PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 1

Prof. Leonardo Gomes

CONVERSA INICIAL

Nesta aula, faremos uma introdução ao tópico de programação orientada a objetos e à linguagem de programação Java. Falaremos sobre os diferentes paradigmas de programação com foco na orientação a objetos. Daremos, também, os primeiros passos no Java, uma importante linguagem de programação que servirá de base para esta disciplina como um todo.

Ao final desta aula esperamos atingir os seguintes objetivos que serão avaliados ao longo da disciplina da forma indicada.

Quadro 1 – Objetivos da aula

Objetivos	**Avaliação**
1. Contextualização sobre o paradigma de programação orientado a objetos e a Linguagem de programação Java.	Questionário e questões dissertativas
2. Capacidade de criar projetos Java simples dentro da IDE Eclipse.	Questionário e questões dissertativas
3. Desenvolver programas básicos utilizando Java.	Questionário e questões dissertativas

TEMA 1 – PARADIGMAS DE PROGRAMAÇÃO E HISTÓRIA DA PROGRAMAÇÃO ORIENTADA A OBJETOS

Neste Tema, discutiremos os diferentes paradigmas de programação e um pouco de seu histórico com foco na Orientação a Objetos.

Antes mesmo do surgimento dos primeiros computadores na década de 40, já existiam modelos e regras formais para descrever algoritmos que serviram de base para as primeiras linguagens de programação propriamente ditas.

Chamamos de paradigma de programação um dos meios de classificar linguagens de programação de acordo com sua estruturação, abstração e funcionalidades. Nestes primeiros anos da

programação, destacamos o paradigma **procedural**, **paradigma funcional**, **paradigma lógico** e o **paradigma orientado a objetos**.

Por meio do **paradigma procedural** que geralmente iniciamos o aprendizado na programação, podemos citar como exemplo a linguagem C, que utiliza exclusivamente esse paradigma. No paradigma procedural baseamos nosso código em comandos que mudam o estado da memória de forma detalhada e sequencial, ou seja, procedural. Reservamos espaços de memória por meio de variáveis para armazenar nossos dados e criamos funções que definem comportamentos desejados para esses dados. Esse paradigma se aproxima da forma com que o processador interpreta os comandos e trabalha com os dados efetivamente, dando maior liberdade para o programador desenvolver algoritmos eficientes.

Junto ao paradigma procedural surgiu também o **paradigma funcional**, no qual o código é pensado e descrito por meio da resolução de funções matemáticas. Esse paradigma facilita o desenvolvimento de certos algoritmos que são mais facilmente representados de forma puramente matemática. Trata-se de um paradigma ainda muito utilizado em certas linguagens que combinam paradigmas como a linguagem *Scala*.

Outro importante paradigma que surgiu na década de 50 foi o **paradigma lógico**, no qual uma base de declarações lógicas matemáticas é gerada pelo programador com a qual o computador se baseia para calcular respostas fundamentadas na base inicialmente criada. Uma linguagem proeminente que adota esse paradigma de forma exclusiva é o *Prolog*, que possui diversas aplicações na inteligência artificial.

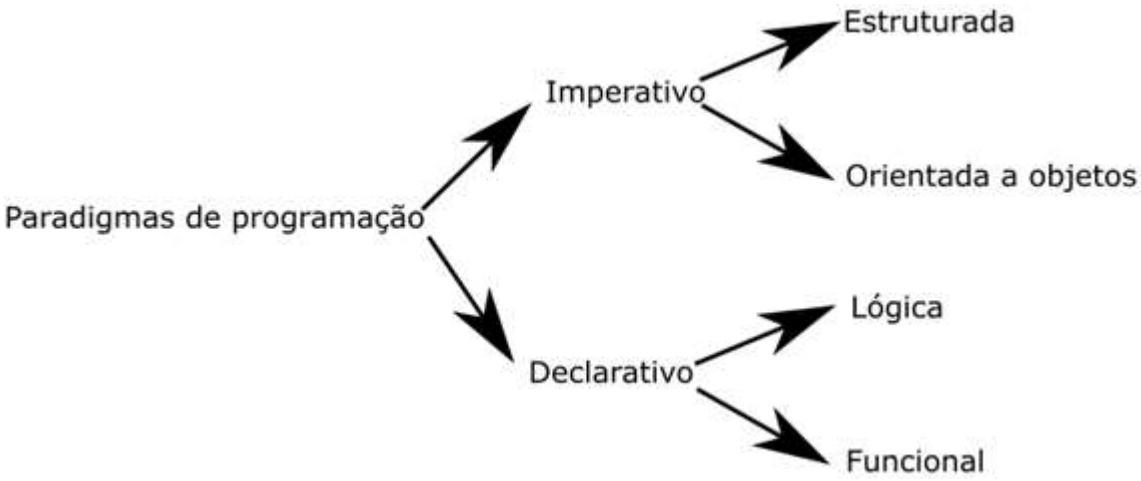
Em meados dos anos 60, um pesquisador chamado Alan Kay, influenciado por sua formação em biologia e matemática, além de outras tecnologias da época, como *Sketchpad* e *Simula*, pensou em uma nova arquitetura de programação que chamou de **paradigma orientado a objetos**. Em sua concepção original, a programação orientada a objeto deveria se basear em células independentes trocando mensagens entre si, retirando o foco dos dados. As linguagens de programação *Simula* e *Smalltalk* foram as primeiras a adotar as práticas propostas por Alan Kay.

Os problemas computacionais são muito diversos e, com isso, foram desenvolvidos paradigmas diferentes para classes diferentes de problemas. Portanto cada paradigma é ótimo para um tipo de situação diferente, de modo que um programador versado em diversos paradigmas terá um ferramental maior na hora de gerar soluções para problemas complexos. As linguagens modernas de

programação combinam estratégias de diferentes paradigmas, oferecendo diversas opções para o desenvolvimento de algoritmos.

Nesta disciplina, vamos abordar o paradigma orientado a objetos que é muito popular em soluções comerciais hoje, por ser especialmente adequado para projetos de grande escala que necessitam de constante manutenção e ampliação. Para ilustrar e classificar os principais paradigmas de programação, veja a Figura 1.

Figura 1 – Representação dos paradigmas de programação



Na Figura 1, temos representado os paradigmas de tipo Imperativo, voltados para representar os comandos que resolvem o problema, focado no COMO resolver. Em contrapartida temos os paradigmas do tipo declarativos, que não focam nas mudanças de estado sequencial de um programa, mas sim O QUE se deve resolver. Dessas duas classes derivam os paradigmas que discutimos até aqui.

1.1 PARADIGMA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

O paradigma orientado a objeto foi pela, primeira vez, aplicado de forma adaptada na linguagem de programação *Simula 67*, nos anos de 1960, posteriormente também sendo utilizada de forma exclusiva na linguagem *Smalltalk*, da *Xerox*. Sua grande popularidade influenciou todas as principais linguagens de programação de hoje, *C++, C#, PHP, Python* e *Java* que é a linguagem base de nossa disciplina.

O pesquisador Alan Kay, que atuou na empresa *Xerox*, foi quem liderou o projeto que encabeçou as primeiras linguagens de programação baseada em orientação a objetos e inclusive cunhou o termo em si. Alan Kay propôs uma abstração em que a programação de computadores poderia ser encarada como um organismo vivo, no qual cada célula é entendida como um elemento independente, relacionando-se com outras células de forma a manter o funcionamento de todo um organismo.

Alan Kay expandiu essa ideia para outras relações além das intercelulares, a forma como entendemos o mundo por meio é por meio das relações de objetos e pessoas, cada um com suas características individuais, realizando ações uns sobre os outros, permitindo a construção de sistemas complexos. Essa forma de pensar é natural e intuitiva para nós, pois emula como vemos o mundo.

Em programação estruturada o foco está nas ações, por exemplo: em um programa de computador de vendas, o conjunto de instruções que efetua a compra de itens por um cliente geralmente seria agrupado em uma função chamada ***comprar()***. Porém, em um sistema orientado a objetos, pensamos primeiro no objeto, "Quem está realizando essa compra?", e teríamos um objeto cliente com todos os seus dados em específico e teríamos um comando como ***cliente.comprar()***. Associamos sempre os objetos (cliente neste exemplo) na ação (comprar), que deixa mais claro e intuitiva a leitura dos códigos, pois tem maior contexto pensar em um "cliente comprando" do que simplesmente na ação comprar isolada, trazendo o código mais perto de como entendemos o mundo. Associando essa prática com outros conceitos como herança, polimorfismo, encapsulamento, entre outros, promovemos ganhos de produtividade especialmente na manutenção de códigos pela maior facilidade no entendimento.

No próximo Tema, será apresentada a linguagem de programação Java, que é completamente voltada ao paradigma de orientação a objetos.

TEMA 2 - HISTÓRIA DO JAVA

Neste Tema, vamos discutir o histórico, origem e importância da linguagem Java, de modo que ajude na compreensão da função dos principais atores tecnológicos da computação.

2.1 PRIMEIROS ANOS

A Linguagem Java surgiu no início dos anos 90, em uma importante empresa de tecnologia chamada *Sun Microsystems*. A equipe responsável pelo desenvolvimento dessa linguagem atuava no chamado *Green Project* e foram liderados pelo cientista da computação James Gosling.

O *Green Project* tinha por objetivo gerar tecnologias voltadas para conectividade de equipamentos domésticos. O primeiro produto foi um dispositivo chamado StarSeven, que, semelhante a um tablet, poderia receber comandos de toque e controlar os demais dispositivos. A comunicação entre os dispositivos deveria se dar por meio de uma linguagem de programação que fosse independente de plataforma. Sendo assim, equipe de Gosling, a princípio, tentou adaptar o C++ para essa tarefa, mas, por fim, optaram por desenvolver uma linguagem própria que chamaram, na época, de Oak (traduz como carvalho).

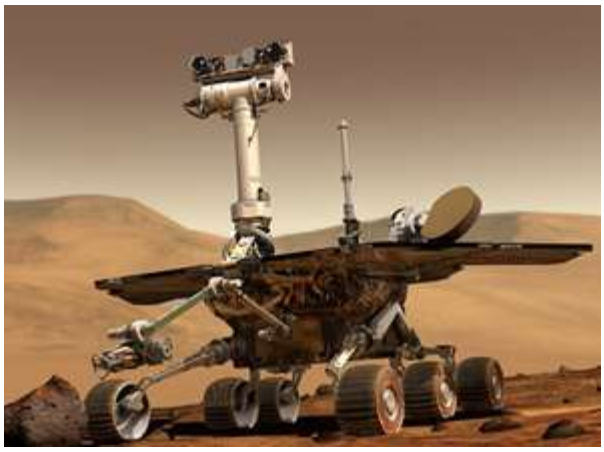
A linguagem Oak possuía o diferencial de ser uma linguagem interpretada por uma máquina virtual fornecida pela Sun. Todo o dispositivo que rodasse a máquina virtual da Sun, seria capaz de executar códigos Oak sem necessidade de compilação específica para o dispositivo em questão. Embora o projeto de conectividade doméstica não tenha emplacado como planejado, a tecnologia de uma linguagem independente de plataforma casou muito bem com outra demanda que surgiu na mesma época, que foi a popularização da internet e dos navegadores. Por questões legais e de registro de marca, a linguagem Oak, em 1995, mudou seu nome para Java.

Com isso se popularizou muito o uso de pequenos programas chamados *applets*, que eram baixados de um servidor web e poderiam ser executados na máquina dos clientes independentemente da plataforma, desde que ele possuísse uma máquina virtual Java previamente instalada.

2.2 O JAVA MODERNO

Na década de 1990 e 2000, a popularização da internet levou uma grande popularização da linguagem Java, que recebeu suporte de grandes companhias de informática, como IBM. E, de certa forma, o objetivo inicial Green Project foi atingido com Java sendo utilizado para conectar todo o tipo de dispositivo móvel, celulares, tablets, computadores, e até uma das primeiras sondas espaciais robóticas a atingir solo marciano em 2004 utilizou linguagem Java. Na Figura 2, temos um exemplo de bastante destaque de uma aplicação do Java, a sonda espacial *Opportunity*.

Figura 2 – Opportunity, sonda espacial que realizou missão exploratória do solo marciano



Fonte: NASA/JPL/Cornell University, Maas Digital LLC/C.C.

A linguagem Java adotou licença de software livre GPL v3[1] em 2006, o que significa que os programas feitos pela Sun para permitir o funcionamento do Java, assim como suas bibliotecas, possuem código aberto para consulta, cópia e modificação, desde que o desenvolvedor que faça modificações também disponibilize seu código livremente. A *Sun Microsystems* foi adquirida pela Oracle, em 2010, que é quem oferece suporte ao Java até hoje.

Embora *applets* não sejam mais usualmente adotadas, a popularidade do Java se mantém, ela é adotada nos aplicativos do sistema operacional Android, diversos tipos de servidores, leitores de livros digitais como Kindle, TV digital DTVI e até o tradicional programa do Imposto de renda brasileiro dentre outros muitos exemplos.

No momento da edição deste documento, a linguagem Java se encontra na versão 13, lançada em setembro de 2019 e disponibilizado no site da Oracle[2]. Trata-se de uma linguagem Orientada a Objetos com sintaxe baseada na linguagem C. No próximo Tema discutiremos questões técnicas da arquitetura e organização das soluções Java.

TEMA 3 - ORGANIZAÇÃO DO JAVA

Neste Tema, vamos debater, em detalhes, a tecnologia Java em si. Mais do que uma linguagem e bibliotecas, o Java necessita de um ambiente próprio para seu funcionamento. Ele também acompanha um conjunto completo de programas que iremos apresentar.

Tradicionalmente as linguagens de programação passam por um processo denominado *compilação*, que transforma o código alto nível escrito pelo programador no que chamamos de código de máquina, ou binário. Esse código nativo é lido pelo processador que executa as instruções. Os programas .exe do windows são um exemplo de binário.

Em contrapartida existem também linguagens que são interpretadas que não passam por esse processo de compilação, de modo que o código escrito pelo programador em tempo de execução é traduzido para código de máquina.

Códigos interpretados são essencialmente menos eficientes do que códigos compilados, pela quantidade extra de instruções que requer sua interpretação. Porém possuem a vantagem de serem facilmente portados para diferentes plataformas justamente por não o necessitar de recompilação para cada plataforma, por exemplo um mesmo código interpretado sem alterações pode ser executado em diferentes computadores com sistema operacional Linux, Windows ou Mac.

Quanto ao Java dependendo do ambiente de execução é possível trabalhar com ele tanto interpretado quanto compilado. Porém tipicamente ele funciona em um processo em dois passos. Primeiro o código alto nível é compilado para um conjunto de instruções internas do java denominado *bytecode*. Esse código *bytecode*, posteriormente, é interpretado por um programa chamado *máquina virtual Java*, em inglês *Java Virtual Machine*, e ao longo das aulas chamaremos pela sigla JVM. As JVM para as principais plataformas são mantidas pela Oracle, mas podem ser desenvolvidas de forma independente para os mais diversos dispositivos por qualquer equipe, visto que possui licença livre. Portanto um mesmo *bytecode* pode ser executado em qualquer sistema que possua uma JVM.

Pelo fato de o Java utilizar JVM para interpretar seus *bytecodes*, existe uma perda em desempenho quando comparado a um código compilado nativo. Porém as JVM evoluíram muito ao longo dos anos, e uma das principais tecnologias nesse sentido é o chamado *Hotspot*.

Estudos estatísticos mostram que, na grande maioria dos programas, 80% do processamento se concentra em somente 20% do código. O *Hotspot* é uma tecnologia que identifica esses trechos de código com muito processamento e executa uma compilação dos mesmos durante a execução do código. Essa tecnologia de compilação em tempo de execução é chamada de *Just in time compilation*, mais conhecida pela sigla JIT. A combinação das duas tecnologias, dentre outras

melhorias, tornou o Java muito eficiente, diminuindo a distância em relação às linguagens compiladas.

No contexto da computação, nós chamamos de *Benchmark* os testes que buscam comparar desempenho. Na Tabela 1 vemos um *benchmark* do Java comparado ao C++ em três algoritmos diferentes. Em um dos testes o Java chega a apresentar um tempo de execução menor do que o C++, embora no caso geral o C++, por ser uma linguagem compilada, apresenta desempenho em tempo e uso de memória significativamente melhores.

Tabela 1 – *Benchmark* [3] comparando Java ao C++ em três algoritmos distintos. Versão dos compiladores: Java, openjdk13 17-9-2019; C++, g++ 9.2.1-9ubuntu2

Algoritmo: Reverse-complement		
Linguagem	**Tempo em segundos**	**Memória (bytes)**
Java	3.16	712.368
C++	4.17	500.036
Algoritmo: Fannkuch-redux		
Linguagem	**Tempo em segundos**	**Memória (bytes)**
Java	14.33	30.888
C++	10.69	1.896
Algoritmo: Fasta		
Linguagem	**Tempo em segundos**	**Memória (bytes)**
Java	2.22	45.172
C++	1.46	2.216

Quando se deseja apenas executar *bytecodes* do Java, é necessário instalar em sua máquina o Ambiente de execução Java, em inglês, *Java Runtime Environment,* mais conhecido pela sigla JRE. Ela é composta principalmente pela JVM e bibliotecas padrão do Java, podendo ser encontrada para as mais diversas plataformas na página oficial da Oracle.

Agora quando desejamos programar em Java, precisamos instalar o Kit de desenvolvimento Java, em inglês *Java Development Kit*, mais conhecido pela sigla JDK. Ele é composto por um conjunto de utilitários como o compilador de *bytecode* além de uma JRE. A JDK também é encontra na página oficial da Oracle.

Neste Tema, vimos muitas siglas e termos, e para facilitar o entendimento e servir de rápida referência, vamos agrupar todos no quadro 2.

Quadro 2 – Termos e siglas relacionados ao Java

Nome	Tradução	Definição
Java Virtual Machine ***JVM***	Máquina Virtual Java	Programa responsável por interpretar e executar código *Bytecode* Java.
Bytecode	Código em byte	O equivalente ao executável Java, o *Bytecode* é gerado após o processo de compilação dos códigos fontes Java.
Java Development Kit ***JDK***	Kit de desenvolvimento Java	Conjunto de bibliotecas, compiladores e demais ferramentas para o desenvolvimento de programas Java.
Java Runtime Environment ***JRE***	Ambiente de execução Java	Conjunto de biblioteca padrão Java e JVM para execução de códigos *Bytecode*.
Just in time compilation ***JIT***	Compilação dinâmica	Técnica que permite a JVM compilar partes críticas do código em linguagem de máquina em tempo de execução. Oferecendo significativo ganho de memória
Garbage Collection	Coletor de Lixo	Técnica que isenta o programador da responsabilidade de desalocar memória, a JVM regularmente se encarrega de liberar memória alocada não utilizada.

No próximo Tema, daremos sequência ao nosso estudo e faremos nosso primeiro código Java.

TEMA 4 – VERSÕES DO JAVA E PRIMEIRO CÓDIGO

Neste Tema, vamos, finalmente, colocar a mão na massa, fazendo nosso primeiro projeto Java por meio do programa Eclipse.

4.1 VERSÕES DO JAVA E A IDE ECLIPSE

No caso do Java, assim como a grande maioria das linguagens de programação, é possível codificar utilizando qualquer editor de texto e, posteriormente, por meio de um compilador dedicado, gerar o seu binário (*Bytecode* no caso do Java). Porém é muito mais produtivo, especialmente em projetos de grande escala, utilizar um programa próprio direcionado ao desenvolvimento de códigos que combine editor de texto, compilador, depurador, bibliotecas entre outras funcionalidades. Esse tipo de programa é conhecido como um ambiente de desenvolvimento integrado, do inglês *Integrated Development Environment* ou apenas IDE.

Existem diversas IDEs de excelente qualidade para o Java, aqui destacamos o Netbeans, da própria Oracle, o IntelliJ IDEA da empresa JetBrains, e o mais popular de todos o Eclipse, inicialmente da IBM mas hoje segue *licença software livre*. O Eclipse é o que utilizaremos em nossas aulas, mas todas funcionam de forma análoga.

Saiba mais

O Eclipse originalmente desenvolvido como IDE para Java foi adaptado por *plugin-ins* desenvolvidos pela comunidade para as mais diversas linguagens de programação, ele é encontrado no site: <https://www.eclipse.org/>. Acesso em: 30 jan. 2021.

O Java conta com três versões principais, *Java Micro Edition* (ME), *Standard Edition* (SE) e *Enterprise Edition* (EE). A seguir, mencionamos suas características:

- O **Java ME** visa a construção de softwares para dispositivos embarcados, sistemas de propósito específico com poucos recursos computacionais. Ela é compatível com uma biblioteca básica de classes e se torna especialmente importante no contexto de soluções desenvolvidas pensando na internet das coisas.
- **Java SE** é a edição padrão do Java com o principal conjunto de bibliotecas, perfeita para desenvolver programas desktop e de console. Por console entenda programas com interface puramente em modo texto que são geralmente executados por prompt de comando sistema operacional Windows ou terminal do Linux.
- Por fim **Java EE** é a edição mais completa, já vem equipada com bibliotecas prontas para soluções empresariais especialmente voltadas para internet e banco de dados. Trata-se de uma

série de especificações que foi desenvolvida integralmente ou parcialmente na forma de servidor de aplicações por diversos fornecedores. Uma importante tecnologia que ajuda a formar a espinha dorsal da internet hoje.

A IDE do Eclipse conta com diversas opções para instalação. Para nossa disciplina, busque a versão SE ou EE.

4.2 PRIMEIRO CÓDIGO

As principais IDEs incluindo Eclipse possuem o conceito de projeto. O projeto engloba todas as classes e bibliotecas necessárias para a geração de um programa. Na Figura 3, vemos a estrutura de um projeto Java na IDE Eclipse.

Os principais elementos que compõem o projeto são:

- Bibliotecas: Que são *bytecodes* com funcionalidades específicas implementadas. Permitem o programador reaproveitar códigos geralmente desenvolvidos por equipes diversas e que já são muito bem testados e eficientes. No caso o Eclipse já inclui uma biblioteca básica em todos os projetos.
- Pacotes: Um conceito semelhante ao de pasta/diretório para organizar a estrutura dos códigos Java. Supondo que temos um projeto grande com muitos códigos, podemos agrupar os arquivos ligados a bancos de dados em um pacote e que estão ligados a interface visual em outro por exemplo. Usualmente o pacote principal de um projeto é nomeado com o inverso do domínio da sua instituição. Por exemplo *empresa.com* se torna *com.empresa* essa é uma prática comum porém não é necessário. Esse pacote principal fica inserido dentro de uma pasta nomeada src (que vem da palavra *source*, código em inglês).
- Classe: Os códigos são descritos em arquivos com extensão *.java* e geralmente temos um arquivo por classe.

Figura 3 – Estrutura de projeto básico Eclipse com uma única classe

A seguir os passos necessários para iniciar um projeto:

1. Uma vez aberto o programa Eclipse para criar um projeto novo basta ir na opção, **File/New Project** dê um nome apropriado ao projeto e utilize as opções padrões.
2. O projeto criado estará vazio, para criar uma primeira classe java que receberá o método principal basta clicar com o botão direito sobre a pasta com nome do projeto e ir na opção **New/Class** dê um nome para a classe e um nome para o pacote.
3. Por se tratar da primeira classe com o método principal (equivalente ao main do C/C++) é interessante marcar no checkbox a opção **public static void main(String[] args)**.

Um arquivo com aproximadamente o seguinte código deve surgir:

```
package com.empresa;
public class PrimeiraClasse {
      public static void main(String[] args) {
      }
}
```

Vamos analisar linha por linha o código acima:

- **package com.empresa;**

Indica o nome do pacote na qual a classe está.

- **public class PrimeiraClasse**

Esta é a linha na qual se informa o nome da classe. O comando **public** indica que a classe pode ser acessada de forma pública por outras classes, esse conceito de classes públicas e privadas e suas implicações serão discutidos em detalhes posteriormente.

- **public static void main(String[] args) {**

Esta linha é a declaração do método, **static** indica que o método pertence à classe e não ao objeto, o conceito de métodos estáticos será discutido com detalhes em outra aula. **main** é o nome do método principal, equivalente à função principal em linguagens como C/C++ e indica que esse

método será o primeiro a ser executado pelo programa. **String [] args** é a declaração de um array de objetos da Classe String como parâmetro de entrada do método. Caso o programa seja executado em modo console eventuais parâmetros de execução na chamada do programa, serão lidos e direcionados para a variável **args**.

As chaves **{ }** representam blocos de código, marcam onde começa e termina a classe e onde começa e termina o método.

Como primeiro comando escreva dentro do bloco de código do método main o seguinte comando:

```
System.out.println("Alo Mamae");
```

Esse é o comando que imprime a mensagem que estiver entre aspas na tela em modo console.

Para executar e testar o seu primeiro programa vá na opção **Run/run** ou utilize o atalho CTRL + F11.

Se tudo ocorrer bem verá a sua mensagem impressa na tela. Parabéns pelo seu primeiro programa Java!

Dica: O Eclipse conta com um atalho para o comando acima, basta escrever **sout** e fazer o comando **CTRL + ESPAÇO.** Existem diversos atalhos dentro no Eclipse, procure os principais na internet ou no próprio Eclipse dentro de **Help/Show Active Keybindings**. Dominar alguns atalhos contribuirá para um aumento significativo na produtividade em longo prazo.

Alguns dos comandos Java parecem longos e intimidadores quando comparados com os de outras linguagens, porém com a experiência se notará que segue uma padronização que se torna intuitiva e clara com o tempo. Java conta com uma biblioteca padrão muito extensa que permite um grande reaproveitamento de códigos.

No próximo Tema, faremos um apanhado geral sobre os principais comandos Java.

TEMA 5 – VISÃO GERAL SOBRE O CÓDIGO JAVA

Neste Tema vamos avançar a discussão sobre a linguagem Java. Sem entrar em detalhes sobre a orientação a objetos ainda vamos passar brevemente sobre os principais comandos e estruturas de dados disponíveis nessa linguagem.

5.1 PRINCIPAIS COMANDOS

A sintaxe da linguagem Java é baseada em C/C++, portanto quem as conhece automaticamente já domina grande parte dos principais comandos Java. Abaixo seguem alguns exemplos:

Entrada e saída:

Já vimos, no Tema 4, um primeiro comando de impressão no console, porém, a seguir, vemos algumas de suas variantes:

```
System.out.print("msg1"); //Imprime uma mensagem
System.out.println("msg2"); //Imprime uma mensagem e pula linha.
System.out.printf("msg3 %d",10); //Imprime mensagens formatadas,
análogo ao printf da linguagem C.
```

Quanto à leitura de dados, é necessário realizar alguns passos. A seguir temos um código Java que utilizaremos para exemplificar.

```
package com.empresa;
import java.util.Scanner; // Obs 1
public class ExemploLeitura {
	public static void main(String[] args) {
		Scanner teclado = new Scanner(System.in); // Obs 2
		System.out.println("Digite sua idade:");
		int idade=teclado.nextInt(); // Obs 3
```

Primeiro precisamos importar a biblioteca *java.util.Scanner,* que possui as definições da classe Scanner, necessária para a leitura, veja o código abaixo na linha com a **Obs 1** o comando **import** é utilizado para esta tarefa, as importações devem vir sempre logo abaixo da declaração do nome do pacote.

Segundo declaramos uma variável (objeto) do tipo (classe) Scanner. Essa classe é responsável por ler os dados de alguma fonte passada por parâmetro, no caso o parâmetro deve ser *System.in*, que aponta a entrada padrão do sistema, o teclado, veja a **Obs 2** no código, o nome do objeto pode ser qualquer um, no caso optou-se pelo nome *teclado*.

Terceiro passo, devemos utilizar o objeto que declaramos para fazer a leitura. Para ler um valor inteiro *teclado.nextInt()* e para um valor real *teclado.nextFloat()* ou *teclado.nextDouble()* para ler uma string *teclado.next()*. Veja a **Obs 3** no código para um exemplo.

Comandos de desvio:

Assim como nas principais linguagens de programação o principal comando de desvio é o **if().** Entre parênteses colocamos a expressão associada, se a expressão for verdadeira o fluxo do código é desviado. As variantes com **else if** e a variante com **else** também estão presentes na linguagem. No exemplo a seguir, o código que imprime verifica o valor de uma variável idade e imprime a mensagem *Criança, Adolescente* ou *Adulto* dependo do valor.

```
if(idade < 10) {
	System.out.println("Criança");
}
else if(idade < 18) {
	System.out.println("Adolescente");
}
else {
	System.out.println("Adulto");
}
```

O comando *switch case* também é presente na linguagem Java. Permitindo desvio de fluxo. Primeiro a expressão dentro do *switch* é avaliada e o código é desviado o *case* pertinente ou default caso nenhum se adeque. A seguir um exemplo genérico do seu uso.

```
switch (expressao) {
    case constante1:
      // comandos primeiro caso
      break;
    case constante2:
      // comandos segundo caso
      break;
    default:
      //caso padrão
}
```

Comandos de repetição:

Os principais comandos de repetição também estão presentes em Java. *while, do-while* e *for.* A sintaxe é a mesma do C/C++.

```
while (condição) {
  //Bloco de código executado
}

do {
  //Bloco de código executado
}while (condição);

for (inicialização ; expressão enquanto ; interação ) {
  //Bloco de código executado
 }
```

5.2 TIPOS DE DADOS

No Java, os dados são armazenados em primitivas que são os tipos básicos presentes nas principais linguagens de programação. E em um nível de abstração maior os dados podem ser armazenados também em tipos não primitivos como String, Array e Classes. No Quadro 3, temos as primitivas.

Quadro 3 – Tipos básicos no Java

Tipo	Tamanho	Descrição
byte	1 byte	Números inteiros (-128 até 127)
short	2 bytes	Números inteiros (-32.768 até 32.767)
int	4 bytes	Números inteiros (-2.147.483,648 até 2.147.483.647)
long	8 bytes	Números inteiros (-9.223.372.036.854.775.808 até 9.223.372.036.854.775.807)
float	4 bytes	Armazena números inteiros e fracionários até 6 a 7 dígitos decimais.
double	8 bytes	Armazena números inteiros e fracionários até 15 dígitos decimais.
boolean	1 bit	Armazena apenas 0 ou 1 (false ou true)
char	2 bytes	Armazena um único caractere, letra Stores a single character/letter or ASCII values

String:

As strings, ou sequência de caracteres, no Java são representadas com uma classe chamada justamente de *String*. No Java constantes da classe String devem ser escritas entre aspas duplas. Elas possuem diversos métodos internos. Neste código temos um exemplo com comentários:

```
String msg = "mario"; //Declaração da string

msg = "super " + msg; //Concatenação, msg virou “super mario”
int tamanho = msg.length(); //Total de caracteres no caso 11
msg = msg.toUpperCase();//Todas as letras ficam maiúsculas
System.out.println(msg);//Impressão
```

Abaixo mais alguns exemplos de exercícios:

- Código que descobre se o texto de uma string aparece em outra:

```
String str1 = "frase de exemplo para teste";
String str2 = "exemplo";
System.out.println("String original: " + str1);
System.out.println("String que será buscada: " + str2);
// str1 está contida na str2 portanto entrará no if
if(str1.contains(str2)){
      System.out.println("Palavra aparece na string1");
}else{
      System.out.println("Palavra não aparece na string1");
}
```

- Código que compara duas Strings, observe que a String por ser um tipo primitivo (int,float,char,...) não pode ser simplesmente comparado utilizando == mas sim conta com um método próprio chamado *equals*:

```
String str1 = "mario";
String str2 = "Mario";
// str1 é diferente da str2 devido a letra maiúscula, código entra
no else
if(str1.equals(str2)){
      System.out.println("Palavras iguais");
}else{
      System.out.println("Palavras diferentes");
}
// para que a comparação ignore maiúscula e minúscula é possível
transformar as palavras inteiramente em maiúsculas antes de comparar
no exemplo abaixo a comparação será verdadeira e entrará no if

if(str1.toUpperCase().equals(str2.toUpperCase())){
      System.out.println("Palavras iguais");
}else{
      System.out.println("Palavras diferentes");
}
```

Arrays:

Os arrays também são essencialmente classes e trabalham de forma análoga ao C/C++. Eles podem ser declarados entre chaves e acessados com colchetes indexados a partir do valor zero. Segue o exemplo:

```
String[] nomes = {"Mario", "Luigi", "Peach", "Yoshi"};
nomes[0] = "Bowser";
System.out.println(nomes[0]);// no array nomes posição zero será
impresso a palavra Bowser
```

Os arrays também contam com diversos métodos, por exemplo o métodos *length()* que como na classe String retorna a quantidade de itens do array. Considerando o exemplo acima, o comando

nomes.length() retornaria o número 4.

A classe Arrays possui uma série de métodos que podem ser utilizados para funções comuns com arrays, tal qual ordenação, preencher os elementos com um mesmo valor, comparar arrays, entre outros. Para ter acesso a essas funções, é necessário primeiro realizar o comando import java.util.Arrays e, na sequência, acessar com o comando Arrays.

Ao colocar o ponto depois da palavra Arrays, o Eclipse deve apresentar as várias opções, como desafio fica a proposta para o(a) aluno(a) investigar e junto da documentação encontrada na internet descobrir o que algumas dessas funções fazem.

FINALIZANDO

Nesta aula, iniciamos nosso aprendizado introduzindo o contexto histórico dos paradigmas de programação e da linguagem Java. Aprendemos a arquitetura da linguagem, quais suas versões e fizemos nosso primeiro programa. Por fim realizamos uma visão geral sobre a linguagem Java, os principais comandos.

Neste momento ainda não entramos em contato direto com Programação Orientada a Objetos em si, mas preparamos o terreno oferecendo o ferramental para nos aprofundarmos utilizando o Java como base para o estudo. Posteriormente, exploraremos mais da criação e utilização das classes, o principal conceito dentro da programação orientada a objetos.

REFERÊNCIAS

DEITEL, P.; DEITEL, H. **Java Como programar**. 10. ed. São Paulo: Pearson, 2017.

SINTES, T. **Aprenda programação orientada a objetos em 21 dias**. 5. reimpressão. São Paulo: Pearson Education do Brasil, 2014.

LARMAN, C. **Utilizando UML e Padrões**: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo. 3. ed. Porto Alegre: Bookman, 2007.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Pearson, 2011.

BARNES, D. J.; KÖLLING, M. **Programação orientada a objetos com Java**. 4. ed. São Paulo: Pearson Prentice Hall, 2009.

MEDEIROS, E. S. de. **Desenvolvendo software com UML 2.0**: definitivo. São Paulo: Pearson Makron Books, 2004.

PAGE-JONES, M. **Fundamentos do desenho orientado a objetos com UML**. São Paulo: Makron Book, 2001

PFLEEGER, S. L. **Engenharia de software**: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

---

[1]<https://www.gnu.org/licenses/gpl-3.0.en.html>. Acesso em: 30 jan. 2021.

[2] <https://www.oracle.com/java/technologies/javase-downloads.html>. Acesso em: 30 jan. 2021.

[3] Fonte: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/java-gpp.htm