CAPÍTULO 2

O que é Java

"Computadores são inúteis, eles apenas dão respostas"
— Picasso

Chegou a hora de responder as perguntas mais básicas sobre Java. Ao término desse capítulo, você será capaz de:

- responder o que é Java;
- mostrar as vantagens e desvantagens do Java;
- entender bem o conceito de máquina virtual;
- compilar e executar um programa simples.

2.1 – 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.

O Java foi criado pela antiga Sun Microsystems e mantida através de um comitê (http://www.jcp.org). Seu site principal era o java.sun.com, e java.com um site mais institucional, voltado ao consumidor de produtos e usuários leigos, não desenvolvedores. Com a compra da Sun pela Oracle em 2009, muitas URLs e nomes tem sido trocados para refletir a marca da Oracle. A página principal do Java é: http://www.oracle.com/technetwork/java/

No Brasil, diversos grupos de usuários se formaram para tentar disseminar o conhecimento da linguagem. Um deles é o **GUJ** (http://www.guj.com.br), uma comunidade virtual com artigos, tutoriais e fórum para tirar dúvidas, o maior em língua portuguesa com mais de cem mil usuários e 1 milhão de mensagens.

Encorajamos todos os alunos a usar muito os fóruns do mesmo, pois é uma das melhores maneiras para achar soluções para pequenos problemas que acontecem com grande frequência.

2.2 - Uma breve história do Java

A Sun criou um time (conhecido como Green Team) para desenvolver inovações tecnológicas em 1992. Esse time foi liderado por James Gosling, considerado o pai do Java. O time voltou com a ideia de criar um interpretador (já era uma máquina virtual, veremos o que é isso mais a frente) para pequenos dispositivos, facilitando a reescrita de software para aparelhos eletrônicos, como vídeo cassete, televisão e aparelhos de TV a cabo.

A ideia não deu certo. Tentaram fechar diversos contratos com grandes fabricantes de eletrônicos, como Panasonic, mas não houve êxito devido ao conflito de interesses e custos. Hoje, sabemos que o Java domina o mercado de aplicações para celulares com mais de 2.5 bilhões de dispositivos compatíveis, porém em 1994 ainda era muito cedo para isso.

Com o advento da web, a Sun percebeu que poderia utilizar a ideia criada em

1992 para rodar pequenas aplicações dentro do browser. A semelhança era que na internet havia uma grande quantidade de sistemas operacionais e browsers, e com isso seria grande vantagem poder programar numa única linguagem, independente da plataforma. Foi aí que o Java 1.0 foi lançado: focado em transformar o browser de apenas um cliente magro (*thin client* ou terminal burro) em uma aplicação que possa também realizar operações avançadas, e não apenas renderizar html.

Os applets deixaram de ser o foco da Sun, e nem a Oracle nunca teve interesse. É curioso notar que a tecnologia Java nasceu com um objetivo em mente, foi lançado com outro, mas, no final, decolou mesmo no desenvolvimento de aplicações do lado do servidor. Sorte? Há hoje o Java FX, tentando dar força para o Java não só no desktop mas como aplicações ricas na web, mas muitos não acreditam que haja espaço para tal, considerando o destino de tecnologias como Adobe Flex e Microsoft Silverlight.

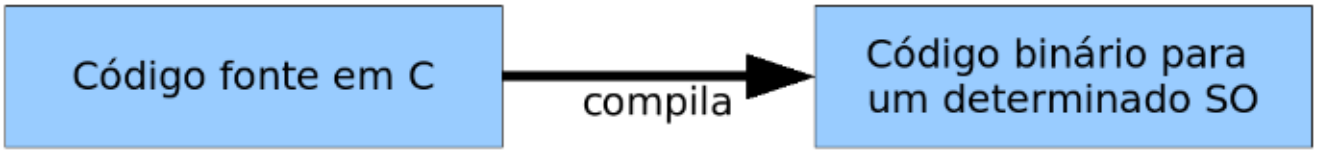
Você pode ler a história da linguagem Java em:
http://www.java.com/en/javahistory/

E um vídeo interessante: http://tinyurl.com/histjava

Em 2009 a Oracle comprou a Sun, fortalecendo a marca. A Oracle sempre foi, junto com a IBM, uma das empresas que mais investiram e fizeram negócios através do uso da plataforma Java. Em 2014 surge a versão Java 8 com mudanças interessantes na linguagem.

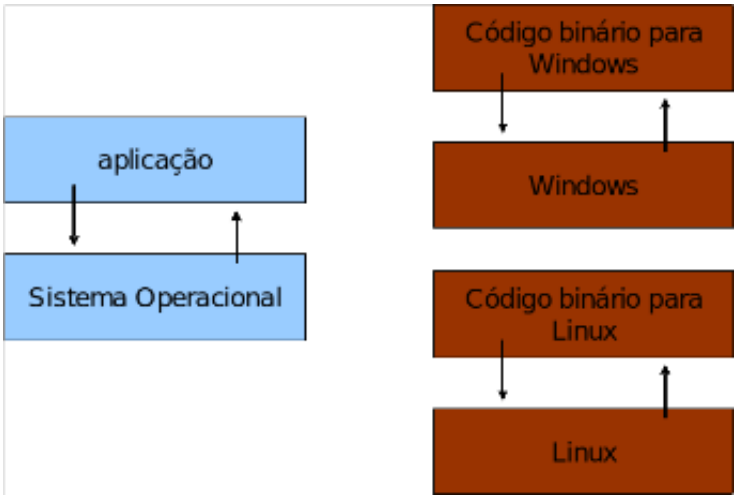
2.3 - Máquina Virtual

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.



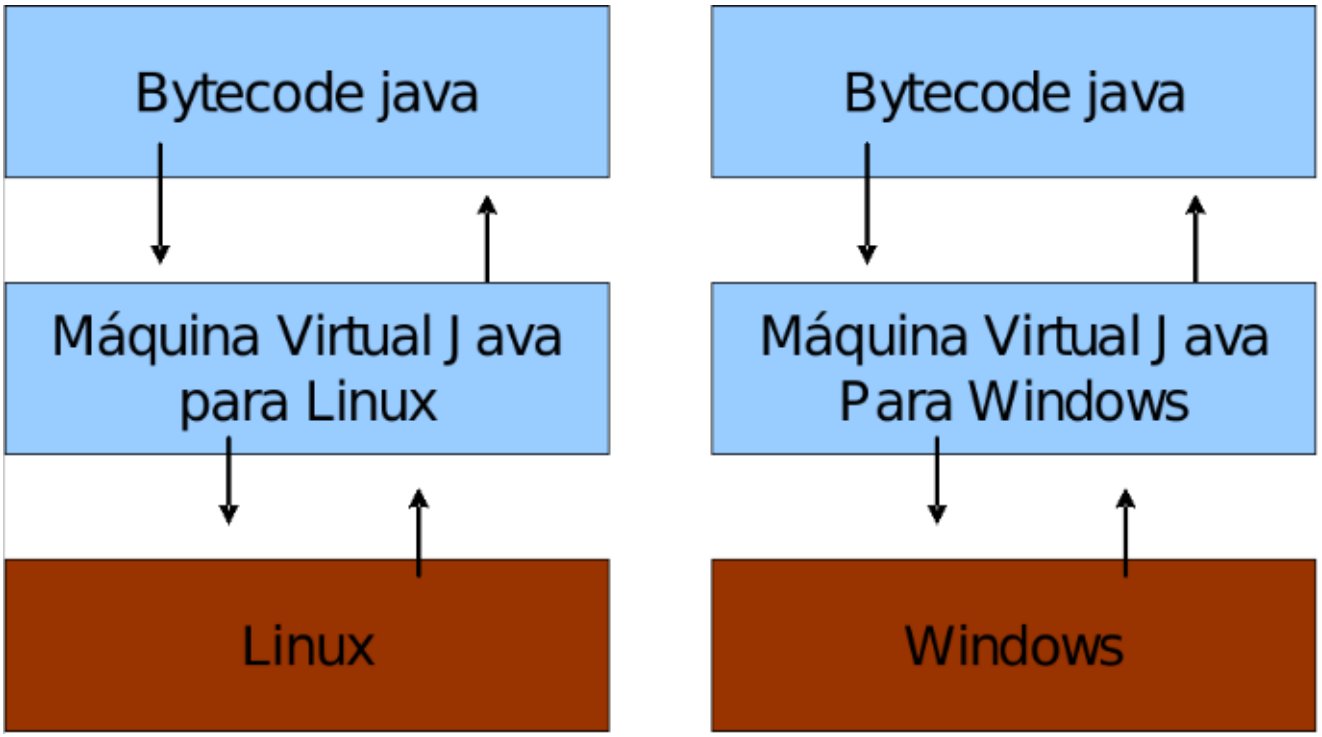
Isto é, temos um código executável para cada sistema operacional. É necessário compilar uma vez para Windows, outra para o Linux, e assim por diante, caso a gente queira que esse nosso software possa ser utilizado em várias plataformas. Esse é o caso de aplicativos como o OpenOffice, Firefox e outros.

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 existem 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.

Write once, run anywhere

Esse era um slogan que a Sun usava para o Java, já que você não precisa reescrever partes da sua aplicação toda vez que quiser mudar de sistema operacional.

2.4 - Java lento? Hotspot e JIT

Hotspot é a tecnologia que a JVM utiliza para detectar *pontos quentes* da sua aplicação: código que é executado muito, provavelmente dentro de um ou mais loops. Quando a JVM julgar necessário, ela vai **compilar** estes códigos para instruções realmente nativas da plataforma, tendo em vista que isso vai provavelmente melhorar a performance da sua aplicação. Esse compilador é o *JIT: Just inTime Compiler*, o compilador que aparece "bem na hora" que você precisa.

Você pode pensar então: porque a JVM não compila tudo antes de executar a aplicação? É que teoricamente compilar dinamicamente, a medida do necessário, pode gerar uma performance melhor. O motivo é simples: imagine um .exe gerado pelo VisualBasic, pelo gcc ou pelo Delphi; ele é estático. Ele já foi otimizado baseado em heurísticas, o compilador pode ter tomado uma decisão não tão boa.

Já a JVM, por estar compilando dinamicamente durante a execução, pode perceber que um determinado código não está com performance adequada e otimizar mais um pouco aquele trecho, ou ainda mudar a estratégia de otimização. É por esse motivo que as JVMs mais recentes em alguns casos chegam a ganhar de códigos C compilados com o GCC 3.x.

2.5 - Versões do Java e a confusão do Java2

Java 1.0 e 1.1 são as versões muito antigas do Java, mas já traziam bibliotecas importantes como o JDBC e o java.io.

Com o Java 1.2 houve um aumento grande no tamanho da API, e foi nesse momento em que trocaram a nomenclatura de Java para Java2, com o objetivo de diminuir a confusão que havia entre Java e Javascript. Mas lembre-se, não há versão "Java 2.0", o 2 foi incorporado ao nome, tornando-se Java2 1.2.

Depois vieram o Java2 1.3 e 1.4, e o Java 1.5 passou a se chamar Java 5, tanto por uma questão de marketing e porque mudanças significativas na linguagem foram incluídas. É nesse momento que o "2" do nome Java desaparece. Repare que para fins de desenvolvimento, o Java 5 ainda é referido como Java 1.5.

Hoje a última versão disponível do Java é a 8.

2.6 - JVM? JRE? JDK? O que devo baixar?

O que gostaríamos de baixar no site da Oracle?

- JVM = apenas a virtual machine, esse download não existe, ela sempre vem acompanhada.
- JRE = **Java Runtime Environment**, ambiente de execução Java, formado pela JVM e bibliotecas, tudo que você precisa para executar uma aplicação Java. Mas nós precisamos de mais.
- JDK = **Java Development Kit**: Nós, desenvolvedores, faremos o download do JDK

do Java SE (Standard Edition). Ele é formado pela JRE somado a ferramentas, como o compilador.

Tanto o JRE e o JDK podem ser baixados do site http://www.oracle.com/technetwork/java/. Para encontrá-los, acesse o link Java SE dentro dos top downloads. Consulte o apêndice de instalação do JDK para maiores detalhes.

2.7 - Onde usar e os objetivos do Java

No decorrer do curso, você pode achar que o Java tem menor produtividade quando comparada com a linguagem que você está acostumado.

É preciso ficar claro que a premissa do Java não é a de criar sistemas pequenos, onde temos um ou dois desenvolvedores, mais rapidamente que linguagens como php, perl, e outras.

O foco da plataforma é outro: aplicações de *médio a grande porte*, onde o time de desenvolvedores tem *várias pessoas* e sempre pode vir a *mudar* e *crescer*. Não tenha dúvidas que criar a primeira versão de uma aplicação usando Java, mesmo utilizando IDEs e ferramentas poderosas, será mais trabalhoso que muitas linguagens script ou de alta produtividade. Porém, com uma linguagem orientada a objetos e madura como o Java, será extremamente mais fácil e rápido fazer alterações no sistema, desde que você siga as boas práticas e recomendações sobre *design* orientado a objetos.

Além disso, a quantidade enorme de bibliotecas gratuitas para realizar os mais diversos trabalhos (tais como relatórios, gráficos, sistemas de busca, geração de código de barra, manipulação de XML, tocadores de vídeo, manipuladores de texto, persistência transparente, impressão, etc) é um ponto fortíssimo para adoção do Java: você pode criar uma aplicação sofisticada, usando diversos recursos, sem precisar comprar um componente específico, que costuma ser caro. O ecossistema do Java é enorme.

Cada linguagem tem seu espaço e seu melhor uso. O uso do Java é interessante em aplicações que virão a crescer, em que a legibilidade do código é importante, onde temos muita conectividade e se há muitas plataformas (ambientes e sistemas operacionais) heterogêneas (Linux, Unix, OSX e Windows misturados).

Você pode ver isso pela quantidade enorme de ofertas de emprego procurando desenvolvedores Java para trabalhar com sistemas web e aplicações de integração

no servidor.

Apesar disto, a Sun empenhou-se em tentar popularizar o uso do Java em aplicações desktop, mesmo com o fraco marketshare do Swing/AWT/SWT em relação às tecnologias concorrentes (em especial Microsoft .NET). A atual tentativa é o Java FX, onde a Oracle tem investido bastante.

2.8 - Especificação versus implementação

Outro ponto importante: quando falamos de Java Virtual Machine, estamos falando de uma especificação. Ela diz como o bytecode deve ser interpretado pela JVM. Quando fazemos o download no site da Oracle, o que vem junto é a Oracle JVM. Em outras palavras, existem outras JVMs disponíveis, como a JRockit da BEA (também adquirida pela Oracle), a J9 da IBM, entre outras.

Esse é outro ponto interessante para as empresas. Caso não estejam gostando de algum detalhe da JVM da Oracle ou prefiram trabalhar com outra empresa, pagando por suporte, elas podem trocar de JVM com a garantia absoluta de que todo o sistema continuará funcionando. Isso porque toda JVM deve ser certificada pela Oracle, provando a sua compatibilidade. Não há nem necessidade de recompilar nenhuma de suas classes.

Além de independência de hardware e sistema operacional, você tem a independência de *vendor* (fabricante): graças a ideia da JVM ser uma especificação e não um software.

2.9 - Como o FJ-11 está organizado

Java é uma linguagem simples: existem poucas regras, muito bem definidas.

Porém quebrar o paradigma procedural para mergulhar na orientação a objetos não é simples. Quebrar o paradigma e ganhar fluência com a linguagem e API são os objetivos do FJ-11.

O começo pode ser um pouco frustrante: exemplos simples, controle de fluxo com o `if`, `for`, `while` e criação de pequenos programas que nem ao menos captam dados do teclado. Apesar de isso tudo ser necessário, é só nos 20% finais do curso que utilizaremos bibliotecas para, no final, criarmos um chat entre duas máquinas que transferem Strings por TCP/IP. Neste ponto, teremos tudo que é necessário para entender completamente como a API funciona, quem estende quem, e o porquê.

Depois desse capítulo no qual o Java, a JVM e primeiros conceitos são passados, veremos os comandos básicos do java para controle de fluxo e utilização de variáveis do tipo primitivo. Criaremos classes para testar esse pequeno aprendizado, sem saber exatamente o que é uma classe. Isso dificulta ainda mais a curva de aprendizado, porém cada conceito será introduzido no momento considerado mais apropriado pelos instrutores.

Passamos para o capítulo de orientação a objetos básico, mostrando os problemas do paradigma procedural e a necessidade de algo diferente para resolvê-los. Atributos, métodos, variáveis do tipo referência e outros. Passamos então para um pouco de arrays.

Os capítulos de modificadores de acesso, herança, classes abstratas e interfaces demonstram o conceito fundamental que o curso quer passar: encapsule, exponha o mínimo de suas classes, foque no que elas fazem, no relacionamento entre elas. Com um bom design, a codificação fica fácil e a modificação e expansão do sistema também.

No decorrer desses capítulos, o Eclipse é introduzido de forma natural, evitando-se ao máximo wizards e menus, priorizando mostrar os chamados *code assists* e *quick fixes*. Isso faz com que o Eclipse trabalhe de forma simbiótica com o desenvolvedor, sem se intrometer, sem fazer mágica.

Pacotes, javadoc, jars e `java.lang` apresentam os últimos conceitos fundamentais do Java, dando toda a fundação para, então, passarmos a estudar as

principais e mais utilizadas APIs do Java SE.

As APIs estudadas serão `java.util`, `java.io` e `java.net`. Todas elas usam e abusam dos conceitos vistos no decorrer do curso, ajudando a sedimentá-los. Juntamente, temos os conceitos básicos do uso de Threads, e os problemas e perigos da programação concorrente quando dados são compartilhados.

Resumindo: o objetivo do curso é apresentar o Java ao mesmo tempo que os fundamentos da orientação a objetos são introduzidos. Bateremos muito no ponto de dizer que o importante é como as classes se relacionam e qual é o papel de cada uma, e não em como elas realizam as suas obrigações. *Programe voltado à interface, e não à implementação.*

2.10 - Compilando o primeiro programa

Vamos para o nosso primeiro código! O programa que imprime uma linha simples.

Para mostrar uma linha, podemos fazer:

```
System.out.println("Minha primeira aplicação Java!");
```

Mas esse código não será aceito pelo compilador java. O Java é uma linguagem bastante burocrática, e precisa de mais do que isso para iniciar uma execução. Veremos os detalhes e os porquês durante os próximos capítulos. O mínimo que precisaríamos escrever é algo como:

```
1 class MeuPrograma {
2   public static void main(String[] args) {
3     System.out.println("Minha primeira aplicação Java!");
4   }
5 }
```

Notação

Todos os códigos apresentados na apostila estão formatados com recursos visuais para auxiliar a leitura e compreensão dos mesmos. Quando for digitar os códigos no computador, trate os códigos como texto simples.

A numeração das linhas **não** faz parte do código e não deve ser digitada; é apenas um recurso didático. O Java é case sensitive: tome cuidado com maiúsculas e minúsculas.

Após digitar o código acima, grave-o como **MeuPrograma.java** em algum diretório. Para compilar, você deve pedir para que o compilador de Java da Oracle, chamado `javac`, gere o bytecode correspondente ao seu código Java.

```
teste@andrade:/home/moreira/java$ javac MeuPrograma.java
teste@andrade:/home/moreira/java$ ls -l
total 8
-rw-r--r-- 1 teste teste 442 2006-09-06 16:41 MeuPrograma.class
-rw-r--r-- 1 teste teste 119 2006-09-06 16:36 MeuPrograma.java
teste@andrade:/home/moreira/java$
```

Depois de compilar, o **bytecode** foi gerado. Quando o sistema operacional listar os arquivos contidos no diretório atual, você poderá ver que um arquivo **.class** foi gerado, com o mesmo nome da sua classe Java.

Assustado com o código?

Para quem já tem uma experiência com Java, esse primeiro código é muito simples. Mas, se é seu primeiro código em Java, pode ser um pouco traumatizante. Não deixe de ler o prefácio do curso, que deixará você mais tranquilo em relação a curva de aprendizado da linguagem, conhecendo como o curso está organizado.

Preciso sempre programar usando o Notepad ou similar?

Não é necessário digitar sempre seu programa em um simples aplicativo como o Notepad. Você pode usar um editor que tenha **syntax highlighting** e outros benefícios.

Mas, no começo, é interessante você usar algo que não possua ferramentas, para que você possa se acostumar com os erros de compilação, sintaxe e outros. Depois do capítulo de polimorfismo e herança sugerimos a utilização do Eclipse (http://www.eclipse.org), a IDE líder no mercado, e gratuita. Existe um capítulo a parte para o uso do Eclipse nesta apostila.

No Linux, recomendamos o uso do gedit, kate e vi. No Windows, você pode usar o Notepad++ ou o TextPad. No Mac, TextMate, Sublime ou mesmo o vi.

2.11 – Executando seu primeiro programa

Os procedimentos para executar seu programa são muito simples. O javac é o compilador Java, e o java é o responsável por invocar a máquina virtual para interpretar o seu programa.

```
teste@andrade:/home/moreira/java$ java MeuPrograma
Meu primeiro programa java
teste@andrade:/home/moreira/java$
```

Ao executar, pode ser que a acentuação resultante saia errada devido a algumas configurações que deixamos de fazer. Sem problemas.

2.12 – O que aconteceu?

```
1 class MeuPrograma {
2   public static void main(String[] args) {
3
4     // miolo do programa começa aqui!
5     System.out.println("Minha primeira aplicação Java!!");
6     // fim do miolo do programa
7
8   }
9 }
```

O miolo do programa é o que será executado quando chamamos a máquina virtual. Por enquanto, todas as linhas anteriores, onde há a declaração de uma classe e a de um método, não importam para nós nesse momento. Mas devemos saber que toda aplicação Java começa por um ponto de entrada, e este ponto de entrada é o método `main`.

Ainda não sabemos o que é método, mas veremos no capítulo 4. Até lá, não se preocupe com essas declarações. Sempre que um exercício for feito, o código que nos importa sempre estará nesse miolo.

No caso do nosso código, a linha do `System.out.println` faz com que o conteúdo entre aspas seja colocado na tela.

2.13 – Para saber mais: como é o bytecode?

O `MeuPrograma.class` gerado não é legível por seres humanos (não que seja impossível). Ele está escrito no formato que a virtual machine sabe entender e que foi especificado que ela entendesse.

É como um assembly, escrito para esta máquina em específico. Podemos ler os mnemônicos utilizando a ferramenta javap que acompanha o JDK:

```
javap -c MeuPrograma
```

E a saída:

```
MeuPrograma();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return

public static void main(java.lang.String[]);
  Code:
  0:  getstatic  #2; //Field java/lang/System.out:Ljava/io/PrintStream;
  3:   ldc     #3; //String Minha primeira aplicação Java!!
  5:   invokevirtual   #4; //Method java/io/PrintStream.println:
                (Ljava/lang/String;)V
  8:   return

}
```

É o código acima, que a JVM sabe ler. É o "código de máquina", da máquina virtual.

Um bytecode pode ser revertido para o .java original (com perda de comentários e nomes de variáveis locais). Caso seu software vá virar um produto de prateleira, é fundamental usar um ofuscador no seu código, que vai embaralhar classes, métodos e um monte de outros recursos (indicamos o http://proguard.sf.net).

2.14 - Exercícios: Modificando o Hello World

1. Altere seu programa para imprimir uma mensagem diferente.

2. Altere seu programa para imprimir duas linhas de texto usando duas linhas de código System.out.

3. Sabendo que os caracteres `\n` representam uma quebra de linhas, imprima duas linhas de texto usando uma única linha de código `System.out`.

2.15 - O que pode dar errado?

Muitos erros podem ocorrer no momento que você rodar seu primeiro código. Vamos ver alguns deles:

Código:

```
1 class X {
2   public static void main (String[] args) {
3     System.out.println("Falta ponto e vírgula")
4   }
5 }
```

Erro:

```
X.java:4: ';' expected
        }
        ^
1 error
```

Esse é o erro de compilação mais comum: aquele onde um ponto e vírgula fora esquecido. Repare que o compilador é explícito em dizer que a linha 4 é a com problemas. Outros erros de compilação podem ocorrer se você escreveu palavras

chaves (as que colocamos em negrito) em maiúsculas, esqueceu de abrir e fechar as {}, etc.

Durante a execução, outros erros podem aparecer:

- Se você declarar a classe como X, compilá-la e depois tentar usá-la como x minúsculo (java x), o Java te avisa:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
                    X (wrong name: x)
```

- Se tentar acessar uma classe no diretório ou classpath errado, ou se o nome estiver errado, ocorrerá o seguinte erro:

```
Exception in thread "main" java.lang.NoClassDefFoundError: X
```

- Se esquecer de colocar `static` ou o argumento `String[] args` no método `main`:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Por exemplo:

```
class X {
  public void main (String[] args) {
    System.out.println("Faltou o static, tente executar!");
  }
}
```

- Se não colocar o método `main` como `public`:

```
Main method not public.
```

Por exemplo:

```
class X {
  static void main (String[] args) {
    System.out.println("Faltou o public");
  }
}
```

2.16 - Um pouco mais...

1. Procure um colega, ou algum conhecido, que esteja em um projeto Java. Descubra porque Java foi escolhido como tecnologia. O que é importante para esse projeto e o que acabou fazendo do Java a melhor escolha?

2.17 – Exercícios adicionais

1. Um arquivo fonte Java deve sempre ter a extensão `.java`, ou o compilador o rejeitará. Além disso, existem algumas outras regras na hora de dar o nome de um arquivo Java. Experimente gravar o código deste capítulo com `OutroNome.java` ou algo similar.

Compile e verifique o nome do arquivo gerado. Como executar a sua aplicação?

CAPÍTULO ANTERIOR:

Como Aprender Java

PRÓXIMO CAPÍTULO:

Variáveis primitivas e Controle de fluxo

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter