****

1

Introdução ao Java

**O que é Java?**

Java é uma linguagem de programação de propósito geral, orientada a objetos e projetada para ser portátil e altamente performática. Foi desenvolvida originalmente por James Gosling e sua equipe na Sun Microsystems e lançada em 1995.

**História e Evolução**

A história do Java começa em 1991 na empresa Sun Microsystems, onde foi iniciado o Green Project, o berço do **Java**, uma linguagem de programação orientada a objetos. Os mentores do projeto eram Patrick Naughton, Mike Sheridan, e James Gosling. Eles acreditavam que eventualmente, haveria uma convergência dos computadores com os equipamentos e eletrodomésticos frequentemente usados pelas pessoas no seu dia-a-dia.

Para provar a viabilidade desta ideia, treze pessoas trabalharam durante dezoito meses. No verão de 1992 eles emergiram de um escritório de Sand Hill Road (no Menlo Park), com uma demonstração funcional da ideia inicial. O protótipo se chamava **\*7**(lê-se “Star Seven”), um controle remoto com uma ***interface gráfica touchscreen,*** acompanhado de um mascote, hoje amplamente conhecido no mundo java, o Duke, que tinha a função de ser um guia virtual ajudando usuários a utilizar o equipamento. O star-seven tinha a habilidade de controlar diversos dispositivos e aplicações. James Gosling especificou uma nova linguagem de programação para o \*7. Chamada de “Oak”, que quer dizer carvalho, uma árvore que ele podia observar quando olhava através da sua janela.

Com tudo, o estouro da internet aconteceu rapidamente uma grande rede interativa estava se estabelecendo. Era este tipo de rede interativa que a equipe do \*7 estava tentando vender para as empresas de TV a cabo. E, da noite para o dia, não era mais necessário construir a infraestrutura para a rede, ela simplesmente estava lá. Gosling foi incumbindo de adaptar o Oak para a internet e em janeiro de 1995 foi lançada uma nova versão do Oak que foi rebatizada para Java – diz-se que inspirado no café que o time de desenvolvimento consumia, oriundo da ilha de Java, e que também está presente na logomarca Java. A tecnologia Java tinha sido projetada para se mover por meio das redes de dispositivos heterogêneos, redes como a internet. Agora aplicações poderiam ser executadas dentro dos navegadores nos Appets Java e tudo seria disponibilizado pela internet instantaneamente. Foi o estático HTML dos navegadores que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos seguintes foi assustadora, o número de usuários cresceu rapidamente, grandes fornecedores de tecnologia, como a IBM anunciaram suporte para a tecnologia Java.

Desde seu lançamento, em maio de 1995, a plataforma Java foi adotada mais rapidamente do que qualquer outra linguagem de programação na história da computação. Em 2004 Java atingiu a marca de 3 milhões de desenvolvedores em todo mundo. Java continuou crescendo e hoje é uma referência no mercado de desenvolvimento de software. Java tornou-se popular pelo seu uso na internet e hoje possui seu ambiente de execução presente em navegadores, mainframes, sistemas operacionais, celulares, palmtops, cartões inteligentes etc.

Em 1997 a Sun Microsystems tentou submeter a linguagem a padronização pelos órgãos ISO/IEC e ECMA, mas acabou desistindo. Java ainda é um padrão de fato, que é controlada através da JCP (Java Community Process). Em novembro de 2006, a Sun liberou a maior parte do Java como um software livre sob os termos da GNU GPL (General Public License), finalizando o processo em maio de 2007, tornando praticamente todo o código Java como software de código aberto, menos uma pequena porção da qual a Sun não possui os direitos legais.

Em 2008 a Oracle Corporation adquire a empresa responsável pela linguagem Java, a Sun Microsystems, por US$: 7,4 bilhões, com o objetivo de levar o Java e outros produtos da Sun ao dispor dos consumidores. Desde então, Java tem evoluído constante mente, com versões lançadas regularmente.

**Principais Características**

* **Portabilidade:** “Escreva uma vez, execute em qualquer lugar”.
* **Segurança:** Gerenciamento automático de memória e verificação de código.
* **Multithreading:** Suporte nativo para desenvolvimento concorrente.
* **Robustez:** Gestão de exceções e checagem rigorosa em tempo de compilação e execução.
* **Desempenho:** Just-In-Time (JIT) compiler e otimizações avançadas.

**2**

**Configuração**

**do**

**Ambiente**

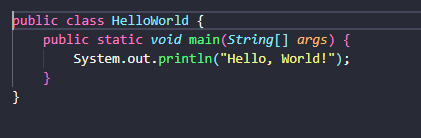
**Instalando o JDK**

* Baixe o JDK (Java Development Kit) do site oficial da Oracle (<https://www.oracle.com/br/java/technologies/downloads/> ) ou de uma distribuição open-source, com o OpenJDK.
* Siga as instruções de instalação para o seu sistema operacional.
* Configure a variável de ambiente `JAVA\_HOME` e adicione o diretório `bin` do JDK ao PATH do sistema.

**Configurando o IDE**

Java é suportado por diversos IDEs (Integrated Development Environments), cada um com suas características:

* Visual Studio Code: Ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente.
* IntelliJ IDE: Muito popular por suas funcionalidades avançadas e suporte a múltiplas linguagens.
* Eclipse: Um dos IDEs mais antigos e amplamente usados para Java.
* NetBeans: Um IDE open-source oficial da Oracle.

**Estrutura de um programa Java**

3

**Sintaxe Básica**

* **Classe:** Definida pela palavra-chave `class`, representa o blueprint do objeto.
* **Método `main`:**  Ponto de entrada da aplicação Java.

**Tipos de Dados e Variáveis**

Java possui tipos de dados primitivos e não primitivos (referência):

* **Primitivos:** `int`, `byte`, `short`, `long`, `float`, `double`, `boolean`, `char`.
* **Referência:** Classes, Interfaces, arrays.

**Operadores**

Java oferece diversos operadores como aritméticos, relacionais, lógicos, bit a bit, de atribuição e outros.

* **Aritméticos:** (`+`, `-`, `\*`, `/`, `%`);
* **Relacionais:** (`==`, `!=`, `>`, `<`);
* **Lógicos:** (`&&`, `||`, `!`);

**Estrutura de Controle**

* **Condicionais:** Utiliza `if `, `else if `, `else` e `switch` para controle de fluxo.
* **Loops:** Os principais laços são `for`, `while` e `do-while`.

**Conceitos Fundamentais**

**4**

**Programação Orientada a Objetos**

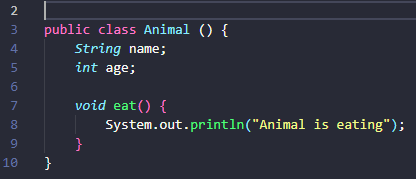
A programação orientanda a objetos surgiu como uma alternativa a essas características da programação estruturada. O intuito da sua criação também foi o de aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real, daí o nome “objeto” como algo genérico, que pode representar qualquer coisa tangível.

Este novo paradigma se baseia em alguns conceitos chaves:

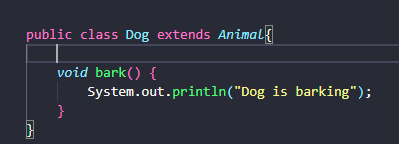
* **Encapsulamento:** Agrupamento de dados e métodos que operam sobre os dados.
* **Herança:** Mecanismo onde uma classe herda propriedades e métodos de outra.
* **Polimorfismo:** Capacidade de tratar objetos de diferentes classes através da mesma interface.

**Classes e Objetos**

Uma classe é uma forma de definir um tipo de dado na linguem orientada a objeto. Ela é formada por dados e comportamentos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe.

****A figura abaixo mostra a definição da classe Animal, que temos os atributos nome, uma ação.

**Herança**

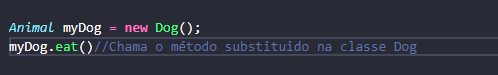
****A herança é um tipo de relacionamento que define uma classe “é um” de outra classe como, por exemplo, Dog que é um cachorro, que tem um relacionamento de herança com a classe Animal. Em algumas linguagens, como C#, é possível fazer herança múltipla, isto é, uma classe pode herdar de diversas outras classes, mas em Java isso não é permitido, pois cada classe pode herdar de apenas outra classe.

**Polimorfismo**

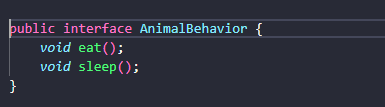
O polimorfismo é a possibilidade de em uma hierarquia de classes implementar métodos com a mesma assinatura e, assim, implementar um mesmo código que funcione para qualquer classe dessa hierarquia sem a necessidade de implementações específicas para cada classe. O principal objetivo do polimorfismo é diminuir a quantidade de código escrito, aumentando a clareza e a facilidade de manutenção.

A imagem abaixo mostra um exemplo de polimorfismo, onde estamos criando um objeto do tipo `Dog` e atribuindo-o a uma varável de referência do tipo `Animal`. `Animal` é a classe base (ou classe pai) e `Dog` é uma classe derivada (ou classe filha) que estende `Animal`. Isto é permitido porque `Dog` é um tipo de `Animal`, então uma referência `Animal` pode apontar para um objeto `Dog`. `myDog` é uma variável de referência do tipo `Animal`, mas aponta para um objeto `Dog`. Quando chamamos `eat()` em `myDog`, o método `eat` da classe `Dog` será executado, não o da classe `Animal`, assumindo que o método `eat` está sobrescrito na classe `Dog`.

Isso é conhecido como polimorfismo de tempo de execução, onde o método a ser chamado é determinado durante a execução, e não durante a compilação

****

**Interfaces**

****A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objetos, para “obrigar” a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

Neste exemplo: A interface `AnimalBehavior` define os métodos `eat` e `sleep`.

5

**Bibliotecas e Apis**

**Biblioteca Padrão do Java**

Assim como qualquer outra linguagem, o estudo de Java limita-se não só ao estudo das suas estruturas de controle e conjunto de palavras reservadas, mas também ao estudo das bibliotecas de classes. A API da linguagem Java disponibiliza oito bibliotecas de classes para utilização no desenvolvimento de applets e aplicações. As bibliotecas da linguagem Java são chamadas pacotes (ou packages), e são as seguintes:

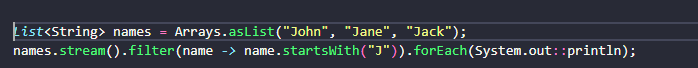
* **Java.lang:** A biblioteca Java de uso geral, java.lang provê classes que formam o núcleo da linguagem Java e da máquina virtual. Por exemplo, as classes Object, String, e Thread, que são usadas por quase todos os programas Java são definidas neste pacote.
* **Java.io:** A biblioteca de I/O, fornece um conjunto de classes de filas de entrada e saída de dados usadas para ler e escrever dados em arquivos ou outras fontes de entrada e saída. As filas de entrada e saída de Java são orientadas a bytes, e as classes definidas por esta biblioteca podem ser usadas na implementação de funcionalidades mais sofisticadas de leitura e gravação.
* **Java.util:** A biblioteca de utilidades diversas, contém uma coleção de classes de utilidades. Ela também inclui classes que fornecem estruturas de dados genéricas como dicionários, hashtables, pilhas e vetores, assim como classes para manipulação de strings, e utilidade para de calendários e datas (classe Date). É nesta classe que são definidas a interface Observer e a classe Observable, que permite que um objeto notifique outro objeto quando ele sofre uma alteração.

**Introdução às Coleções**

* **List:** Tem duas implementações `ArrayList`, e `LinkedList`. ArrayList oferece acesso aleatório rápido através do índice. Já em LinkedList o acesso aleatório é lento e necessita de um objeto nó para cada elemento, que é composto pelo dado propriamente dito e uma referência para o próximo nó, ou seja, consome mais memória.
* Set: `HashSet`, `TreeSet`;
* Map: `HashMap`, `TreeMap`;

**Streams e Expressões Lambda**

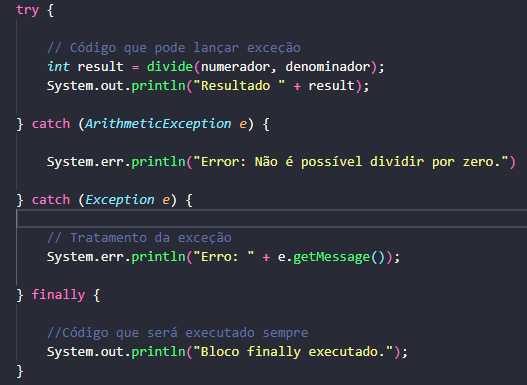
Introduzidas no Java 8, permitem operações funcionais e processamento de coleções de forma eficiente.



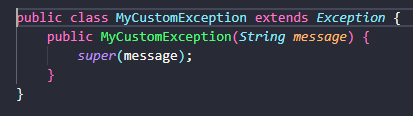
1. **Exceções e Tratamento de Erros**

**Tipos de Exceções**

* **Checked:** Exceções verificadas em tempo de compilação (`IOException`, `SQLException`).
* **Unchecked:** Exceções verificadas em tempo de execução (`NullPointerException`, ` ArrayIndexOutOfBoundsException`).

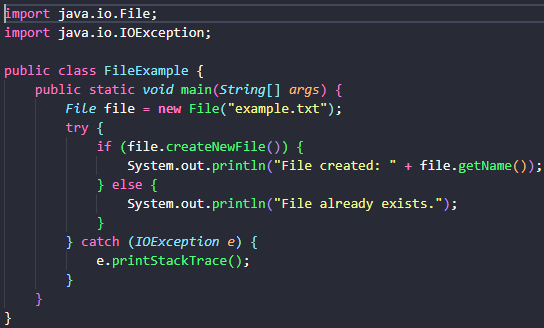
**Bloco try-catch-finally**

**Exceções Personalizadas**

****

1. **Entrada e Saída**

**Manipulação de Arquivos**

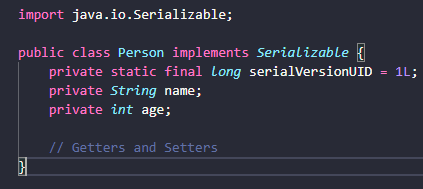
****

**Entrada e Saída com Streams**

Java oferece diversas classes para manipulação de fluxo de dados, como `FileInputStream`, `FileOutputStream`, `BufferedReader`, `BufferedWrite`.

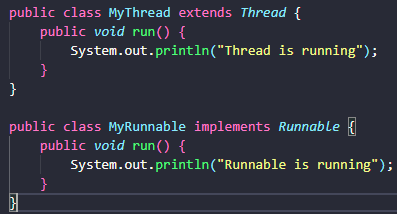
**Serialização**

Permite converter um objeto em um fluxo de bytes para armazenamento ou transmissão.



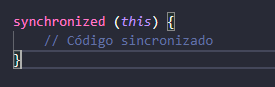
1. **Programação Concorrente**

**Threads e Runnable**

****

**Sincronização**

Evita condições de corrida ao acessar recursos compartilhados.



**Executors e Concurrency Utilities**

Java fornece o pacote `java.util.concurrent`com classes para gerenciar threads e tarefas assíncronas.

1. **Desenvolvimento Web**

**Introdução ao Java EE**

Java Plataform, Enterprise Edition (Java EE) oferece um conjunto de especificações para o desenvolvimento de aplicações corporativas.

**Servlets e JSP**

Servlets são classes Java que respondem a requisições HTTP, enquanto JSP (Java Server Pages) é uma tecnologia que permite a criação de páginas web dinâmicas.

**Frameworks Populares**

* **Spring:** Framework completo para desenvolvimento de aplicações Java corporativas.
* **Hibernate:** Framework de mapeamento objeto-relacional (ORM).

1. **Boas Práticas e Padrões de Projeto**

**Código Limpo**

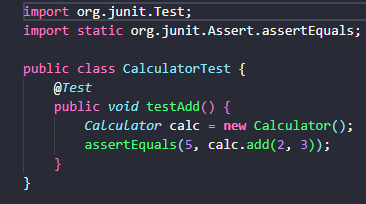
Escreva código legível, modular e bem documentado.

**Principais Padrões de Projeto**

* **Singleton:** Garante que uma classe tenha apenas uma instância.
* **Factory:** Cria objetos sem expor a lógica de criação ao cliente.
* **Observer:** Define uma dependência um-para-muitos entre objetos.

**Testes Unitários com JUnit**

JUnit é uma biblioteca para escrever e rodar testes automatizados em Java.



1. **Ferramentas e Biuld**

**Maven e Gradle**

Ferramentas de automação de biuld que gerenciam dependências e constroem o projeto.

**Controle de Versão com Git**

Sistema de controle de versão distribuído que permite rastrear mudanças no código-fonte.

**Integração Contínua**

Automatiza o processo de integração de código e deploy de aplicações.

1. **Tópicos Avançados**

**Java 8 e Funcionalidades Posteriores**

Introduziu expressões lambda, API de Strams, e melhorias na API de Data e Hora.

**Modulação com Java 9+**

Introduziu o sistema de módulos, que melhora a encapsulação e modularização do código.

**Otimização e Performance**

Técnicas para melhorar a performance de aplicações Java, como profiling e tuning da JVM.

1. **Conclusão e Próximos Passos**

**Recursos e Comunidades**

* **Documentação Oficial:** [Oracle Java Documentation](https://docs.oracle.com/en/java/)
* **Comunidades:** Stack Overflow, GitHub, Reddit (r/java), meetup.

**Livros Recomendados**

* **“Effective Java” por Joshua Bloch**
* **“Java: The Complete Reference” por Herbert Schildt**
* **“Head First Java” por Kathy Sierra e Bert Bates**

**Certificações Java**

* **Oracle Certified Associate (OCA)**
* **Oracle Certified Professional (OCP)**