Capítulo 1

Introdução

"Sempre desejei que o meu computador fosse tão fácil de usar como o meu telefone. O meu desejo realizou-se. Já não sei usar o meu telefone"
Bjarne Stroustrup, criador da linguagem C

Orientação à objetos é um paradigma de programação, isto é, uma forma ou estilo de programar. Basicamente, este modelo de programação utiliza o conceito de objetos para agrupar as informações e operações de entidades abstratas objetivando a solução de problemas. Assim, dizemos que uma linguagem é orientada a objetos quando ela disponibiliza recursos linguísticos que favorecem a implementação de programas usando este estilo de programação. Com isso, podemos de certa forma tratar as linguagens como ferramentas de implementação deste modelo de programação, sendo algumas delas mais apropriadas para a construção de programas neste estilo, e outras menos.

Este material está inserido em um contexto no qual os alunos já cursaram a Unidade Curricular denominada *Algoritmos e Lógica de*

Programação que introduz os aspectos fundamentais de implementação de programas sob a forma procedural e estruturada[1] e utiliza, neste contexto, a linguagem C como ferramenta de implementação. Como, nesta Unidade Curricular será utilizada a linguagem Java como ferramenta de implementação dos conceitos de Orientação a Objetos, é necessário apresentar alguns detalhes e pricípios básicos da linguagem antes de aprender o novo paradigma. Apesar de serem linguagens totalmente diferentes, a sintaxe do Java é morfologicamente muito parecida com a linguagem C. Este Capítulo apresenta um breve resumo da sintaxe, tipos de dados, operadores, comandos básicos, estruturas linguísticas, convenções de código e características tecnológicas oferecidas pela Linguagem de programação Java.

1.1 Conceito de Máquina Virtual Java

Algumas linguagens de programação, como por exemplo, C e Pascal, compilam seus programas fontes para uma plataforma e um sistema operacional específico. O código binário gerado pela compilação é então executado pelo sistema operacional nativo para qual ele foi compilado. Isto é, este código executável poderá ser apenas executado neste sistema operacional. Para que o mesmo programa funcione em outro sistema operacional é necessário compilar o código fonte novamente utilizando um compilador adequado para este novo sistema em questão.

O Java adota um conceito de máquina virtual que adiciona uma camada extra entre o sistema operacional e a aplicação. Esta camada é responsável por executar o código Java compilado, interpretando os comandos em chamadas do sistema operacional no qual está rodando no momento. A Máquina Virtual Java ou *Java Virtual Machine (JVM)* é responsável pela interpretação dos *bytecodes* [2].

A Figura 1.1 descreve o conceito geral da Máquina Virtual Java. Um código de programa fonte em Java é compilado para bytecodes e podendo ser executado por máquinas virtuais Java específicas para diferentes plataformas.

[1] As diferenças entre os paradigmas *Orientado a Objetos* do paradigma *Procedural e Estruturado* serão discutidas no Capítulo 2.

[2] Bytecode é o termo dado ao código binário gerado pelo compilador Java.

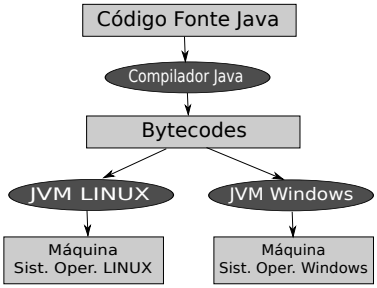


Figura 1.1: Conceito geral da Máquina Virtual Java.

1.2 Plataforma de Desenvolvimento Java

O processo de criação de aplicações ou programas na linguagem Java passa pela edição do código fonte, compilação para bytecodes e execução pela máquina virtual Java. A edição do código fonte pode ser feita por meio de um editor de texto simples. Entretanto, existem ferramentas que facilitam o desenvolvimento de aplicações Java. Estas ferramentas também são conhecidas como IDE (*Integrated Development Environment*), ou seja, um Ambiente de Desenvolvimento Integrado. Estes ambientes disponibilizam uma série de recursos para dar produtividade. Dois exemplos interessantes de IDE são: NetBeans (http://www.netbeans.org) e Eclipse(www.eclipse.org).

O JDK 5.0 é ambiente de desenvolvimento da linguagem Java, contendo o conjunto de ferramentas para compilar, depurar, executar e documentar um programa escrito em Java. O arquivo de instalação pode ser encontrado no site oficial da Sun[3], aproximadente 60 Mb de tamanho.

O procedimento de instalação geralmente é simples, executando o arquivo e seguindo os passos, escolhendo o diretório desejado. Após a instalação, é necessário configurar algumas variáveis de ambiente no sistema operacional. Basicamente são duas variáveis importantes que o Java necessita que você adicione: `JAVA_HOME` e `CLASSPATH`.

O Java utiliza a variável de ambiente chamada `CLASSPATH` para

[3] http://java.sun.com

procurar pelas classes e pacotes sempre que for necessário pelos programas. A variável `JAVA_HOME` torna possível executar o compilador Java e a máquina virtual de qualquer diretório. No Windows (NT, 2000, e XP), você poderá adicionar novas variáveis de ambiente em: *Iniciar* → *Painel de Controle* → *Sistema* → *Avançado* → *Variáveis de Sistema*.

A compilação do código fonte será realizada pelo comando `javac` seguido do nome do arquivo (`.java`) para que o compilador do Java gere o bytecode correspondente. O bytecode gerado estará nos arquivos contidos no diretório atual com a extensão (`.class`) e o mesmo nome da sua classe Java compilada. A execução do bytecode será por meio do comando `java` seguido do nome da classe, invocando a máquina virtual para interpretar o seu programa.

1.3 Estrutura da Linguagem

Como foi dito anteriormente a sintaxe da Linguagem Java segue parecida com a linguagem C. A seguir, apresentamos de forma resumida os recursos linguisticos básicos do Java.

1.3.1 Comentários

Os comentários podem ser descritos de duas formas gerais:

```
// comentário de uma única linha
/* comentário de uma ou mais linhas */
```

1.3.2 Estilo e organização

Os blocos de código são demarcados entre chaves { } e ao final de cada instrução deverá ser usado o ; (ponto e vírgula). A linguagem Java é *Case Sensitive*, isto é, significa que o compilador irá fazer a diferenciação entre letras maiusculas e minusculas.

Toda classe em Java deverá ser definida com o mesmo nome do arquivo (`.java`), sendo adotado como padrão que o primeiro caractere de todas as palavras que compõem o identificador da classe deve ser maiúsculo. Exemplos: `Pessoa`, `ContaCorrente` e `TimeDeFutebol`. Os Métodos, Atributos e Variáveis adotam como padrão o primeiro caracter minúsculo e demais palavras iniciando com Maiusculas. Exemplos: `media`, `calculaMedia()` e

codigoVip. Já as constantes devem adotar o padrão com todos os caracteres maiúsculos e divisão de palavras utilizando _. Exemplos: AZUL e AZUL_CLARO.

1.4 Tipos de Dados

Em Java, como em outras linguagens fortemente tipadas, toda variável deve ser declarada descrevendo o tipo da informação a ser armazenada. A sintaxe de declaração segue identica a utilizada pela linguagem C.

```
<Tipo> <identificador>;
ou
<Tipo> <identificador> = valor;
ou
<Tipo> <identificador> = valor, <identificador> = valor...  ;
```

Tipo Java	Tipo de Dado	Tamanho em Bytes
int	inteiro	4
byte	inteiro 0 até 255	1
long	inteiro longo	8
float	real	4
double	real longo	8
boolean	lógico	1
char	1 caractere	2

Tabela 1.1: Tipos primitivos Java.

A Tabela 1.4 apresenta alguns tipos primitivos da linguagem Java. As variáveis definas para esses tipos armazenam o valor diretamente no espaço alocado em memória. As variáveis do tipo referência armazenam o endereço de memória para um determinado valor ou objeto, que dependendo da informação e execução do programa, pode ter espaço de memória variável, como por exemplo, o tipo `String` que será descrito mais adiante.

Os valores literais que representam uma informação do tipo `float` devem receber o caractere `f` após a informação. Exemplos: `100.0f`, `5.34f` e `3.1416f`.

1.5 Operadores

Os operadores na linguagem Java adotados são os mesmos da linguagem C / C++.

1.5.1 Aritméticos

Operador	Descrição
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão (mod)
++	incremento
–	decremento

Tabela 1.2: Operadores Aritméticos Java.

1.5.2 Lógicos

Operador	Descrição
&&	conjunção (E)
||	disjunção (OU)
!	negação (NÃO)

Tabela 1.3: Operadores Lógicos Java.

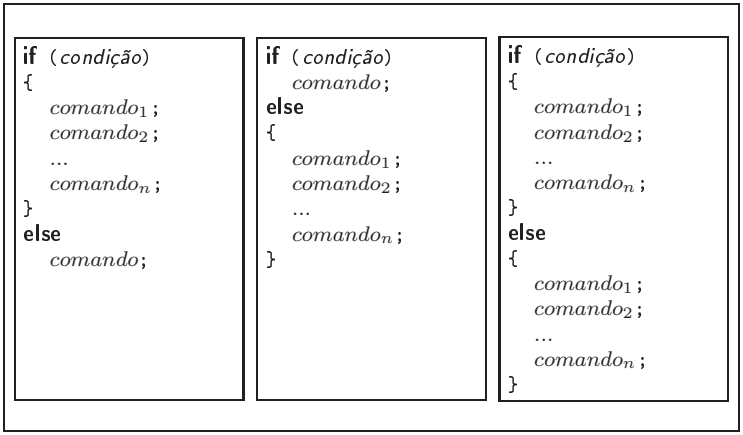
1.5.3 Relacionais

Operador	Descrição
>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
==	igualdade
!=	diferente

Tabela 1.4: Operadores Relacionais Java.

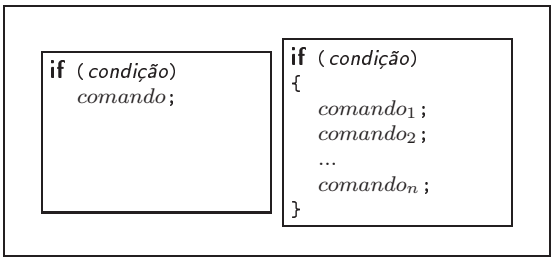
1.6 Desvios Condicionais

A Linguagem Java, como qualquer outra linguagem de programação, suporta desvios condicionais utilizando a estrutura **if** ..**else** . Quando temos mais de um comando a ser executado para um determinada condição, devemos agrupar estes comandos em um bloco delimitado por chaves, conforme indicado no Exemplo 1.1.



Exemplo 1.1 : Padrões de desvios com blocos.

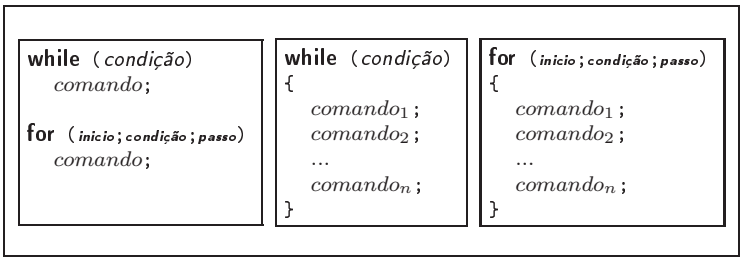
Logo, quando não há nenhum comando a ser executado no caso da condição ser falsa, devemos omitir a parte **else** do comando **if** , como indicado no Exemplo 1.2.



Exemplo 1.2 : Padrões de desvios com omissão de **else** .

1.7 Laços de Repetição

A Linguagem Java, disponibiliza diversos tipos de laços de repetição. As mais comuns são ofereciadas por meio das estruturas **while** e **for** conforme indicado no Exemplo 1.3.



Exemplo 1.3: Padrões de laços de repetição.

1.8 Manipulação de Strings

Na linguagem Java, o tipo *String* é especial. Não podemos considerá-lo como tipo primitivo. Na verdade, String é uma classe Java para manipular uma cadeia de caracteres. Como o conceito de classe apenas será mostrado em capítulos futuros, pretendemos, neste momento, apenas apresentar como manipular esse tipo de dado sem conhecimentos dos detalhes conceituais. Diferente dos tipos primitivos, a comparação de duas strings não pode ser feita explicitamente por meio dos operadoradores relacionais == != > <,etc ... Além disso, uma variável String disponibiliza algumas operações pré-definidas que ajudam a sua manipulação. Outro detalhe é que podemos utilizar operador + para concatenar uma String com uma outra, ou até mesmo um valor, produzindo uma nova String.

```
01   String s1 = "Oi gente.";
02   String s2 = "Tudo bem?";
03   String s3;
04
05   s3 = s1 + s2;
06   s2 = s1 + 85;
07   if ( s1.equals("Oi Pessoal") )
08     System.out.print("sim.  as strings são iguais");
09   else
10     System.out.print("sim.  as strings são iguais");
```

Exemplo 1.4 : Manipulação de Strings em Java.

Na linha 5 do Exemplo 1.4, a string s_3 recebe o valor *"Oi gente.Tudo Bem?"* representando a concatenação da string s_1 com s_2. O operador de concatenação $+$ pode ser aplicado a outros tipos de valores para uma string conforme indicado pela linha 6. Neste caso, a string s_2 receberá o valor *"Oi gente.85"*. A linha 7 exemplifica o uso da operação *equals* para comparação de duas strings.

Método	Descrição
charAt(n)	devolve o caracter existente na posição n
length()	devolve o tamanho da string
equals(s)	compara uma string com s

Tabela 1.5: Alguns métodos para manipulação de Strings.

1.9 Entrada e Saída de Dados

O Java disponibiliza diversas classes que manipulam a entrada e saída básicas para ler e escrever em arquivos, envio de dados via rede e o console[4].

A classe *System* disponibiliza o acesso para a saída e entrada padrão por meio do *System.out* e *System.in* respectivamente. Estes paquecotes oferecem métodos de impressão e leitura usados para manipular dados em um programa no console.

[4]Console é o nome dado a janela de linha de comandos também conhecida como *prompt* (Windows) ou *terminal* (Linux).

1.9.1 Saída de Dados

O pacote de classes *System.out* oferece dois métodos básicos para impressão. O *print* para apresentar dados na tela sem quebra de linha e *println* para apresentar dados na tela com quebra de linha ao final da impressão.

```
01  int qtde = 5;
02  System.out.print("Tenho "+ qtde + "amigos.");
03
04  float valor = 1.55f;
05  System.out.println("Tenho apenas R$:"+ valor);
```

Exemplo 1.5: Uso do System.out.print() e System.out.println().

☹ ...pensei que eu ia me livrar da linguagem C...

O Java disponibiliza a mesma função `printf()` da linguagem C como um método do pacote de classes *System.out*, conforme apresentado no Exemplo 1.6.

```
01  int idade = 20;
02  System.out.printf("idade:  %2d",idade);
03
04  float preco = 123.35f;
05  System.out.printf("Preço:  %.2f",preco);
```

Exemplo 1.6: Uso do System.out.printf().

1.9.2 Leitura de Dados

☞ ...não se preocupe, logo saberá o que é instanciar um objeto...

O Java disponibiliza a classe `Scanner` do pacote *java.util*. Essa classe implementa as operações de entrada de dados pelo teclado no *console*. Para utilizar a classe `Scanner` deve-se importar o respectivo pacote, adicionando a linha `import java.util.Scanner;` no início do arquivo de código. Além disso, deve-se instanciar um objeto desta classe, conforme apresentado no Exemplo 1.7.

```
01  import java.util.Scanner;
02
03  class class TesteLeitura {
04    public static void main(String args[]) {
06      Scanner leitor = new Scanner(System.in);
07      System.out.println("Informe uma idade:");
08      int n = leitor.nextInt();
09      System.out.println("Valor digitado:  "+n);
10    }
11  }
```

Exemplo 1.7: Uso da classe Scanner.

Tipo	Exemplo de uso
inteiro	`int n = leitor.nextInt();`
real (float)	`float preco = leitor.nextFloat();`
real (double)	`double salario = leitor.nextDouble();`
String (palavra)	`String palavra = leitor.next();`
String (texto)	`String texto = leitor.nextLine();`

Tabela 1.6: Leitura dos diversos tipos de valores.

A leitura dos diversos tipos diferentes é apresentada pela Tabela 1.9.2. Enquanto `leitor.next()` é usado na leitura de palavras simples, ou seja, não são separados pelo caractere de espaço, o comando `leitor.nextLine()` é usado na leitura de palavras compostas, como por exemplo, *"oi gente"*.