LINGUAGEM DE PROGRAMAÇÃO ORIENTADA À OBJETOS – LPOO

Aula 04 – Using Data (Parte I e II)

Profa. Thais Rocha – thais.rocha@docente.unip.br

TÓPICOS DA AULA

- *Aprender a declarar e usar variáveis e constantes*
- *Data types: Integer, Double, Float, Char e Boolean*
- *Operações aritméticas*

Leitura:

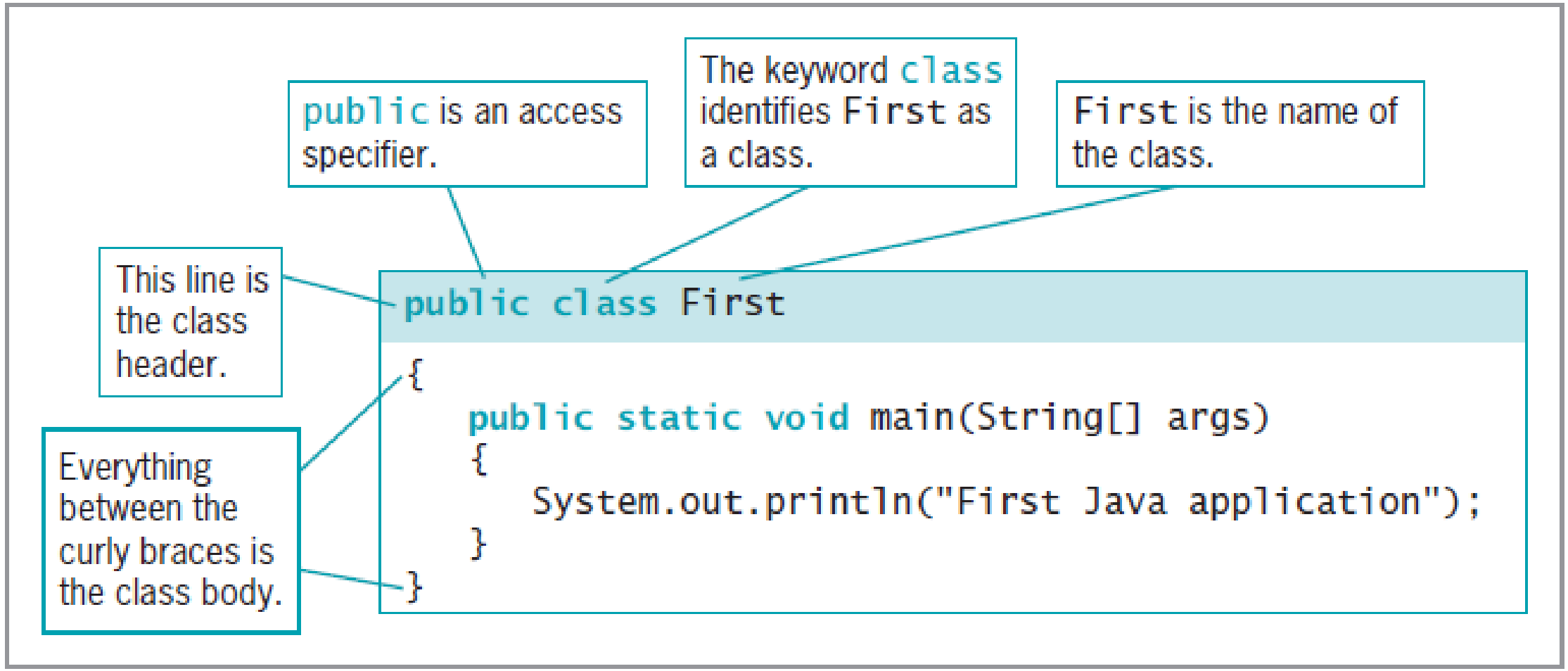
- *Java Programming for Beginners – Mark Lassoff (paginas 62 a 97)*
- *Java Programming – Joyce Farrell (Chapter 2 – Using Data)*

JAVA APPLICATION - CONSOLE OUTPUT

- Vamos agora analisar com detalhe, uma classe Java que produz saída do console

```
public class First
{
    public static void main(String[] args)
    {
        System.out.println("First Java application");
    }
}
```

ENTENDENDO O CÓDIGO-FONTE – VISÃO GERAL



BOAS PRÁTICAS NA PROGRAMAÇÃO - CONVENÇÕES JAVA

- É um padrão Java, embora não seja um requisito, começar identificadores de classe com **letras maiúsculas** e use outras letras maiúsculas conforme necessário para **melhorar a legibilidade.**

Você deve seguir as convenções estabelecidas para Java para seus programas sejam mais fáceis de serem interpretados e entendidos por outros programadores.

COMENTÁRIOS EM JAVA – IMPORTANTE

- Existem 3 tipos de comentários em Java:
  Linha, blocos e Javadoc

- **Line comments** start with two forward slashes (//) and continue to the end of the current line. A line comment can appear on a line by itself or at the end (and to the right) of a line following executable code. Line comments do not require an ending symbol.

- **Block comments** start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). A block comment can appear on a line by itself, on a line before executable code, or on a line after executable code. Block comments also can extend across as many lines as needed.

- **Javadoc** comments are a special case of block comments called **documentation comments** because they are used to automatically generate nicely formatted program documentation with a program named javadoc. Javadoc comments begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/). Appendix E teaches you how to create javadoc comments.

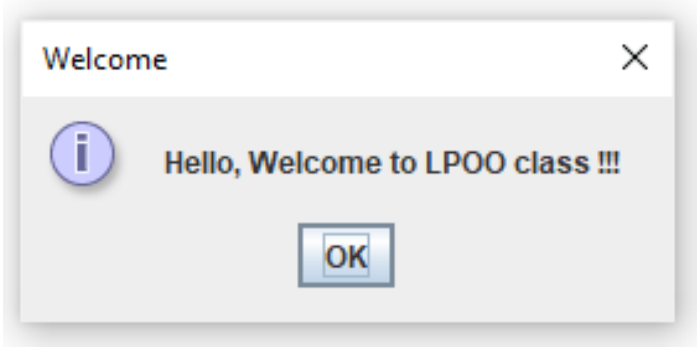
JAVA APPLICATION - GUI OUTPUT

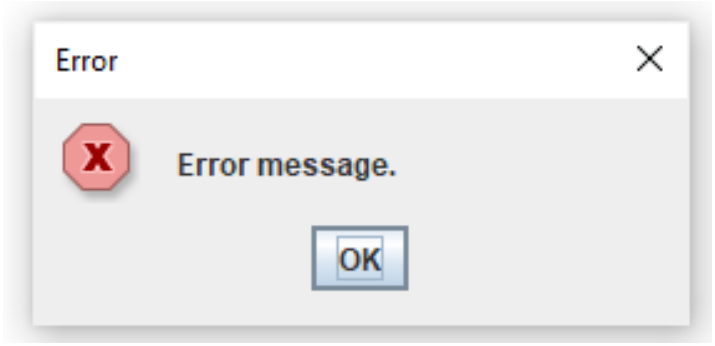
- Além de permitir que você use a classe System para produzir saída de janela de comando, Java fornece classes integradas que produzem saída GUI.

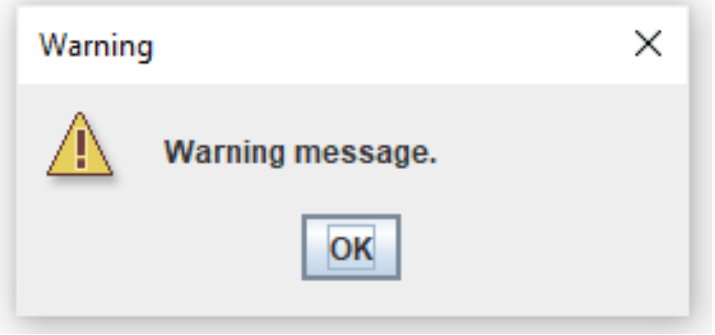
- Java contém uma classe chamada **JOptionPane** que permite a você produzir **caixas de diálogo.**

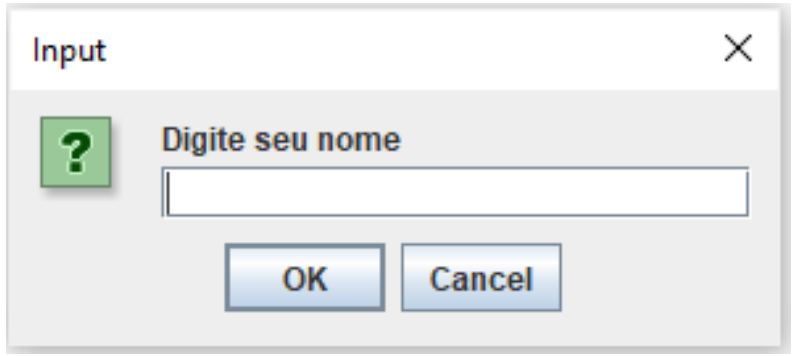
- Uma caixa de diálogo é um objeto GUI semelhante a uma janela na qual você pode colocar as mensagens que deseja exibir.

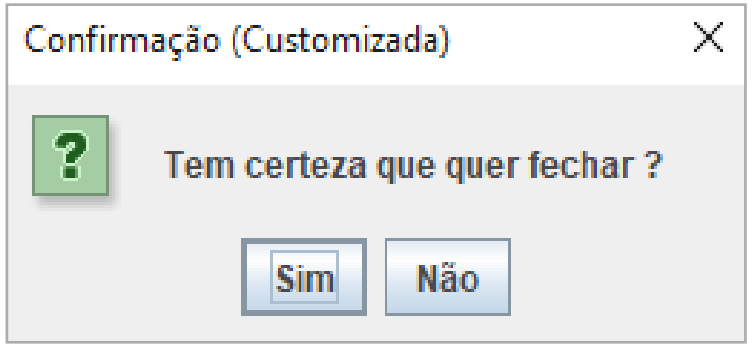
JOptionPane – showMessageDialog / showInputDialog / showOptionDialog











https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

SCANNER CLASS - KEYBOARD INPUT

- Embora possamos atribuir valores às variáveis declaradas, os programas normalmente se tornam mais útil **quando um usuário pode fornecer valores diferentes** para variáveis cada vez que um programa é executado.

- Nas aulas anterior, aprendemos como exibir a saída usando a propriedade **System.out.**

- Aprendemos também que pode usar os métodos **print( )** e **println( )** para exibir muitos dados de tipos; por exemplo, você pode usá-los para exibir um double, int ou String ....

SCANNER CLASS - KEYBOARD INPUT

- **System.in** não é tão flexível; ele é projetado para ler apenas bytes. Isso é um problema, porque você frequentemente deseja aceitar dados de outros tipos.

- Felizmente, os designers de Java criaram uma classe chamada **Scanner** que torna System.in mais flexível.

- Para criar um objeto **Scanner** e conectá-lo ao objeto **System.in,** você escreve uma declaração semelhante ao seguinte

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

SCANNER CLASS - KEYBOARD INPUT

- A classe Scanner **contém métodos que recuperam valores** de um dispositivo de entrada.
- Cada o valor recuperado é um **token**, que é um conjunto de caracteres separado do próximo conjunto por espaço em branco.
- Na maioria das vezes, isso significa que os dados são aceitos quando um usuário pressiona a tecla Enter
- A Tabela 2-7 resume alguns dos métodos mais úteis que lêem diferentes tipos de dados da entrada. Cada um recupera um valor do teclado e o **retorna se o próximo token for o tipo de dados correto.**

SCANNER CLASS - KEYBOARD INPUT

Method	Description
`nextDouble()`	Retrieves input as a `double`
`nextInt()`	Retrieves input as an `int`
`nextLine()`	Retrieves the next line of data and returns it as a `String`
`next()`	Retrieves the next complete token as a `String`
`nextShort()`	Retrieves input as a `short`
`nextByte()`	Retrieves input as a `byte`
`nextFloat()`	Retrieves input as a `float`. Note that when you enter an input value that will be stored as a `float`, you do not type an *F*. The *F* is used only with constants coded within a program.
`nextLong()`	Retrieves input as a `long`. Note that when you enter an input value that will be stored as a `long`, you do not type an *L*. The *L* is used only with constants coded within a program.

Table 2-7 Selected `Scanner` class methods

PRATICANDO ....

```
import java.util.Scanner;
public class GetUserInfo
{
   public static void main(String[] args)
   {
      String name;
      int age;
      Scanner inputDevice = new Scanner(System.in);
      System.out.print("Please enter your name >> ");
      name = inputDevice.nextLine();
      System.out.print("Please enter your age >> ");
      age = inputDevice.nextInt();
      System.out.println("Your name is " + name +
         " and you are " + age + " years old.");
   }
}
```

Repeating as output what a user has entered as input is called **echoing the input**. Echoing input is a good programming practice; it helps eliminate misunderstandings when the user can visually confirm what was entered.

Figure 2-17 The GetUserInfo class

JOPTIONPANE CLASS - GUI INPUT

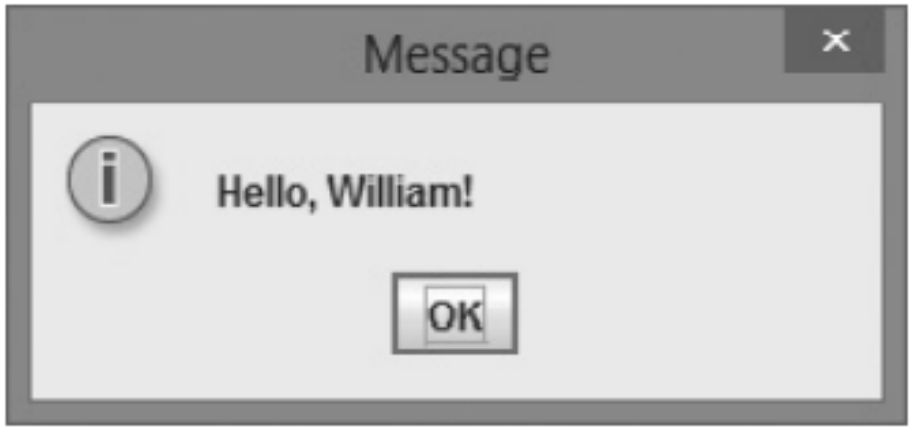
- Nas aulas anteriores, aprendemos como exibir a saída na linha de comando e como criar **caixas de mensagem** para exibir objetos String.

- Podemos também pode **aceitar a entrada em uma caixa de diálogo** GUI usando a classe JOptionPane.

- Podem ser usadas para aceitar a entrada do usuário :
  - **InputDialog** - solicita ao usuário a entrada de texto (importante)
  - **ConfirmDialog** - faz uma pergunta ao usuário, fornecendo botões nos quais o usuário pode clicar Sim, Não e Cancelar como respostas

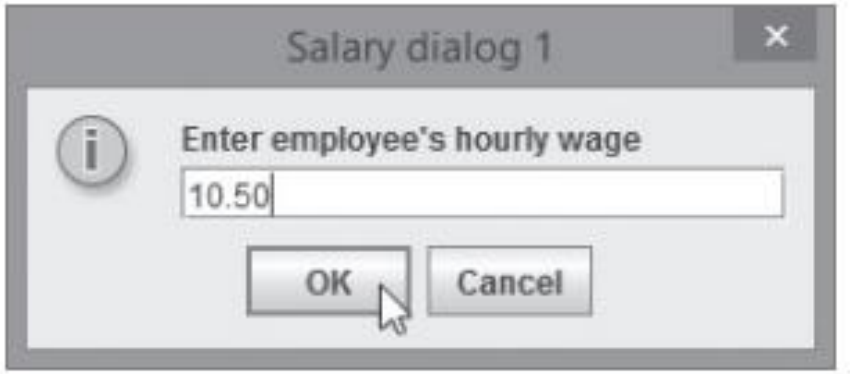
JOPTIONPANE CLASS – EXEMPLO I

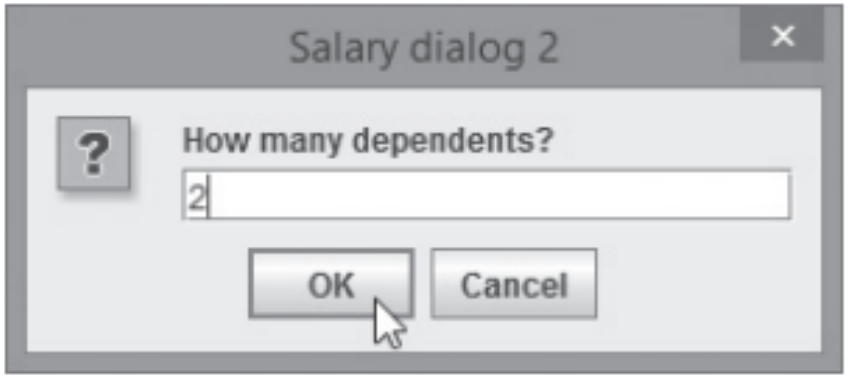
```
import javax.swing.JOptionPane;
public class HelloNameDialog
{
   public static void main(String[] args)
   {
      String result;
      result = JOptionPane.showInputDialog(null, "What is your name?");
      JOptionPane.showMessageDialog(null, "Hello, " + result + "!");
   }
}
```

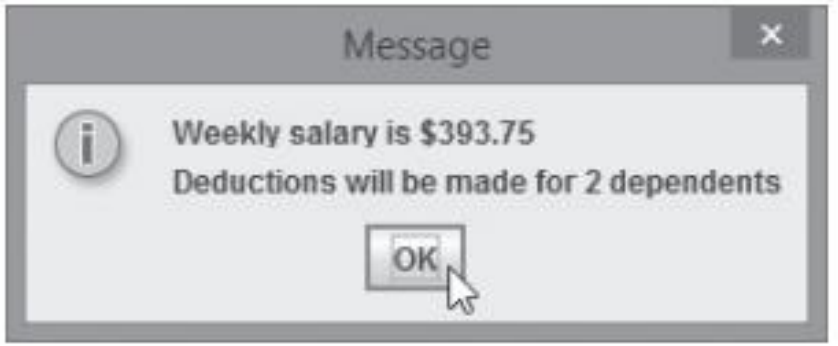


JOPTIONPANE CLASS – EXEMPLO 2

```
import javax.swing.JOptionPane;
public class SalaryDialog
{
   public static void main(String[] args)
   {
        String wageString, dependentsString;
        double wage, weeklyPay;
        int dependents;
        final double HOURS_IN_WEEK = 37.5;
        wageString = JOptionPane.showInputDialog(null,
           "Enter employee's hourly wage", "Salary dialog 1",
           JOptionPane.INFORMATION_MESSAGE);
        weeklyPay = Double.parseDouble(wageString) *
           HOURS_IN_WEEK;
        dependentsString = JOptionPane.showInputDialog(null,
           "How many dependents?", "Salary dialog 2",
           JOptionPane.QUESTION_MESSAGE);
        dependents = Integer.parseInt(dependentsString);
        JOptionPane.showMessageDialog(null, "Weekly salary is $" +
           weeklyPay + "\nDeductions will be made for " +
           dependents + " dependents");
   }
}
```







USING CONFIRM DIALOG BOXES

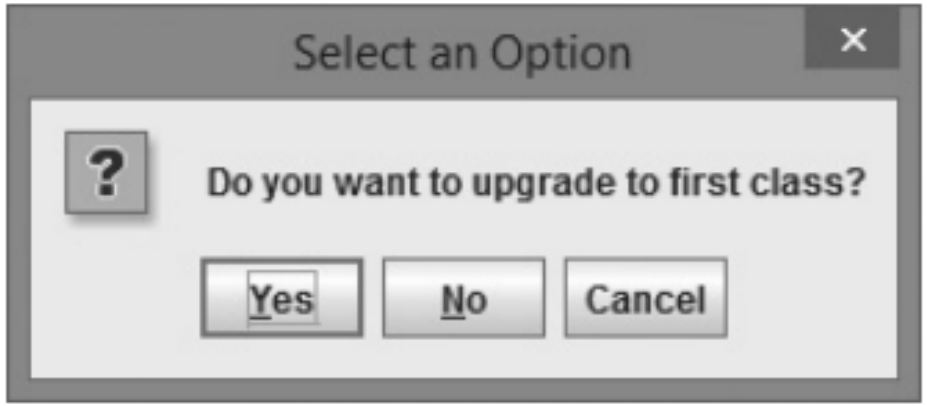
Às vezes, a entrada que você deseja de um usuário não precisa ser digitada no teclado.

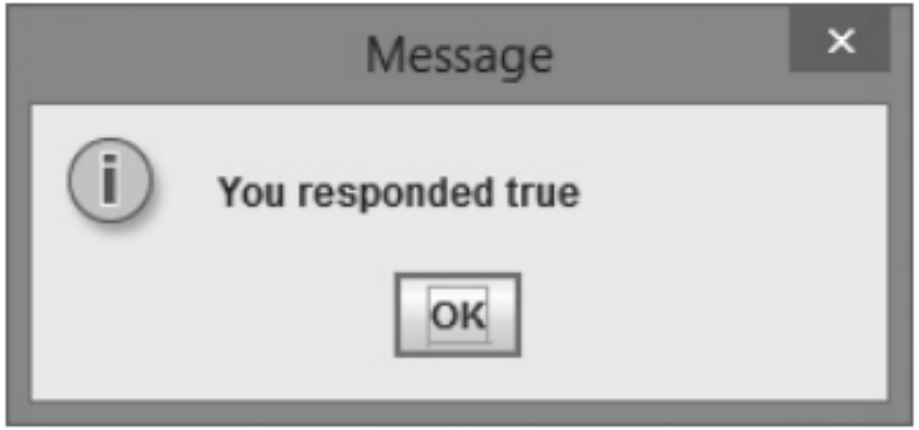
Quando você apresenta opções simples a um usuário, pode oferecer botões nos quais o usuário pode clicar para confirmar uma escolha.

Uma caixa de diálogo de confirmação que exibe as opções Sim, Não e Cancelar pode ser criada usando o método **showConfirmDialog ( )** na classe **JOptionPane**

USING CONFIRM DIALOG BOXES

```
import javax.swing.JOptionPane;
public class AirlineDialog
{
   public static void main(String[] args)
   {
      int selection;
      boolean isYes;
      selection = JOptionPane.showConfirmDialog(null,
         "Do you want to upgrade to first class?");
      isYes = (selection == JOptionPane.YES_OPTION);
      JOptionPane.showMessageDialog(null,
         "You responded " + isYes);
   }
}
```





VÁRIAVEIS E CONSTANTES

- Um item de dados é **CONSTANTE** quando seu valor **não pode ser alterado** durante a execução de um programa.

- Em vez de usar dados constantes, você pode configurar um item de dados para ser **VARIÁVEL.** Uma variável é um nome localização da memória que pode armazenar um valor que sofrerá alteração durante a execução do programa.

- Um **TIPO PRIMITIVO** é um tipo de dados simples. Existem oito tipos de dados primitivos em Java
  - *byte, short, int, long, float, double, char and boolean.*

Obs: String não é um tipo de dado primitivo

DECLARANDO VARIAVEIS

Uma declaração de variável é uma declaração que reserva um local de memória e contem a seguintes sintaxe:

- **Modificador de acesso:** public ou private;
- **Tipo de dados:** identifica o tipo de dados que a variável irá armazenar;
- **Nome da variável:** começando com letra minúscula;
- Um **operador de atribuição** (opcional) e **valor atribuído**, se você quiser que uma variável contenha um valor inicial
- **Um ponto e vírgula**

DECLARANDO CONSTANTES

O valor da constante não deve mudar durante a execução de um programa, você pode criá-lo para ser um constante nomeada da seguinte forma:

Exemplos:

- *private **final** int NUMBER_OF_DEPTS = 20;*
- *private **final** double PI = 3.14159;*
- *private **final** double TAX_RATE = 0.015;*
- *private **final** string COMPANY = "ABC Manufacturing";*

Indica que esse atributo não sofrerá alterações

INTEGER DATA TYPES

- Em Java, você pode usar variáveis de tipos de dados byte, short, int e long para armazenar inteiros. **Inteiro é um número inteiro sem casas decimais.**

- O tipo de dados **int** é o tipo inteiro mais comumente usado

- Os tipos de dados **byte, short e long** são variações do tipo inteiro.

- O byte e short **ocupam menos memória** e podem conter apenas valores menores; o tipo long **ocupa mais memória** e pode conter valores maiores

INTEGER DATA TYPES

Type	Minimum Value	Maximum Value	Size in Bytes
byte	–128	127	1
short	–32,768	32,767	2
int	–2,147,483,648	2,147,483,647	4
long	–9,223,372,036,854,775,808	9,223,372,036,854,775,807	8

Table 2-2 Limits on integer values by type

É importante escolher os **tipos apropriados** para as variáveis que você usará em seus programas. **Se você escolher um tipo de dados que seja maior do que você precisa, você desperdiça memória**

PRATICANDO ...

1. Crei uma classe chamada IntegerDemo;

2. Crie também o método main ();

3. No corpo do método main, crie variáveis **inteiras**, cada um dela com um tipo diferente (int, byte, short e long). Inicia elas com o valor 100.

4. Exibe os valores das variáveis no console, usando o método System.out.println (...)

You Do It (livro Farrell paginas 66 e 67)

PRATICANDO ...

5. Agora modifique os valores iniciais das variáveis para **12345.**

6. Salve o arquivo e em seguida compile ... Run as

O que aconteceu e porque ??? **Justifique a resposta**

Exemplo: You Do It (livro Farrell paginas 66 e 67)

BOOLEAN DATA TYPE

A lógica booleana é baseada em comparações verdadeiras ou falsas.

Considerando que uma variável int pode conter milhões de valores diferentes (em momentos diferentes), **uma variável booleana pode conter apenas um de dois valores – verdadeiro ou falso.**

Exemplo:

```
private boolean isValid = true;
private boolean validaCamposObrigatorios = false;
```

Java suporta 6 operadores relacionais que são usados para fazer **comparações**. Um operador relacional **compara dois itens**

BOOLEAN DATA TYPE

Operator	Description	True Example	False Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <= 5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

Table 2-3 Relational operators

FLOATING-POINT DATA TYPES

Um número de ponto flutuante contém posições decimais. Java oferece suporte a dois tipos de dados de ponto flutuante: **float e double.**

Um tipo de dados **float** pode conter valores de até **seis ou sete dígitos de precisão.**

Um tipo de dados **double** requer mais memória do que um flutuante e pode conter 14 ou 15 dígitos significativos de precisão.

Type	Minimum	Maximum	Size in Bytes
`float`	$-3.4 * 10^{38}$	$3.4 * 10^{38}$	4
`double`	$-1.7 * 10^{308}$	$1.7 * 10^{308}$	8

Table 2-4 Limits on floating-point values

CHAR DATA TYPE

O tipo de dados char é usado para armazenar **apenas um caracter**

Por exemplo, o que se segue são declarações típicas de caracteres:

```
private char middleInitial = 'M';
private char gradeInChemistry = 'A';
private char aStar = '*';
```

CLASSES WRAPPERS EM JAVA

- O Java conta com diversos Wrappers que adicionam funcionalidades a outras classes ou **tipos primitivos, um exemplo dele é o Integer, que é o Wrapper do tipo primitivo int.**

- Uma outra situação em que as classes Wrapper são úteis é quando precisamos **converter um objeto em um tipo de dado primitivo.**

- Por exemplo, digamos que em nossa aplicação, uma interface gráfica do usuário solicite que seja digitada a idade de uma pessoa em uma caixa de texto JTextField. Contudo, sabemos que caixas de textos retornam seu conteúdo em uma String, o que requer uma conversão para int de maneira que possamos realizar cálculos com a idade na aplicação.

CLASSES WRAPPERS EM JAVA

Então se temos Wrappers porque usar os tipos primitivos?

Simples, eles são mais rápidos e consomem menos memória, afinal não tem implementação de métodos ou qualquer outro algoritmo complexo que venha a consumir mais tempo da JVM.

Primitive Types	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

TIPOS PRIMITIVOS X WRAPPERS, QUANDO USAR?

- Até entre os **programadores mais experientes** surgem dúvidas na hora de declarar uma variável.

- Como Java é uma **linguagem fortemente tipada**, é necessário que todas suas variáveis obrigatoriamente sejam declaradas de acordo com um tipo (Como por ex: String, int, Integer, double, Double etc…)

- Em Java existe os **tipos primitivos e os Wrappers** e com isso, sempre surge a dúvida de quando usar ou não usar um tipo primitivo e qual usar.

Por isso, vou tentar esclarecer essa dúvida para sabermos o que estamos fazendo quando surgir a questão: **Qual tipo devo declarar essa variável?**

OPERAÇÕES ARITMÉTICAS

A tabela abaixo descreve os cinco operadores aritméticos padrão que você usa para realizar cálculos com valores em seus programas.

Operator	Description	Example
+	Addition	45 + 2, the result is 47
–	Subtraction	45 – 2, the result is 43
*	Multiplication	45 * 2, the result is 90
/	Division	45.0 / 2, the result is 22.5 45 / 2, the result is 22 (not 22.5)
%	Remainder (modulus)	45 % 2, the result is 1 (that is, 45 / 2 = 22 with a remainder of 1)

Table 2-8 Arithmetic operators

ASSOCIATIVIDADE E PRECEDÊNCIA

Quando você combina operações matemáticas em uma única instrução, você deve entender tanto **associatividade quanto precedência.**

A associatividade de operadores aritméticos com o mesmo a precedência é da esquerda para a direita.

Em uma afirmação como answer = x + y + z;

o x e y são adicionados primeiro, produzindo um resultado temporário e, em seguida, z é adicionado à soma temporária.

Depois da soma é calculado, o resultado é atribuído à answer.

ASSOCIATIVIDADE E PRECEDÊNCIA (cont)

A precedência do operador se refere às regras para a ordem em que as partes de um expressão são avaliados.

Os operadores de ***multiplicação (*), divisão (/) e resto (%)*** têm o **mesma precedência**, e é **maior** do que a precedência da adição (+) e subtração (-).

Em outras palavras, uma expressão aritmética é avaliada da esquerda para a direita e qualquer multiplicação, divisão e operações de resto ocorrem. E depois as operações de adição e subtração são executadas..

ENTENDENDO A CONVERSÃO DE TIPOS DE DADOS

- Quando você faz aritmética com variáveis ou constantes do **mesmo tipo**, o resultado do operação **mantem o mesmo tipo.**
  - Por exemplo, quando você divide dois ints, o resultado é um int, e quando você subtrai dois double, o resultado é um double.

- Muitas vezes, no entanto, você pode querer para realizar operações matemáticas em operandos com **tipos diferentes.** O processo de converter um tipo de dados em outro é **conversão de tipo**.

- Java realiza algumas conversões para você automaticamente ou implicitamente, **mas outras conversões devem ser solicitadas explicitamente pelo programador.**

CONVERSÃO AUTOMÁTICA DE DADOS

- Quando você executa operações aritméticas com operandos de **tipos diferentes**, Java escolhe um **tipo unificador** para o resultado.

- Java realiza uma conversão **implícita**; ou seja, ele converte automaticamente operandos não-conformes para o unificador modelo.

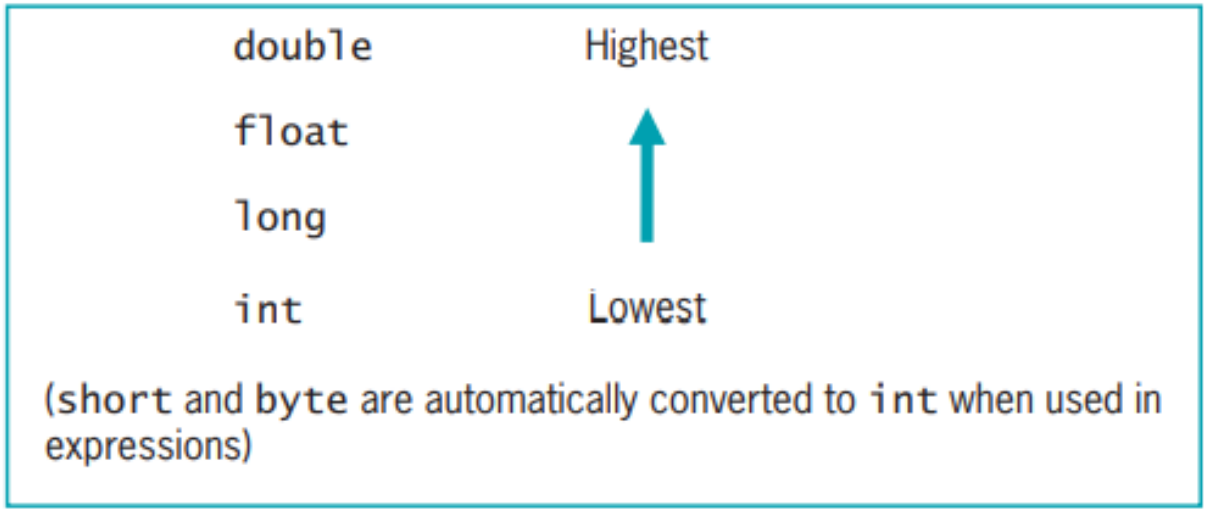


Figure 2-41 Order for establishing unifying data types

CONVERSÃO AUTOMÁTICA DE DADOS

Vamos analisar as seguintes linhas de códigos:

```
private int hoursWorked = 37;
private double payRate = 16.73;
private double totalPay = hoursWorked * payRate;
```

Quando dois tipos diferentes são usados em uma expressão, o **tipo unificador é aquele que é mais alto** (Figura 2-41).

Por exemplo, a adição de um **double** e um **int** resulta em um **double**, e a subtração de um **long** de um **float** resulta em uma **float**

CONVERSÃO EXPLICITA DE DADOS

- Você pode **substituir propositalmente** o tipo de unificação imposto por Java executando uma conversão de tipo.

- A conversão de tipo força um valor de um tipo de dados a ser usado como um valor de outro tipo.

- Executar um tipo de conversão, você usa um **operador de conversão, que é criado colocando o tipo de resultado desejado em parênteses**.

CONVERSÃO EXPLICITA DE DADOS

Por exemplo, uma conversão de tipo é realizada na seguinte código:

```
double bankBalance = 189.66;
float weeklyBudget = (float) (bankBalance / 4);
```

Neste exemplo, o valor **double** bankBalance é dividido pelo **int** 4 e o resultado é um em **double**.

Em seguida, o resultado double é convertido em um float antes de ser armazenado no weeklyBudget.

EXERCÍCIOS DE FIXAÇÃO

1) Escreva uma classe Java que para resolver e imprime os valores das seguintes operações aritméticas:

```
4 + 6 * 2
10 / 5 + 8
12 / 4 + 16 / 2
17 / 2
22 / 5
```

```
39 / 10
19 % (2 + 3)
3 + 4 * 20 / 3
36 % (6 + 2)
8 % 2 * 0
```

```
15 > 13
8 <= (2 + 6)
5 == 15
3 >= 3
3 * 3 == 2 * 4
```

```
5 < 8 – 3
7 != 7
8 != (2 + 5)
10 – 20 == -10
3 + 2 * 6 == 15
```

EXERCÍCIOS DE FIXAÇÃO

2) Escolha o melhor tipo de dados para cada um das seguintes informações. **Explique por que você escolheu o tipo**

a) Número de irmãos que você tem
b) Média final de uma determinado turma do curso de Computação
c) População de Limeira
d) Numero de passageiros de um ônibus
e) Numero de jogadores de um time de futebol
f) Ano de um acontecimento histórico
g) Preço de um automóvel

EXERCÍCIOS DE FIXAÇÃO

3) Crei uma classe chamada **KmToMiles** que conversa uma distancia em KM para um distância em Milha (padrão US), seguindo as seguintes informações:

- *1 milha = 1.60934 km*
- *Para testar, crie algumas variáveis que represente valores em km (simulando um valor informado pelo usuário)*
- *Exibe no console os resultados dos testes, usando o seguinte comando:*
  - *System.out.println("O valor convertido de Km para Milha é :" + valorMilha)*

4) Crei uma classe em Java chamada **MinutesConversion** que converte um numero de minutos (inteiro) e converte para horas e dias.

Por exemplo: 6000 minutos equivale a 100 horas e 4.17 dias

PRÓXIMA AULA

- Classes, métodos e objetos em Java

LEITURA:

- *Java Programming – Joyce Farrell (Chapter 3 – Using Methods, Classes, and Objects)*