modernas.
- b) Pequenos programas independentes.
- c) Sequências de passos lógicos para resolver problemas.
- d) Elementos visuais usados na criação de software.

4- Qual dos seguintes não é um exemplo de estrutura de controle em programação?

- a) Sequência
- b) Laço de repetição
- c) Vetor
- d) Tomada de decisão

5- Qual é o propósito da estrutura de controle de repetição (loop) em lógica de programação?

- a) Executar um bloco de código apenas uma vez.
- b) Controlar a entrada e saída de dados.
- c) Repetir um bloco de código várias vezes.
- d) Realizar cálculos matemáticos complexos.

6- O que é uma condição em lógica de programação?

- a) Um valor numérico constante.
- b) Um operador matemático.
- c) Uma instrução que verifica uma situação específica.
- d) Uma sequência de caracteres.

7- Qual termo é usado para se referir a valores verdadeiros ou falsos em lógica de programação?

- a) Dados binários
- b) Dados complexos
- c) Dados booleanos
- d) Dados numéricos

8- O que é um pseudocódigo?

- a) Um código de programação altamente otimizado.
- b) Um tipo de gráfico usado para representar algoritmos.
- c) Uma linguagem de programação orientada a objetos.
- d) Uma representação textual simplificada de um algoritmo.

9- Qual é o principal objetivo da estrutura de decisão em lógica de programação?

- a) Executar um bloco de código repetidamente.
- b) Realizar cálculos matemáticos.
- c) Tomar decisões com base em condições específicas.
- d) Criar interfaces de usuário interativas.

10- O que é um diagrama de fluxo?

- a) Um dispositivo de hardware usado para programar computadores.
- b) Um tipo de algoritmo.
- c) Uma representação gráfica de um algoritmo.
- d) Um tipo de linguagem de programação.

TEMA 03

CLASSES JAVA: STRINGS

Habilidades

- Codificar programas orientados a objetos.



Disponível

em: <<https://tinyurl.com/3cawa5yc>>. Acesso em 11 set. 2023.



Acesse o QR Code do vídeo Classes em Java - Fonte do vídeo: <<https://youtu.be/MCprYW6NaY0>>

Introdução

As classes de strings em Java são fundamentais para o tratamento de texto. Elas permitem a manipulação, concatenação e análise de cadeias de caracteres. As strings são imutáveis, ou seja, uma vez criadas, não podem ser alteradas diretamente. Isso garante estabilidade e segurança na manipulação de informações textuais. Por meio das classes de strings, os programadores podem realizar tarefas como combinar palavras, buscar padrões e modificar conteúdo de maneira eficiente. As operações com strings são essenciais em muitos aplicativos, desde processamento de dados até a criação de interfaces de usuário.

A definição de String é: uma sequência organizada de caracteres.

Criação de uma string

Existem duas maneiras de criar strings em Java:

1. Literal de string

```
String s = "CursodeJAVA";
```

2. Usando a palavra-chave **new**

```
String s = new String ("CursodeJAVA");
```

Construtores

1. `String (byte [] byte_arr)` - Constrói uma nova String decodificando a matriz de bytes .

Ele usa o conjunto de caracteres padrão da plataforma para decodificação.

Exemplo:

```
byte [] b_arr = {71, 101, 101, 107, 115};
```

```
String s_byte = new String (b_arr);
```

2. `String (byte [] byte_arr, Charset char_set)` - Constrói uma nova String decodificando a matriz de bytes . Ele usa o `char_set` para decodificação.

Exemplo:

```
byte [] b_arr = {71, 101, 101, 107, 115};
```

```
Charset cs = Charset.defaultCharset (); String s_byte_char = new String (b_arr, cs);
```

3. `String (byte [] byte_arr, String char_set_name)` - Construa uma nova String decodificando a matriz de bytes . Ele usa o `char_set_name` para decodificação. É semelhante às construções acima e elas aparecem antes de funções semelhantes, mas leva o `String` (que contém `char_set_name`) como parâmetro enquanto o construtor acima leva `CharSet`.

Exemplo:

```
byte [] b = {1, 11, 21, 77, 15};
```

```
String s = new String (b, "US-ASCII");
```

4. `String (byte [] byte_arr, int start_index, int length)` - Constrói uma nova string a partir da matriz de bytes dependendo do `start_index` (localização inicial) e comprimento (número de caracteres da localização inicial).

Exemplo:

```
byte [] b = {1, 11, 21, 77, 15};
```

```
String s = nova String (b, 1, 3);
```

- String (`byte [] byte_arr, int start_index, int length, Charset char_set`) - Constrói uma nova string da matriz de bytes dependendo do `start_index` (localização inicial) e comprimento (número de caracteres da localização inicial). Usa `char_set` para decodificação.

Exemplo:

```
byte [] b = {1, 11, 21, 77, 15};
```

```
Charset cs = Charset.defaultCharset (); String s = nova String (b, 1, 3, cs);
```

5. `String (byte [] byte_arr, int start_index, int length, String char_set_name)` - Constrói

uma nova string da matriz de bytes dependendo do start_index (localização inicial) e comprimento (número de caracteres da localização inicial). Usa char_set_name para decodificação.

Exemplo: byte [] b = {1, 11, 21, 77, 15};

```
String s = new String (b, 1, 4, "US-ASCII");
```

6. String (char [] char_arr) - Aloca uma nova String da matriz de caracteres fornecida .

Exemplo:

```
char c [] = {'J', 'a', 'v', 'a'}; String s = new String (c);
```

Métodos de String

1. int length (): Retorna o número de caracteres na String.

```
"CursodeJAVA".length () // retorna 11
```

2. char charAt (int i) : Retorna o caractere no i ésmo índice.

```
"CursodeJAVA".charAt (3) // retorna 's' (o vetor inicia de 0)
```

3. String substring (int i) : Retorna a substring do i ésmo caractere de índice até o final.

```
" CursodeJAVA".substring(3) // retorna "odeJAVA"
```

4. String substring (int i, int j) : Retorna a substring de i para o índice j-1.

```
" CursodeJAVA".substring (2, 5) // retorna "rso"
```

5. String concat (String str) : Concatena a string especificada ao final desta string.

```
String s1 = "Cursode"; String s2 = "JAVA";
```

```
String saida = s1.concat (s2); // retorna "CursodeJAVA"
```

6. int indexOf (String s): Retorna o índice dentro da string da primeira ocorrência da string especificada.

```
String s = "Aprenda, Compartilhe";
```

```
int output = s.indexOf ("Compartilhe"); // retorna 9
```

7. int indexOf (String s, int i) : Retorna o índice dentro da string da primeira ocorrência da string especificada, começando no índice especificado.

```
String s = "Aprenda, Compartilhe";
```

```
int output = s.indexOf ("ar", 3); // retorna 13
```

8. Int lastIndexOf (String s): Retorna o índice dentro da string da última ocorrência da string especificada.

```
String s = "Aprenda, Compartilhe";
```

```
int saida = s.lastIndexOf ("a"); // retorna 14
```

9. boolean equals (Object otherObj): Compara esta string com o objeto especificado.

```
Boolean out = "Java".equals ("Java"); // retorna verdadeiro Boolean out = "JAVA" .equals ("java"); // retorna falso
```

10. boolean equalsIgnoreCase (String anotherString) : Compara uma string a outra string, ignorando se são maiúsculas e minúsculas.

```
Boolean out = "Java".equalsIgnoreCase ("Java"); // retorna verdadeiro Boolean out = "JAVA".equalsIgnoreCase ("java"); // retorna verdadeiro
```

11. int compareTo (String anotherString) : Compara duas strings lexicograficamente.

```
int out = s1.compareTo(s2); // s1 e s2 são strings a serem comparadas
```

Isso retorna a diferença s1-s2. Se :

out < 0 // s1 vem antes de s2 out = 0 // s1 e s2 são iguais. out > 0 // s1 vem depois de s2.

12. int compareToIgnoreCase (String anotherString): Compara duas strings lexicograficamente, ignorando as considerações de maiúsculas e minúsculas.

```
int out = s1.compareToIgnoreCase(s2);
```

// onde s1 e s2 são strings a serem comparadas

Isso retorna a diferença s1-s2. Se:

out < 0 // s1 vem antes de s2 out = 0 // s1 e s2 são iguais. out > 0 // s1 vem depois de s2.

Neste caso, não irá considerar maiúsculas e minúsculas (irá ignorar se é maiúscula ou minúscula).

String toLowerCase () : Converte todos os caracteres da String em minúsculas.

```
String palavra1 = "OLÁ";
```

```
String palavra3 = palavra1.toLowerCase(); // retorna "olá"
```

13. String toUpperCase () : Converte todos os caracteres da String em maiúsculas.

```
String palavra1 = "olá";
```

```
String palavra2 = palavra1.toUpperCase(); // retorna "OLÁ"
```

14. String trim () : Retorna a cópia da String, removendo os espaços em branco em ambas as extremidades. Não afeta os espaços em branco no meio.

```
String w1 = " Aprenda, Compartilhe Aprenda ";
```

```
String w2 = w1.trim(); // retorna "Aprender, Compartilhar Aprender"
```

15. Substituição de string (char oldChar, char newChar) : Retorna uma nova string substituindo todas as ocorrências de oldChar por newChar.

```
String s1 = "CursodeJBVB";
```

```
String s2 = "CursodeJBVB".replace ('B', 'A'); // retorna "CursodeJAVA"
```

Observação: - s1 ainda é CursodeJBVB e s2 é CursodeJAVA

Programa para ilustrar todos os métodos de string:

```
import java.io.*; import java.util.*; class Teste
{
    public static void main (String[] args)
    {
        String s = "CursodeJAVA";
        // ou String s = new String ("CursodeJAVA");
        // Retora o número de caracteres na String System.out.println("Tamanho da String = " +
s.length());
        // Returns the character at ith index. System.out.println("Caracter na 3ª posição = "
+ s.charAt(3));
        // Retorna a substring no índice 3 System.out.println("Substring = " + s.substring(3));
        // Retorna a substring do índice i até j-1 (2 a 5)
    }
}
```

Resultado :

```
System.out.println("Substring = " + s.substring(2,5));
```

```
// Concatena a string2 no fim da string1. String s1 = "JAVA";
```

```
String s2 = "Cursode"; System.out.println("String concatenada = " +  
s1.concat(s2));  
// Retorna o índice da ocorrência da palavra indicada String s4 = "Learn Share Learn";  
System.out.println("Índice da palavra Share = " +  
s4.indexOf("Share"));  
// Retorna o índice da letra a em sua primeira ocorrência System.out.println("Índice de a = " +  
s4.indexOf('a',3));  
// Checando a igualdade das Strings Boolean out = "JAVA".equals("java");  
System.out.println("Checando igualdade " + out); out = "JAVA".equals("JAVA");  
System.out.println("Checando igualdade " + out); out = "JAVA".equalsIgnoreCase("JAvA");  
System.out.println("Checando igualdade " + out);  
//Comparando 2 Strings  
int out1 = s1.compareTo(s2);  
System.out.println("A diferença na tabela ASCII vale = "+out1);  
// Convertendo maiúsculas e minúsculas String word1 = "JaVa";  
System.out.println("Minúsculas: " +  
word1.toLowerCase());  
// Convertendo maiúsculas e minúsculas String word2 = " JaVa ";  
System.out.println("Maiúsculas: " +  
word2.toUpperCase());  
// Trocando caracteres String str1 = "CursodeJBVB";  
System.out.println("String Original " + str1); String str2 = " CursodeJBVB ".replace('B', 'A');  
System.out.println("Trocando A por B -> " + str2);  
Tamanho da String = 11 Caracter na 3ª posição = s Substring = sodeJAVA Substring = rso  
String concatenada = JAVA Cursode Índice da palavra Share = 6 Índice de a = 8  
Checando igualdade false Checando igualdade true Checando igualdade false  
A diferença na tabela ASCII vale = 7 Minúsculas: java  
Maiúsculas: JAVA  
String Original CursodeJBVB Trocando A por B -> CursodeJAVA  
Existem muitos outros métodos utilizados com a classe String, porém são menos relevantes  
que os citados.
```



ATIVIDADE DE FIXAÇÃO

- 1- Em Java, qual é o objetivo principal das classes de strings? }
- a) Manipular objetos gráficos.
 - b) Gerenciar entradas do teclado.
 - c) Manipular e processar cadeias de caracteres.
 - d) Criar interfaces de usuário.

2- Qual característica das strings em Java as torna imutáveis?

- a) Elas não podem ser declaradas como variáveis.
- b) Uma vez criadas, seu conteúdo não pode ser alterado.
- c) Elas só podem ser utilizadas em loops.
- d) São apenas usadas para comentários em código.

3- Qual método é usado para determinar o comprimento de uma string em Java?

- a) length()
- b) size()
- c) count()
- d) total().

4- Qual operador é utilizado para concatenar strings em Java?

- a) +
- b) -
- c) *
- d) /

5- Qual método é usado para transformar uma string em letras maiúsculas em Java?

- a) toLowerCase()
- b) uppercase()
- c) toUpper()
- d) upperCase()

6- Como se compara o conteúdo de duas strings em Java?

- a) Utilizando o operador ==.
- b) Utilizando o método compare().
- c) Usando o método equals().
- d) Não é possível comparar strings em Java.

7- Qual método é usado para extrair uma parte específica de uma string em Java?

- a) substring()
- b) extract()
- c) section()
- d) split()

8- Qual classe Java é utilizada para criar objetos de strings?

- a) StringObject
- b) TextString
- c) String
- d) TextObject

9- Qual método é usado para encontrar a posição de uma subcadeia dentro de uma string em Java?

- a) find()
- b) locate()
- c) search()
- d) indexOf()

10- Em Java, o que é um "escape sequence" em uma string?

- a) Um método para copiar strings.
- b) Um caractere especial que representa um valor numérico.
- c) Um conjunto de caracteres que não podem ser usados em strings.
- d) Uma sequência de caracteres que representa um caractere especial, começando com uma barra invertida () .

TEMA 04

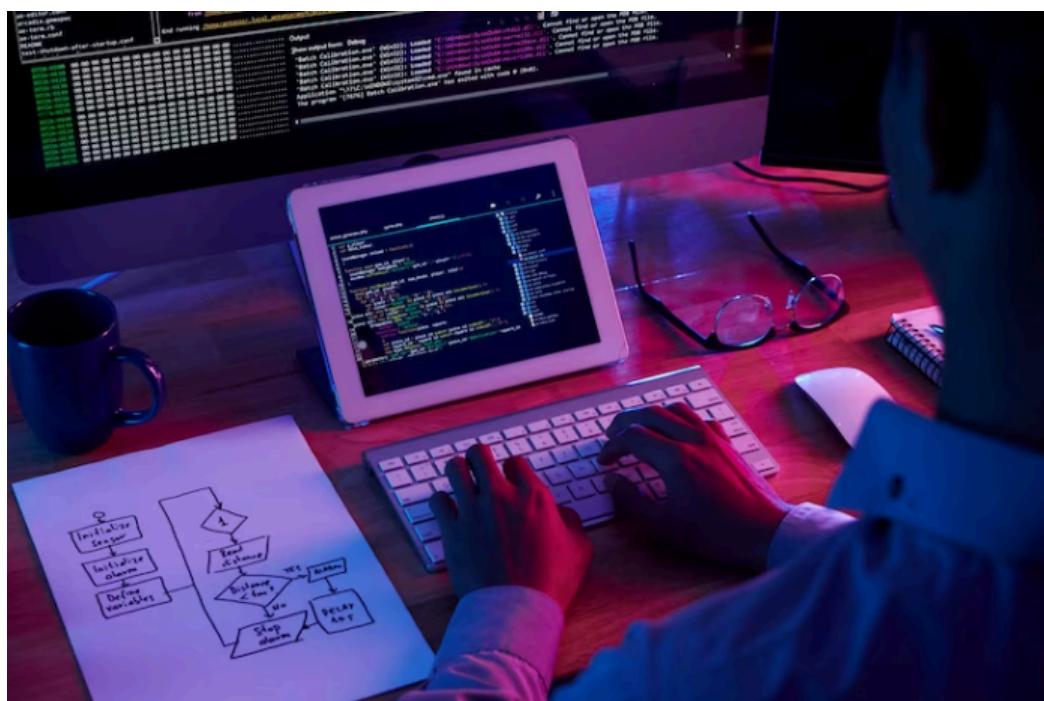
ESTRUTURAS DE CONTROLE - CONDICIONAIS

Habilidades

- Utilizar ambientes de desenvolvimento para desenvolvimento desktop
- Aplicar técnicas de orientação a objetos.

Introdução

As estruturas de controle condicionais são blocos essenciais na construção de programas, permitindo que a lógica tome diferentes caminhos com base em condições específicas. Elas desempenham um papel fundamental na tomada de decisões em programação, permitindo que os desenvolvedores criem software capaz de se adaptar a situações variáveis. Através do uso de estruturas condicionais, como "if-else" e "switch", os programadores podem criar algoritmos inteligentes e interativos, tornando o código mais flexível e eficiente ao responder de forma dinâmica às necessidades e aos dados fornecidos.



Disponível em: <<https://tinyurl.com/4kazbhah>>. Acesso em 11 set. 2023.

Condicional If

Essa é a estrutura de controle mais utilizada dentro da programação. A palavra if significa "se".

Sua utilização depende de uma condição, definida por uma expressão. O resultado desta expressão (racional, lógica ou de variável) será um valor booleano (Verdadeiro ou Falso). Esse valor pode ser também um literal true ou false.

Por exemplo:

```
if (true) { // o próprio literal "true" sendo utilizado
    // comandos
}
if (7 > 2) { // Expressão relacional
    // comandos
}
if (true || false) { // Expressão lógica
    // comandos
}
if (7 > 2 && 7 > 5) { // Expressão lógica e expressão relacional
    // comandos
}
if (VarBooleana) { //Utilizando variável
    // comandos
}
if (VarBooleana && OutraVarBooleana) { //Variável e expressão lógica
    // comandos
}
```

Condisional Switch Case

O switch é uma estrutura de decisão que, de acordo com determinado valor, executa um bloco de código específico. Ao contrário do if, ele lida com valores que não são booleanos.

O Switch permite executar uma entre várias opções de comandos. Assim, pode-se substituir vários ifs encadeados por um código simples para criação, entendimento e manutenção.

Por exemplo, um algoritmo que proponha a escolha de um número qualquer entre 1 e 3 :

```
System.out.println("Digite um número entre 1 e 3:");
int n = entrada.nextInt();
switch (n) {
    case 1:
        System.out.println("Você escolheu 1");
        break;
    case 2:
        System.out.println("Você escolheu 2");
        break;
    case 3:
        System.out.println("Você escolheu 3");
        break;
    default:
```

```
        System.out.println("Número inválido");
    }
```

Exemplos de exercícios em JAVA:

1. Escreva um programa que, com base em uma temperatura em graus celsius (15°), a converta e exiba em Kelvin (K), Réaumur (Re), Rankine (Ra) e Fahrenheit (F), seguindo as fórmulas:

$$F = C * 1.8 + 32; K = C + 273.15; Re = C * 0.8;$$

$$Ra = C * 1.8 + 32 + 459.67$$

```
public class Temperatura {
    public static void main(String[]
        args) { double C, K, F, Re, Ra;
        C = 15;
        F = C * 1.8 +
        32; K = C +
        273.15;
        Ra = C * 1.8 + 32 +
        459.67; Re = C * 0.8;
        System.out.println("A temperatura em Fahrenheit é: " + F);
        System.out.println("A temperatura em Kelvin é: " + K);
        System.out.println("A temperatura em Reaumur é: " + Ra);
        System.out.println("A temperatura em Rankine é: " + Re);
    }
}
```

Fonte: Elaborado pelo autor (2023).

2. Escreva um programa que entre com um número e o imprima caso seja maior do que 20.

```
import
java.util.Scanner;
public class
Maiorque20 {
    public static void main(String[] args)
    { Scanner input = new
        Scanner(System.in); int num;
        System.out.print("digite um número:
"); num = input.nextInt();
        if ( num >= 20 ){
            System.out.print( "numero " +num+ " é maior que 20");
        } else {
            if( num < 20 ){
                System.out.print( "numero " +num+ " é menor que 20");
            }
        }
    }
}
```

Fonte: Elaborado pelo autor (2023).

3. Entrar com dois números e imprimir o menor número (suponha números diferentes).

```
import
java.util.Scanner;
public class
MenorNumero{
    public static void main(String[] args)
    { Scanner input = new
    Scanner(System.in);
        int x, y;
        System.out.print("Digite um número:
"); x = input.nextInt();
        System.out.print("Digite um número diferente do primeiro:
"); y = input.nextInt();
        if ( x < y ) {
            System.out.printf("O número " +x+ " é o menor");
        } else {
            if ( x > y ){
                System.out.printf("O número " +y+ " é menor");
            }
        }
    }
}
```

Fonte: Elaborado pelo autor (2023).

4. Entrar com dois números e imprimi-los em ordem decrescente (suponha números diferentes).

```
import
java.util.Scanner;
public class
MenorNumero{
    public static void main(String[] args)
    { Scanner input = new
    Scanner(System.in);
        int x, y;
        System.out.print("Digite um número:
"); x = input.nextInt();
        System.out.print("Digite um número diferente do primeiro:
"); y = input.nextInt();
        if ( x > y ){
            System.out.println( x+ " é maior que " +y);
        }else{
            if ( x < y ) {
                System.out.print ( y+ " é maior que " +x);
            }
        }
    }
}
```

Fonte: Elaborado pelo autor (2023).

Exercício para testar as condições – Qual será o resultado?

```
public class Teste{  
    public static void main(String[] args) { int A,B,C,D,F,G;  
        A=4,B=3,C=9,D=9,F=9,G=7;  
        if(C==6){  
            F=A  
            *G;  
            F=A  
            +F;  
            G=B  
            +G;  
        }  
        System.out.println(D+F+A+C);  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

E este outro?

```
public class Teste{  
    public static void main(String[] args) { int A,B,C,D,F,G;  
        A=8,B=9,C=4,D=6,F=5,G=5;  
        if((!(C<6))&&(D!=5)){  
            G=C-G;  
            if(G<=  
                5){  
                G=F-A;  
            }else{  
                if((C!=7)|| (C<3  
                    )){ D=F-A;  
                }  
            }  
        }else{  
            if(C<=5){  
                G=D+F;  
            }  
            A=G-C;  
        }  
        System.out.println(B+F+G-C);  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

Teste na sua IDE Netbeans esses códigos e verifique o que acontece a cada etapa do código. Se necessário use alguns System.out.println ao longo do código, para verificar a saída de cada variável e seu comportamento a cada condição.



ATIVIDADE DE FIXAÇÃO

1- Qual é o propósito principal das estruturas de controle condicionais na programação?

- a) Gerenciar a entrada e saída de dados.
- b) Executar um bloco de código repetidamente.
- c) Realizar cálculos matemáticos complexos.
- d) Permitir que um programa tome decisões baseadas em condições.

2- O que é uma expressão lógica em relação às estruturas de controle condicionais?

- a) Um cálculo matemático complexo.
- b) Um nome de variável.
- c) Uma combinação de operadores que avalia como verdadeiro ou falso.
- d) Uma string de caracteres.

3- Qual é a diferença entre as estruturas "if" e "if-else" em programação?

- a) Não há diferença entre elas.
- b) "if" verifica múltiplas condições, enquanto "if-else" verifica apenas uma.
- c) "if" é utilizado para loops, e "if-else" para decisões únicas.
- d) "if" é usado para números, e "if-else" para strings.

4- Qual das seguintes opções é um operador de comparação em programação?

- a) @
- b) *
- c) &&
- d) ==

5- O que é um bloco de código em uma estrutura condicional?

- a) Um programa completo.
- b) Uma variável especial.
- c) Um conjunto de instruções agrupadas entre chaves { }.
- d) Uma expressão booleana.

6- Em uma estrutura "switch", qual é a função do bloco "default"?

- a) Indicar um caso específico.
- b) Executar código somente se não houver correspondência com nenhum caso.
- c) Encerrar a estrutura "switch".
- d) Realizar cálculos matemáticos.

7- Como você expressa uma comparação "se maior que" em uma estrutura condicional?

- a) >
- b) <
- c) >=
- d) <=

8- Qual é o resultado de uma expressão lógica que usa o operador "&&" (AND)?

- a) Verdadeiro se pelo menos um dos operandos for verdadeiro.
- b) Verdadeiro apenas se ambos os operandos forem verdadeiros.
- c) Falso se pelo menos um dos operandos for falso.
- d) Falso apenas se ambos os operandos forem falsos.

9- O que a estrutura "else if" permite fazer?

- a) Executar um bloco de código sem qualquer condição.
- b) Criar uma cópia de uma estrutura "if" anterior.
- c) Avaliar múltiplas condições em sequência.
- d) Ignorar uma estrutura "if" e passar para a próxima.

10- Qual das seguintes opções representa uma estrutura condicional aninhada?

- a) switch
- b) for
- c) if-else
- d) while

TEMA 05

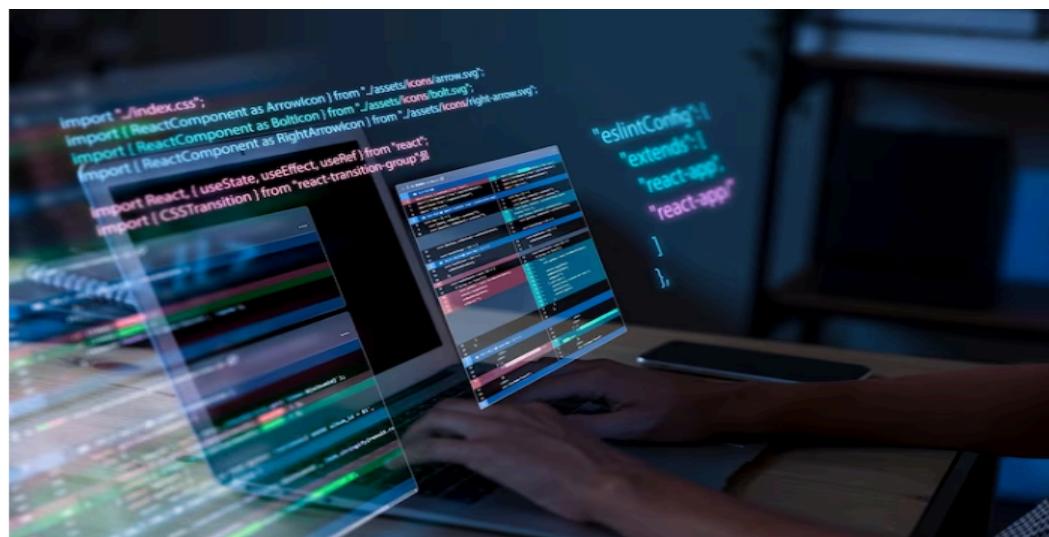
ESTRUTURAS DE CONTROLE – REPETIÇÃO

Habilidades

- Codificar programas orientados a objetos.

Introdução

As estruturas de controle de repetição são um componente fundamental na programação, permitindo que um conjunto de instruções seja executado repetidamente com base em uma condição específica. Essa capacidade de repetir ações é essencial para automatizar tarefas e processar grandes volumes de dados de maneira eficiente. Ao utilizar estruturas de repetição, como "for", "while" e "do-while", os programadores podem criar algoritmos que executam tarefas interativas, como contagem, busca, processamento de listas e muito mais. Isso não apenas economiza tempo, mas também contribui para a criação de programas mais dinâmicos e poderosos, capazes de lidar com desafios complexos e variáveis.



Disponível em: <[freepix-https://tinyurl.com/2p8dmkeh](https://tinyurl.com/2p8dmkeh)>. Acesso em 11 set. 2023.

Estruturas de Repetição

As estruturas de repetição, também conhecidas como laços (loops), são utilizadas para executar, repetidamente, um comando ou bloco de comandos atendendo a determinada condição.

Elas se classificam em dois tipos:

- Laços Contados: Sabe-se (ou o usuário sabe) exatamente quantas vezes o comando no interior da construção será executado;

Exemplo: Comando for.

- Laços Condicionais: Inicialmente não se sabe o número de vezes que o conjunto de comandos no interior do laço será repetido. Dependerá de uma condição sujeita à modificação pelas instruções do interior do laço.

'Exemplo: Comandos while e do while.

Laço Contado (for)

A estrutura de um laço for:

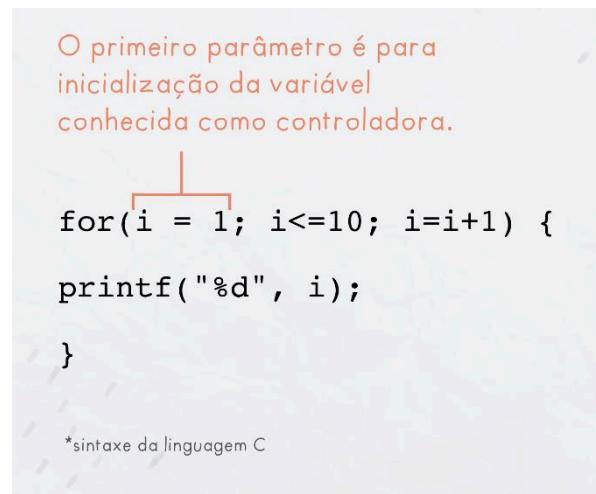
```
for ( ; ; ) {  
    // Bloco do laço for  
}
```

Entre os parênteses do for, há 3 posições separadas por ponto e vírgula.

A primeira posição é utilizada para inclusão de uma expressão para iniciar nossas iterações.

Por exemplo:

```
int i = 0;
```



Disponível em: <[medium.com-https://tinyurl.com/eappnena](https://tinyurl.com/eappnena)>. Acesso em 11 set. 2023.

Na segunda posição, coloca-se uma expressão que precisa retornar um valor booleano. É a posição onde existe a relação entre o número de iterações e a quantidade de iterações desejadas.

Por exemplo:

```
i < 5;
```

O segundo parâmetro é para a condição, que será testada em todo início de ciclo.

```
for(i = 1; i<=10; i=i+1) {  
    printf("%d", i);  
}
```

*sintaxe da linguagem C

Disponível em: <[medium.com-https://tinyurl.com/eappnena](https://tinyurl.com/eappnena)>. Acesso em 11 set. 2023.

Neste exemplo, enquanto a variável i for menor do que 5, a estrutura for irá repetir e, quando i chegar ao valor 5, as iterações irão parar.

A terceira posição do laço terá a expressão de incremento (de quanto em quanto o i irá ser incrementado). É chamada “expressão de iteração”. Em geral, a variável i aumenta de uma em uma unidade.

Por exemplo:

i = i + 1;

O terceiro parâmetro é para a incremento, que modificará a variável controladora.

```
for(i = 1; i<=10; i=i+1) {  
    printf("%d", i);  
}
```

*sintaxe da linguagem C

Disponível em: <[medium.com-https://tinyurl.com/eappnena](https://tinyurl.com/eappnena)>. Acesso em 11 set. 2023.

Exemplo da estrutura de repetição completa:

```
for (int i = 0; i < 5; i++) {  
    System.out.println("Acompanhe a repetição, em que o i = " + i);  
}
```

Aparecerá na tela:

Acompanhe a repetição, em que o i = 0
Acompanhe a repetição, em que o i = 1
Acompanhe a

repetição, em que o i = 2 Acompanhe a repetição, em que o i = 3 Acompanhe a repetição, em que o i = 4

Exercícios:

1. Escreva um programa em JAVA que receba o nome de um aluno com suas respectivas 2 notas, em seguida calcular a média do aluno e apresentar ao final a média calculada e a situação de Aprovação do aluno. (aprovado com média ≥ 6).

Observações:

- A turma tem 30 alunos.
- Calcular e mostrar a média geral da turma

```
import java.util.*; public class MediaTurma{  
    public static void main(String[] args) { Scanner teclado = new Scanner(System.in); String  
aluno;  
    float nota1, nota2, media, soma, mediaTurma; soma = 0;  
    for(int i=0; i<30; i++){ System.out.print("Nome do Aluno: "); aluno = teclado.nextLine();  
    System.out.print("Nota 1: "); nota1 = teclado.nextFloat();  
    System.out.print("Nota 2: "); nota2 = teclado.nextFloat(); teclado.nextLine();  
    media = (nota1+nota2)/2; soma = soma + media;  
    System.out.printf("Média do aluno é %.1f\n", media);  
    if (media >= 6)  
        System.out.println("Aluno Aprovado. Parabéns."); else  
        System.out.println("Em recuperação! Estude bastante.");  
    }  
    mediaTurma = soma/30;  
    System.out.printf("Média da Turma = %.1f\n", mediaTurma);  
    }  
}
```

Laço Condisional (while)

A estrutura de um laço while é definida por uma condição. Esta condição é uma expressão que gera um valor booleano (true ou false).

Por exemplo:

```
while () {  
    // Bloco do while  
}
```

Entre os parênteses estará a condição de parada ou continuidade da estrutura de repetição while.

Por exemplo:

```
int iteracao = 0; while (iteracao < 5) {  
    System.out.println("Acompanhe a repetição, em que o i = " + iteracao); iteracao = iteracao +
```

```
1;
```

```
}
```

Neste exemplo, são impressas as iterações de 0 até 5, da mesma forma feita com o for, porém o incremento não está mais no escopo da estrutura, mas conta no bloco de comandos.

Exemplo de uma condição de parada em que o usuário decide quando quer parar:

```
Scanner scanner = new Scanner(System.in); int numero, op =1;  
while (op == 1) {  
    System.out.print("Digite um número: "); numero = scanner.nextInt();  
    System.out.print("Digite 1 para continuar ou qualquer número para sair: ");  
    op = scanner.nextInt();  
}
```

Quando decidir se deve usar o laço “for” ou o laço “while”?

O for é ideal quando é sabido o número de iterações. Já o while repete dependendo da condição que o define.

Interessante destacar que ambas as estruturas de repetição podem executar da mesma forma. A estrutura da iteração deverá ser adaptada, mas é possível fazer as mesmas coisas.

Em resumo:

- O número de iterações é conhecido? Use o for;
- O usuário ou a condição decidem quando deve parar a repetição? Use o while.



ATIVIDADE DE FIXAÇÃO

1- Qual é o objetivo principal das estruturas de controle de repetição em programação?

- A) Organizar os dados de um programa.
- B) Realizar operações matemáticas complexas.
- C) Executar um conjunto de instruções repetidamente até que uma condição seja atendida.
- D) Fornecer opções de personalização de interface.

2- Qual estrutura de controle de repetição é mais adequada quando você sabe exatamente quantas vezes deseja executar um bloco de código?

- A) while loop
- B) do-while loop
- C) for loop
- D) if statement

3- Em qual estrutura de repetição o bloco de código é executado pelo menos uma vez, independentemente da condição?

- A) while loop
- B) do-while loop
- C) for loop
- D) if statement

4- O que a instrução "break" faz em uma estrutura de repetição?

- A) Encerra a execução do programa.
- B) Pula para o próximo ciclo da repetição.
- C) Encerra imediatamente o loop.
- D) Imprime uma mensagem de erro.

5- Qual é a principal diferença entre o "while loop" e o "do-while loop"?

- A) O "while loop" não é uma estrutura de repetição.
- B) O "while loop" executa o bloco de código pelo menos uma vez.
- C) O "do-while loop" sempre executa o bloco de código pelo menos uma vez.
- D) O "do-while loop" não existe em programação.

6- Como você pode evitar um loop infinito?

- A) Usando a instrução "continue".
- B) Não usando estruturas de repetição.
- C) Certificando-se de que a condição de saída do loop será eventualmente atendida.
- D) Fechando o programa abruptamente.

7- Qual é a finalidade da declaração "continue" em uma estrutura de repetição?

- A) Parar a execução do programa.
- B) Ignorar o restante do bloco de código e passar para a próxima iteração do loop.
- C) Voltar ao início do loop.
- D) Pular para o final do loop.

8- Qual é a função da variável de controle em um "for loop"?

- A) Armazenar o resultado final do loop.
- B) Determinar o número de iterações do loop.
- C) Armazenar o valor de saída do loop.
- D) Definir as condições de entrada do loop.

9- Em um "for loop", qual é a ordem das partes no cabeçalho da estrutura?

- A) Condição, inicialização, iteração.
- B) Inicialização, iteração, condição.
- C) Condição, iteração, inicialização.
- D) Iteração, inicialização, condição.

10- Qual das seguintes estruturas de repetição é mais apropriada quando você não sabe quantas vezes deseja executar um bloco de código?



- A) while loop
- B) do-while loop
- C) for loop
- D) if statement

TEMA 06

VETORES E MATRIZES EM JAVA

Habilidades

- Codificar programas orientados a objetos.
- Aplicar técnicas de orientação a objetos.



Disponível em: <[rawpixel-https://tinyurl.com/yrhpkbah](https://tinyurl.com/yrhpkbah)>. Acesso em 11 set. 2023.

Introdução

Vetores e matrizes são conceitos fundamentais na programação em Java, proporcionando uma maneira eficaz de armazenar e organizar dados de forma estruturada. Um vetor é uma coleção unidimensional de elementos do mesmo tipo, onde cada elemento é acessado através de um índice. Isso permite a criação de listas de dados facilmente manipuláveis, tornando-se essenciais para lidar com conjuntos de informações repetitivas ou sequenciais.

Por outro lado, as matrizes expandem essa ideia para duas ou mais dimensões, criando uma estrutura tabular de elementos. Em essência, uma matriz pode ser vista como uma coleção de vetores, onde cada linha ou coluna representa um vetor individual. Isso é especialmente útil para representar dados em formato de grade, como tabelas, mapas ou imagens, permitindo uma organização sistemática e acesso eficiente aos elementos.

Em Java, a utilização de vetores e matrizes envolve declaração, alocação de memória e manipulação dos elementos armazenados. Através dessas estruturas, é possível simplificar tarefas

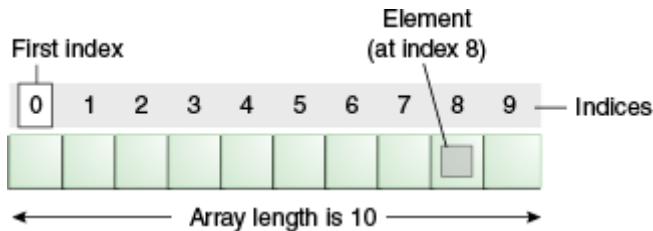
complexas, como cálculos estatísticos, processamento de imagens e resolução de problemas algorítmicos. Dominar a manipulação de vetores e matrizes é crucial para desenvolver aplicações eficientes e flexíveis, tornando-se uma habilidade essencial para programadores que desejam criar soluções robustas e bem estruturadas em Java.

Normalmente, uma matriz é uma coleção de tipos semelhantes de elementos que possuem localização de memória contígua. Em Java, um vetor ou array, é um objeto que contém elementos de um tipo de dados semelhante. Além disso, os elementos de um vetor são armazenados em um local de memória contíguo.

É uma estrutura de dados onde armazenamos elementos semelhantes. Podemos armazenar apenas um conjunto fixo de elementos em um array Java, que é baseado em índice, sendo o primeiro elemento do array armazenado no 0º índice, o segundo elemento é armazenado no 1º índice e assim por diante.

Ao contrário de C / C ++, podemos obter o comprimento da matriz usando o comprimento da estrutura (length). Em C / C ++, precisamos usar o operador sizeof.

Em Java, array é um objeto de uma classe gerada dinamicamente. Podemos armazenar valores ou objetos primitivos em um array em Java. Assim como em C / C ++, também podemos criar matrizes unidimensionais ou multidimensionais em Java.



Disponível em: <[stackoverflow-https://tinyurl.com/223m75eu](https://tinyurl.com/223m75eu)>. Acesso em 11 set. 2023.

Vantagens

- Otimização de código: Torna o código otimizado, podemos recuperar ou classificar os dados de forma eficiente.
- Acesso aleatório: podemos obter quaisquer dados localizados em uma posição de índice.

Desvantagens

- Limite de tamanho: podemos armazenar apenas o tamanho fixo dos elementos na matriz. Ele não aumenta seu tamanho em tempo de execução. Para resolver este problema, a estrutura de coleção é usada em Java, que cresce automaticamente.

Tipos de array em java

Existem dois tipos de array:

- Matriz unidimensional
- Matriz Multidimensional

Matriz unidimensional em Java

Sintaxe para declarar um array em Java

tipo[] arr; (ou)

tipo []arr; (ou) tipo arr[];

Instanciação de um Array em Java

arrayVar = new tipo[tamanho];

Programa para demonstrar como declarar, instanciar, inicializar e percorrer o array Java.

```
class TesteArray {
```

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

```
        int a [] = new int [5]; // declaração e instanciação
```

```
                ; // inicialização
```

```
        0]     0
```

```
            ;
```

```
        1]     0
```

```
            ;
```

```
        2]     0
```

```
            ;
```

```
        3]     0
```

```
            ;
```

```
        4]     0
```

```
            ;
```

```
// percorrendo a matriz
```

```
for (int i=0; i<a.length; i ++) // comprimento é a propriedade da matriz System.out.println
```

```
(a[i]);
```

```
}
```

```
}
```

Resultado:

10

20

70

40

50

Repetição For-Each para Matriz em JAVA

Também podemos imprimir o array Java usando for-each loop . O loop for-each do Java imprime os elementos do array um por um.

Ele mantém um elemento da matriz em uma variável e, em seguida, executa o corpo do loop.
A sintaxe da repetição for-each é:

```
for (tipo variável: array) {  
    // comandos  
}  
  
Programa Java para imprimir os elementos do array usando a repetição for-each  
class TesteArray1 {  
    public static void main (String args []) { int arr[] = { 33 , 3 , 4 , 5 };  
        // imprimindo array usando for-each loop for (int i: arr)  
        System.out.println (i);  
    }  
}  
  
Resultado:  
33  
3  
4  
5
```

Passando Array para um Método em Java

Podemos passar o array java para o método para que possamos reutilizar a mesma lógica em qualquer array.

Programa Java para demonstrar a maneira de passar um array para o método.

```
class TesteArray2{  
    // criando um método que recebe um array como parâmetro static void min(int arr[]){  
    int min = arr[0];  
    for(int i=1;i<arr.length;i++) if(min>arr[i])  
        min=arr[i];  
    System.out.println(min);  
}  
  
public static void main(String args[]){  
    int a[]={33,3,4,5};//declarando e inicializando um array min(a);// passando array para método  
}
```

Resultado:

```
3
```

Matriz multidimensional em Java

Nesse caso, os dados são armazenados em um índice baseado em linha e coluna (também conhecido como formato de matriz).

Sintaxe para declarar array multidimensional em Java:

tipo[][] array; (ou)
tipo [][]arrayr; (ou)
tipo array[][]; (ou)
tipo []array[];

Exemplo para instanciar Array Bidimensional em Java

```
int [][]arr = new int[3][3]; // 3 linhas e 3 colunas
```

Em seguida o programa mostra como declarar, instanciar, inicializar e imprimir a array Bidimensional.

Programa Java para ilustrar o uso de array multidimensional

```
class Testarray3 {  
    public static void main (String args []) {  
        // declarando e inicializando a array  
        int arr [] [] = {{ 1 , 2 , 3 }, { 2 , 4 , 5 }, { 4 , 4 , 5 }};  
        // imprimindo matriz 2D  
        for (int i = 0 ; i < 3 ; i ++) { for (int j = 0 ; j < 3 ; j ++) {  
            System.out.print (arr[i][j] + "");  
        }  
        System.out.println ();  
    }  
}
```

Resultado:

```
1 2 3  
2 4 5  
4 4 5
```



ATIVIDADE DE FIXAÇÃO

1- Qual é a principal característica de um vetor em Java?

- a) Armazenar elementos de diferentes tipos.
- b) Armazenar elementos de diferentes tamanhos.
- c) Armazenar elementos do mesmo tipo em sequência.
- d) Armazenar elementos em ordem aleatória.

2- Em Java, como é acessado um elemento específico de um vetor?

- a) Através de uma chave.
- b) Através de um índice.
- c) Através de um nome descritivo.
- d) Através de um ponteiro.

3- Qual é a diferença fundamental entre um vetor e uma matriz em Java?

- a) Um vetor armazena elementos de tipos primitivos, enquanto uma matriz armazena objetos.
- b) Um vetor é unidimensional, enquanto uma matriz é bidimensional ou multidimensional.
- c) Um vetor pode armazenar mais elementos do que uma matriz.
- d) Um vetor tem um tamanho fixo, enquanto uma matriz é dinamicamente ajustável.

4- Como se declara um vetor de inteiros em Java com tamanho 5?

- a) int[] vetor = new int[5];
- b) vetor<int> = int[5];
- c) int vetor = {5};
- d) vetor[5] = new int[];

5- Qual método é usado para obter o tamanho de um vetor em Java?

- a) length()
- b) size()
- c) count()
- d) getLength()

6- Qual é a sintaxe correta para acessar o elemento de índice 2 em um vetor chamado "numeros"?

- a) numeros.get(2)
- b) numeros[2]
- c) numeros.at(2)
- d) numeros.getElement(2)

7- Qual é a dimensão de uma matriz 3x3 em Java?

- a) 1x3
- b) 2x3
- c) 3x1
- d) 3x3

8- Como se declara uma matriz 2D de inteiros em Java com 3 linhas e 4 colunas?

- a) int[][] matriz = new int[3, 4];
- b) int[][] matriz = new int[3][4];
- c) int[3][4] matriz = new int[][];
- d) matriz<int>[3][4] = new int[][];

9- Qual método é usado para acessar um elemento específico de uma matriz em Java?

- a) get()
- b) at()
- c) getElement()
- d) [linha][coluna]

10- Qual é a vantagem de utilizar vetores e matrizes em Java na resolução de problemas?

- a) Acesso rápido a elementos individuais.
- b) Menos consumo de memória.
- c) Maior flexibilidade para armazenar tipos diferentes.
- d) Melhor desempenho em operações de entrada e saída.

TEMA 07

POO – CONCEITOS INICIAIS

Habilidades

- Aplicar técnicas de orientação a objetos.
- Codificar programas orientados a objetos.



Disponível em: <<https://www.dio.me/articles/programacao-orientada-a-objeto>>. Acesso em 11 set. 2023.



Acesse o QR Code do vídeo Vetores - Fonte do vídeo: <https://youtu.be/fubUTQ_PL2k>

Introdução

A Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia na representação de objetos, que são unidades individuais que combinam dados (atributos) e comportamentos (métodos). Essa abordagem visa modelar problemas do mundo real de maneira mais próxima, permitindo a criação de estruturas mais organizadas e reutilizáveis. No âmbito da POO, os conceitos iniciais incluem classes e objetos, onde as classes são as definições dos tipos de objetos que podem ser criados, e os objetos são instâncias dessas classes, capazes de interagir entre si por meio de mensagens, promovendo a modularidade, encapsulamento e extensibilidade do código.

Em POO, as classes servem como moldes para a criação de objetos, encapsulando

características e comportamentos relacionados. Os objetos são as instâncias dessas classes, capazes de armazenar informações em seus atributos e executar operações específicas por meio de seus métodos. A POO oferece uma maneira de organizar e estruturar o código de forma mais intuitiva e próxima da realidade, facilitando a manutenção e o desenvolvimento de sistemas complexos.

Programação Orientada a Objetos (POO)

Programação Orientada a Objetos, ou POO, é um conceito popularmente conhecido e amplamente usado em linguagens de programação modernas, como Java, e trabalha com os princípios de abstração, encapsulamento, herança e polimorfismo. Ele permite que os usuários criem objetos que desejem e criem métodos para lidar com esses objetos.

O conceito básico de POO é criar objetos, reutilizá-los em todo o programa e manipular esses objetos para obter resultados.

Os principais pilares do POO são:



Fonte: Elaborado pelo autor (2023).

- Abstração: significa usar coisas simples para representar a complexidade. Todos nós sabemos como ligar a TV, mas não precisamos saber como funciona para aproveitá-la. Em Java, abstração significa coisas simples como objetos, classes e variáveis que representam códigos e dados

subjacentes mais complexos. Isso é importante porque permite evitar a repetição do mesmo trabalho várias vezes.

- Encapsulamento: é a prática de manter os campos dentro de uma classe privada e, em seguida, fornecer acesso a eles por meio de métodos públicos. É uma barreira protetora que mantém os dados e o código seguros dentro da própria classe. Dessa forma, podemos reutilizar objetos como componentes de código ou variáveis sem permitir acesso aberto aos dados de todo o sistema.
- Herança: é um recurso especial da Programação Orientada a Objetos em Java. Ele permite que os programadores criem novas classes que compartilham alguns dos atributos das classes existentes. Isso nos permite desenvolver o trabalho anterior sem reinventar a roda.
- Polimorfismo: conceito Java POO permite que os programadores usem a mesma palavra para significar coisas diferentes em contextos diferentes. Uma forma de polimorfismo em Java é a sobrecarga de método. É quando significados diferentes estão implícitos no próprio código. A outra forma é a substituição de métodos. É quando os diferentes significados estão implícitos nos valores das variáveis fornecidas. Veja mais sobre isso abaixo.

Todos estes conceitos funcionam permitindo que os programadores criem componentes que podem ser reutilizados de diferentes maneiras, mas ainda mantêm a segurança. Eles são aprendidos a medida em que se evoluí no conhecimento de JAVA. Neste curso aprenderemos o Encapsulamento e Abstração.

Como funcionam estes conceitos?

- Como funciona a abstração?

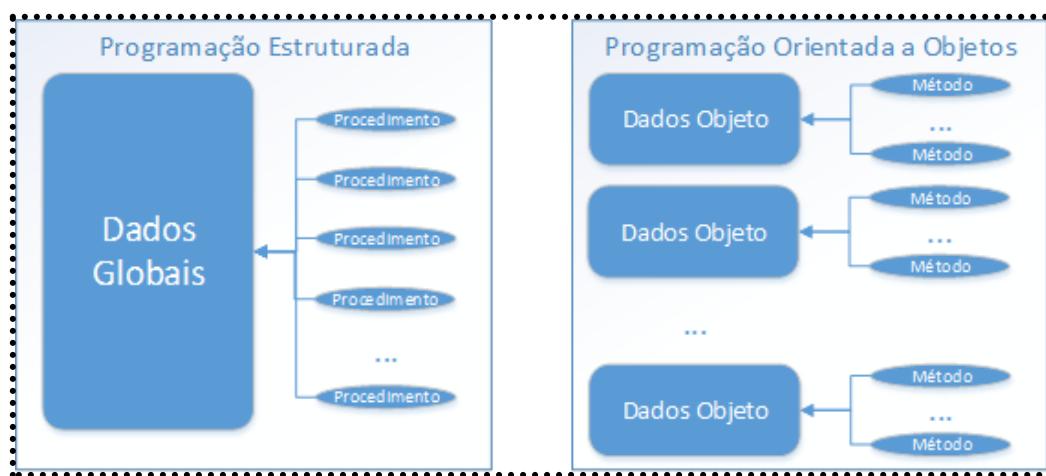
Abstração permite que os programadores criem ferramentas úteis e reutilizáveis. Por exemplo, um programador pode criar vários tipos diferentes de objetos. Podem ser variáveis, funções ou estruturas de dados. Os programadores também podem criar diferentes classes de objetos. Estas são formas de definir os objetos.

Por exemplo, uma classe de variável pode ser um endereço. A classe pode especificar que cada objeto de endereço deve ter um nome, rua, cidade e código postal. Os objetos, neste caso, podem ser endereços de funcionários, endereços de clientes ou endereços de fornecedores.

- Como funciona o encapsulamento?

O encapsulamento nos permite reutilizar a funcionalidade sem comprometer a segurança. É um conceito POO poderoso em Java porque nos ajuda a economizar muito tempo. Por exemplo, podemos criar um pedaço de código que chama dados específicos de um banco de dados. Pode ser útil reutilizar esse código com outros bancos de dados ou processos. O encapsulamento nos permite fazer isso enquanto mantemos nossos dados originais privados. Também nos permite alterar nosso código original sem quebrá-lo para outras pessoas que o adotaram nesse ínterim.

A POO, em comparação com o paradigma estruturado de programação, se diferencia de forma que o paradigma estruturado temos procedimentos (ou funções) que são aplicados globalmente em nossa aplicação. Já na POO há métodos aplicados aos dados de cada objeto. Fundamentalmente, os procedimentos e métodos são iguais, sendo diferenciados apenas pelo seu escopo.



Disponível em: <[devmedia-https://tinyurl.com/383rb833](https://tinyurl.com/383rb833)>. Acesso em 11 set. 2023.

A construção da programação POO é executada através de Classes e objetos, que são instanciados, definidos com características, comportamentos e estados.

Na figura em seguida, percebe-se cada definição dessas características e diferenças entre Classes, Objetos, a instância e seus atributos.

Classe	As classes de programação são modelos de objetos, as quais contêm suas características e seus comportamentos, permitindo, assim, armazenar atributos e métodos dentro dela.
Atributos	São as características de um objeto; essas características também são conhecidas como variáveis, mas na OO, são chamadas de atributos.
Métodos	São as ações que os objetos podem exercer quando solicitados, podem interagir e se comunicar com outros objetos.
Instância	Uma instância de uma classe é um novo objeto criado dessa classe. Instanciar uma classe é criar um novo objeto do mesmo tipo dessa classe.

Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.

Objeto

Objeto significa uma entidade do mundo real, como uma caneta, cadeira, mesa, computador, relógio, etc. A Programação Orientada a Objetos é uma metodologia ou paradigma para projetar um programa usando classes e objetos. É uma coisa Natural ou abstrata, percebida pelos sentidos e descrita por suas características, comportamento e estado atual.

Um objeto é um conceito, uma abstração ou uma coisa com identidade que possui significado para uma aplicação, que normalmente parecem como nomes próprios ou referências específicas nas descrições de problemas e discussões com os usuários e clientes durante o processo de desenvolvimento de sistemas de software. Todos os objetos possuem identidade e são distinguíveis.

Um objeto pode ser definido como uma instância de uma classe. Um objeto contém um endereço e ocupa algum espaço na memória.

Os objetos podem se comunicar sem conhecer os detalhes dos dados ou código uns dos outros. A única coisa necessária é o tipo de mensagem aceita e o tipo de resposta retornada pelos objetos.

Exemplo de objetos concretos e abstratos: Smartphone (concreto)

- Características (modelo, marca, capacidade);
- Comportamento (ligar, desligar, clicar, rolarTela);
- Estado (ligado, desligado, travado); Compromisso (abstrato)
- Características (horário, comQuem, local);
- Comportamento (Marcar, desmarcar, realocar);
- Estado (marcado, cancelado)

Nas figuras a seguir, você pode notar diferentes tipos de motocicletas que podem ser utilizadas para diferentes propósitos. Algumas pessoas utilizam para trabalho, outras utilizam para passeio. Mas todas são motocicletas e têm propriedades em comum.

Moto de Passeio



Disponível em: <jcomp-<https://tinyurl.com/f8v68md6>>. Acesso em 11 set. 2023.

Moto de trabalho



Disponível em: <senivpetro-<https://tinyurl.com/yc2s6fpt>>. Acesso em 11 set. 2023.

Note que os diferentes tipos de motos têm diferentes características e propriedades e que são distinguíveis entre si.

OBJETO - Definição em exemplo: MOTO

Coisas que tenho (TEM)	modelo, marca, cor, nroPlaca, nroChassi	ATRIBUTOS
Coisas que faço (FAZ)	licenciar, checaChassi	MÉTODOS
Como estou agora (ESTADO ATUAL)	nova, quebrada, velha, turbinada	STATUS

Este exemplo nos leva às definições que envolvem a construção dos objetos, como suas características, comportamento e estado.

Classe

Uma classe é um projeto ou protótipo definido pelo usuário a partir do qual os objetos são criados. Ele representa o conjunto de propriedades ou métodos comuns a todos os objetos de um tipo. A classe é o “Molde”, aquele que se encaixa em todos os objetos;

Por exemplo, todas as motos das figuras anteriores pertencem à classe Motocicleta e têm as mesmas propriedades, que são: modelo, marca, número do chassi, placa, cor, número de cilindradas.

Essas são propriedades relevantes que descrevem as propriedades em comum dessas diferentes motocicletas.

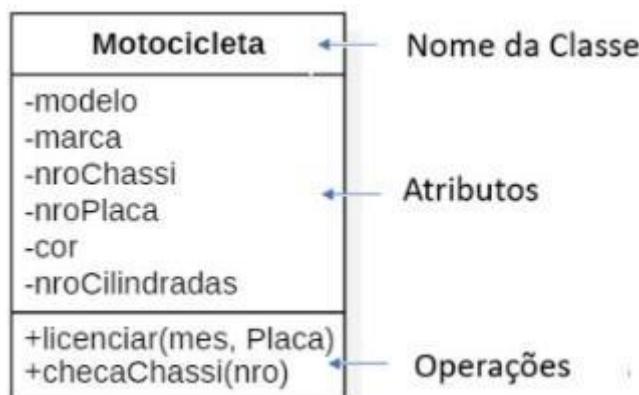
Todos esses diferentes objetos Motocicletas pertencem à mesma Classe Motocicleta, que poderia ser representada como é apresentada na figura a seguir por meio da notação de classe e da notação de objeto em UML:



Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.

Os objetos da classe Motocicleta possuem características em comum, como:

- modelo,
- marca,
- nroChassi,
- nroPlaca,
- Cor
- nroCilindradas.



Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.



Você pode perceber que os objetos da mesma classe também têm as mesmas operações que são: licenciar e checarChassi.

Veja a figura a seguir.

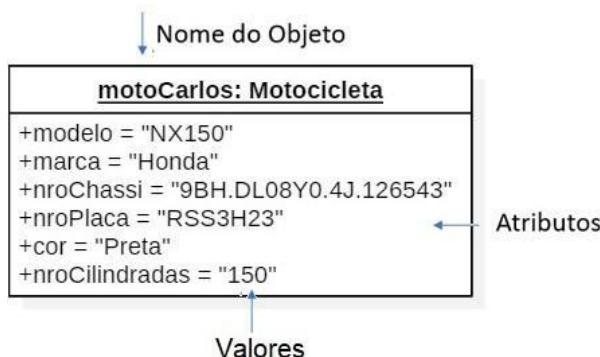
<u>motoCarlos: Motocicleta</u>	<u>motoMaria: Motocicleta</u>
-modelo = "NX150" -marca = "Honda" -nroChassi = "9BH.DL08Y0.4J.126543" -nroPlaca = "RSS3H23" -cor = "Preta" -nroCilindradas = "150"	-modelo = "CG125" -marca = "Honda" -nroChassi = "9BG.RD08X0.4G.117974" -nroPlaca = "ABC1D23" -cor = "Vermelha" -nroCilindradas = "125"

Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.

A representação de dois objetos diferentes que pertencem à mesma classe Motocicletas:

- a moto de Carlos é uma Motocicleta de modelo NX 150 da marca Honda;
- a moto de Maria é uma CG 125 também da marca Honda, mas a Motocicleta da Maria é da cor vermelha, e a motocicleta de Carlos é da cor preta.

A representação dos objetos é definida da seguinte forma:



Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.

Atributo

Atributo é um dado (informação de estado) para o qual cada Objeto em uma Classe tem seu próprio valor.

Valor

Um valor é um elemento dos dados. Você pode achar valores por meio do exame da documentação do problema.

Analogia: objeto está para classe assim como valor está para atributo.

Declarando objetos (instanciando objetos de uma classe)

Quando um objeto de uma classe é criado, diz-se que a classe está **instanciada**.

Todas as instâncias compartilham os atributos e o comportamento da classe. Mas os valores desses atributos, ou seja, os estados são únicos para cada objeto.

Uma única classe pode ter qualquer número de instâncias, sendo devidamente diferenciadas por seus nomes, conforme declaramos as variáveis. Isso notifica o compilador de que usaremos o nome para referir aos dados da Classe a que esse objeto pertence (o seu tipo).

Assim como uma variável primitiva, a declaração na instância da Classe também reserva uma quantidade adequada de memória para a variável.

Exemplo:

```
Motocicleta m1;//declaração da variável do tipo Motocicleta  
Motocicleta m2;
```

Método

Um método na programação orientada a objetos é um procedimento associado a uma classe. Um método define o comportamento dos objetos que são criados a partir da classe.

Outra maneira de dizer isso é que um método é uma ação que um objeto é capaz de realizar. A associação entre método e classe é chamada de ligação.

Os métodos determinam o comportamento dos objetos de uma classe e são análogos às funções ou procedimentos da programação estruturada.

Todo método tem como característica própria o uso de parênteses para parâmetros que tenha ou não, mas sempre terá parênteses sempre em sua estrutura.

Vamos criar nossa primeira Classe JAVA:

```
public class Moto {  
    String modelo, marca, nroChassi, nroPlaca, cor;  
    boolean licenciada;  
    int nroCilindradas;  
  
    void licenciar(){  
        licenciada = true;  
    }  
    void verificaLicenca(){  
        //condição para verificar  
    }  
}
```

Atributos

Métodos

Disponível em: <<https://youtu.be/aBagKMTkWv8>>. Acesso em 11 set. 2023.

Agora vamos instanciar nossos objetos da Classe:

```
public class ClasseMoto {  
    public static void main(String[] args) {  
        Moto m1 = new Moto();  
        m1.cor = "Azul";  
        m1.marca = "Honda";  
        m1.modelo = "Twister";  
        m1.nroChassi ="909090333333111";  
        m1.nroCilindradas = 250;  
        m1.nroPlaca = "ABC12C4";  
  
        m1.licenciar();  
        m1.status();  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

Note que as características criadas na classe Moto() são definidas nas instâncias, atribuindo valores a elas. Já os métodos são invocados.



ATIVIDADE DE FIXAÇÃO

1- O que é a Programação Orientada a Objetos (POO)?

- a) Um paradigma de programação baseado em estruturas de controle condicionais.
- b) Um padrão de design gráfico para interfaces de usuário.
- c) Um estilo de codificação que se concentra apenas em loops e iterações.
- d) Um paradigma de programação que se baseia na criação de objetos que combinam dados e comportamentos.

2- Qual é a principal unidade de construção em POO?

- a) Funções
- b) Variáveis
- c) Laços de repetição
- d) Objetos

3- O que são classes em POO?

- a) Variáveis que armazenam valores.
- b) Estruturas de controle usadas para tomar decisões.
- c) Tipos de dados primitivos.
- d) Definições de tipos de objetos, incluindo atributos e métodos.

4- Qual é o objetivo do encapsulamento em POO?

- a) Criar múltiplas instâncias de uma classe.
- b) Expor todos os atributos de uma classe.
- c) Esconder detalhes internos da implementação e fornecer uma interface controlada.
- d) Definir atributos e métodos estáticos em uma classe.

5- O que são objetos em POO?

- a) Variáveis que armazenam números inteiros.
- b) Instâncias de classes, com atributos e métodos específicos.
- c) Funções usadas para realizar cálculos complexos.
- d) Elementos de estruturas de dados lineares.

6- Qual é a relação entre classes e objetos em POO?

- a) As classes são instâncias dos objetos.
- b) Os objetos são instâncias das classes.
- c) Classes e objetos são termos usados de forma intercambiável.
- d) Classes e objetos não têm relação entre si.

7- O que são atributos em um objeto POO?

- a) Métodos que realizam operações em objetos.
- b) Variáveis que armazenam valores específicos para um objeto.
- c) Estruturas de controle usadas para iterações.
- d) Funções que desempenham ações específicas.

8- O que são métodos em POO?

- a) Variáveis que armazenam valores específicos para um objeto.
- b) Operações que podem ser realizadas em objetos, representadas como funções.
- c) Atributos que armazenam informações sobre objetos.
- d) Estruturas de controle usadas para tomada de decisões.

9- O que é herança em POO?

- a) Um conceito que envolve criar novos objetos.
- b) A capacidade de compartilhar atributos e comportamentos entre classes.
- c) Um termo usado para criar cópias de objetos existentes.
- d) Uma técnica para limitar a visibilidade dos métodos em uma classe.

10. Qual é a importância da POO no desenvolvimento de software?

- a) Não tem impacto no desenvolvimento de software.
- b) Simplifica a sintaxe da linguagem de programação.
- c) Torna o código mais difícil de manter.
- d) Organiza o código de forma modular, promovendo reutilização, extensibilidade e manutenção mais fácil.

TEMA 08

CONSTRUTORES

Habilidades

- Aplicar técnicas de orientação a objetos.

Introdução

Os construtores desempenham um papel fundamental na Programação Orientada a Objetos (POO), fornecendo uma maneira de inicializar e configurar objetos de forma eficiente. Um construtor é um método especial em uma classe que é chamado automaticamente quando um novo objeto é criado a partir dessa classe. Essencialmente, os construtores permitem que você defina como os objetos devem ser configurados logo após sua criação, garantindo que estejam em um estado válido e pronto para uso.

Ao criar uma instância de uma classe, o construtor associado a essa classe é acionado, permitindo que você forneça argumentos iniciais para configurar os atributos do objeto. Isso é particularmente útil para garantir que o objeto tenha valores coerentes e válidos desde o início. Além disso, construtores podem ser sobreescritos, permitindo diferentes formas de inicialização para atender a várias necessidades. Em suma, os construtores desempenham um papel crucial na criação de objetos bem-estruturados e prontos para serem usados em uma aplicação orientada a objetos.



Disponível em: <[freepix-https://tinyurl.com/2djs7pa3](https://tinyurl.com/2djs7pa3)>. Acesso em 11 set. 2023.

Construtor

Um construtor determina as ações que podem ser executadas quando um objeto é criado em JAVA. Ele é definido como um método, que deve ter o mesmo nome da classe e sem definição do tipo de retorno, nem mesmo void.

O construtor é unicamente invocado no momento da criação do objeto através do operador new. O retorno do operador new é uma referência para o objeto recém-criado.

O construtor pode receber argumentos, como qualquer método. Usando o mecanismo de sobrecarga, mais de um construtor pode ser definido para uma classe.

Um construtor é um método de grande importância dentro de uma classe Java. Ele possui algumas características próprias:

- É um método que tem o mesmo nome que a classe
- O padrão (default) não possui um valor de retorno, nem mesmo void.
- Pode conter parâmetros (argumentos)
- Toda classe deve possuir ao menos um construtor
- Se nenhum construtor for definido em uma classe, a JVM irá prover um construtor padrão, sem argumentos, conhecido como default.

Ao ser criada, uma classe inicializa seus atributos da seguinte maneira:

- Tipos primitivos numéricos (int, short, byte, long, float, double) são inicializados com valor 0 (Zero)
- Tipos primitivos booleanos (boolean) são inicializados com valor false
- Tipos primitivos caracteres (char) são inicializar com valor " (vazio)
- Objetos são inicializados como null.

Quando criamos a classe Moto e instanciamos o objeto m1, ele foi criado usando o construtor default Moto().

Exemplo:

```
Moto m1 = new Moto();
```

Fonte: Elaborado pelo autor (2023).

A figura a seguir representa a construção de métodos construtores, sendo o default (sem indicação de retorno) e os outros com a sobrecarga (tem parâmetros de entrada)/ :

```
//Construtor
default public
Moto() {
}
//sobrecarga do construtor
public Moto (String cor, String marca){
    this.cor = cor;
    this.marca =
        marca;
}
//sobrecarga do construtor
public Moto (String cor, String marca, String modelo){
    this.cor = cor;
    this.marca = marca;
    this.modelo =
        modelo;
}
```

Fonte: Elaborado pelo autor (2023).

O primeiro método construtor é o default, aquele padrão necessário para a construção da classe. O segundo possui dois parâmetros (cor e marca). Já o terceiro método construtor tem três parâmetros, sendo eles cor, marca e modelo, todos sendo definidos já quando o objeto seja instanciado no programa principal.

No próximo exemplo, teremos um código para o cálculo de vários aspectos de um círculo. Vamos ter os parâmetros de raio e cor, calculando ainda a sua área.

```
public class Circulo {
    private double raio,
    area; private String
    cor;
//Construtor default com definições
    internas public Circulo(){
        this.raio =
            1.0f; this.cor =
            "Verde";
    }
//sobrecarga do construtor
    public Circulo(double
        raio) {
        this.raio=raio;
        this.area =
            raio*raio; this.cor
            = "Amarelo";
    }
}
```

Fonte: Elaborado pelo autor (2023).

No programa principal, os objetos c1 e c2 são instanciados, cada um com um construtor, sendo o c1 definido pelo construtor sobreescrito, com o parâmetro de entrada e o c2 com o

construtor default.

```
public class mainCirculo {  
    public static void main(String[]  
        args) { Circulo c1 = new Circulo  
        (3.0f); Circulo c2 = new Circulo  
        ();  
  
        System.out.println("Raio de c1: "+c1.getRaio());  
        System.out.println("Cor do circulo: "+ c1.getCor());  
  
        c2.setRaio(5.0f);  
        c2.setCor("azul");  
  
        System.out.println("Raio de c2: "+ c2.getRaio());  
        System.out.println("Cor do circulo: "+ c2.getCor());  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

Não foi definida no programa principal a cor do círculo c1, porém ela aparece como amarela, pois estava pré definida no construtor sobrecarregado, ao qual o objeto c1 foi instanciado. Já c2, instanciado com o construtor default (que tinha como cor definida “Verde”) teve uma alteração com o modificador SET, que será desenvolvido a seguir, sendo definida a cor no desenvolvimento do código como azul.

A saída deste código será desta forma:

```
Saída - circulo (run) ×  
run:  
Raio de c1: 3.0  
Cor do círculo: Amarelo  
Raio de c2: 5.0  
Cor do círculo: azul  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Fonte: Elaborado pelo autor (2023).



ATIVIDADE DE FIXAÇÃO

1- O que é um construtor em Programação Orientada a Objetos (POO)?

- a) Um método que define atributos públicos.

- b) Um método especial para inicializar objetos.
- c) Um método que manipula exceções.
- d) Um método que define herança entre classes.

2- Qual é o principal propósito de um construtor em uma classe?

- a) Realizar operações matemáticas.
- b) Definir os atributos da classe como privados.
- c) Criar e inicializar objetos da classe.
- d) Realizar a serialização dos objetos.

3- Qual das seguintes afirmações é verdadeira sobre os construtores?

- a) Um construtor deve ter o mesmo nome da classe.
- b) Um construtor nunca pode receber parâmetros.
- c) Um construtor só pode ser chamado dentro de outro método da classe.
- d) Um construtor não pode ser sobre carregado.

4- É possível ter mais de um construtor em uma classe?

- a) Sim, mas todos os construtores devem ter o mesmo nome.
- b) Sim, desde que todos os construtores tenham a mesma assinatura.
- c) Não, uma classe só pode ter um construtor.
- d) Sim, é possível ter construtores com diferentes parâmetros.

5- Qual é a diferença entre um construtor padrão e um construtor parametrizado?

- a) Um construtor padrão não aceita parâmetros, enquanto um construtor parametrizado aceita.
- b) Um construtor padrão sempre inicia todos os atributos como nulos.
- c) Um construtor parametrizado é sempre privado.
- d) Um construtor padrão é automaticamente fornecido pelo compilador.

6- Qual é a função do operador "this" em um construtor?

- a) Ele define a herança entre classes.
- b) Ele indica que a classe não possui construtores.
- c) Ele referencia o próprio objeto sendo construído.
- d) Ele é usado para chamar outros construtores de outras classes.

7- Em Java, qual é o modificador de acesso mais apropriado para um construtor que deve ser acessível somente dentro da classe?

- a) private
- b) public
- c) protected
- d) static

8- O que acontece se uma classe não tiver um construtor definido explicitamente?

- a) A classe não pode ser instanciada.
- b) O Java automaticamente cria um construtor padrão vazio.
- c) O Java gera um erro de compilação.
- d) A classe não pode ser usada como uma superclasse.

9- Qual é o objetivo de usar a palavra-chave "super" em um construtor?

- a) Ela indica que um construtor está sendo sobreescrito.
- b) Ela chama explicitamente um construtor da classe pai.
- c) Ela cria uma instância da classe pai.
- d) Ela inicializa todos os atributos da classe atual.

10- Ao criar uma subclasse, se o construtor da classe pai não for explicitamente chamado, qual construtor da classe pai será chamado automaticamente?

- a) O construtor padrão.
- b) O construtor que aceita mais parâmetros.
- c) O construtor que aceita menos parâmetros.
- d) O construtor parametrizado com os mesmos parâmetros da subclasse.

TEMA 09

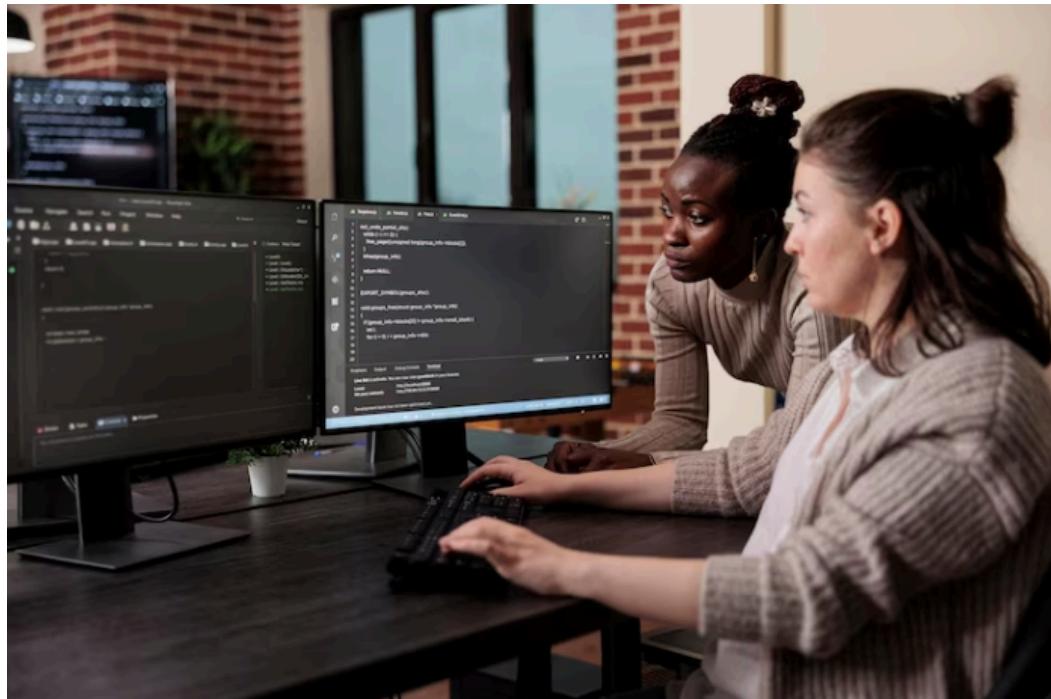
ENCAPSULAMENTO E OS MÉTODOS MODIFICADORES E ACESSORES

Habilidades

- Codificar programas orientados a objetos.
- Aplicar técnicas de orientação a objetos.

Introdução

O encapsulamento é um conceito fundamental na Programação Orientada a Objetos (POO) que se concentra em ocultar os detalhes internos de uma classe e fornecer interfaces controladas para acessar e modificar seus atributos. Isso promove a segurança e a organização do código, permitindo que as alterações internas de uma classe não afetem diretamente outras partes do programa. Juntamente com o encapsulamento, os métodos modificadores (também conhecidos como métodos setters) e os métodos acessores (ou métodos getters) são mecanismos cruciais para interagir com os atributos de uma classe de maneira controlada e consistente. Os métodos modificadores permitem a alteração controlada dos atributos, enquanto os métodos acessores possibilitam a obtenção desses valores, tornando possível implementar regras de negócio e validações ao manipular os dados internos de uma classe. Juntos, encapsulamento e métodos modificadores e acessores ajudam a criar uma estrutura robusta e eficiente, fortalecendo os princípios da orientação a objetos.



Disponível em: <DCStudio-><https://tinyurl.com/2jm9ma94>

Encapsulamento

O processo de vincular dados e métodos correspondentes (comportamento) juntos em uma única unidade é chamado de encapsulamento em Java. Em outras palavras, o encapsulamento é uma técnica de programação que une os membros da classe (variáveis e métodos) e evita que sejam acessados por outras classes.

Assim, podemos manter variáveis e métodos protegidos de interferências externas e uso indevido.

Cada classe Java é um exemplo de encapsulamento porque escrevemos tudo dentro da classe apenas o que vincula variáveis e métodos e esconde sua complexidade de outras classes.

Outro exemplo de encapsulamento é uma cápsula. Basicamente, a cápsula encapsula várias combinações de medicamentos. Se as combinações de medicamentos são variáveis e métodos, a cápsula funcionará como uma classe e todo o processo é denominado Encapsulamento, conforme mostrado na figura abaixo.

Na técnica de encapsulamento, declaramos os campos como privados na classe para evitar que outras classes os acessem diretamente. Os dados encapsulados necessários podem ser acessados usando os métodos getter e setter públicos (serão explicados na sequência).

Se o campo for declarado privado na classe, ele não poderá ser acessado por ninguém de fora da classe e oculta o campo dentro da classe. Portanto, também é chamado de ocultação de dados.

Por exemplo:

A mochila escolar é um exemplo de encapsulamento. Ela pode guardar nossos livros, canetas etc.

Outro exemplo é a validação de acesso. Quando você faz login em suas contas de e-mail, como Gmail, por exemplo, há muitos processos internos ocorrendo no back-end e você não tem

controle sobre eles.

Quando você insere a senha para registro, eles são recuperados de forma criptografada e verificados, e então você recebe acesso à sua conta. Você não tem controle sobre como a senha foi verificada. Assim, ele mantém nossa conta protegida contra uso indevido.

Como implementar encapsulamento em Java?

Existem duas formas pelas quais podemos alcançar ou implementar o encapsulamento em JAVA:

1. Declarando a variável de instância da classe como privada. para que não possa ser acessado diretamente por ninguém de fora da classe.
2. Fornecendo os métodos setter e getter públicos na classe para acessar e/ou modificar os valores das variáveis.

Vantagens do encapsulamento em Java

- O código encapsulado é mais flexível e fácil de alterar com novos requisitos.
- Impede que outras classes accessem os campos privados.
- O encapsulamento permite modificar o código implementado sem quebrar outro código que o implementou.
 - Ele mantém os dados e códigos protegidos de herança externa. Assim, o encapsulamento ajuda a alcançar a segurança.
 - Melhora a capacidade de manutenção da aplicação.
 - Se você não definir o método setter na classe, os campos podem se tornar somente leitura.
 - Se você não definir o método getter na classe, os campos podem ser feitos somente para gravação.

Desvantagem do encapsulamento em Java

A principal desvantagem do encapsulamento em Java é que aumenta o comprimento do código e atrasa o fim da execução.

Vamos entender alguns programas de exemplo baseados em encapsulamento em Java.

Classe Estudante:

```
public class estudante {  
    private String nome;  
    public String getNome  
    () {  
        return nome;  
    }  
    public void setNome (String  
        nomeEstudante) { name = nomeEstudante;  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

Programa principal

```
public class Encapsulamento {  
    public static void main(String[]  
        arg) Estudante el = new Estudante  
        ();  
        el.nome = "Joaozinho"; //vai dar erro de compilação!!!  
        //nome é uma variável privada (private)  
        //então tento declarar uma String nova para receber nome  
        String nomEstudante = el.nome; //novamente erro de compilação!!  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

Qual seria então a solução correta, já que encapsulamos a variável nome, de forma privada?

```
public class Encapsulamento {  
    public static void main(String[]  
        arg) Estudante el = new Estudante  
        ();  
        el.setNome("Joaozinho");//será “setado” o nome lá na variável  
        privada  
        String nomEstudante = el.getNome();//neste momento eu busco o  
        nome //  
        //atribuo a esta nova variável  
        System.out.println(nomEstudante);  
    }  
}
```

Fonte: Elaborado pelo autor (2023).

A saída será:

Joaozinho

Modificadores e Acessores

Getters e setters são conhecidos como métodos assessores (getters) e modificadores (setters), usados para proteger os dados do código, principalmente ao criar classes. Para cada variável de instância, um método getter retorna seu valor enquanto um método setter define ou atualiza seu valor. Por isso, getters e setters também são conhecidos como assessores e modificadores,

respectivamente.

Por convenção, getters começam com a palavra "get" e setters com a palavra "set", seguido por um nome de variável. Em ambos os casos, a primeira letra do nome da variável é maiúscula, por exemplo:

```
public class Veiculo { private String cor;  
    // Getter  
    public String getCor() { return cor;  
    }  
    // Setter  
    public void setCor(String c) { this.cor = c;  
    }  
}
```

O método getter retorna o valor do atributo. O método setter pega um parâmetro e o atribui ao atributo. Uma vez que getter e setter foram definidos, podemos usá-los no nosso programa principal:

```
public static void main(String[] args) { Veiculo v1 = new Veiculo(); v1.setCor("Verde");  
System.out.println(v1.getCor());  
}  
// A saída será "Verde"
```

Getters e setters permitem controle sobre os valores. Você pode validar o valor fornecido no configurador antes de definir o valor.

É obrigatório usar getters e setters?

Não, mas é uma boa prática. Os getters e setters permitem controlar como variáveis importantes são acessadas e atualizadas no código. Por exemplo, considere este método setter:

```
public void setNum(int num) { if (num < 1 || num > 10) {  
    System.out.println("Erro do valor numérico");  
}  
this.num = num;  
}
```

Ao usar o método setNum, você pode ter certeza de que o valor de num está sempre entre 1 e 10. Isso é muito melhor do que atualizar a variável num diretamente, como por exemplo:

```
obj.num = 13;
```

Se você atualizar num diretamente, é possível que cause efeitos colaterais indesejados em algum outro lugar do seu código. Aqui, definir num como 13 viola a restrição de 1 a 10 que queremos estabelecer. Criar num como uma variável privada e usar o método setNum evitaria que isso acontecesse. Por outro lado, a única maneira de ler o valor de num é usando um método getter:

```
public int getNum() { return this.num;  
}
```

Resumindo:

1. Se você definir apenas o método getter, ele pode se tornar somente leitura.
2. Se você definir apenas o método setter, ele pode ser feito somente para gravação.
3. Se você definir os métodos getter e setter, eles poderão ser utilizados para leitura e gravação.

Voltando ao exemplo do Círculo, agora com a Classe completa:

```
//Primeiro os construtores... e depois os
métodos public double getArea() {
    return area;
}
public void setArea(float
    area) { this.area = area;
}
public double
    getRaio() { return
        raio;
}
public void setRaio(float
    raio) { this.raio = raio;
}
public String
    getCor() { return
        cor;
}
public void setCor(String
    cor) { this.cor = cor;
}
```

Fonte: Elaborado pelo autor (2023).

Para a compreensão completa de Encapsulamento, é necessário entender os modificadores de acesso em JAVA.



ATIVIDADE DE FIXAÇÃO

1- O que é encapsulamento na programação orientada a objetos?

- a) Um processo de criação de objetos em Java.
- b) Um princípio que envolve a ocultação dos detalhes internos de uma classe e a exposição de uma interface controlada.
- c) Um método que permite acessar atributos privados diretamente.
- d) Um recurso para evitar a criação de classes.

2- Qual é a principal vantagem do encapsulamento?

- a) Tornar o código mais complicado.
- b) Facilitar a criação de objetos.

- c) Permitir a reutilização de código.
- d) Isolar os detalhes internos e evitar alterações indesejadas.

3- Qual é a função dos métodos modificadores (setters) em Java?

- a) Obter valores dos atributos de uma classe.
- b) Acessar os atributos privados de uma classe.
- c) Modificar os valores dos atributos privados de uma classe.
- d) Criar novos atributos em uma classe.

4- Qual é o propósito dos métodos acessores (getters)?

- a) Modificar os valores dos atributos privados de uma classe.
- b) Criar novos atributos em uma classe.
- c) Obter valores dos atributos de uma classe.
- d) Validar dados de entrada.

5- Quais são as palavras-chave usadas para definir métodos modificadores e acessores em Java?

- a) modify e access
- b) set e get
- c) private e public
- d) encapsulate e retrieve

6- Um método acessor em Java deve ser declarado como:

- a) void
- b) private
- c) public
- d) static

7- Qual é a convenção de nomenclatura comum para um método acessor em Java?

- a) getValorAtributo()
- b) atributoGet()
- c) obterAtributo()
- d) acessarAtributo()

8- Qual é a principal diferença entre um atributo público e um atributo privado em relação ao encapsulamento?

- a) Atributos públicos podem ser acessados de qualquer classe, enquanto atributos privados só podem ser acessados dentro da própria classe.
- b) Atributos públicos são mais seguros em relação à manipulação de dados.
- c) Atributos privados não podem ser modificados.
- d) Atributos privados não podem ser declarados em Java.

9- Qual é o objetivo principal dos métodos modificadores e acessores?

- a) Acelerar a execução do programa.
- b) Facilitar a criação de objetos.
- c) Controlar o acesso e a modificação dos atributos de uma classe.
- d) Substituir atributos por métodos.

10- Em um exemplo de classe chamada "Pessoa", como um método modificador poderia ser definido para alterar o nome da pessoa?

- a) setNomePessoa(nome)
- b) alterarNome(nome)
- c) mudarNome(nome)
- d) nome = novoNome(nome)

TEMA 10

MODIFICADORES DE ACESSO EM JAVA

Habilidades

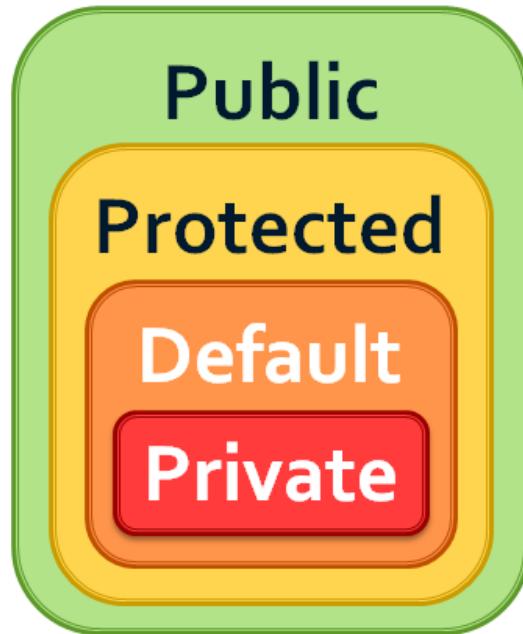
- Utilizar ambientes de desenvolvimento para desenvolvimento desktop.
- Construir interface gráfica.

Introdução

Os modificadores de acesso desempenham um papel de extrema importância no contexto da linguagem de programação Java, desempenhando um papel crucial na definição da visibilidade e da acessibilidade dos membros de uma classe. Em outras palavras, eles são a chave para controlar quem pode ver e interagir com os atributos e métodos de uma classe, um aspecto fundamental da programação orientada a objetos.

Em Java, existem quatro principais modificadores de acesso: public, private, protected e o padrão (ausência de modificador). Cada um desses modificadores possui um propósito específico e determina como os membros de uma classe podem ser acessados e manipulados. Isso é fundamental para a aplicação do princípio de encapsulação, que é uma das pedras angulares da programação orientada a objetos.

Ao designar um atributo como private, por exemplo, você está estabelecendo que ele só pode ser acessado dentro da própria classe em que está definido. Essa restrição é crucial para evitar que outros componentes externos accessem ou modifiquem diretamente os atributos, o que poderia levar a erros ou comportamentos inesperados no programa. A privacidade oferecida pelo modificador private é essencial para proteger os detalhes internos de uma classe.



Disponível em: <<http://www.mauda.com.br/?p=1433>>. Acesso em 11 set. 2023.

Por outro lado, quando um atributo ou método é definido como public, ele se torna acessível a partir de qualquer classe, possibilitando uma interação mais aberta e flexível. Isso pode ser útil quando você deseja que determinadas funcionalidades da classe sejam utilizadas livremente por outras partes do código.

Os modificadores de acesso em Java desempenham um papel crucial na organização interna de uma classe, permitindo que você estabeleça barreiras de proteção quando necessário e torne partes específicas da classe acessíveis quando apropriado. Ao fazer isso, eles contribuem para a criação de um código mais seguro e compreensível, facilitando o desenvolvimento, a manutenção e a colaboração em projetos de software. Portanto, compreender e aplicar adequadamente esses modificadores é um passo essencial para qualquer programador Java que busca escrever código de qualidade.

São 4 os modificadores de acesso básicos da linguagem Java:

private, padrão, protected e public

Eles servem para tornar componentes da sua aplicação mais ou menos acessíveis por outras partes do seu programa.

Modificador	Funcionalidade
public	permite que qualquer outra parte da aplicação tenha acesso ao membro
Padrão (default)	membros que não foram marcados com nenhum modificador explicitamente. Só podem ser acessados por outras classes dentro do mesmo pacote
protected	os membros são acessíveis por classes dentro do mesmo pacote e por classes derivadas (mesmo em pacotes diferentes)

private	só é acessível dentro da própria classe em que foi declarado
---------	--

Fonte: Elaborado pelo autor (2023).

public

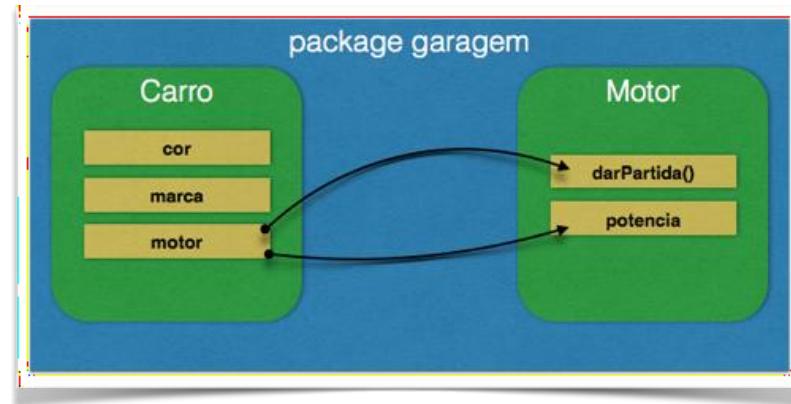
O modificador de acesso public é o menos restritivo de todos. Ele permite que qualquer outra parte da sua aplicação tenha acesso ao componente marcado como public.

Dentro do mesmo pacote

Classes Carro e Motor:

```
package garagem; public class Carro {  
    public String marca; public String cor; public Motor motor;  
    public void ligar()  
    {  
        this.motor.darPartida();  
    }  
  
    public String toString()  
    {  
        return marca + " " + cor + " " + motor.potencia;  
    }  
}  
  
package garagem; public class  
Motor {  
    public int potencia; public void  
darPartida(){}  
}
```

A classe Carro faz uso da classe Motor. Na classe Carro temos acesso à propriedade potencia e ao método darPartida() da classe Motor.



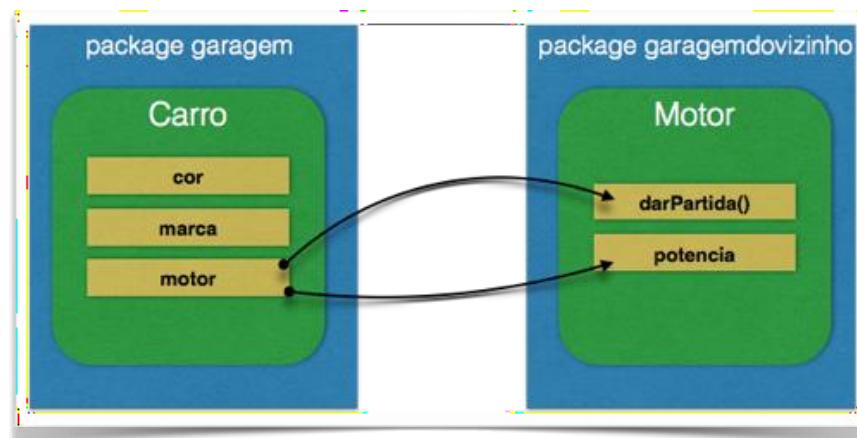
Fonte: Elaborado pelo autor (2023).

A classe Carro tem acesso a todos os membros marcados como **public** da classe Motor mesmo que as duas classes estejam em pacotes distintos.

Dentro de pacotes diferentes

Vamos colocar a classe Motor no pacote garagemdovizinho e fazer a importação na classe Carro:

```
package garagemdovizinho; public class Motor {  
    public int potencia; public void darPartida(){}
}  
  
package garagem;  
import garagemdovizinho.Motor; public class Carro {  
    public String marca; public String cor; public Motor motor;  
    public void ligar()  
    {  
        this.motor.darPartida();  
    }  
    public String toString()  
    {  
        return marca + " " + cor + " " + motor.potencia;  
    }
}
```



Fonte: Elaborado pelo autor (2023).

Mesmo com a classe Motor dentro do um pacote diferente, a classe Carro continua conseguindo acessar o método `darPartida()` e a propriedade `potencia` da classe Motor.

Classes derivadas

Os membros de uma classe que são explicitamente marcados com o modificador de acesso

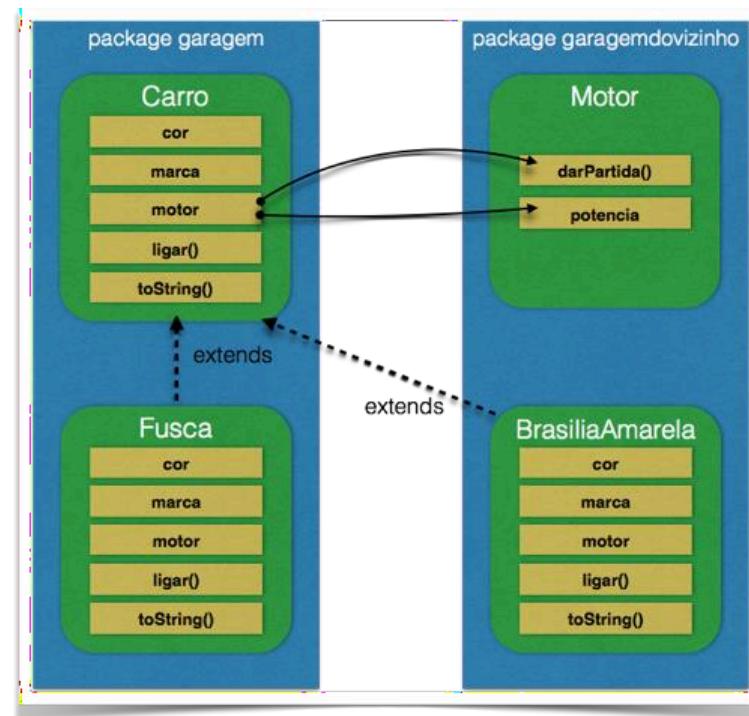
"public" estão disponíveis e podem ser acessados por outras classes que derivam da mesma classe. Para ilustrar essa ideia, consideremos as declarações das classes "Fusca" e "BrasiliaAmarela", que estão localizadas em diferentes pacotes: "garagem" e "garagemdovizinho", respectivamente. Ambas essas classes são subtipos da classe base chamada "Carro".

Isso significa que, mesmo estando em pacotes distintos, as classes "Fusca" e "BrasiliaAmarela" podem acessar os membros públicos da classe "Carro". O modificador de acesso "public" permite uma visibilidade ampla, garantindo que as classes derivadas tenham a capacidade de utilizar esses membros sem restrições.

Essa abordagem é fundamental para promover a reutilização de código e a extensibilidade das classes em Java. Ao herdar uma classe base, as classes derivadas podem herdar e usar os membros públicos da classe base, simplificando o desenvolvimento e a manutenção do código, mesmo quando as classes estão localizadas em diferentes pacotes. Essa flexibilidade é um dos princípios fundamentais da programação orientada a objetos em Java e permite criar hierarquias de classes eficientes e organizadas. Portanto, a capacidade de acesso aos membros públicos em classes derivadas contribui significativamente para a modularidade e a flexibilidade do sistema de classes em Java. Ambas as classes são derivadas da classe Carro:

```
package garagem;
public class Fusca extends Carro {

    public Fusca()
    {
        this.cor = "Branco"; this.marca = "VW"; this.ligar();
        this.toString();
    }
}
package garagemdovizinho; import garagem.Carro;
public class BrasiliaAmarela extends Carro { public
BrasiliaAmarela() {
    this.cor = "Amarelo"; this.marca = "VW"; this.ligar();
    this.toString();
}
}
```



Fonte: Elaborado pelo autor (2023).

De todos os modificadores de acesso, o **public** é o menos restritivo. Veja o resumo da acessibilidade na imagem abaixo:



Fonte: Elaborado pelo autor (2023).

Protected

Os membros das classes marcados com o modificador de acesso **protected** serão acessíveis por classes e interfaces dentro do mesmo pacote e por classes derivadas mesmo que estejam em pacotes diferentes.

Dentro do mesmo pacote

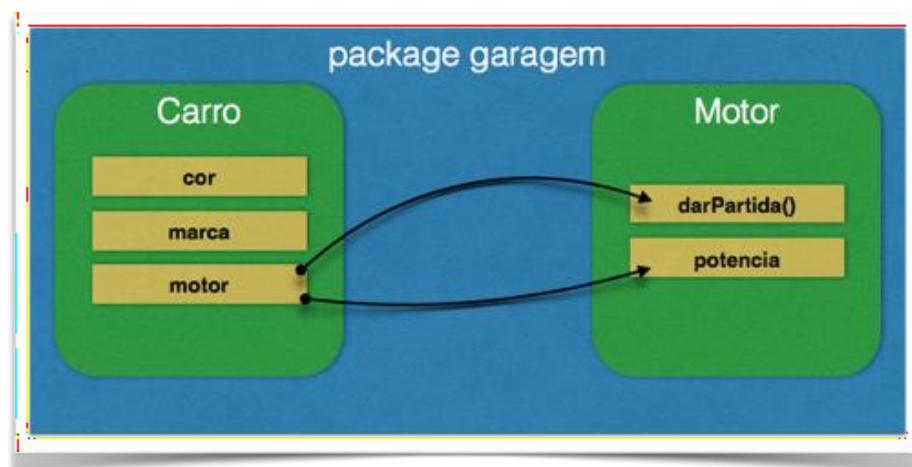
Classes Carro e Motor dentro do pacote garagem:

```
package garagem; public class Motor {  
    public int potencia; protected void darPartida(){}  
}
```

O método `darPartida()` da classe Motor está marcado com o modificador de acesso `protected`.

```
package garagem; public class Carro {  
    protected String marca; protected String cor; public Motor motor;  
    protected void ligar()  
    {  
        this.motor.darPartida();  
    }  
    public String toString()  
    {  
        return marca + " " + cor + " " + motor.potencia;  
    }  
}
```

Os membros `marca`, `cor` e `ligar()` da classe Carro estão marcados com o modificador de acesso `protected`.



Fonte: Elaborado pelo autor (2023).

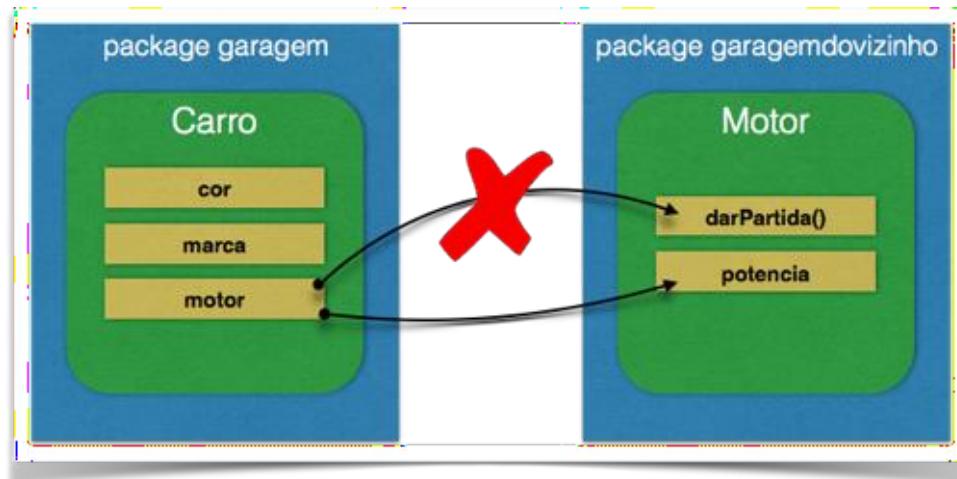
O método `darPartida()` da classe Motor é acessível pela classe Carro.

Dentro de pacotes diferentes

Vamos mover a classe Motor para o pacote garagemvizinho e observar as mudanças.

Veja a declaração das classes Carro e Motor:

```
package garagemdovizinho; public class Motor {  
    public int potencia; protected void darPartida(){}
}  
  
package garagem;  
import garagemdovizinho.Motor; public class Carro {  
    protected String marca; protected String cor; public Motor motor;  
  
    protected void ligar()  
    {  
        this.motor.darPartida();  
        // O método darPartida() do tipo Motor não é visível.  
    }  
  
    public String toString()  
    {  
        return marca + " " + cor + " " + motor.potencia;  
    }  
}
```



Fonte: Elaborado pelo autor (2023).

A classe Carro não compila e recebemos a mensagem de que o método `darPartida()` do tipo Motor não é visível. Um membro marcado com o modificador de acesso `protected` só é visível para outras classes e interfaces localizadas dentro do mesmo pacote ou por seus herdeiros. Veremos como as classes derivadas se comportam com o uso de membros `protected` a seguir.

Classes derivadas

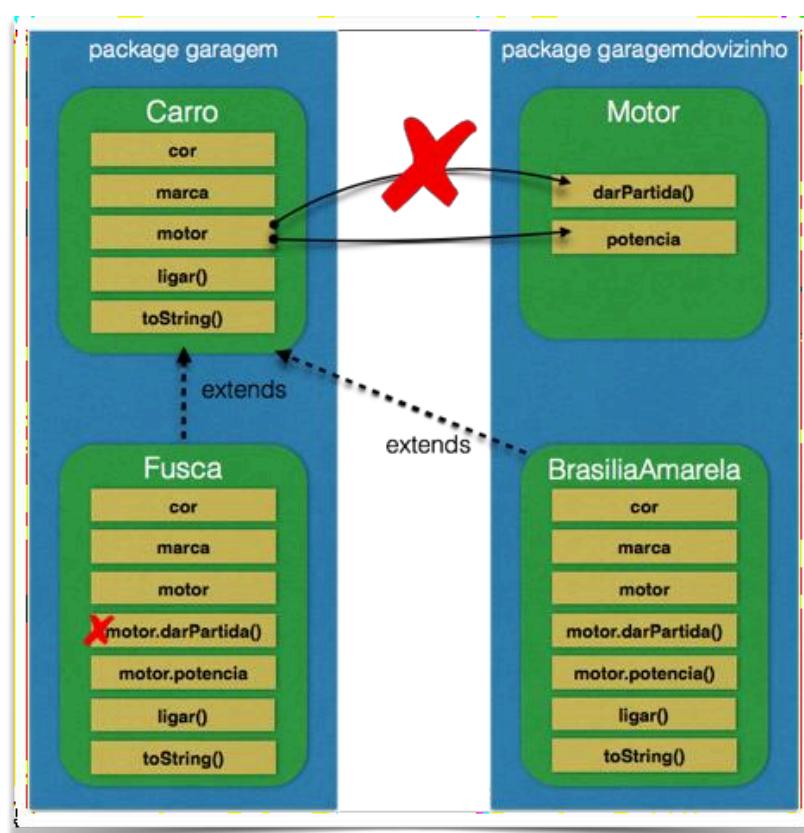
As classes que herdam de outras classes têm a capacidade de acessar os membros que foram declarados como "protected" na classe mãe, independentemente do pacote em que essas classes

estejam localizadas. Vamos examinar a declaração das classes Fusca e BrasiliaAmarela para entender melhor esse conceito.

Primeiramente, é importante ressaltar que, no contexto das classes Fusca e BrasiliaAmarela, o método `darPartida()` da classe Motor não está acessível diretamente pela classe Fusca. Isso significa que a classe Fusca não pode chamar o método `darPartida()` diretamente, provavelmente devido a restrições de acesso impostas pela classe Motor ou por razões de encapsulamento específicas.

No entanto, é fundamental observar que todos os membros das classes Carro e Motor são acessíveis pela classe BrasiliaAmarela. Isso implica que a classe BrasiliaAmarela pode interagir com todos os membros (variáveis e métodos) que foram declarados como "protected" nas classes Carro e Motor, independentemente do pacote em que essas classes estejam localizadas. Esse acesso ampliado proporciona à classe BrasiliaAmarela uma maior flexibilidade e capacidade de utilizar e modificar os membros herdados dessas classes.

Em resumo, as classes derivadas, como a classe BrasiliaAmarela, têm o privilégio de acessar os membros marcados como "protected" na classe mãe, permitindo uma extensão eficaz das funcionalidades e o uso de recursos herdados. No entanto, é importante observar que restrições de acesso específicas podem ser aplicadas, como no caso do método `darPartida()` da classe Motor, que não é acessível diretamente pela classe Fusca.



Fonte: Elaborado pelo autor (2023).

```
package garagem;
public class Fusca extends Carro {
    public Fusca()
    {
        this.cor = "Branco"; this.marca = "VW"; this.ligar();
        this.motor.darPartida(); // O método darPartida() do tipo Motor não é visível this.motor.potencia = 100;
        this.toString();
    }
}

package garagemdovizinho; import garagem.Carro;
public class BrasiliaAmarela extends Carro { public BrasiliaAmarela() {
    this.cor = "Branco"; this.marca = "VW"; this.ligar(); this.motor.darPartida(); this.motor.potencia = 100;
    this.toString();
}
}
```

O modificador de acesso `protected` é mais restritivo do que o modificador `public`, mas é menos restritivo do que os modificadores `padrão` e `private`.



Fonte: Elaborado pelo autor (2023).

Padrão

O modificador de acesso padrão, comumente referido como "acessibilidade de pacote", é um atributo que é automaticamente aplicado aos membros de uma classe quando nenhum outro modificador de acesso específico é definido para eles. Esses membros que possuem acessibilidade de

pacote só podem ser acessados por outras classes ou interfaces que também estão definidas no mesmo pacote.

Em outras palavras, quando você cria uma classe e declara variáveis, métodos ou outros membros nessa classe sem especificar um modificador de acesso como "public", "private" ou "protected", esses membros herdam o modificador de acesso padrão, que é o acesso de pacote.

A principal característica desse modificador é que ele limita o acesso aos membros da classe apenas ao escopo do mesmo pacote em que a classe está contida. Isso significa que outras classes fora do pacote não podem acessar diretamente esses membros, a menos que herdem essa classe ou tenham alguma relação de herança com ela.

Portanto, ao utilizar o modificador de acesso padrão ou "acessibilidade de pacote", você está restrito a permitir que apenas as classes dentro do mesmo pacote acessem esses membros, proporcionando um nível de encapsulamento e controle sobre a visibilidade e o uso desses elementos dentro do contexto específico do pacote em questão.

Dentro do mesmo pacote

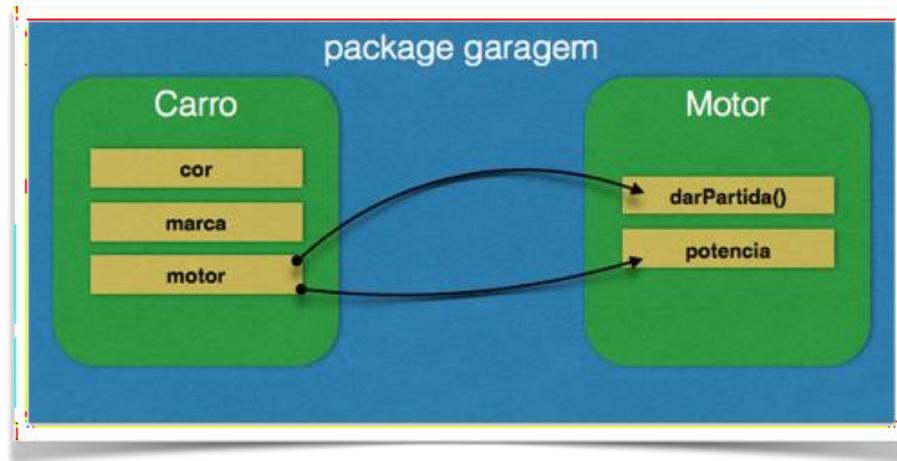
A definição da classe Carro e Motor dentro do pacote garagem com alguns dos seus elementos com a acessibilidade de pacote:

```
package garagem; public class Motor {  
    public int potencia; void darPartida(){}
}
```

O método `darPartida()` tem a acessibilidade padrão.

```
package garagem; public class Carro {  
    String marca;  
    String cor;  
    public Motor motor;  
    void ligar()  
    {  
        this.motor.darPartida();  
    }  
  
    public String toString()  
    {  
        return marca + " " + cor + " " + motor.potencia;  
    }
}
```

As variáveis de classe `marca`, `cor` e o método `ligar()` têm a acessibilidade padrão.



Fonte: Elaborado pelo autor (2023).

Como as duas classes estão dentro do mesmo pacote, o método `darPartida()`, mesmo tendo a acessibilidade de pacote, continua sendo acessível pela classe `Carro`.

Dentro de pacotes diferentes

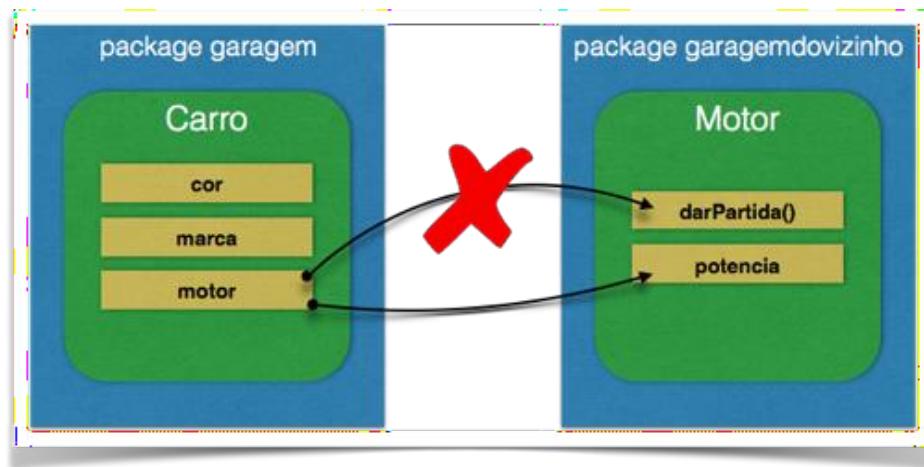
Ao colocar a classe `Motor` dentro do pacote `garagemdovizinho`, a classe `Carro` não consegue mais ter o acesso ao método `darPartida()` da classe `Motor`.

Por exemplo:

```
package garagemdovizinho;
public class Motor { public int potencia; void darPartida(){}
}
```

```
package garagem;
import garagemdovizinho.Motor; public class Carro {
String marca;
String cor;
public Motor motor;
void ligar()
{
this.motor.darPartida();
// O método darPartida() do tipo Motor não é visível.
}
public String toString()
{
return marca + " " + cor + " " + motor.potencia;
}
}
```

Perdemos o acesso ao método `darPartida()` da classe `Motor`.



Fonte: Elaborado pelo autor (2023).

A compilação da classe Carro falha e recebemos a mensagem de que o método darPartida() do tipo Motor não é visível.

Classes derivadas

Com classes derivadas a regra continua a mesma, só as classes derivadas declaradas dentro do mesmo pacote têm acesso aos membros com acessibilidade de pacote da classe mãe.

Por exemplo:

```
package garagem;  
public class Fusca extends Carro {  
  
    public Fusca()  
    {  
        this.cor = "Branco"; this.marca = "VW"; this.ligar();  
        this.motor.darPartida(); // O método darPartida() do tipo Motor não é visível  
        this.motor.potencia = 100;  
        this.toString();  
    }  
}
```

Apesar da variável de classe motor do tipo Motor estar visível, o método darPartida() não está.

```
package garagemdovizinho; import garagem.Carro;  
public class BrasiliaAmarela extends Carro { public BrasiliaAmarela() {  
    this.cor = "Branco"; // O campo Carro.cor não é visível.
```

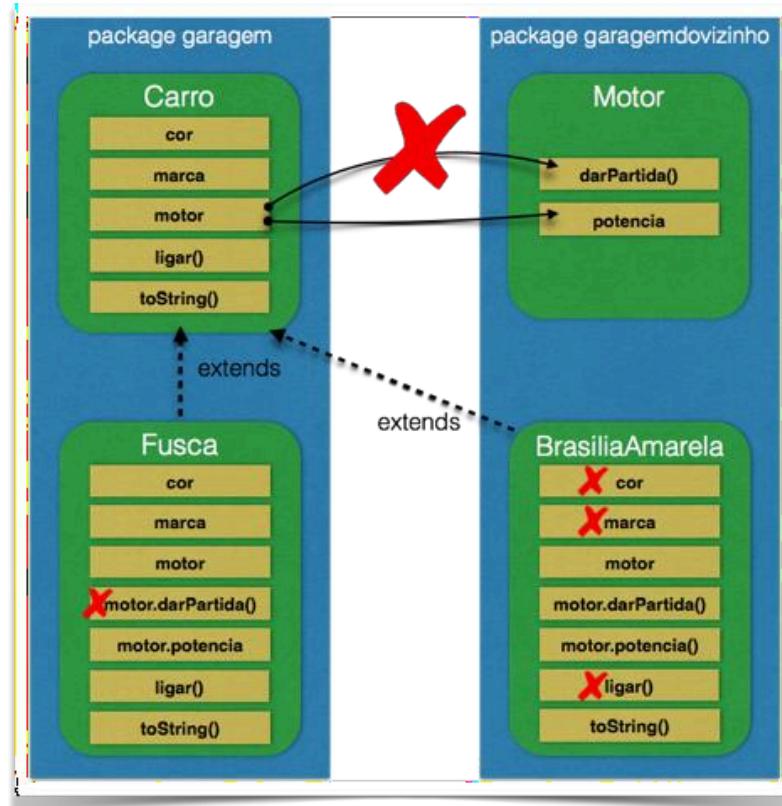
```

this.marca = "VW"; // O campo Carro.marca não é visível. this.ligar(); // O método ligar() do
tipo Carro não é visível. this.motor.darPartida();
this.motor.potencia = 100; this.toString();
}
}

```

A classe Carro, em que a classe BrasiliaAmarela tem um relacionamento de herança, o método darPartida() da classe Motor não está visível. Mas na classe BrasiliaAmarela, o mesmo método é visível.

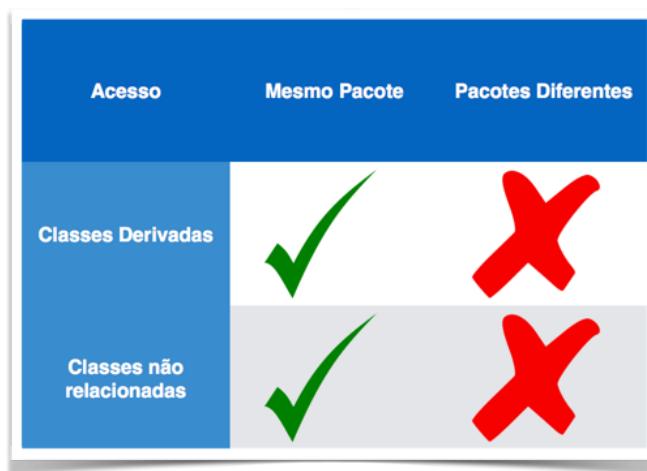
Detalhamento das classes e pacotes:



Fonte: Elaborado pelo autor (2023).

Apesar das classes Fusca e BrasiliaAmarela terem acesso à classe Motor por herdarem da classe Carro, a acessibilidade aos membros da classe Carro por essas duas classes filhas é diferente. A classe Fusca só tem acesso à variável de classe potencia, enquanto a classe BrasiliaAmarela tem acesso à variável de classe potencia e ao método darPartida().

O modificador de acesso padrão é mais restritivo do que o modificador public e protected, mas ele é menos restritivo do que o modificador private.



Fonte: Elaborado pelo autor (2023).

Private

O modificador de acesso "private" representa a restrição mais rigorosa no controle de acesso em programação. Qualquer membro de uma classe que seja definido como "private" está acessível somente a essa própria classe, independentemente de onde ela esteja localizada em pacotes ou se a classe é ou não herdada por outras. Em outras palavras, um membro com o modificador "private" só pode ser acessado e manipulado dentro da mesma classe na qual ele foi originalmente declarado.

Este nível extremamente restrito de acesso é uma característica fundamental da programação orientada a objetos e é usado para garantir a encapsulação e a integridade dos dados dentro de uma classe. Ele impede que outros elementos do programa, mesmo aqueles dentro do mesmo pacote ou subclasses, acessem diretamente os detalhes internos da classe. Isso ajuda a manter a coesão da classe, garantindo que somente os métodos e operações especificamente projetados para interagir com seus membros "private" possam fazê-lo.

Assim, ao utilizar o modificador "private", você está estabelecendo um nível máximo de isolamento e controle sobre os membros da sua classe, contribuindo para a manutenção da segurança e organização do seu código. Isso permite que você projete classes mais robustas e evite o acesso não autorizado ou modificações inadvertidas nos dados internos da classe, contribuindo para um código mais seguro e confiável.

```
package garagem;  
import garagemdovizinho.Motor;  
public class Carro { private String marca; private String cor; public Motor motor;  
  
    private void ligar()  
    {  
        this.motor.darPartida();  
        // O método darPartida() do tipo Motor não é visível.  
    }  
}
```

```
public String toString()
{
    return marca + " " + cor + " " + motor.potencia;
}
}

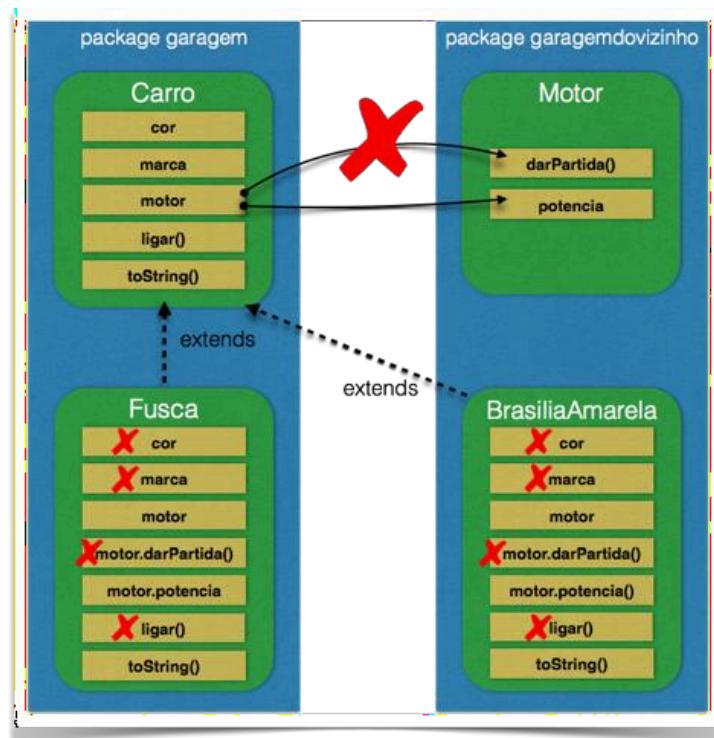
package garagemdovizinho; public class Motor {
public int potencia; private void darPartida(){}
}

package garagem;
public class Fusca extends Carro { public Fusca()
{
    this.cor = "Branco"; // O campo Carro.cor não é visível. this.marca = "VW"; // O campo
Carro.marca não é visível. this.ligar(); // O método ligar() do tipo Carro não é visível.
this.motor.darPartida();
    // O método darPartida() do tipo Motor não é visível. this.motor.potencia = 100;
    this.toString();
}
}

package garagemdovizinho; import garagem.Carro;
public class BrasiliaAmarela extends Carro {

    public BrasiliaAmarela() {
        this.cor = "Branco"; // O campo Carro.cor não é visível. this.marca = "VW"; // O campo
Carro.marca não é visível. this.ligar(); // O método ligar() do tipo Carro não é visível.
this.motor.darPartida();
    // O método darPartida() do tipo Motor não é visível. this.motor.potencia = 100;
    this.toString();
}
}
```

Nota-se que todo elemento marcado com o modificador `private` não é acessível por outras classes.



Fonte: Elaborado pelo autor (2023).

Comportamento do modificador de acesso private na imagem a seguir:



Fonte: Elaborado pelo autor (2023).

Resumo dos modificadores de nível de membro

	Classe	Pacote	Subclasse	Todos
public	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
protected	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
default	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
private	<input checked="" type="checkbox"/>			

Disponível em: <<https://tinyurl.com/24f2xbua>>. Acesso em 11 set. 2023.



ATIVIDADE DE FIXAÇÃO

1- Qual é o objetivo principal dos modificadores de acesso em Java?

- a) Definir o tipo de dado de um atributo.
- b) Controlar a visibilidade e o acesso aos membros de uma classe.
- c) Modificar o comportamento de um método.
- d) Determinar a ordem de execução das instruções.

2- Qual é o modificador de acesso padrão em Java?

- a) public
- b) private
- c) protected
- d) default

3- Qual é o resultado de declarar um método como private em uma classe?

- a) O método só pode ser acessado por outras classes do mesmo pacote.
- b) O método pode ser acessado por qualquer classe.
- c) O método pode ser acessado somente dentro da mesma classe.
- d) O método pode ser acessado por classes filhas.

4- Quais são os quatro principais modificadores de acesso em Java?

- a) open, close, read, write
- b) inside, outside, protected, public
- c) public, private, protected, default
- d) access, restrict, expose, permit

5- Qual é a visibilidade de um atributo declarado como protected?

- a) Atributos protegidos só podem ser acessados dentro da mesma classe.
- b) Atributos protegidos podem ser acessados por qualquer classe.
- c) Atributos protegidos só podem ser acessados por classes do mesmo pacote ou classes filhas.
- d) Atributos protegidos só podem ser acessados por classes filhas.

6- Qual é a visibilidade de um membro de classe quando nenhum modificador de acesso é especificado?

- a) private
- b) public
- c) default
- d) protected

7- Em que cenário um atributo ou método pode ser acessado por qualquer classe em Java?

- a) Quando declarado como public.
- b) Quando declarado como private.
- c) Quando declarado como protected.
- d) Quando declarado como default.

8- Qual é o principal objetivo do encapsulamento em Java?

- a) Permitir o acesso irrestrito aos atributos da classe.
- b) Isolar uma classe das demais, impossibilitando a interação entre elas.
- c) Controlar o acesso aos detalhes internos de uma classe e promover a segurança e integridade do código.
- d) Tornar todos os membros de uma classe visíveis para todas as outras classes.

9- Qual é a principal vantagem de utilizar modificadores de acesso em Java?

- a) Simplificar a sintaxe do código.
- b) Facilitar a reutilização de código.
- c) Promover o reuso de classes.
- d) Controlar o acesso e a visibilidade dos membros de uma classe.

10- O que acontece quando um membro de uma classe é declarado como private?

- a) Ele só pode ser acessado por métodos da mesma classe.
- b) Ele só pode ser acessado por classes do mesmo pacote.

- c) Ele pode ser acessado por qualquer classe.
 - d) Ele pode ser acessado por classes filhas.
erro de forma controlada durante a execução.
- D) Uma abordagem para evitar o uso de blocos try e catch no código.

10- Qual é o principal objetivo do uso de streams de caracteres ao trabalhar com leitura e escrita de dados em Java?

- A) Reduzir o tamanho dos arquivos.
- B) Melhorar o desempenho das operações.
- C) Manipular dados binários de forma eficiente.
- D) Trabalhar com dados em formato de texto.



REFERÊNCIAS

GUEDES. Gilleanes T. A. **UML 2 – Uma abordagem prática.** - 1. ed. – Rio de Janeiro: Luiz Antônio de Moraes Pereira. p.31, p.32, p.33. p.37. 2011.

SOARES. Bruno C. **Requisitos para utilização de prototipagem evolutiva nos processos de desenvolvimento de software baseado na web.** Departamento de Ciência da Computação - UFMG – Universidade Federal de Minas Gerais (UFMG). 2007.



Viva sua profissão!