Nem tudo depende
do tempo, às vezes as
coisas dependem só de
uma atitude.


# Desenvolvimento de Aplicações Desktop

*Variáveis, Constantes, Entrada e Saída*

*Professor: Charles Leite*

# Um Programa em Java

Java é uma LP TOTALMENTE Orientada a Objetos (OO)

A construção de um programa em Java é, fundamentalmente, baseada no conceito de CLASSE

# Um Programa em Java

Um programa em Java é constituído por uma ou mais classes

Uma classe contém um ou mais atributos e métodos

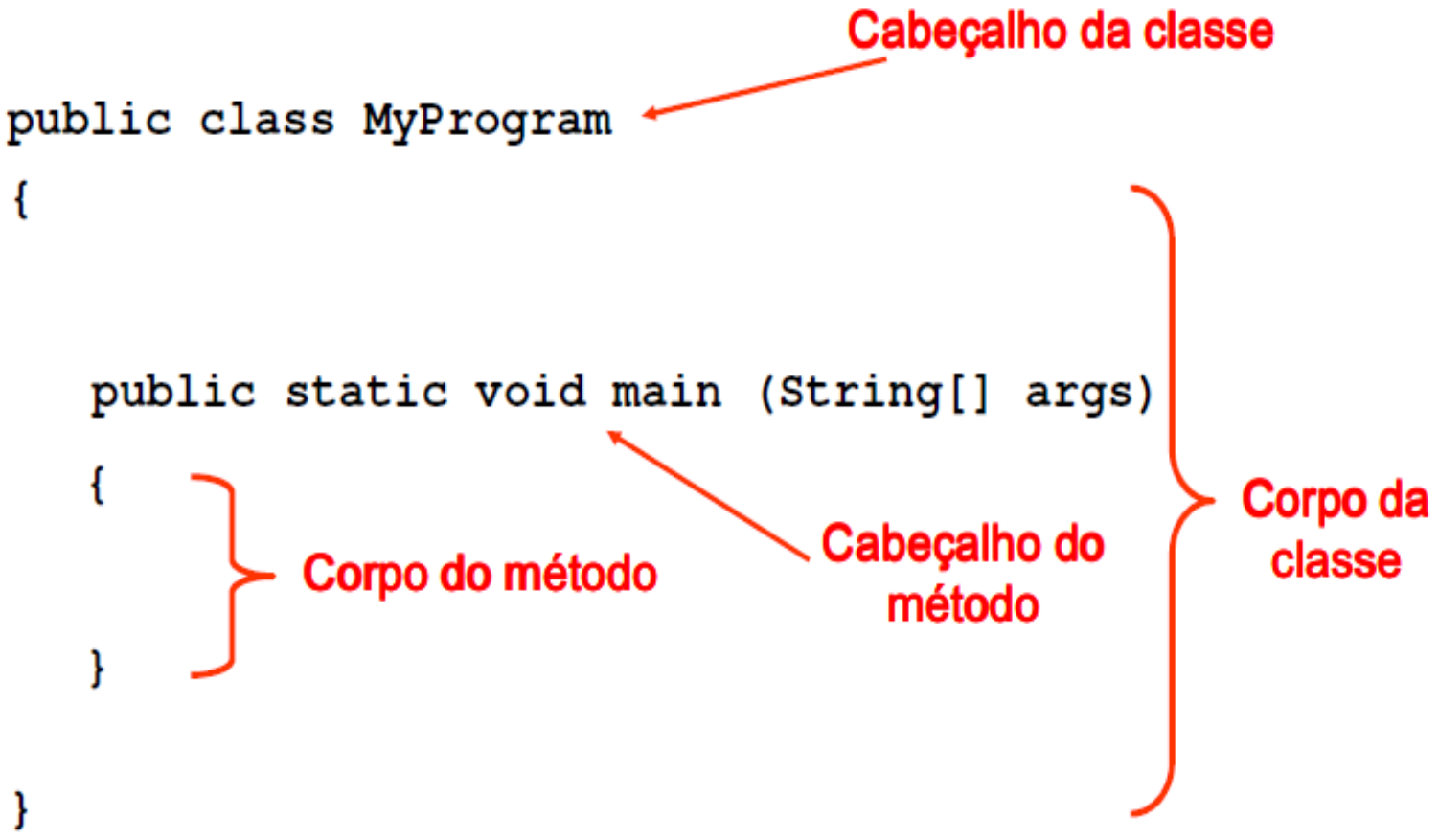
- Atributos (variáveis de instância) → Características
- Métodos → Comportamentos (funcionalidades)
  - Compostos por instruções

# Um Programa em Java

- Toda aplicação (programa) em Java deve conter, pelo menos, uma classe onde existirá um método chamado **`main`**
  - Esse método é responsável por disparar a execução da aplicação
  - Ou seja, a aplicação inicia a sua execução a partir do método **`main`**

```
public static void main(String args[])
```

# Um Programa em Java



# Comentários

Comentário é um mecanismo oferecido pelas linguagens de programação que permite ao programador expressar, em linguagem natural, a lógica pensada para escrever um programa ou trecho de programa

# Comentários

- Um programa deve ser bem documentado
- Para tal, deve-se escrever comentários sempre que necessário
  - Eles não afetam a funcionalidade dos programas
- Permitem que os programadores definam e comuniquem suas idéias, independente do código
  - Eles fornecem uma compreensão melhor da intenção do programador
  - Um programa pode ser usado por muito tempo e, portanto, pode sofrer muitas alterações
    - O programador original pode não lembrar de detalhes da codificação
    - Um novo programador pode assumir o comando

# Comentários

A documentação de um programa pode ser:

- Documentação externa para usuários
- Documentação interna (*inline*) no próprio programa
  - Aumenta a legibilidade do código

# (Formas de) Comentários

1. Comentários de uma linha
   - Devem ser precedidos por //
   - Exemplo:

```
// Este é um comentário de uma
linha
```

# (Formas de) Comentários

2. Comentários de múltiplas linhas

   - Devem estar entre /*  */
   - Exemplo:

```
/* Este tipo de comentário só
termina quando o asterisco-
barra é encontrado */
```

# (Formas de) Comentários

3. Comentários *Javadoc*

   - Comentários que podem ser usados para gerar automaticamente uma documentação externa de um programa, através da ferramenta *javadoc*
   - Devem estar entre /** */
   - Exemplo:

```
/** Isto é um comentário javadoc */
```

# (Formas de) Comentários

- Os dois primeiros tipos de comentários podem ser usados em conjunto, criando diversos estilos de documentação

```
// Este é um comentário de uma linha

// ---------------------------------
// Diversos comentários de uma linha
// Podem ser usados para explicar as
  diversas
// característica do programa:
// propósito, autor, data de criação, etc.

/*
  Este é um comentário de diversas linhas
*/
```

# Identificadores

- São palavras que os programadores usam para descrever seus programas
  - Nomes de variáveis, métodos, classes, pacotes, etc.
- Os identificadores podem, também, ser nomes que terceiros escolheram para construir seus próprios programas
  - São, na verdade, partes de bibliotecas
    - Ou seja, conjunto de métodos e classes escritos (código) por outras pessoas com finalidades específicas
  - Por exemplo, o nome de uma classe ou método de uma API

# Identificadores

- Os identificadores podem ser, também, PALAVRAS RESERVADAS da linguagem
  - Identificadores que possuem um propósito específico na linguagem
    - Só podem ser usados de forma pré-definida
  - Não podem ser usados para outros propósitos

# Identificadores

- Palavras reservadas

| | | | | |
|---|---|---|---|---|
| abstract | default | goto* | package | this |
| assert | do | if | private | throw |
| boolean | double | implements | protected | throws |
| break | else | import | public | transient |
| byte | enum | instanceof | return | true |
| case | extends | int | short | try |
| catch | false | interface | static | void |
| char | final | long | strictfp | volatile |
| class | finally | native | super | while |
| const* | float | new | switch | |
| continue | for | null | synchronized | |

# Identificadores

- Um identificador é formado por um conjunto de letras, dígitos, o caractere *underscore* (_) e o caractere *dollar* ($)
- Identificadores não podem começar com um dígito
  - Exemplos válidos:
    - `_a`, `a3`, `bom_dia`
  - Exemplos inválidos:
    - `2a`, `a-b`, `a b`
- Java é *case-sensitive*
  - Os identificadores `casa` e `CASA` são diferentes

# Identificadores

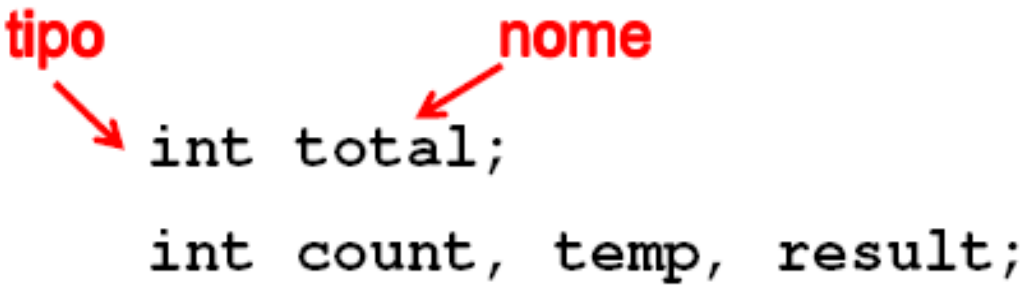
- Dicas para construção de identificadores:
  - Escolher nomes que representem bem a entidade (classe, variável, constante, método, ...) que está sendo modelada
  - Os nomes devem ser auto-descritivos
  - Java permite identificadores (nomes) de qualquer tamanho
    - Porém, deve-se ter o cuidado para não escrever nomes muito grandes ou pequenos

# Variáveis

- Programas manipulam dados e esses dados são armazenados em VARIÁVEIS
- Uma variável é uma posição (localização) na memória, referenciada por um identificador (nome)
- Uma variável deve ser declarada informando o tipo de dado que ela armazenará, seguido pelo seu nome

# Variáveis

## Declaração de variáveis




Muitas variáveis podem ser criadas em uma declaração

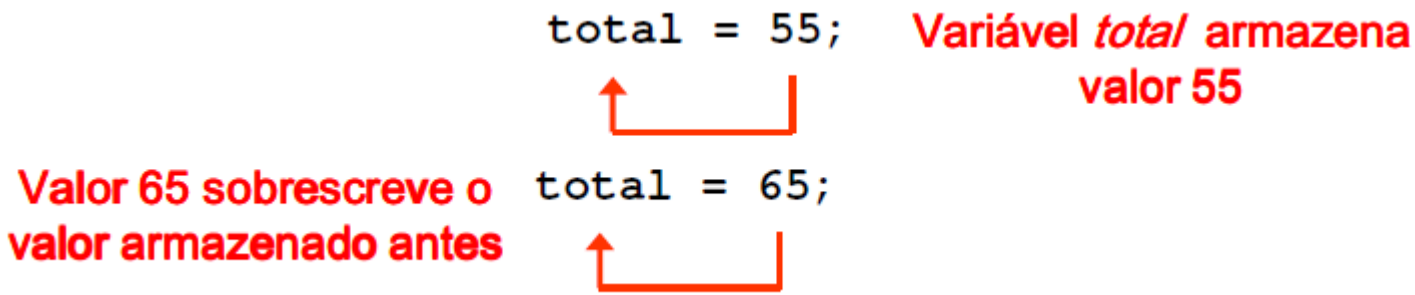
# Variáveis

- Uma declaração de VARIÁVEL instrui o compilador a reservar um espaço de memória suficiente para armazenar um valor do tipo de dado declarado
  - É o nome ao qual iremos referenciar essa posição de memória
- Só após a declaração da variável, é que ela poderá ser referenciada (usada)
- Quando uma variável é referenciada no programa, o valor armazenado nela é usado

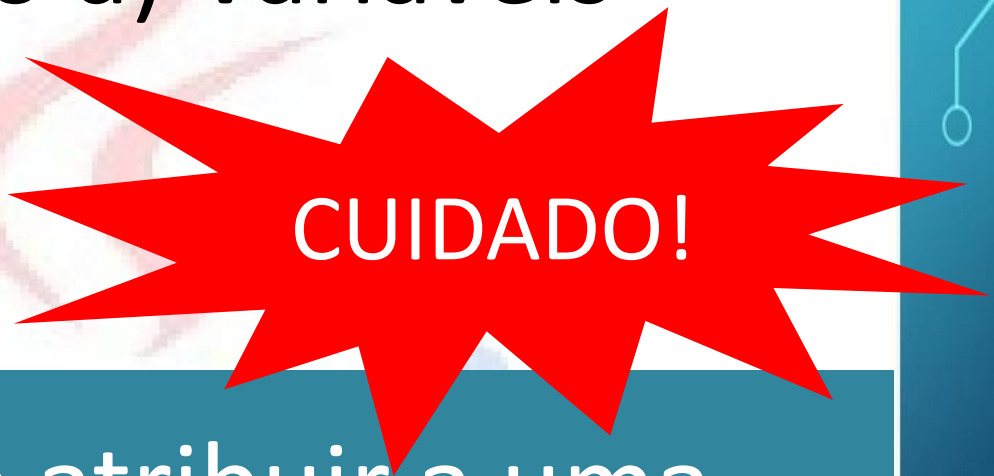
# (Atribuição a) Variáveis

- Um COMANDO DE ATRIBUIÇÃO modifica o valor armazenado numa variável

- O operador de atribuição, em Java, é o sinal "="



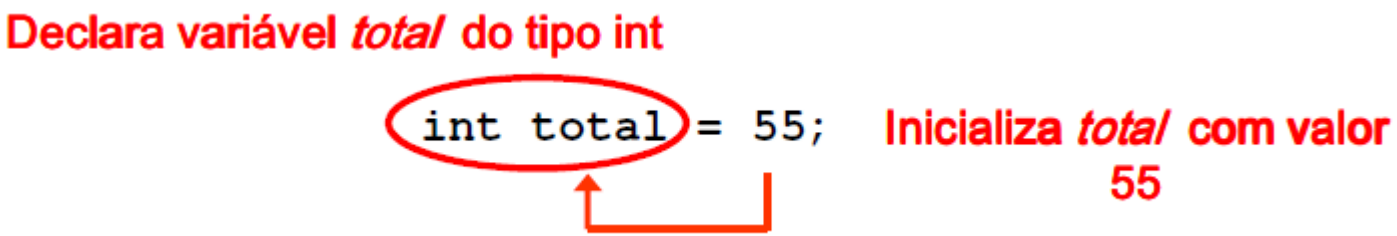
# (Atribuição a) Variáveis




Só se pode atribuir a uma variável valores compatíveis com o tipo declarado da variável – Java é FORTEMENTE TIPADA

# (Inicialização de) Variáveis

- Uma variável pode ser INICIALIZADA com o comando de atribuição na hora de sua declaração



# (Inicialização de) Variáveis

- Caso variáveis não sejam inicializadas, Java atribui valores padrões que dependem do tipo da variável
  - `0` para tipos numéricos
  - Caractere de código `\u0000` para caracteres
  - `false` para booleanos
  - `null` para referências a objetos

Esta regra não vale para variáveis declaradas dentro de métodos – variáveis locais

# Constantes

- Uma CONSTANTE é um identificador semelhante a uma variável, exceto pelo fato de que ela armazena o mesmo valor durante toda a sua existência
- São úteis para dar um significado mais compreensível a determinados valores
  - Por exemplo, `LIMITE_VAGAS` é mais compreensível que o valor `2546`
- Facilitam a manutenção do programa
  - Caso uma alteração no programa, que acarrete uma mudança no valor da constante, seja necessária e essa constante seja referenciada em vários lugares do programa, só precisamos alterar o valor da constante no lugar em que foi declarada
- Explicitam formalmente que um determinado valor não pode ser alterado
  - Evitam erros de outros programadores

# Constantes

- Uma constante é declarada usando a palavra reservada **`final`**
- Deve-se inicializar a constante no ato da sua declaração

```
final float PI = 3.1416;
PI = 3.141618;
```

Esse comando gera um erro de compilação

Não se pode mudar o valor de uma constante

# Tipos de Dados

Em Java, os tipos de dados podem ser:

- **Primitivos** para representar valores numéricos, caracteres e valores booleanos
- **Referências** para referenciar objetos de classes existentes
  - Tipos referências são tipos de classes

# Valores Numéricos

- Java pode representar dois tipos de valores numéricos:
  - Inteiros
  - Ponto flutuante (reais)
- A diferença entre os diversos tipos numéricos está na quantidade de espaço ocupada pela memória para armazená-lo
  - Determina a faixa de valores que pode ser representada

# Valores Numéricos

| Tipo | Tamanho | Valor Min. | Valor Max. |
|---|---|---|---|
| `byte` | 8 bits | -128 | 127 |
| `short` | 16 bits | -32768 | 32767 |
| `int` | 32 bits | - 2147483648 | 2147483647 |
| `long` | 64 bits | - 92233720368547580 8 | 92233720368547580 7 |
| `float` | 32 bits | Aproximadamente, -/+3.4E+38 com 7 dígitos significativos | |
| `double` | 64 bits | Aproximadamente, -/+1.7E+308 com 15 dígitos significativos | |

# Caracteres

- Caracteres, em Java, são representados pelo tipo **`char`**
- Uma variável do tipo **`char`** ocupa 16 bits
- Um literal caractere é delimitado por aspas simples
- Exemplos:

```
char a = 'A';
char b = 'g';
char c = 65; // representa o
  caractere A
char d = ',';
```

# Valores Booleanos

- Apenas dois valores são representados:
  - `True`
  - `False`
- Em Java, um valor booleano é representado pelo tipo **`boolean`**

# E Cadeias de Caracteres?

- Os valores de cadeias (*strings*) de caracteres, em Java, não são considerados tipos primitivos
- Uma *string* de caracteres, em Java, é considerada um objeto, definido pela classe `String`
- O valor (literal) de uma *string* é delimitado por aspas duplas e pode conter quaisquer caracteres válidos, incluindo números, símbolos de pontuação e outros caracteres especiais
  - Por exemplo, “Java é a LP da disciplina Programação OO”
- Para concatenar duas ou mais *strings*, usamos o operador de adição (+)
  - Por exemplo, “Java é a LP da “ + “disciplina Programação OO”

# Entrada e Saída de Dados

Lembram dos comandos **leia** e **escreva** de Lógica de Programação?

# Entrada e Saída de Dados

- Para a **saída** de dados, Java utiliza o objeto `System.out`, que representa um dispositivo de saída ou um arquivo
  - Por *default*, esse objeto representa a tela de um monitor
  - Para ser mais preciso, o nome do objeto é `out`, do tipo `PrintScream`, que está armazenado na classe `System`, parte do pacote `lang` da biblioteca de classes padrão do JDK

# Entrada e Saída de Dados

- O objeto `System.out` possui vários métodos, dentre eles, os métodos (e suas variações):
  - `print()`
  - `println()`
  - Enquanto o método `println()` imprime uma informação na tela e pula para o início da próxima linha, o método `print()` permanece na mesma linha

# Entrada e Saída de Dados

- Exemplo:
  - A seguinte sequência de instruções

```
System.out.print("10, ");
System.out.print("9 e ");
System.out.println("8.");
System.out.println("Essas serão as minhas notas em Java, no mínimo.");
```

  - Gera a seguinte saída na tela:

```
10, 9 e 8.
Essas serão as minhas notas em Java, no mínimo.
```

# Entrada e Saída de Dados

- Para **entrada** de dados, Java utiliza a classe `Scanner`, parte do pacote `util` da biblioteca de classes padrão do JDK
- Ela fornece métodos para leitura de valores de diversos tipos
- A entrada pode vir de várias fontes, como, dados digitados pelo usuário ou armazenados em um arquivo
- A classe `Scanner` pode também ser usada para avaliar uma string de caracteres em partes separadas

# Entrada e Saída de Dados

- Por exemplo, se quisermos ler dados digitados pelo usuário, a partir do teclado:

```
Scanner leia = new Scanner(System.in);
```

Objetos em Java são criados com o operador **new**

A fonte de entrada; neste caso, um objeto que representa uma *stream* de entrada padrão, ou seja, o teclado

# Entrada e Saída de Dados

- Em seguida, podemos usar os métodos da classe `Scanner`
- Por exemplo:

```
Scanner leia = new Scanner(System.in);
String nome = leia.nextLine();
String nome = leia.next();
int idade = leia.nextInt();
double salario = leia.nextDouble();
```

Os métodos `next`... servem para ler os próximos valores digitados pelo usuário

# DÚVIDAS ...