# **Unidade 1**: Introdução à Linguagem Java

## Variáveis e Operadores

Prof. Daniel Caetano

**Objetivo**: Apresentar a linguagem Java, suas principais regras, declarações de variáveis e os principais operadores.

**Bibliografia**: DEITEL, 2005; CAELUM, 2010; HOFF, 1996.

## INTRODUÇÃO

O curso de Linguagem de Programação tem o objetivo de iniciar o aluno na área da programação e prepará-lo para o desenvolvimento de aplicações básicas em Java, além de prepará-lo para compreender códigos e identificar falhas. Nesta primeira unidade, serão conhecidos os fundamentos da linguagem Java, desde suas origens e objetivos até a sintaxe básica, tipos de variáveis e operadores.

## 1. O QUE É UMA LINGUAGEM DE PROGRAMAÇÃO ?

Muitas pessoas podem pensar que uma linguagem de programação seja algo complexo. Embora não sejam de aprendizado automático, as linguagens de programação são linguagens bastante simples, com poucos elementos e grande parte das pessoas são capazes de aprendê-las em poucos dias.

O aluno, neste instante, pode não ter muita certeza disso mas, para ajudá-lo a compreender a razão pela qual uma linguagem de computador é bastante simples, é útil entender o conceito de linguagem ou língua. O que é uma língua?

Uma língua é um conjunto de símbolos e regras com as quais estes símbolos são combinados, para que seja possível transmitir uma idéia. Na língua portuguesa, por exemplo, teríamos as palavras, sinais de pontuações, ortografia e a gramática.

A parte mais complicada da linguagem de programação talvez seja, então, a programação, e não a linguagem. Mas o que é programação? Programação é a definição de uma sequência de passos para cumprir um determinado objetivo. Um exemplo clássico seria uma agenda de um projeto ou uma receita de bolo.

Assim, a linguagem de programação é uma linguagem que serve para descrever passos para se atingir um determinado objetivo. Usar essa linguagem não é difícil, como veremos; a parte mais complexa é, muitas vezes, definir qual é a sequência de passos que precisa ser realizada.

## 1.1. Definindo um Programa

Como apresentado anteriormente, uma linguagem de programação serve para descrever uma sequência de passos para cumprir um objetivo. Tal sequência de passos pode ser chamada de um **programa**.

Os programas estão em nosso dia-a-dia, ainda que nem sempre pensemos nisso. São programas uma lista de compras, uma lista de afazeres, instruções de uso de aparelhos... Entretanto, nem sempre estes programas são completos, isto é, descrições completas do que precisa ser feito. Uma lista de compras não costuma indicar onde comprar os produtos, ou a data e hora em que isso deve ser feito.

O mesmo ocorre com manuais de instruções. Um manual de DVD pode descrever a seguinte sequência para seu uso:

a) Ligue os cabos
b) Ligue o aparelho de TV e o DVD
c) Insira um DVD

Observe que são instruções genéricas e uma porção de dúvidas surgem: ligar quais cabos e onde? Ligar qual aparelho de TV? Inserir um DVD onde? O que é uma TV e um DVD? O que é um cabo?

Está claro, assim, que a linguagem em si - no caso, o português - não é suficiente para que as instruções sejam compreendidas. O que está faltando?

Basicamente, o que está faltando é definir as instruções de forma não ambígua, isto é, sem nenhum tipo de dúvida acerca do que estas instruções significam. Adicionalmente, talvez seja necessário detalhar um pouco mais estas instruções.

Esta costuma ser a maior dificuldade em se **programar um computador**: estamos acostumados a criar programas; só que estamos acostumados a criar programas que são interpretados por pessoas, que "adivinham" o que queremos com este programa e o cumprem - às vezes a contento, às vezes não.

Quando vamos programar um computador, não podemos agir desta maneira, assim como não aceitaremos se "às vezes ele agir a contento e às vezes não". Ao programar um computador, precisamos ser **específicos**, isto é, **dizer exatamente** o que queremos, e o computador seguirá à risca.

### 1.2. Usando a Enciclopédia

Obviamente ter que explicar **tudo** em detalhe para um computador pode ser uma tarefa bastante chata e, em alguns casos, árdua. Até mesmo um tarefa simples, como imprimir uma letra na tela, pode exigir centenas e centenas de linhas de código.

Como imprimir um caractere é uma tarefa muito comum - assim como diversas outras, as linguagens de programação costumam vir acompanhadas de um pacote contendo muitas e muitas destas tarefas pré-programadas, cada uma delas com um nome. Desta forma, para imprimir um caractere, basta pedir à linguagem que use o código pronto que imprime o caractere. Isso é feito pelo **nome** da tarefa, que pode ser algo simples como "**imprimir**".

Estes pacotes de tarefas prontas, que podem ser encarados como mini-programas, são chamados de **bibliotecas** e sua utilidade é enorme, pois eles reduzem em muito o nosso trabalho como desenvolvedores.

## 2. A LINGUAGEM JAVA

Existem diversas linguagens de programação disponíveis no mercado. Praticamente todas elas possuem características semelhantes, como possibilitarem a descrição precisa dos passos que uma tarefa precisa para ser concluída e disponibilizarem uma biblioteca com tarefas complexas pré-programadas.

Os símbolos e regras são bastante similares na maioria destas linguagens, estando a maior diferença entre elas justamente nas bibliotecas. Dependendo do tipo de uso que foi imaginado para uma linguagem - isto é, se ela é para a web, para banco de dados, para cálculos matemáticos etc., sua biblioteca englobará tarefas diferentes.

Assim, antes de nos aprofundarmos no estudo das linguagens de programação com o uso da linguagem Java, vamos conhecer um pouco do histórico da linguagem Java.

### 2.1. Histórico do Java

A linguagem Java foi concebida como uma linguagem para desenvolvimento de produtos eletrônicos de consumo (eletrodomésticos e eletro-eletrônicos), com software embarcado. Entretanto, ela acabou se popularizando apenas com o advento da World Wide Web e apenas recentemente vem voltando à sua vocação incial.

#### Origens

No início da década de 1990 estavam se popularizando os equipamentos eletro-eletrônicos programáveis/programados, indo desde televisores até fornos de microondas e geladeiras. Embora muitas empresas tivessem notado que as linguagens

existentes traziam problemas para o desenvolvimento destes equipamentos, foi a Sun Microsystems quem primeiro propôs uma solução.

Antes de entendermos qualquer tipo de solução, é importante entendermos qual era o problema, que talvez não seja óbvio para aqueles que nunca trabalharam com projeto de equipamentos eletro-eletrônicos.

Sempre que um projeto é realizado, uma decisão importante que deve ser feita é a definição de quais serão os componentes do equipamento que está sendo projetado. No caso de um equipamento eletrônico, componentes importantes são os eletrônicos, em especial os circuitos integrados e, no caso dos eletro-eletrônicos programáveis (ou programados), os microprocessadores.

Via de regra, o processador selecionado é aquele que tiver o menor custo, dado que atende às características básicas do projeto. Entretanto, um eletro-eletrônico pode continuar sendo produzido e vendido por vários anos; por outro lado, o preço dos processadores não é estático ao longo destes mesmos anos, fazendo com que o "processador mais barato que atenda às necessidades" possa mudar com o tempo. Nestas situações, em geral os equipamentos voltam para a prancheta e são redesenhados para acomodar um novo processador, por exemplo.

É importante ressaltar que uma economia de alguns reais em cada unidade pode levar a grandes lucros para a empresa, visto que dezenas de milhares de unidades daquele eletro-eletrônico são produzidas ao longo de um ano: um aumento de lucro que as empresas em geral não desprezam. Exemplos, em casos de video-games (SMS1/2/3/Compact, MD1/2/3, PS/PSOne, PS2/PS2Slim, PS3/PS3Slim, XBox/XBox360, GameCube/NintendoWii...)

Entretanto, a troca de um processador muitas vezes implica em troca de todo o software, já que usualmente processadores distintos têm linguagens de máquina distintas. O problema então surge: a necessidade de se re-compilar e, muitas vezes, reescrever um software para o novo processador... acaba com grande parte do lucro obtido com a troca do processador. E, mesmo quando isso não ocorria, muitas vezes significava novos "bugs" e problemas, algo bastante indesejável.

De olho nisso, em 1990, James Gosling começou a trabalhar em uma linguagem que funcionasse de tal forma que os programas raramente precisassem ser reescritos quando a plataforma onde são executados fosse substituída, desde que ambas oferecessem recursos similares. Essa linguagem acabou por ficar conhecida como Linguagem Java.

## Projetos Iniciais

Raramente uma linguagem baseada apenas em teoria e sem experimentação prática consegue ter sucesso. Por esta razão, os técnicos da Sun Microsystems, durante o desenvolvimento do Java desenvolveram projetos em Java, para testar suas funcionalidades.

O primeiro destes projetos foi o Projeto Green, que visava a criação de uma nova interface com o usuário para o equipamento "*7" (Star Seven), que tinha o objetivo de controlar os eletrodomésticos de uma casa através de ícones animados e uma touch screen. Um outro projeto foi o de VoD (Video On Demand), com uma função similar ao que hoje se chama de TV Interativa.

Entretanto, foi com o surgimento da Web que a nova linguagem realmente apareceu a público: os navegadores web estavam em franca evolução quando a Sun apresentou o WebRunner, mais tarde renomeado para HotJava. A principal característica destes browsers não era exatamente a renderização HTML (o que eles faziam de forma similar aos já existentes Mosaic e Netscape), mas sim o fato de terem capacidade de executar applets java, pequenos programas que rodavam no computador do usuário, fosse esse computador IBM PC ou Apple MacIntosh.

A inovação fez tanto sucesso que em poucas semanas a Netscape lançava sua primeira versão capaz de executar a Java Virtual Machine da Sun como plugin e, com isso, executar também applets java. Mais tarde foi incorporado no browser da Netscape também o JavaScript e, rapidamente, ambos se tornaram padrões tão importantes que é quase impossível navegar hoje sem os mesmos instalados, juntamente com o Macromedia Flash.

### O Java Hoje

O tempo foi passando e mostrou que a Sun Microsystems, de alguma forma, estava adiante de seu tempo. Com o surgimento dos PDAs (Personal Data Assistants, os "PALMs") e telefones celulares capazes de executar aplicativos, tornou-se bastante atrativa uma tecnologia que permitisse que um programa pudesse ser executado em máquinas diferentes: afinal de contas, não só os recursos disponíveis nestes equipamentos, como também seus processadores e arquiteturas podem ser bastante diferentes até mesmo de um modelo para outro!

Assim, hoje o Java voltou a ter sua vocação inicial: desenvolvimento de software embaracado em eletro-eletrônicos. Ainda não é muito comum, mas vem crescendo o número de equipamentos como Set-Top-Boxes (HDTV), modems ADSL, computadores portáteis, DVD players, TVs e outros equipamentos que se utilizam de programas escritos na linguagem Java para permitir que o usuário se comunique com o equipamento.

## 2.2. Como Funciona o Java

Como um programa em Java consegue rodar em qualquer lugar? Como um código feito para "máquina nenhuma" consegue rodar em qualquer lugar? Na verdade, o segredo está no **Interpretador Java**, também conhecido como **Java Virtual Machine (JVM)**, que é um programa que precisa ser reescrito para cada processador e equipamento.

A JVM exerce o papel de um "tradutor simultâneo". É ela quem lê o programa Java e diz para um computador específico o que deve ser feito para realizar aquela tarefa. Ela

funciona como um intermediário. É como um intérprete de um técnico de futebol que não fala a língua dos jogadores:

| *Nome do Técnico* | *Língua do Técnico* | *Conversão* | *Língua dos Jogadores* |
|---|---|---|---|
| Luis Felipe | Português | Intérprete P/A | Árabe |
| Luis Felipe | Português | Intérprete P/I | Inglês |
| Luis Felipe | Português | Intérprete P/J | Japonês |

| *Nome do Programa* | *Línguagem do Programa* | *Conversão* | *Línguagem do Processador* |
|---|---|---|---|
| MeuPrograma | Java | JVM J/P4 | Pentium IV ASM |
| MeuPrograma | Java | JVM J/PPC | PowerPC ASM |
| MeuPrograma | Java | JVM J/A7 | ARM7 ASM |

Perceba que ao trocar a língua do time, não é preciso trocar o técnico nem a língua que ele fala, pois existe um intérprete que faz as traduções. Se trocar o time e mantiver o técnico, basta trocar o intérprete. No caso do programa em Java, ocorre o mesmo: não é preciso trocar o programa nem a linguagem dele quando se troca de processador: basta trocar a JVM.

Como existe um passo a mais de tradução, isso tem influência direta no desempenho das aplicações Java. Apesar de aplicações Java possuírem um desempenho bastante superior ao de linguagens script normais, seu desempenho pode ser bastante mais lento que uma linguagem compilada como C. Entretanto, os fabricantes não têm se mostrado muito preocupados com esse "problema", dado que os equipamentos têm poder de processamento cada vez maior a custos cada vez menores: preservar o investimento em software desenvolvido acaba sendo muito mais importante quando se visa lucro em alguns mercados (como o dos celulares).

Nas versões mais recentes, a Sun se empenhou em resolver o problema "desempenho", sempre associado à linguagem Java. Para isso criaram um sistema chamado de "hotspots", com o uso da tecnologia JIT (Just-in-Time), que compilam o código à medida em que ele é executado, com grande otimização, permitindo que, em muitos casos, programas em Java de versão recente sejam executados em velocidade similar a programas em C ou C++.

## 2.3. O Java é bom para o Programador?

Além de todas estas vantagens, o Java ainda tem uma porção de vantagens para os programadores, pois o interpretador Java cuida de uma série de aspectos, como gerenciamento de memória, e a programação é feita de tal maneira que ajuda o programador a organizar melhor o seu programa e, para finalizar, possui uma vasta biblioteca com milhares de tarefas prontas.

## 2.4. Ok, Mas QUAL Java?

O Java passou por muitas evoluções ao longo dos anos e a nomenclatura não foi feita de uma maneira organizada. Por esta razão, os programadores novatos normalmente se confundem com a mesma. Vejamos quais foram as versões:

| Nome Popular | Versão | Mudanças |
|---|---|---|
| Java | 1.0/1.1 | Versão original |
| Java2 | 1.2, 1.3 e 1.4 | Muitos novos recursos |
| Java 5 | 1.5 | Muitos novos recursos |
| Java 6 | 1.6 | Melhorias de performance |

Existem também algumas siglas usadas, que são importantes:

- JVM: Java Virtual Machine
- JRE: Java Runtime Machine (A JVM e bibliotecas básicas para execução)
- JDK: Java Development Kit (JRE + tudo que é necessário para o desenvolvimento)

## 3. PRIMEIRO PROGRAMA EM JAVA

Usando o NotePad++ ou a interface do NetBeans, digite o seguinte programa e grave com o nome EXATAMENTE como indicado:

**UmPrograma.java**

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

Se digitou o programa, aperte o botão de compilação do NetBeans ou, se usou o NotePad++, digite, no prompt, o seguinte comando:

**javac UmPrograma.java**

Isso irá gerar um arquivo chamado **UmPrograma.class**, que é o seu programa já na pronto para ser executado pelo Java!

Execute-o com o seguinte comando:

**java UmPrograma**

O resultado será:

```
Olá, mundo!
```

Antes de continuarmos, vamos entender um pouco o que aconteceu. Vamos entender o programa linha por linha.

Inicialmente, encontramos essa linha:

**UmPrograma.java**

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

Antes de mais nada, é preciso entender que todo programa Java está organizado em blocos chamados Classes, que esturademos melhor no futuro. Cada classe principal deve ter seu próprio arquivo, que deve ter o nome EXATAMENTE igual ao da classe (incluindo maiúsculas e minúsculas, não use caracteres especiais e acentos). Convenciona-se que todas as palavras que compõem um nome de classe sempre começam com uma letra maiúscula e as outras são minúsculas. Daí o nome da nossa classe: UmPrograma.

Como uma classe é um bloco, é necessário usar indicadores para marcar onde ela começa e onde ela acaba. O java usa as chaves como marcadores, usando o abre chaves { para indicar o início de um bloco e o fecha chaves } para indicar o fim de um bloco.

E tudo que estiver entre o abre e o fecha chaves será colocado pelo Java dentro do bloco "UmPrograma". Dentro deste bloco, a primeira linha é:

**UmPrograma.java**

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

Esta linha indica que um bloco de tarefa será definido (note o abre chaves). Um bloco de tarefas é chamado de **método**.

Uma classe pode conter vários métodos e, em geral, possui pelo menos um. Sendo assim, é útil dar um **nome** a cada método, de maneira que, quando precisarmos cumprir aquela tarefa, possamos pedir para o Java executá-lo simplesmente mencionando o nome.

O método representado no programa chama-se "main" (principal) e é um nome especial de método: sempre que uma classe tiver um método main ele será chamado automaticamente quando executarmos o arquivo da classe. As palavras antes de "main" são informações sobre a função, que estudaremos no futuro.

Um nome de método sempre vem seguido de parênteses ( ) e os valores dentro destes parênteses também possuem um significado, que veremos mais adiante no curso.

Em nosso programa, dentro do método main, temos apenas uma instrução:

**UmPrograma.java**

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

Agora ficará claro a importância de darmos nome para as classes e funções:

**System.out**

É o nome de uma classe que agrega métodos de impressão na tela.

**println**

É o nome de um método da classe System.out que **imprime uma linha** na tela. Assim, quando indicamos:

System.out.println( "..." );

Significa que estamos pedindo que o Java execute o método **println** da classe **System.out**, imprimindo o texto que estiver indicado dentro dos parênteses. A razão para precisar indicar o nome do método e o nome da classe é que outras classes podem ter métodos com o mesmo nome **println**, como, por exemplo, uma classe que imprima na impressora.

É importante observar que após indicar o método existe o sinal de ; (ponto-e-vírgula). Isso é importante porque isso indica que a **instrução** que estamos dando **acabou**. Ou seja: a instrução pode ser executada pelo Java.

### 3.1. Mas o Java Entende Isso que Eu Escrevi?

Sim, entende tudo que escrevemos, como a execução do código mostrou. Entretanto, a JVM não entende a **língua** em que escrevemos. Para isso, é necessário "traduzir" ou, em termos técnicos, "compilar" esse texto, para a língua que a JVM realmente entende, chamada **bytecode**. Isso foi feito com o programa JavaC (Java Compiler), usado no item anterior. As extensões do arquivo nos indicam qual a língua que está sendo usada dentro do arquivo:

| | |
|---|---|
| .java | Programa legível por seres humanos, como o escrevemos |
| .class | Programa legível pela JVM, em bytecode |

## 4. VARIÁVEIS EM JAVA

O programa que escrevemos anteriormente é bastante didático porque permite apresentar algumas das principais partes de um programa em Java. Entretanto, o programa que escrevemos não é muito útil, ele apenas imprime uma frase que nós mesmo escrevemos.

Para tornar um programa mais útil, é interessante usarmos as **variáveis**. Variáveis são locais na memória do computador em que guardamos dados para que possamos realizar operações com eles. Como é muito complicado lidar diretamente com posições de memória de um computador, deixaremos que o Java faça isso por nós. É como se tivéssemos um empregado responsável por cuidar de nossas roupas, pegá-las e guardá-las no armário sempre que precisamos.

Para que o Java possa gerenciar nossas variáveis, ou seja, gerenciar o uso da memória do computador, precisamos informá-lo que **tipo de dado** queremos guardar na memória e, mais importante, dar um **nome** para este lugar da memória já que, sem um nome, não teríamos como recuperar aquele valor no futuro. Essa informação é feita através de uma **declaração de variável**. O formato geral é o seguinte:

```
tipoDeVariavel nomeDaVariavel;
```

Uma variável pode ser deeclarada em qualquer lugar em um programa, mas ela só tem validade dentro do bloco em que foi declarada; em outras palavras, fora daquele bloco em que ela foi criada, é como se ela não existisse.

Há varios tipos de variáveis possíveis, como veremos mais adiante. O tipo mais comum, e o tipo "número inteiro", que é indicado para o Java como **int**. Assim, se quisermos pedir para o Java arrumar um espaço para guardarmos uma idade, que é um número inteiro, podemos indicar da seguinte forma:

**OutroPrograma.java**

```
class OutroPrograma {
   public static void main(String[] args) {
      int idade;
      }
   }
```

Uma variável, entretanto, só é útil quando colocamos algum valor dentro dela. Isso pode ser feito facilmente usando o sinal de igualdade, em uma instrução como a indicada abaixo:

**OutroPrograma.java**

```
class OutroPrograma {
   public static void main(String[] args) {
      int idade;

      idade = 18;
      }
   }
```

Observe que só podemos guardar informações dentro de uma variável **depois** que ela for declarada; se não for respeitada esta ordem, ocorrerá um erro ao compilar ou executar o código.

Podemos imprimir o valor de uma variável, como é indicado a seguir:

**OutroPrograma.java**

```
class OutroPrograma {
   public static void main(String[] args) {
      int idade;

      idade = 18;
      System.out.println( idade );
      }
   }
```

É possível criar outras variáveis e colocar o valor de uma delas na outra, como indica o código seguinte:

**OutroPrograma.java**

```
class OutroPrograma {
   public static void main(String[] args) {
      int idade;
      int outraIdade;

      idade = 18;
      outraIdade = 20;

      idade = outraIdade;
      System.out.println( idade );
      }
   }
```

O resultado agora será diferente, pois o valor de idade não é mais 18 quando o seu valor é impresso: ele recebeu o valor que estava na variável outraidade, que era 20.

Mas isso continua não sendo muito útil. Para tornar isso realmente útil, precisaremos realizar **operações** com estes números!

## 5. OPERAÇÕES EM JAVA

As operações básicas em Java são feitas com os símbolos clássicos:

| | |
|---|---|
| + | soma |
| - | subtração |
| * | multiplicação |
| / | divisão |
| % | resto de divisão |

Um exemplo de uso segue abaixo:

**Operacoes.java**

```
class Operacoes {
   public static void main(String[] args) {
      int idade;
      int idadeMaisUm;

      idade = 18;
      idadeMaisUm = idade + 1;
      System.out.println( idade );
      System.out.println( idadeMaisUm );
      }
   }
```

Qualquer cálculo pode ser feito. Por exemplo, para saber aproximadamente o número de semanas que uma pessoa viveu, podemos multiplicar a idade dela em anos pelo número de semanas de um ano, que é 52:

**Operacoes.java**

```
class Operacoes {
   public static void main(String[] args) {
      int idade;
      int idadeEmSemanas;

      idade = 18;
      idadeEmSemanas = idade * 52;
      System.out.println( idadeEmSemanas );
      }
   }
```

Substitua o valor "18" pela sua idade e descubra, aproximadamente, quantas semanas você viveu do dia em que nasceu até o seu último aniversário.

Um cáclulo mais complexo pode ser feito, combinando várias operações, por exemplo:

**NovasOperacoes.java**

```
class NovasOperacoes {
   public static void main(String[] args) {
      int x;
      int y;

      y = 12;
      x = 2*y + 37;

      System.out.println( x );
      }
   }
```

## 6. TIPOS DE VARIÁVEIS EM JAVA

Como foi dito anteriormente, existem vários tipos básicos de variáveis em Java. Estes tipos estão indicados a seguir, com a faixa de valores que podem ser armazenados nessas variáveis e também com o uso comum.

| Tipo | Tamanho | Uso |
|---|---|---|
| boolean | 1 bit | Valores do tipo "falso" ou "verdadeiro" |
| byte | 1 byte | Valores inteiros de -127 a 128 |
| short | 2 bytes | Valores inteiros de -32767 a 32768 |
| char | 2 bytes | Códigos de caracteres |
| int | 4 bytes | Valores inteiros de aprox. -2.000.000 a +2.000.000 |
| float | 4 bytes | Valores de ponto flutuante de simples precisão |
| long | 8 bytes | Valores inteiros muito grandes |
| double | 8 bytes | Valores de ponto flutuante de dupla precisão |

O programa abaixo mostra como usar uma variável float para multiplicar um valor inteiro por 1.5. Observe, porém, que o resultado desta operação será um número de ponto flutuante, isto é, com casas decimais e, portanto, ele precisa ser armazenado em uma variável de ponto flutuante.

**OperacoesTipos.java**

```
class OperacoesTipos {
   public static void main(String[] args) {
      int idade;
      double indice;
      double resultado;

      idade = 18;
      indice = 1.5;
      resultado = 18 * indice;
      System.out.println( resultado );
      }
   }
```

### 6.1. Casting (opcional)

Caso tentemos armazenar um valor double ou float em uma variável inteira, seja ela de qual tipo for, pode ocorrer "perda de precisão", isto é, os valores depois da vírgula serão perdidos. Assim, se multiplicarmos 1.75 por 10 e armazenarmos o resultado em uma variável do tipo inteira, o valor armazenado será 17, e não 17,5.

Entretanto, se simplesmente escrevermos isso:

```
int x
x = 10 * 1.75;
```

O Java irá reclamar, dizendo que isso está errado. Se quisermos fazer isso, precisamos usar um recurso de "casting". Casting significa dizer para o Java "converte do jeito que eu estou mandando, eu sei que vai ter perda de precisão e você não precisa reclamar". O casting é feito com o nome do tipo "destino" entre parênteses, como indicado no código a seguir:

**Casting.java**

```
class Casting {
   public static void main(String[] args) {
      int resultado;

      resultado = (int) 1.75 * 10;
      System.out.println( resultado );
      }
   }
```

A indicação (int) feita força o Java a converter o resultado para int **antes** de colocar o resultado na variável resultado.

É comum termos de usar casting de uma outra maneira, quando vamos criar um número do tipo float. Por exemplo:

**CastingFloat.java**

```
class CastingFloat {
   public static void main(String[] args) {
      float umaVariavel;

      umaVariavel = 0.5;
      System.out.println( umaVariavel );
      }
   }
```

Se tentar compilar esse código, você receberá um erro na linha:

```
      umaVariavel = 0.5;
```

Mas o que há de errado nesta linha? Simples: por padrão, o java considera que o valor numérico 0.5 digitado é do tipo **double**, e ao colocar um valor double em uma variável float, há perda de precisão.

Para contornar esse erro sem ter que usar o casting **(float)** em um caso tão simples, existe uma forma alternativa de indicar o número 0.5, acrescentando um **f** na frente, indicando se tratar de um número float, não double, como indicado no código a seguir.

**CastingFloat.java**

```
class CastingFloat {
   public static void main(String[] args) {
      float umaVariavel;

      umaVariavel = 0.5f;
      System.out.println( umaVariavel );
      }
   }
```

Este código irá compilar normalmente. Esta mesma regra vale para variáveis do tipo **long**, que precisarão ser associadas com um número seguido da letra **l**, como indicado a seguir.

**CastingLong.java**

```
class CastingFloat {
   public static void main(String[] args) {
      long umaVariavel;

      umaVariavel = 10000000001;
      System.out.println( umaVariavel );
      }
   }
```

## 7. EXERCÍCIOS

Observe o programa abaixo:

**UmPrograma.java**

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

**1)** Altere o programa para imprimir uma mensagem diferente

**2)** Altere o programa para imprimir DUAS mensagens

**3)** Sabendo que o código

\n

serve para "quebrar linha", escreve as mesmas duas linhas do item 2 com um único System.out.println().

Você saberia dizer se a forma do item 3 é melhor ou pior do que a do item 2? E saberia dizer o porquê?

**4)** Na empresa onde trabalhamos, há tabelas com o quanto foi gasto em cada mês. Para fechar o balanço do primeiro trimestre, precisamos somar o gasto total. Sabendo que, em Janeiro, foram gastos 15000 reais, em Fevereiro, 23000 reais e em Março, 17000 reais, faça um programa que calcule e imprima o gasto total no trimestre. Siga esses passos:

a) Crie uma classe chamada BalancoTrimestral com um bloco main, como nos exemplos anteriores;

b) Dentro do main (o miolo do programa), declare uma variável inteira chamada gastosJaneiro e inicialize-a com 15000;

c) Crie também as variáveis gastosFevereiro e gastosMarco, inicializando-as com 23000 e 17000, respectivamente, utilize uma linha para cada declaração;

d) Crie uma variável chamada gastosTrimestre e inicialize-a com a soma das outras 3 variáveis:

```
int gastosTrimestre = gastosJaneiro + gastosFevereiro + gastosMarco;
```

e) Imprima a variável gastosTrimestre.

**Extras**

**4)** Se conhecer alguém que está trabalhando em um projeto Java, pergunte a ele o que ele pensa sobre a linguagem e por que o Java foi escolhido para este projeto.

## 8. BIBLIOGRAFIA

CAELUM, FJ-11: Java e Orientação a Objetos. Acessado em: 10/01/2010. Disponível em: < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos/ >

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

HOFF, A; SHAIO, S; STARBUCK, O. **Ligado em Java**. São Paulo: Makron Books, 1996.