Programação I

Aula 1

Professor: Diogo Passos Ranghetti

Sumário

Java

Entender um pouco da história da plataforma Java é essencial para enxergar os motivos que a levaram ao sucesso. Quais eram os seus maiores problemas quando programava na década de 1990? Ponteiros? Gerenciamento de memória? Organização? Falta de bibliotecas? Ter de reescrever parte do código ao mudar de sistema operacional? Custo financeiro de usar a tecnologia?

A linguagem Java resolve bem esses problemas, que até então apareciam com frequência nas outras linguagens. Alguns desses problemas foram particularmente atacados porque uma das grandes motivações para a criação da plataforma Java era de que essa linguagem fosse usada em pequenos dispositivos, como TVs, videocassetes, aspiradores, liquidificadores e outros. Apesar disso a linguagem teve seu lançamento focado no uso em clientes web (browsers) para rodar pequenas aplicações (applets).

Hoje em dia esse não é o grande mercado do Java: apesar de ter sido idealizado com um propósito e lançado com outro, o Java ganhou destaque no lado do servidor.

Java é capaz de criar tanto aplicativos para desktop, aplicações comerciais, softwares robustos, completos e independentes, aplicativos para a Web. Alem disso, caracteriza-se por ser muito parecida com C++, eliminando as características consideradas complexas, dentre as quais ponteiros e herança múltipla.

História

Em 1991, um pequeno grupo de funcionários da ***Sun*** incluindo James Gosling mudou-se para a *San Hill Road*, uma empresa filial. O grupo estava iniciando um projeto denominado Projeto Green, que consistia na criação de tecnologias modernas de software para empresas eletrônicas de consumo, como dispositivos de controle remoto das TV a cabo.

Logo o grupo percebeu que não poderiam ficar preso as plataformas, pois os clientes não estavam interessados no tipo de processador que estavam utilizando e fazer uma versão do projeto para cada tipo de sistema seria inviável.

Desenvolveram então o sistema operacional *GreenOS*, com a linguagem de programação *Oak*. Eles se basearam no inventor do Pascal, através da linguagem USCD Pascal, que foi o pioneiro da linguagem intermediária ou máquina virtual.

Em 1993, surgiu uma oportunidade para o grupo Green, agora incorporado como *FirstPerson* a *Time-Warner*, uma empresa que estava solicitando propostas de sistemas operacionais de decodificadores e tecnologias de vídeo sob demanda.

Isso foi na mesma época em que o NCSA lançou o MOSAIC 1.0, o primeiro navegador gráfico para Web. A *FirstPerson* apostou nos testes de TV da *Time-Warner*, mas esta empresa preferiu optar pela tecnologia oferecida pela *Silicon Graphics*.

Depois de mais um fracasso, a *FirstPerson* dissolveu-se e metade do pessoal foi trabalhar para a *Sun Interactive* com servidores digitais de vídeo. Entretanto, a equipe restante continuou os trabalhos do projeto na Sun.

Apostando na Web, visto que os projetos estavam sendo todos voltados para a WWW, surgiu a ideia de criar um *browser* com independência de plataforma, que foi o *HotJava*.

Como a equipe de desenvolvimento ingeria muito café enquanto estavam trabalhando, várias xícaras de café foram inseridas até que o projeto estivesse pronto.

Finalmente em maio de 1995, a Sun anunciou um ambiente denominado Java (homenagem às xícaras de café) que obteve sucesso graças a incorporação deste ambiente aos navegadores (*browsers*) populares como o *Netscape Navigator* e padrões tridimensionais como o VRML (*Virtual Reality Modeling Language* – Linguagem de Modelagem para Realidade Virtual).

A Sun considera o sucesso do Java na Internet como sendo o primeiro passo para utilizá-lo em decodificadores da televisão interativa em dispositivos portáteis e outros produtos eletrônicos de consumo – exatamente como o Java tinha começado em 1991.

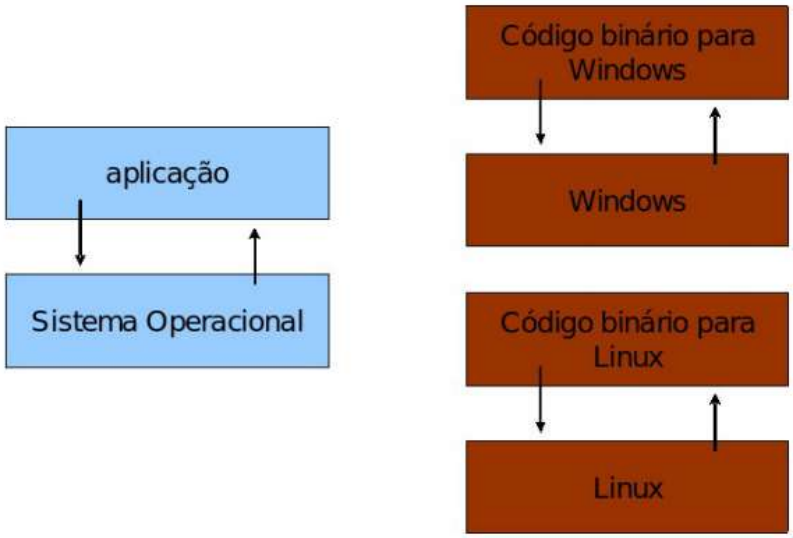
Sua natureza portátil e o projeto robusto permitem o desenvolvimento para múltiplas plataformas, em ambientes tão exigentes como os da eletrônica de consumo. A primeira versão da linguagem Java foi lançada em 1996.

JVM

Em uma linguagem de programação como C e Pascal, temos a seguinte situação quando vamos compilar um programa:

O código fonte é compilado para código de máquina específico de uma plataforma e sistema operacional. Muitas vezes o próprio código fonte é desenvolvido visando uma única plataforma!

Esse código executável (binário) resultante será executado pelo sistema operacional e, por esse motivo, ele deve saber conversar com o sistema operacional em questão.



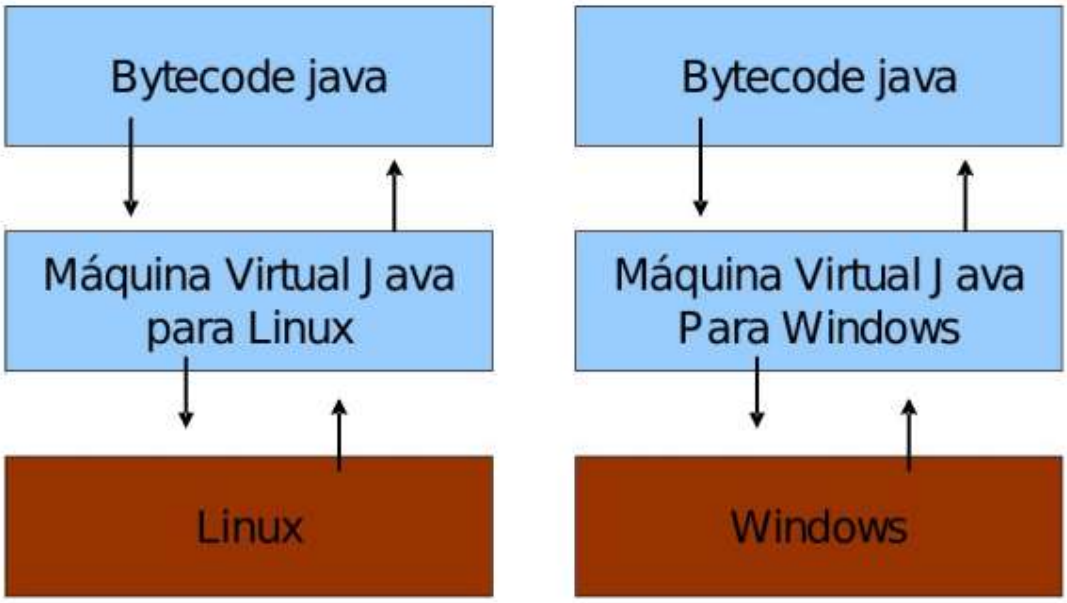
Isto é, temos um código executável para cada sistema operacional. É necessário compilar uma vez para Windows, outra para o Linux, e assim por diante,

caso a gente queira que esse nosso software possa ser utilizado em várias plataformas. Esse é o caso de aplicativos como o OpenOffice, Firefox e outros.

Como foi dito anteriormente, na maioria das vezes, a sua aplicação se utiliza das bibliotecas do sistema operacional, como, por exemplo, a de interface gráfica para desenhar as "telas". A biblioteca de interface gráfica do Windows é bem diferente das do Linux: como criar então uma aplicação que rode de forma parecida nos dois sistemas operacionais?

Precisamos reescrever um mesmo pedaço da aplicação para diferentes sistemas operacionais, já que eles não são compatíveis.

Já o Java utiliza do conceito de máquina virtual, onde existe, entre o sistema operacional e a aplicação, uma camada extra responsável por "traduzir" - mas não apenas isso - o que sua aplicação deseja fazer para as respectivas chamadas do sistema operacional onde ela está rodando no momento:



Dessa forma, a maneira com a qual você abre uma janela no Linux ou no Windows é a mesma: você ganha independência de sistema operacional. Ou, melhor ainda, independência de plataforma em geral: não é preciso se preocupar em qual sistema operacional sua aplicação está rodando, nem em que tipo de máquina, configurações, etc.

Repare que uma máquina virtual é um conceito bem mais amplo que o de um interpretador. Como o próprio nome diz, uma máquina virtual é como um "computador de mentira": tem tudo que um computador tem. Em outras palavras, ela é responsável por gerenciar memória, threads, a pilha de execução, etc.

Sua aplicação roda sem nenhum envolvimento com o sistema operacional! Sempre conversando apenas com a **Java Virtual Machine** (JVM).

Essa característica é interessante: como tudo passa pela JVM, ela pode tirar métricas, decidir onde é melhor alocar a memória, entre outros. Uma JVM isola totalmente a aplicação do sistema operacional.

Se uma JVM termina abruptamente, só as aplicações que estavam rodando nela irão terminar: isso não afetará outras JVMs que estejam rodando no mesmo computador, nem afetará o sistema operacional.

Essa camada de isolamento também é interessante quando pensamos em um servidor que não pode se sujeitar a rodar código que possa interferir na boa execução de outras aplicações.

Essa camada, a máquina virtual, não entende código Java, ela entende um código de máquina específico. Esse código de máquina é gerado por um compilador Java, como o **javac**, e é conhecido por "*bytecode*", pois existe menos de 256 códigos de operação dessa linguagem, e cada "*opcode*" gasta um byte. O compilador Java gera esse *bytecode* que, diferente das linguagens sem máquina virtual, vai servir para diferentes sistemas operacionais, já que ele vai ser "traduzido" pela JVM.

JDK

O JDK é um kit de desenvolvimento Java fornecido livremente pela Sun. Constitui de um conjunto de programas que engloba compilador, interpretador e utilitários. A primeira versão deste Kit foi a 1.0.

Atualmente, o JDK está na versão 8u202 o JDK é separado em três edições: o *Java 2 Standard Edition* (J2SDK), o *Java 2 Enterpise Edition* (J2EE) e o *Java 2 Micro Edition* (J2ME).

Cada uma engloba um conjunto de pacotes diferentes fornecendo aos usuários uma forma organizada e diferenciada para desenvolver aplicações.

Ou seja, os usuários que desejem desenvolver aplicações para Palm Tops, celulares, dispositivos pequenos, deve utilizar o J2ME para desenvolver as suas aplicações.

Os principais componentes do kit de desenvolvimento são:

- **javac** (compilador)
- **java** (interpretador)
- **appletviewer** (visualizador de *applets*)
- **javadoc** (gerador de documentação)
- **jar** (programa de compactação)

A utilização do JDK é feita da seguinte forma:

Primeiro escreve-se o programa fonte em Java, utilizando qualquer editor de texto ou IDE para Java como *Eclipse*, *JCreator*, *JBuilder*, *JDeveloper*, Bloco de Notas, *TextPad* (com *pluggins*), dentre outros.

A seguir, o programa deve ser compilado utilizando o compilador **javac**:
javac <NomeDoArquivo.java>
Exemplo: javac Teste.java

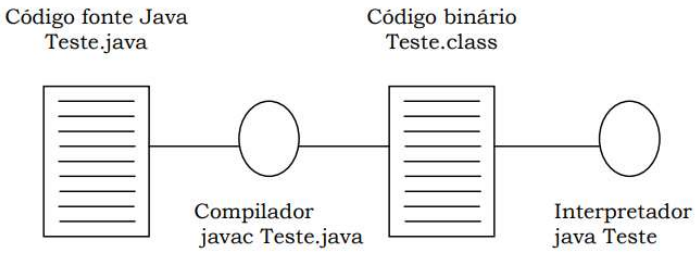
A compilação gera o arquivo em código binário (*bytecode*), com extensão **.class** uma vez compilado, basta executar o programa. Se for uma aplicação, utilizar o interpretador **java**:

java <NomeDaClasse>

Exemplo: java Teste

As IDEs para Java já tem o compilador (javac) e o interpretador (java) embutido no aplicativo, o que basta clicar em botões ou usar teclas de atalho para compilar e interpretar os programas desenvolvidos.

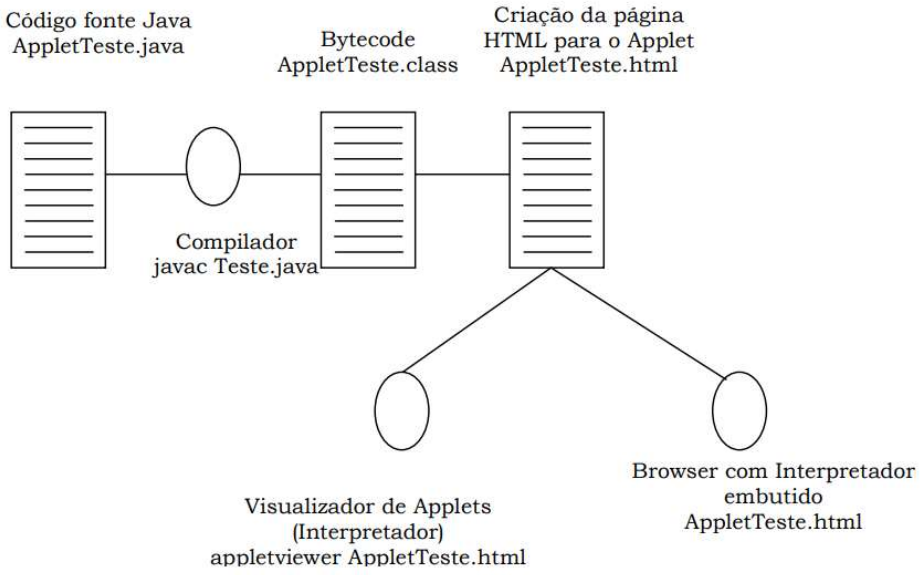
Nas imagens a seguir teremos um exemplo de como funciona a manipulação do JDK (*Java Developement Kit*).



Sendo um *Applet*, deve-se construir uma página HTML para abrigar o *applet* e carregá-la através de um browser ou do **appletviewer** (visualizador de *applets* Java):

appletviewer <NomeDoArquivo.html>

Exemplo: appletviewer Teste.html



Características da Linguagem

Simples e familiar

Linguagem simples e de fácil manipulação, possui sintaxe muito parecida com C++ que é uma das mais conhecidas no meio. Java é muitas vezes considerada uma versão simplificada da linguagem C++, onde Java não possui características como arquivos *headers*, ponteiros, sobrecarga de operadores, classes básicas virtuais, dentre outras que somente aumentavam a dificuldade dos programadores com a linguagem C++.

Orientada a Objetos

Paradigma atualmente mais utilizado na construção de softwares. Permite que se focalize o dado, enfim, o objeto. Java não é uma linguagem 100% orientada a objetos, como *Smaltalk*, onde qualquer elemento, (operadores, sinais, tipos de dados,...) são objetos.

Em Java há os tipos primitivos de dados que não são objetos, mas foram criados e incorporados ao Java para permitir uma melhor forma de utilização da linguagem pelos programadores. Outra característica importante da linguagem Java em relação à linguagem C++, é que Java não suporta herança múltipla.

Compilada e Interpretada

Um programa desenvolvido em Java necessita ser compilado, gerando um *bytecode*. Para executá-lo é necessário então, que um interpretador leia o código binário, o *bytecode* e repasse as instruções ao processador da máquina específica. Esse interpretador é conhecido como JVM (*Java Virtual Machine*).

Os *bytecodes* são conjuntos de instruções, parecidas com código de máquina. É um formato próprio do Java para a representação das instruções no código compilado.

Pronta para Redes

As funcionalidades que são fornecidas pela linguagem Java para desenvolver programas que manipulem as redes através das *API*s são simples e de grande potencialidade.

Através destas *API's* pode-se manipular protocolos como *TCP/IP, HTTP, FTP* e utilizar objetos da grande rede via *URL's*.

Distribuído

Programas Java são "linkados" em tempo de execução. Os *bytecodes* gerados durante a compilação só serão integrados na execução. Um objeto X existente em um arquivo quando instanciado, somente será alocado na memória em tempo de execução. Se alguma alteração ocorrer na classe que define o objeto X, somente o arquivo da classe com a alteração necessita ser compilado.

Multiprocessamento (*Multithread*)

Suporta a utilização de ***threads***. Threads são linhas de execução, executadas concorrentemente dentro de um mesmo processo. Diferentemente de outras linguagens, programar utilizando Threads é simples e fácil na linguagem Java.

Portabilidade

Pode ser executado em qualquer arquitetura de hardware e sistema operacional, sem precisar ser re-compilado. Um programa Java pode ser executado em qualquer plataforma que possua um interpretador Java (ambiente de execução).

Além disso, não há dependência de implementação, como por exemplo, os tamanhos dos tipos primitivos não diferem entre si, são independentes da maquina em que está a aplicação. Assim, o tipo ***int*** possui sempre um tamanho de 32-bits em Java e em qualquer máquina que esteja sendo executado.

Coletor de Lixo (*Garbage Colector*)

Se não houver nenhuma referência a um objeto que tenha sido criado na memória, o coletor de lixo destrói o objeto e libera a memória ocupada por ele. O coletor de lixo é executado de tempos em tempos.

Quando a JVM percebe que o sistema diminuiu a utilização do processador, ela (a JVM) faz com que o coletor de lixo execute e este vasculha a memória em busca de algum objeto criado e não referenciado. Em Java nunca se pode explicitamente liberar a memória de objetos que se tenha alocado anteriormente.

O método abaixo PROPOE/SUGERE que a JVM vai utilizar recursos para reciclar objetos que não são mais utilizados. Mas não garante que daqui a 1

milisegundo ou 100 milisegundos o Garbage Collection vai coletar todos os objetos em desuso.

Runtime.gc();

System.gc();

O *Garbage Collector* é uma grande vantagem para desalocação de memória, que é um grande inconveniente para programadores que trabalham com ponteiros e necessitam liberar o espaço alocado, visto que é o proprio sistema que se encarrega desta limpeza, evitando erros de desalocação de objetos ainda em uso.

Segura

O Java fornece uma série de mecanismos para garantir a segurança dos aplicativos. Um programa em Java não tem contato com o computador real; ele conhece apenas a máquina virtual (JVM). A máquina virtual decide o que pode ou não ser feito. Um programa Java nunca acessa dispositivos de entrada e saída, sistema de arquivos, memória, ao invés disso ele pede a JVM que acesse.

Referências bibliográficas

DEITEL, H. M.; DEITEL, P. J. Java como Programar. 8ª.ed. São Paulo: Pearson Brasil, 2010.

https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf

https://www.faeterj-rio.edu.br/downloads/bbv/0031.pdf