ATITUS
EDUCAÇÃO

A Jornada Fascinante do Java: Do Oak à Oracle

A história da linguagem de programação Java é uma jornada fascinante que começou nas primeiras décadas da era da computação moderna e evoluiu para se tornar uma das linguagens mais populares e influentes do mundo.

Oak: A Semente da Árvore Java

Tudo começou em meados da década de 1990, quando uma equipe de engenheiros liderada por **James Gosling**, na **Sun Microsystems**, começou a trabalhar em um projeto revolucionário de linguagem de programação. Inicialmente chamada de "**Oak**", a linguagem foi concebida com a intenção de ser utilizada em dispositivos eletrônicos de consumo, como TVs interativas e assistentes digitais pessoais.

Essa linguagem de programação buscava atender às demandas da época: ser robusta, segura, portátil e multiplataforma. A ideia era que os programadores escrevessem o código uma vez e o executassem em qualquer dispositivo, independentemente do sistema operacional ou hardware. Essa filosofia, sintetizada no famoso slogan "**Escreva uma vez, execute em qualquer lugar (WORA)**", se tornaria a pedra angular do sucesso do Java.

De Oak para Java: Um Broto Promissor

Em 1995, a Oak foi renomeada para **Java**, uma homenagem ao café java, que na cultura estadunidense da época, era usada para se referir a um café forte, o qual era o preferido pelos desenvolvedores da equipe. Nesse mesmo ano, a Sun Microsystems lançou oficialmente a linguagem Java, junto com o **Java Development Kit (JDK) 1.0**, que incluía o compilador, as bibliotecas padrão e as ferramentas necessárias para desenvolver e executar aplicativos Java.

A característica mais marcante do Java era sua portabilidade. Ao contrário de muitas outras linguagens na época, que eram específicas para uma plataforma, o Java foi projetado para ser executado em qualquer ambiente que suportasse a **Máquina Virtual Java (JVM)**. Isso significava que os programas Java poderiam ser escritos uma vez e executados em qualquer dispositivo ou sistema operacional compatível com a JVM, proporcionando uma enorme flexibilidade e facilitando a distribuição de software.

Ascensão Meteórica e Expansão do Ecossistema

A popularidade do Java explodiu rapidamente. Sua versatilidade e portabilidade a tornaram a escolha ideal para diversos tipos de aplicações, desde sites interativos até softwares Entreprise complexos. A comunidade de desenvolvedores Java cresceu exponencialmente, impulsionada pela ampla gama de bibliotecas, frameworks e ferramentas disponíveis.

A Sun se Despede e a Oracle Assume o Comando

Em 2009, a Oracle adquiriu a Sun Microsystems, incluindo o controle da plataforma Java. Essa mudança gerou incertezas na comunidade, mas a Oracle se comprometeu a manter o desenvolvimento e o suporte ao Java, reconhecendo seu valor inestimável para o cenário tecnológico e mantendo-se como “guardiã” da linguagem.

Em 2017, a Oracle implementou uma mudança significativa no sistema de numeração de versões, que gerou dúvidas e questionamentos na comunidade de desenvolvedores.

Anteriormente, o Java seguia um modelo tradicional de versões sequenciais, com lançamentos anuais incrementais. Cada nova versão, como 1.5, 1.6, 1.7 e 1.8, apresentava aprimoramentos, correções de bugs e novas funcionalidades. Essa cadência era previsível e permitia que os desenvolvedores se planejassem para as atualizações com mais tranquilidade.

Em 2017, a Oracle introduziu um novo modelo de versões, optando por **números redondos** como 9, 10 e 11. Essa mudança acompanhou a adoção de um ritmo de lançamentos mais ágeis, com atualizações a cada seis meses.

A Oracle justificou essa alteração com alguns pontos chave:

- **Maior Frequência de Inovações**: O objetivo era entregar novas funcionalidades e aprimoramentos com mais rapidez, atendendo às demandas dinâmicas do mercado de software.
- **Foco em Lançamentos Estratégicos**: A numeração por números redondos permitiria destacar as versões que representavam marcos mais significativos em termos de inovações e mudanças na plataforma.
- **Simplificação da Comunicação**: O novo sistema visava facilitar a comunicação e identificação das versões, tornando-a mais intuitiva para desenvolvedores e usuários.

A mudança no sistema de versões do Java foi recebida com reações mistas na comunidade. Alguns desenvolvedores elogiaram a maior agilidade na entrega de inovações, enquanto outros expressaram apreensão com a possível fragmentação da plataforma e a necessidade de se adaptar a um novo ritmo de atualizações.

LTS: A Estabilidade para Projetos de Longo Prazo

Para garantir a estabilidade para projetos de longo prazo, a Oracle manteve o modelo de lançamentos LTS (Long Term Support), com versões que recebem atualizações de segurança e correções de bugs por um período mais extenso, geralmente três anos. As versões 8 e 11 do Java são exemplos de lançamentos LTS.

Versões do Java e suas principais melhorias:

1. **Java 1.1 (1997)**:
   - Introdução das classes internas aninhadas.
   - Adição de bibliotecas de coleções (java.util) e de eventos (java.awt.event).
   - Melhorias no desempenho e estabilidade.
2. **Java 1.2 (Java 2, 1998)**:
   - Introdução do Swing GUI toolkit.
   - Adição do Java Naming and Directory Interface (JNDI) para acesso a serviços de diretório.
   - Lançamento da Java Foundation Classes (JFC), incluindo Swing e Java 2D.
   - Melhorias na máquina virtual (JVM) para melhor desempenho e gerenciamento de memória.
3. **Java 1.3 (Java 2, 2000)**:
   - Adição de suporte a hot-spot JVM para melhor desempenho.
   - Introdução do Java Naming and Directory Interface (JNDI) versão 1.2.
   - Melhorias no Java Virtual Machine (JVM) para otimização de desempenho e estabilidade.
4. **Java 1.4 (Java 2, 2002)**:
   - Introdução das Assertions para testes e depuração de código.
   - Adição da API de Logging (java.util.logging).
   - Lançamento da API de Manipulação de XML (JAXP).
   - Adição do Java Web Start para implantação de aplicativos na web.
5. **Java 1.5 (2004)**:
   - Introdução de Generics para permitir tipos parametrizados.
   - Adição de Annotations para metadados no código.
   - Lançamento do autoboxing e unboxing para conversão automática entre tipos primitivos e objetos.
   - Adição das enumerações para representar um conjunto fixo de valores.
6. **Java 1.6 (2006)**:
   - Introdução do Java Compiler API para compilação programática de código Java.
   - Lançamento do Java Database Connectivity (JDBC) 4.0 com melhorias no tratamento de exceções.
   - Adição do suporte a scripting com a inclusão do framework Java Scripting API.
7. **Java 1.7 (2011)**:
   - Introdução do Project Coin, que trouxe pequenas melhorias na linguagem Java, como strings em switch, inferência de tipos em declarações genéricas e multi-catch exceptions.

- Lançamento da API de Fork/Join para facilitar a programação paralela.
- Adição do pacote java.nio para suporte a operações de E/S não bloqueantes.

8. **Java 1.8 (2014)**:
   - Introdução de Lambdas para facilitar a programação funcional.
   - Adição do Streams API para operações de processamento de dados em coleções.
   - Lançamento da nova API de Data e Hora (java.time).
   - Inclusão da PermGen Space com a remoção do Java Virtual Machine (JVM) PermGen Space.
9. **Java 9 (2017)**:
   - Introdução do módulo do sistema para modularizar o JDK.
   - Lançamento do JShell, uma ferramenta interativa de linha de comando para experimentar e testar código Java.
   - Adição do HTTP/2 Client para suportar o protocolo HTTP/2.
10. **Java 10 (2018)**:
    - Introdução do tipo de inferência de variáveis (var) para declarações locais.
    - Lançamento da API de coleta de lixo paralelo (Garbage-Collector) G1 como padrão.
11. **Java 11 (2018)**:
    - Lançamento do Java Platform Module System (JPMS) como padrão.
    - Adição do HTTP Client padrão.
    - Inclusão do JEP 323: Local-Variable Syntax for Lambda Parameters.
12. **Java 12 (2019)**:
    - Introdução do Switch Expressions como recurso de preview.
    - Lançamento do JEP 325: Switch Expressions (Preview).
    - Inclusão do JEP 334: JVM Constants API.
13. **Java 13 (2019)**:
    - Adição do JEP 354: Switch Expressions (Preview) como recurso final.
    - Lançamento do JEP 355: Text Blocks como recurso de preview.
    - Inclusão do JEP 343: Packaging Tool (Incubator).
14. **Java 14 (2020)**:
    - Lançamento do JEP 361: Switch Expressions (Standard).
    - Adição do JEP 368: Text Blocks como recurso final.
    - Inclusão do JEP 359: Records como recurso de preview.
15. **Java 15 (2020)**:
    - Lançamento do JEP 360: Sealed Classes como recurso de preview.
    - Adição do JEP 372: Remove the Nashorn JavaScript Engine.
    - Inclusão do JEP 375: Pattern Matching for instanceof (Preview).
16. **Java 16 (2021)**:
    - Lançamento do JEP 394: Pattern Matching for instanceof (Second Preview).
    - Adição do JEP 395: Records como recurso final.
    - Inclusão do JEP 376: ZGC: Concurrent Thread-Stack Processing.
17. **Java 17 (2021)**:
    - Lançamento do JEP 356: Enhanced Pseudo-Random Number Generators.
    - Adição do JEP 382: New macOS Rendering Pipeline.
    - Inclusão do JEP 391: macOS/AArch64 Port.
18. **Java 18 (2022)**:
    - Lançamento do JEP 388: Windows/AArch64 Port.

- Adição do JEP 392: Packaging Tool.
- Inclusão do JEP 398: Deprecate the Applet API for Removal.

19. **Java 19 (2022)**:
    - Lançamento do JEP 414: Vector API (Second Incubator).
    - Adição do JEP 407: Remove the RMI Activation System.
    - Inclusão do JEP 416: Improve the Test Coverage of Vector API.
20. **Java 20 (2023)**:
    - Lançamento do JEP 419: Vector API (Third Incubator).
    - Adição do JEP 426: Add SocketConnect Lambda for Socket API.
    - Inclusão do JEP 427: Dynamic CDS Archives.
21. **Java 21 (2023)**:
    - Lançamento do JEP 441: Warnings for Ambiguous Java Updates.
    - Adição do JEP 442: Unix-Domain Socket Channels.
    - Inclusão do JEP 443: Remove the java.lang Module.
22. **Java 22 (2024)**:
    - Lançamento do JEP 456: iOS/AArch64 Port.
    - Adição do JEP 458: JVM Interface for JNI.
    - Inclusão do JEP 462: Remove the Java EE and CORBA Modules.