PROGRAMAÇÃO VOLTADA A OBJETOS

AULA 1

Prof. Leonardo Gomes

CONVERSA INICIAL

Nesta etapa, 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 este estudo como um todo.

Ao final desta etapa, esperamos atingir os seguintes objetivos que serão avaliados ao longo do estudo da forma indicada:

Quadro 1 – Objetivos

Objetivos	**Avaliação**
1 – Contextualização sobre o paradigma de programação orientado a objetos e à 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

Fonte: Gomes, 2020.

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

Neste tópico, 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 1940, 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**, o **paradigma funcional**, o **paradigma lógico** e o **paradigma orientado a objetos**. Geralmente, iniciamos o aprendizado na programação por meio do **paradigma procedural**, e a linguagem C utiliza exclusivamente esse paradigma. Nele, 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 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. Tal paradigma ainda é muito utilizado em certas linguagens que combinam paradigmas como a linguagem *Scala*.

Outro importante paradigma que surgiu na década de 1950 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.

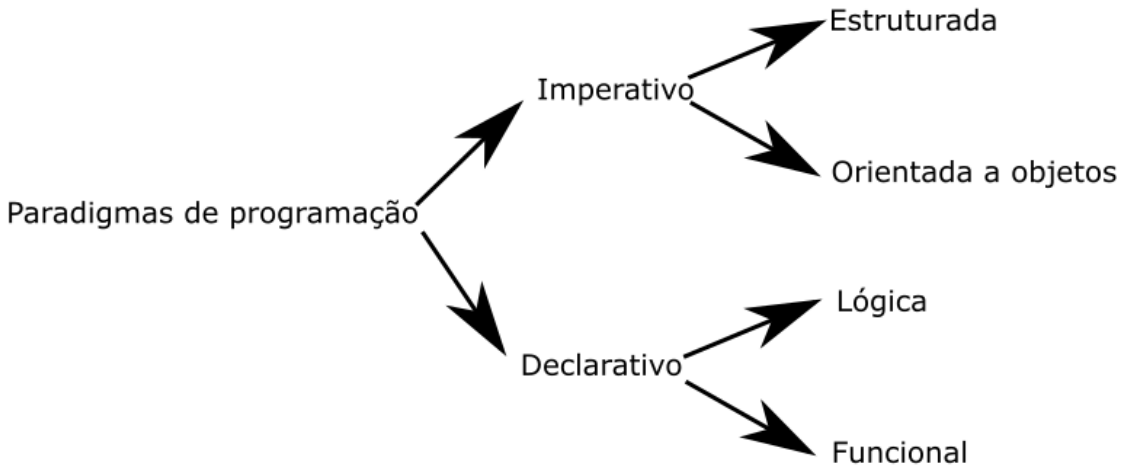
Em meados dos anos 1960, um pesquisador chamado Alan Kay, influenciado por sua formação em biologia e matemática, além de outras tecnologias da época, como o *Sketchpad* e a *Simula*, pensou em uma nova arquitetura de programação que chamou de **paradigma orientado a objetos**. Em sua concepção original, a programação orientada a objeto deveria se basear em células independentes trocando mensagens entre si e retirando o foco dos dados. As linguagens de programação *Simula* e *Smalltalk* foram as primeiras a adotar as práticas propostas por Alan Kay.

As linguagens modernas de programação se inspiram e combinam esses paradigmas oferecendo diversas opções de estratégias para o desenvolvimento de algoritmos.

As soluções aplicadas em um paradigma podem ser de forma direta ou indireta representadas em outro, tornando cada paradigma ótimo para um tipo de situação diferente. Um programador versado em diversos paradigmas terá um ferramental maior na hora de gerar soluções para problemas complexos.

Neste estudo, 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.

Figura 1 – Representação dos paradigmas de programação



Fonte: Gomes, 2020.

Imperativo são os paradigmas voltados para representar os comandos que resolvem o problema, focado em **como** resolver. Em contrapartida, temos os paradigmas declarativos que não focam em mudanças de estado sequencial de um programa, mas sim **no 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 utilizado 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 do nosso estudo.

O pesquisador Alan Kay, que atuou na empresa *Xerox*, foi quem liderou o projeto que encabeçou as primeiras linguagens de programação baseadas em orientação a objetos, sendo inclusive responsável por cunhar o termo.

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, pois a forma como entendemos o mundo se dá por meio das relações de objetos e pessoas, cada um com suas características individuais, realizando ações uns sobre os outros e 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: em um programa de computador de vendas, por exemplo, 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 dados dos clientes, bem como um comando como ***cliente.comprar()***. Assim, associamos sempre os objetos (cliente neste exemplo) à ação (comprar), que deixa mais clara e intuitiva a leitura dos códigos, pois há maior contexto quando pensamos em um "cliente comprando" que simplesmente na ação de comprar isolada, trazendo o código mais perto de como entendemos o mundo. Associando essa prática a outros conceitos, como herança, polimorfismo, encapsulamento, entre outros, promovemos ganhos de produtividade, especialmente na manutenção de códigos pela sua maior facilidade no entendimento.

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

TEMA 2 – HISTÓRIA DO JAVA

Neste tópico vamos discutir o histórico, a origem e a importância da linguagem Java. Conhecer este contexto nos ajuda a compreender a função dos principais atores tecnológicos da computação.

2.1 Primeiros anos

A Linguagem Java surgiu no início dos anos 1990 em uma importante empresa de tecnologia chamada Sun Microsystems.

A equipe responsável pelo desenvolvimento desta linguagem atuava no chamado Green Project e foi liderada 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. A equipe de Gosling a princípio tentou adaptar o C++ para esta tarefa, mas, por fim, optou por desenvolver uma linguagem própria, que chamaram na época de Oak (do inglês, carvalho).

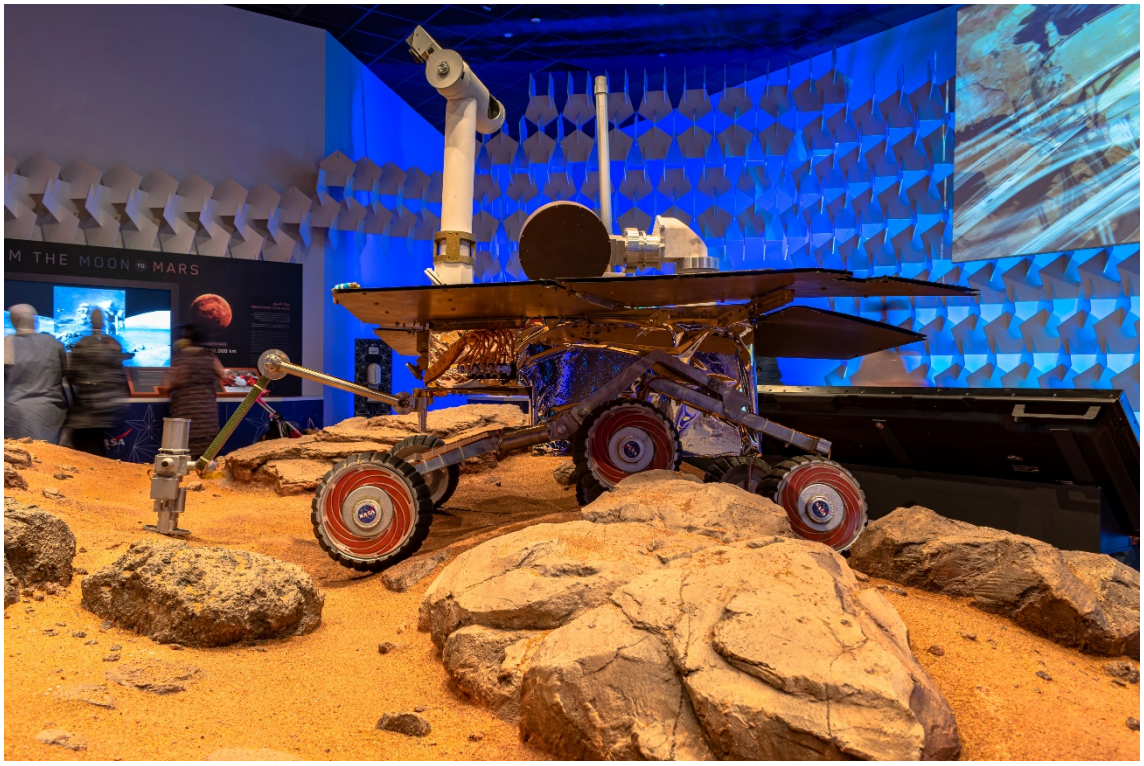
A linguagem Oak possuía o diferencial de ser uma interpretada por uma máquina virtual fornecida pela Sun, e todo o dispositivo que rodasse a máquina virtual da Sun seria capaz de executar códigos Oak sem a necessidade de compilação específica para o dispositivo em questão. Embora o projeto de conectividade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

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

2.2 O Java moderno

Nas décadas de 1990 e 2000, a popularização da internet levou a uma grande popularização da linguagem Java, que recebeu suporte de grandes companhias de informática, tais como a IBM. De certa forma, o objetivo inicial do Green Project foi atingido com o Java sendo utilizado para conectar todo o tipo de dispositivo móvel, tais como celulares, tablets, computadores e até mesmo uma das primeiras sondas espaciais robóticas a atingir o solo marciano em 2004.

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



Créditos: rzulev/Shutterstock.

A linguagem Java adotou licença de software livre GPL v3 em 2006, o que significa que os programas feitos pela Sun para permitir o funcionamento do Java e de 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, de forma que essa linguagem é adotada nos aplicativos do sistema operacional Android, em diversos tipos de servidores, em leitores de livros digitais como Kindle e TV digital DTVI e até no tradicional programa de 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. Trata-se de uma linguagem Orientada a Objetos com sintaxe baseada na linguagem C. No próximo tópico, discutiremos questões técnicas da arquitetura e organização das soluções Java.

TEMA 3 – ORGANIZAÇÃO DO JAVA

Agora, vamos debater em detalhes a tecnologia Java em si. Mais que uma linguagem e bibliotecas, o Java necessita de um ambiente próprio para o seu funcionamento.

Ele também acompanha um conjunto completo de programas que também iremos apresentar. Tradicionalmente, as linguagens de programação passam por um processo denominado compilação, o qual transforma o código de 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. O código escrito pelo programador em tempo de execução é traduzido para código de máquina.

Códigos interpretados são essencialmente menos eficientes que códigos compilados pela quantidade extra de instruções que requer a interpretação dele. Porém, possuem a vantagem de serem facilmente portados para diferentes plataformas justamente por não necessitarem de recompilação para cada plataforma. Dessa forma, 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, ele tipicamente funciona em um processo em dois passos. Primeiro, o código de alto nível é compilado para um formato chamado *bytecode*. Esse código *bytecode* posteriormente é interpretado por um programa chamado máquina virtual Java, em inglês *Java Virtual Machine*. Ao longo dos nossos estudos, 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 possuem 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 neste 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 e diminuiu a distância em relação às linguagens compiladas. No contexto da computação, nós chamamos de *Benchmark* os testes que buscam comparar desempenho. Na figura 3, 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 melhor que o C++. No link presente na legenda da figura, mais testes podem ser vistos.

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

reverse-complement

source	secs	mem	gz	busy	cpu load
Java	**3.16**	712,368	2183	7.08	65% 47% 42% 70%
C++ g++	4.71	500,036	840	4.79	14% 86% 0% 1%

fannkuch-redux

source	secs	mem	gz	busy	cpu load
Java	14.33	34,888	1282	56.56	99% 98% 99% 98%
C++ g++	10.69	1,896	980	42.28	100% 100% 96% 100%

fasta

source	secs	mem	gz	busy	cpu load
Java	2.22	45,172	2473	5.99	61% 50% 98% 60%
C++ g++	1.46	2,216	2711	4.40	75% 75% 76% 75%

Fonte: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/java-gpp.html>.

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. Ele é composto principalmente pela JVM e bibliotecas padrão do Java. O JRE pode 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 tópico, vimos muitas siglas e termos. Para facilitar o entendimento e disponibilizar uma rápida referência, vamos agrupar todos no quadro a seguir:

Quadro 2 – Termos e siglas relacionados ao Java

Nome	**Tradução**	**Definição**
Java Virtual Machine ***JVM***	Máquina Virtual Java	Programa responsável por interpretar e executar o 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

Fonte: Gomes, 2020.

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

TEMA 4 – VERSÕES DO JAVA E PRIMEIRO CÓDIGO

Neste tópico, vamos finalmente colocar a mão na massa. Faremos nosso primeiro projeto Java com base no programa Eclipse.

4.1 Versões do Java e a IDE Eclipse

No caso do Java, assim como na 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, das quais 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, com *licença software livre*. O Eclipse é o que utilizaremos em nossos estudos, mas todas funcionam de forma análoga.

O Eclipse, originalmente desenvolvido como IDE para Java, foi adaptado por *plugins* desenvolvidos pela comunidade para as mais diversas linguagens de programação, podendo ser encontrado no site: <https://www.eclipse.org/>. 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 à construção de softwares para dispositivos embarcados, sistemas de propósito específico com poucos recursos computacionais. Ele 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 do sistema operacional Windows ou terminal do Linux;
- **Java EE** é a edição mais completa que 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 integral ou parcialmente na forma de servidor de aplicações por diversos fornecedores. É uma importante tecnologia que ajuda a formar a espinha dorsal da internet atual.

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

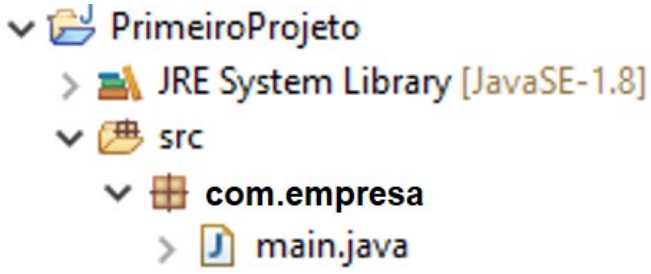
4.2 Primeiro código

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 4, vemos a estrutura de um projeto Java na IDE Eclipse. Os principais elementos que compõem o projeto são:

- Bibliotecas: são *bytecodes* com funcionalidades específicas implementadas. Permitem ao programador reaproveitar códigos geralmente desenvolvidos por equipes diversas e que já são muito bem testados e eficientes. O Eclipse já inclui uma biblioteca básica em todos os projetos;
- Pacotes: 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 de um pacote que esteja ligado à interface visual de 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 necessária. Esse pacote principal fica inserido dentro de uma pasta nomeada src (que vem da palavra *source*, isto é, código, em inglês);

- Classe: os códigos são descritos em arquivos com extensão *.java*, geralmente um arquivo por classe.

Figura 4 – Estrutura de projeto básico Eclipse com uma única classe



Uma vez aberto o programa Eclipse, para criar um projeto novo basta ir à opção **File/New Project**. Em seguida, dê um nome apropriado ao projeto e utilize as opções padrões. 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 o nome do projeto e ir à opção **New/Class**. Dê um nome para a classe e um nome para o pacote. 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 no 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. Os conceitos de classes públicas e privadas e suas implicações serão discutidos em detalhes em conteúdo posterior;
- **`public static void main(String[] args) {`**

- Esta linha é a declaração do método, na qual **static** indica que o método pertence à classe, e não ao objeto – o conceito de métodos estáticos será discutido com detalhes em conteúdo posterior –; **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; e **String [] args** é a declaração de um array (estrutura semelhante a uma lista) 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. Atenção ao **ponto-e-vírgula** ao final de cada comando Java. Para executar e testar o seu primeiro programa, vá na opção **Run/run** ou utilize o atalho CTRL + F11. Se tudo ocorrer bem, você 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 do Eclipse, assim, 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, você notará que eles seguem uma padronização que se torna intuitiva e clara com o tempo. O Java conta com uma biblioteca padrão muito extensa que permite um grande reaproveitamento de códigos.

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

TEMA 5 – VISÃO GERAL SOBRE O CÓDIGO JAVA

Neste tópico, vamos avançar a discussão sobre a linguagem Java, mas sem entrar em detalhes sobre a orientação a objetos. Além disso, também vamos passar brevemente pelos 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. No entanto, mesmo quem vem de outras linguagens como Python terá facilidade, visto que a lógica continua a mesma e muda apenas a forma de descrever certos comandos.

5.1.1 Entrada e saída

Vimos em conteúdo anterior, 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,
%d será substituído pelo número 10
```

Quanto à leitura de dados, é necessário realizar alguns passos. A seguir, temos um código Java que utilizaremos para exemplificar. Primeiro precisamos importar a biblioteca *java.util.Scanner*, que possui as definições da classe Scanner necessárias para a leitura. Veja o código a seguir na linha com a **Obs 1**, o comando **import** é utilizado para esta tarefa, e as importações devem vir sempre logo abaixo da declaração do nome do pacote.

Em seguida, 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*.

Na sequência, devemos utilizar o objeto que declaramos para fazer a leitura. Para ler um valor inteiro, *teclado.nextInt()*; para um valor real,

teclado.nextFloat() ou *teclado.nextDouble()*; e para ler uma string, *teclado.next()*. Veja a **Obs 3** no código para um exemplo.

```
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
```

5.1.2 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*, dependendo do valor. Atenção: o **ponto-e-vírgula** não aparece depois do comando *if*.

```
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 está presente na linguagem Java, permitindo o desvio de fluxo. Primeiro, a expressão dentro do *switch* é avaliada, e o código é desviado para o *case* pertinente ou *default*, caso nenhum esteja adequado. 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
}
```

5.1.3 Comandos de repetição

Os principais comandos de repetição também estão presentes em Java: *while, do-while* e *for.* Entre parênteses vai a condição que deve ser avaliada para analisar se o loop deve repetir ou não. Atenção: o **ponto-e-vírgula** não aparece depois do comando *while*.

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

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

O comando *for* é dividido em três partes separadas por ponto-e-vírgula. Na primeira, trata-se do que deve ser executado antes do loop, sendo geralmente a inicialização de alguma variável de controle. Na sequência, a condição de continuidade do loop, semelhante à condição do *while* e, por fim, o que deve ser executado ao final de cada iteração do loop, geralmente um incremento.

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

```
}
```

5.2 Tipos de dados

No Java, os dados são tipados, ou seja, antes de criar uma variável, é necessário declará-la e indicar o tipo de dado.

No Java, chamamos de primitivas as variáveis dos tipos básicos presentes nas principais linguagens de programação, como a seguir. Em um nível de abstração maior, os dados podem ser armazenados também em tipos não primitivos, como String, Array e Classes. No quadro a seguir, 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 e 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

Fonte: Gomes, 2020.

5.2.1 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. No código a seguir, 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
msg = msg.toUpperCase();//Todas as letras ficam maiúsculas
System.out.println(msg);//Impressão
```

5.2.2 Arrays

Os arrays também são essencialmente classes que trabalham de forma análoga ao C/C++ e muito semelhante às listas do Python. Podem ser declarados entre chaves e acessados com colchetes indexados por meio 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, tais como o método *length()*, que como na classe String, retorna a quantidade de itens do array. Considerando o exemplo anterior, o comando *nomes.length()* retornaria o número 4.

FINALIZANDO

Nesta etapa, 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 e os principais comandos.

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 a criação e a utilização das classes, o principal conceito dentro da programação orientada a objetos.