

Seja bem Vindo!

# **Curso Java na Prática**

**[CursosOnlineSP.com.br](http://CursosOnlineSP.com.br)**

Carga horária: 60hs



## **Conteúdo Programático**

### **Módulo I**

Divisões do Java, Preparando o ambiente de desenvolvimento  
Regras e Convenções

### **Módulo II**

Criando o primeiro projeto Java  
Método Main , Conhecendo o Eclipse, criando a primeira Classe  
Digitando, salvando e executando o primeiro código

### **Módulo III**

Tipos de dados  
Utilizando variáveis, comentários  
Operador de concatenação, continuando a digitar códigos  
Operadores de incremento e decremento, Escopo das variáveis  
Variáveis de instância e variáveis locais, mais código  
Operadores Controle de Fluxo (IF else, While, For), mais códigos

### **Módulo IV**

Métodos, Objetos, Características da P.O.O.  
Exemplo de código orientado a objeto, misturando conceitos

### **Módulo V**

Pacotes, praticando com exemplos mais rebuscados utilizando  
todos os conceitos vistos até aqui  
Classe Scanner  
Classe Math

### **Módulo VI**

Mais métodos da classe Math, Casting de tipos primitivos, mais  
códigos para praticar

## Módulo I – Divisões do Java, Preparando o ambiente....

### Divisões do Java

- JSE (Java Standard Edition): é a base do Java. É composta pelo ambiente de execução e principais bibliotecas.
- JEE (Java Enterprise Edition): compreende aplicações corporativas e Internet.
- JME (Java Micro Edition): para aplicações em dispositivos móveis.

### JSE

#### Kit de desenvolvimento

- JRE (Java Runtime Environment): compreende a JVM + bibliotecas. É justamente a JRE que você deve instalar no seu computador para executar aplicações Java. A JRE pode ser obtida gratuitamente no site da Oracle (a Sun foi comprada pela Oracle).
- JDK (Java Development Kit): compreende JRE + kit completo de desenvolvimento. O JDK serve para criar e executar aplicações Java no seu computador. É o que os desenvolvedores usam e também se encontra gratuitamente no site da Oracle.

#### :: Preparando o ambiente de desenvolvimento



A preparação do ambiente de desenvolvimento é muito importante.

*No decorrer do curso faremos vários downloads e criaremos muitos códigos. Crie uma pasta específica (por exemplo "Java na Prática - Básico") para guardar estes arquivos/programas para mantermos uma organização.*

Primeiramente precisamos baixar o **JDK**. Como a Oracle comprou a Sun, devemos baixar o JDK direto do site da Oracle em:

<http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>

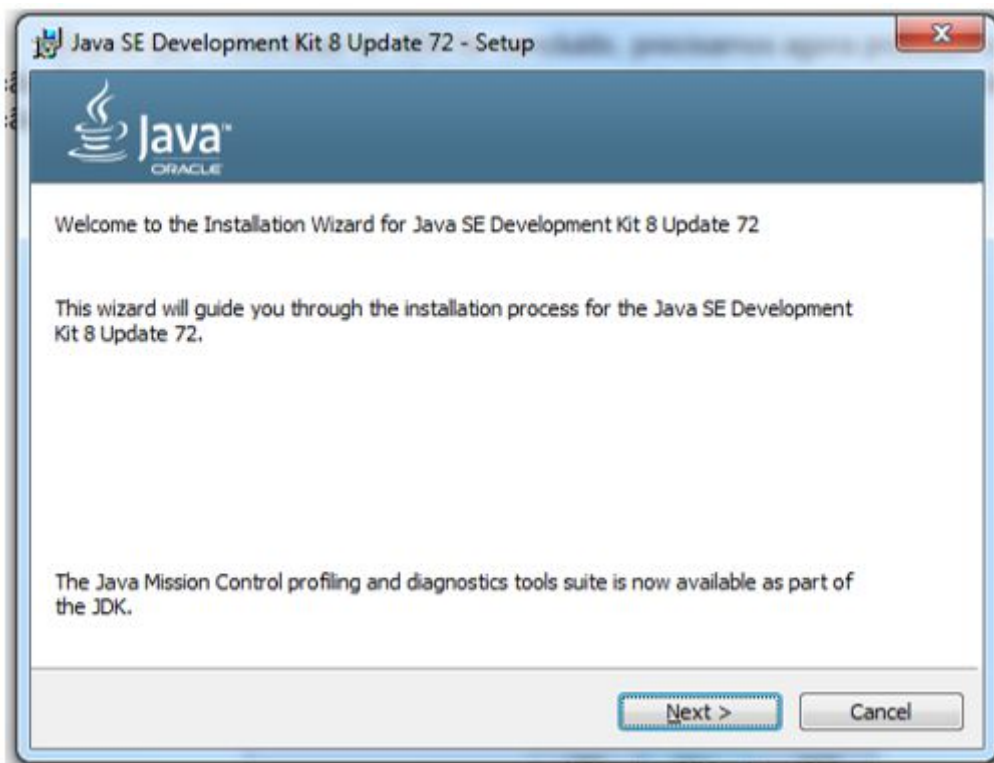
Java SE 8 é a versão mais recente e iremos utilizá-la.

Java SE Development Kit 8u60		
You must accept the <a href="#">Oracle Binary Code License Agreement for Java SE</a> to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	77.69 MB	<a href="#">jdk-8u60-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM v8 Hard Float ABI	74.64 MB	<a href="#">jdk-8u60-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	154.66 MB	<a href="#">jdk-8u60-linux-i586.rpm</a>
Linux x86	174.83 MB	<a href="#">jdk-8u60-linux-i586.tar.gz</a>
Linux x64	152.67 MB	<a href="#">jdk-8u60-linux-x64.rpm</a>
Linux x64	172.84 MB	<a href="#">jdk-8u60-linux-x64.tar.gz</a>
Mac OS X x64	227.07 MB	<a href="#">jdk-8u60-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	139.67 MB	<a href="#">jdk-8u60-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	99.02 MB	<a href="#">jdk-8u60-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	140.18 MB	<a href="#">jdk-8u60-solaris-x64.tar.Z</a>
Solaris x64	96.71 MB	<a href="#">jdk-8u60-solaris-x64.tar.gz</a>
Windows x86	180.82 MB	<a href="#">jdk-8u60-windows-i586.exe</a>
Windows x64	186.16 MB	<a href="#">jdk-8u60-windows-x64.exe</a>

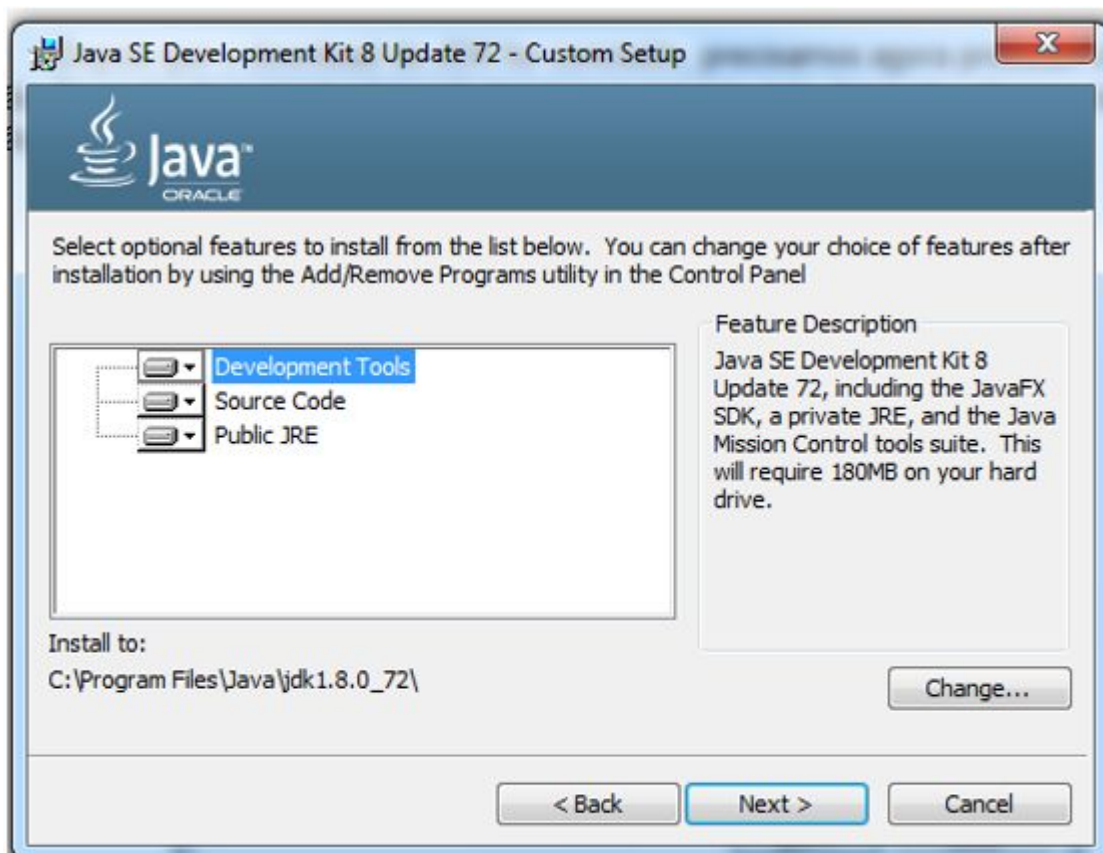
Neste curso trabalharemos com a versão para Windows X86, mas você pode baixar a versão para o seu respectivo sistema operacional que não influenciará nas aulas.

Depois de clicar no JDK para download conforme figura acima, devemos aceitar os termos de licença e escolher a plataforma a ser utilizada: no nosso caso Windows X86.

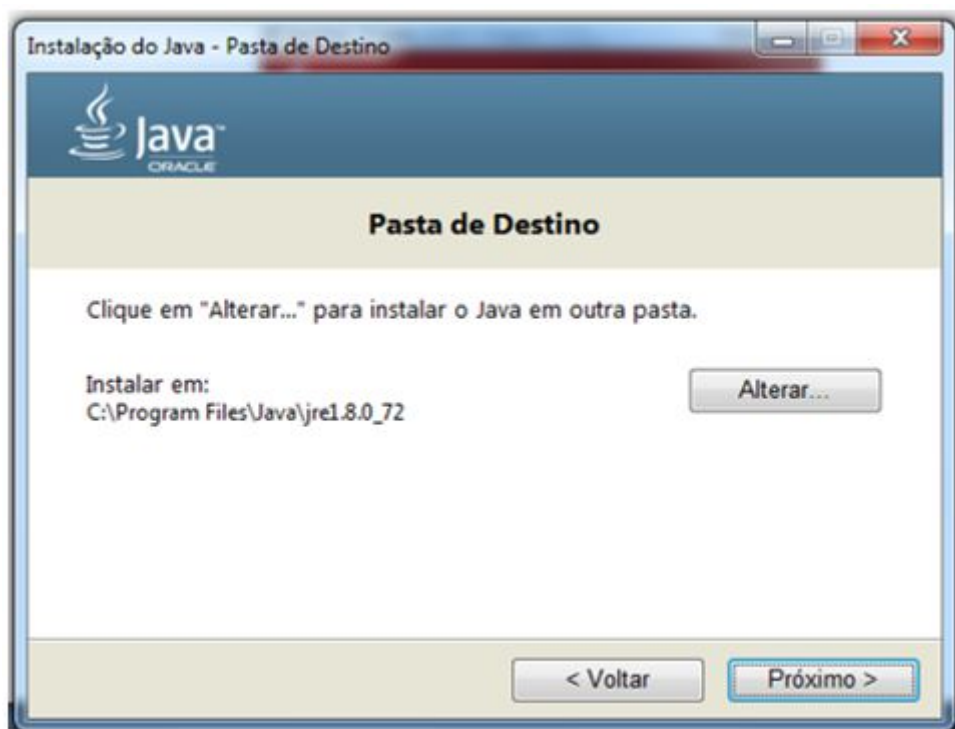
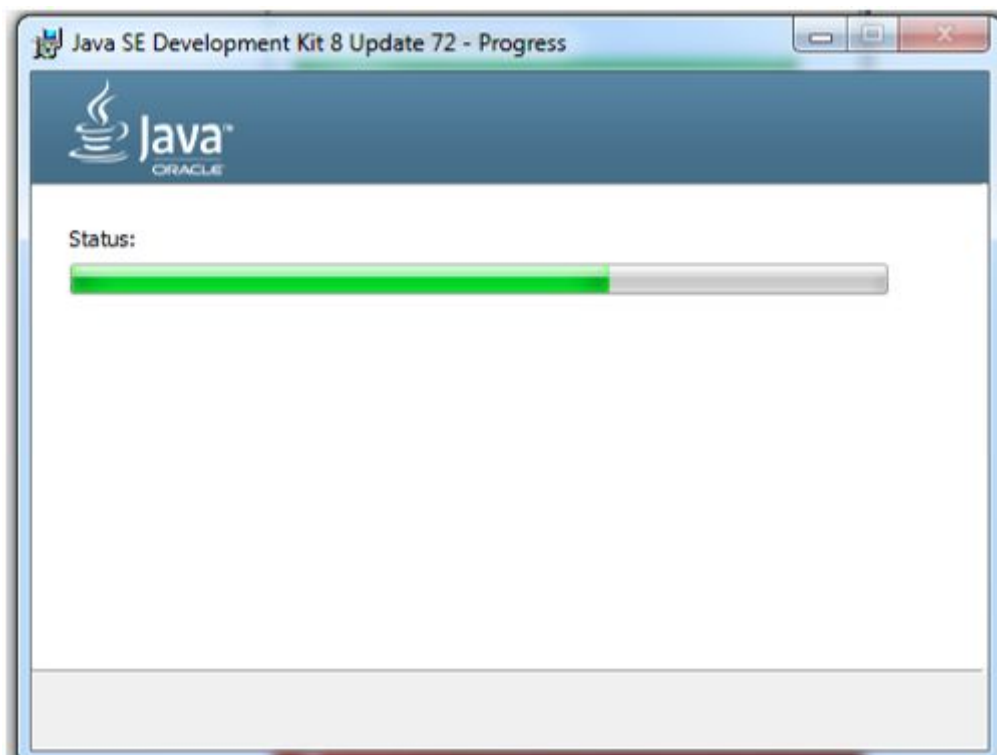
Depois que o download do JDK for concluído, precisamos agora proceder com a instalação. Executando o arquivo baixado teremos a seguinte tela, bastante clássica, de instalação:



Clicando em Next:



Clicando em Next:



Clicando em Next:



E finalmente:



Clicando em Close nossa instalação do JDK está concluída.

Continuando a preparação do ambiente, agora iremos baixar o Eclipse. A IDE Eclipse ([www.eclipse.org](http://www.eclipse.org), é a mais utilizada) é um ambiente gráfico de desenvolvimento gratuito do Java que facilita muito a vida do desenvolvedor e é no Eclipse que desenvolveremos as

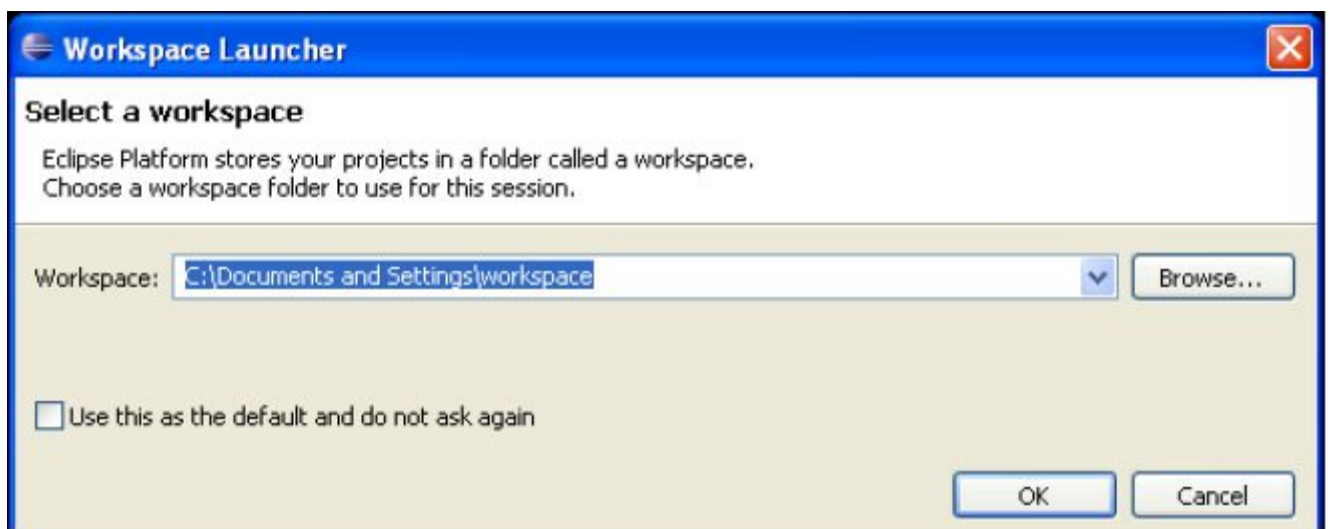
aplicações deste curso. Mas há outras IDEs como NetBeans, JCreator... Lembrando que antes de iniciarmos o Eclipse devemos baixar e instalar o JDK.

Para baixar o Eclipse acesse [www.eclipse.org/downloads](http://www.eclipse.org/downloads) e escolha a opção Eclipse Classic, conforme figura abaixo:



Depois de terminado o download do Eclipse 32 Bit fica muito fácil a sua execução. O Eclipse vem na forma de um arquivo compactado, bastando você descompactá-lo na pasta desejada (pode ser a pasta Eclipse mesmo). O Eclipse não precisa ser instalado, basta descompactar e executar o arquivo `eclipse.exe`.

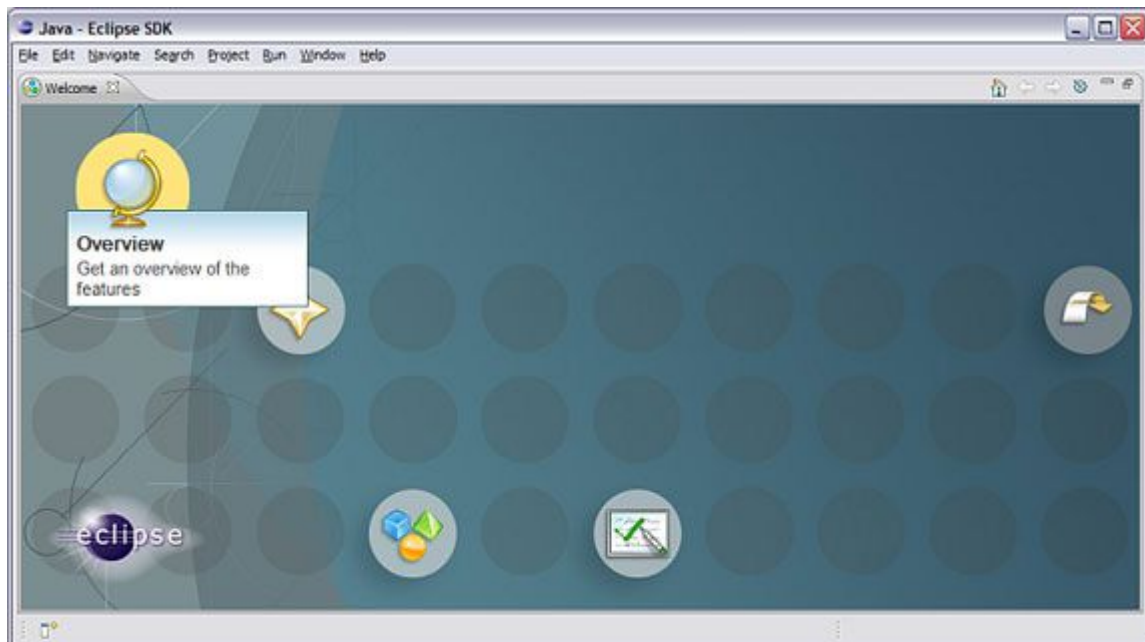
Pronto, já estamos executando o Eclipse.  
Ao executarmos o Eclipse pela primeira vez, teremos a seguinte tela:



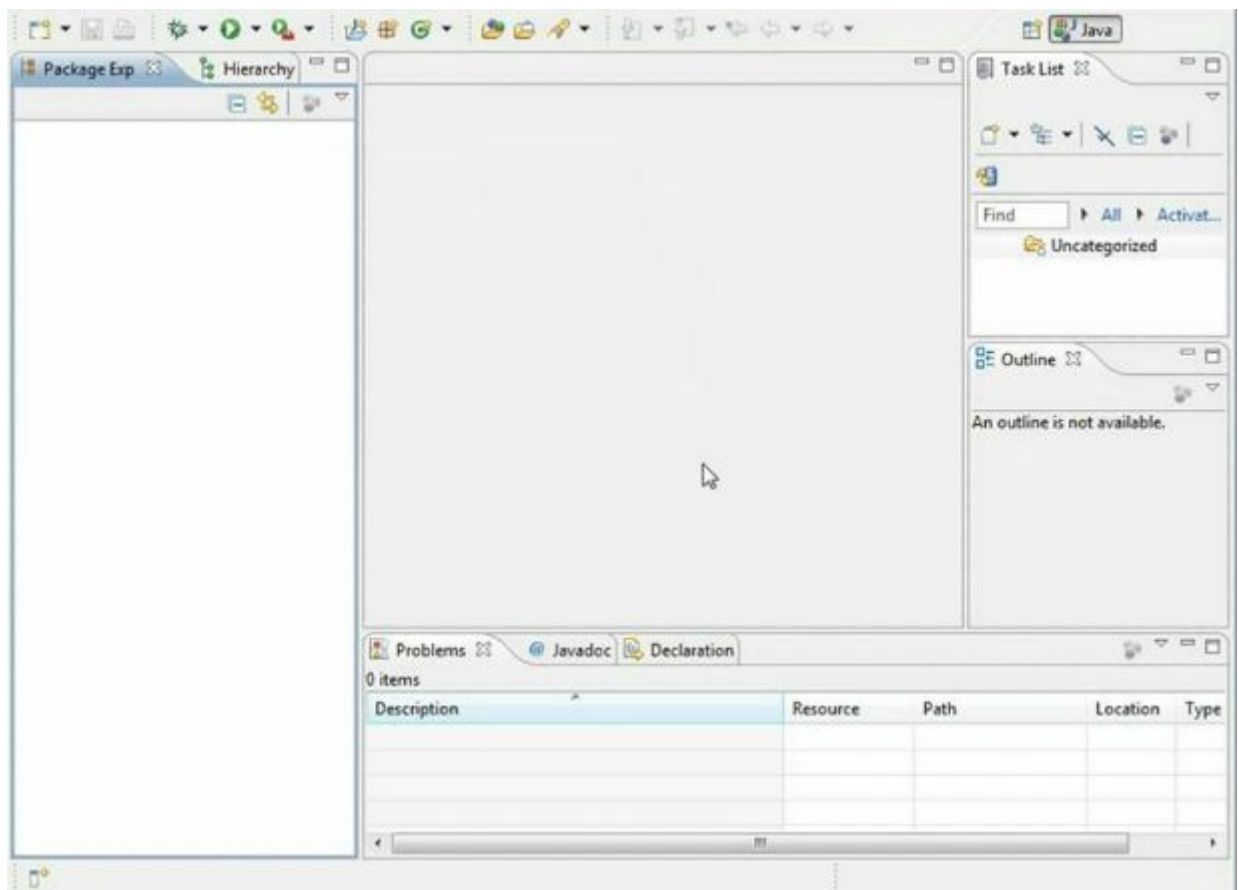
Esta tela está pedindo para escolher onde será definido o seu Workspace. O Workspace nada mais é onde você irá guardar os seus projetos Java. Escolha a pasta de sua preferência (por exemplo crie uma pasta “Projetos”, a utilizaremos para guardar nossos códigos), selecione esta pasta e clique OK.



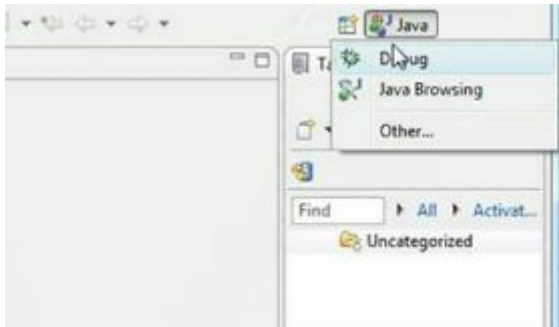
Depois de você ter clicado OK, o Eclipse será inicializado e teremos a seguinte tela de boas vindas que você pode fechá-la normalmente, não servirá para nós:



Fechada a tela de boas vindas temos finalmente o ambiente Eclipse propriamente dito:



Agora vamos dar uma visão geral nesta tela inicial do Eclipse. O Eclipse trabalha com o conceito de perspectivas, onde dependendo da linguagem que você está utilizando pode-se alterar o modo de visualização no canto superior direito da tela. No nosso curso vamos utilizar a perspectiva Java.



### :: Regras e convenções em Java – muito importante

Antes de criarmos o nosso primeiro projeto em Java, temos que nos familiarizar com algumas regras e convenções da linguagem, são elas:

- Todo arquivo Java deve ter uma extensão `.java` (em minúscula);
- Um arquivo a ser executado deve ter obrigatoriamente uma única classe pública (`public`) com um método “main” dentro dela (mais detalhes desta classe e deste método nos próximos capítulos);
- O arquivo deve ter o mesmo nome desta classe pública;
- O Java diferencia letras maiúsculas de minúsculas;
- Os comandos em Java devem ser finalizados com ponto e vírgula “;”.
- O nome de uma classe é um substantivo começando com letra maiúscula, por exemplo: Banco, Cliente, ContaCliente, AgenciaCliente.
- O nome de um método é um verbo representando uma ação começando com letra minúscula, por exemplo: saltar, correr, sacarDinheiro, depositarDinheiro.
- O nome de uma variável que contém duas palavras também

segue o modelo primeira palavra começando em minúscula e a segunda palavra começando em maiúscula.

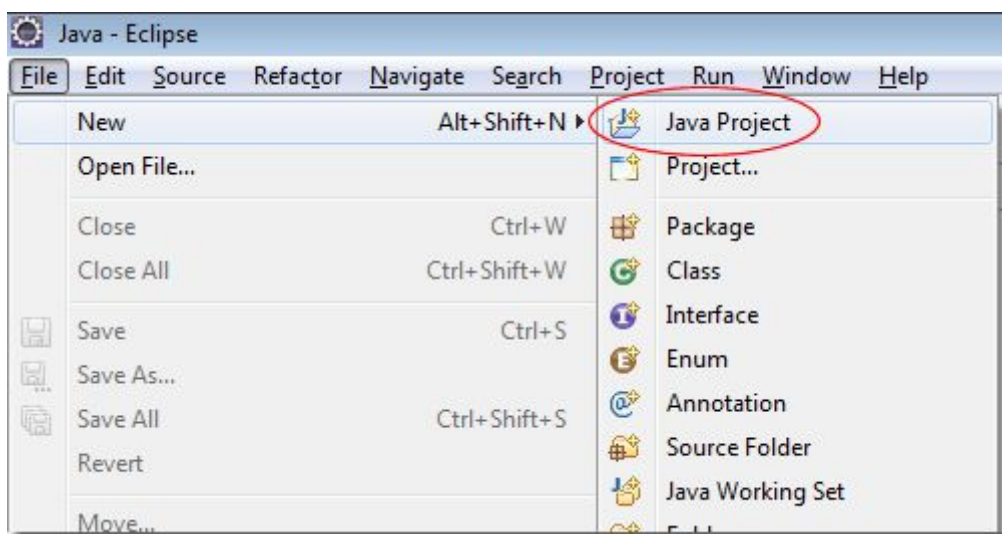


*Mais adiante iremos falar de todos estes conceitos como métodos, classes, pacotes, etc. e também veremos outras regras, convenções e conceitos de orientação a objetos durante o curso.*

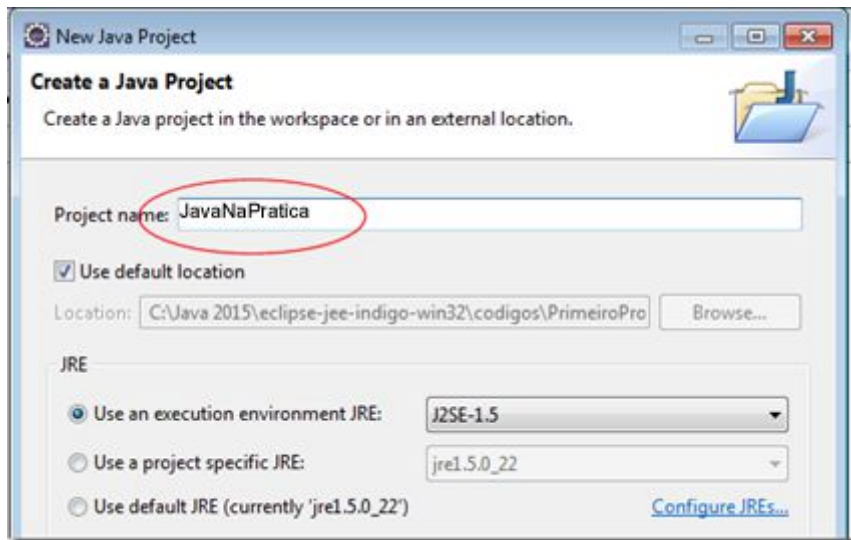
## Módulo II – Criando o primeiro projeto Java....

### Criando o primeiro projeto Java

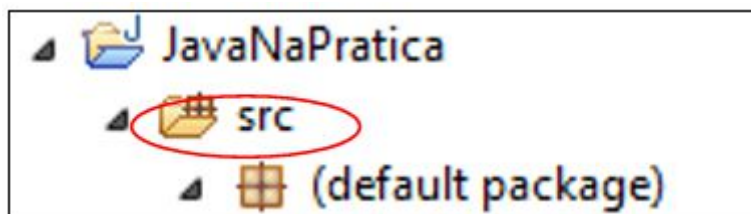
Um Projeto Java (é como se fosse uma pasta) no Eclipse é onde nós iremos guardar/executar nossos códigos. Dentro do Eclipse clique em File, New, Java Project.



Teremos a seguinte tela onde no campo *Project name* daremos o nome de **JavaNaPratica** (note a convenção do nome com duas palavras começando com letra maiúscula e sem espaço), clique em Finish:



Após o Finish, nosso projeto (que agora está no workspace, pasta Projetos) está criado e nossas classes ficarão na pasta `src` (pasta padrão) conforme figura abaixo:



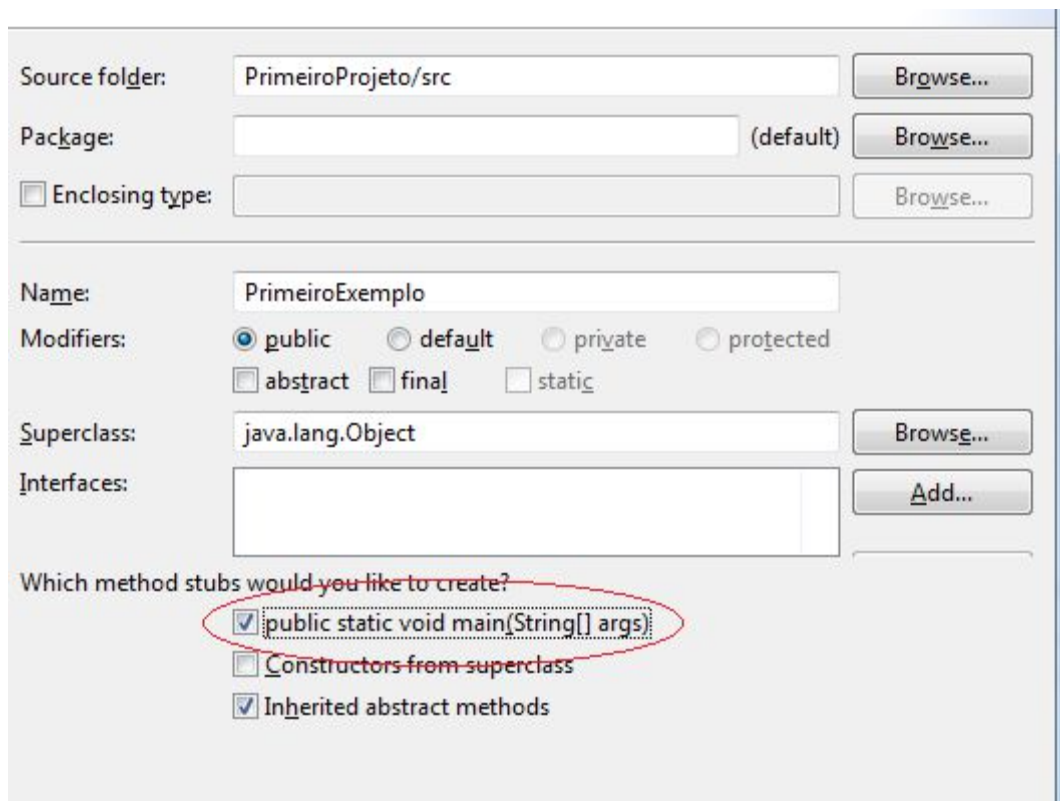
Agora que já temos o nosso Projeto criado, precisamos de uma classe para trabalhar (a classe fica dentro do pacote que fica dentro de um projeto). Lembrando que cada projeto Java obrigatoriamente terá uma classe pública com um método **main**.

**Método main:** para um programa Java executar, é necessário definir um método especial para ser o ponto de entrada do programa, ou seja, para ser o **primeiro** método a ser chamado quando o programa for executado. Ele é o principal método do Java. Quando executarmos nosso programa, é este método que o Java irá procurar primeiro. O Java terá uma, e somente uma, classe pública e nessa classe obrigatoriamente teremos um método main.



No Módulo onde falamos sobre Orientação a Objetos iremos ver mais detalhadamente esses conceitos como classe, método, objeto... e assim entender 100% de cada linha dos nossos códigos. Fique tranquilo (a).

Vamos criar nossa classe: clique com o botão direito na pasta `src` do nosso projeto `PrimeiroProjeto`, New, Class. Teremos a seguinte tela, onde no campo `Name` daremos o nome “`PrimeiroExemplo`” à nossa classe:



Source folder: `PrimeiroProjeto/src` Browse...

Package: `(default)` Browse...

☐ Enclosing type: Browse...

---

Name: `PrimeiroExemplo`

Modifiers: ☒ `public` ☐ `default` ☐ `private` ☐ `protected`  
☐ `abstract` ☐ `final` ☐ `static`

Superclass: `java.lang.Object` Browse...

Interfaces: Add...

Which method stubs would you like to create?

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

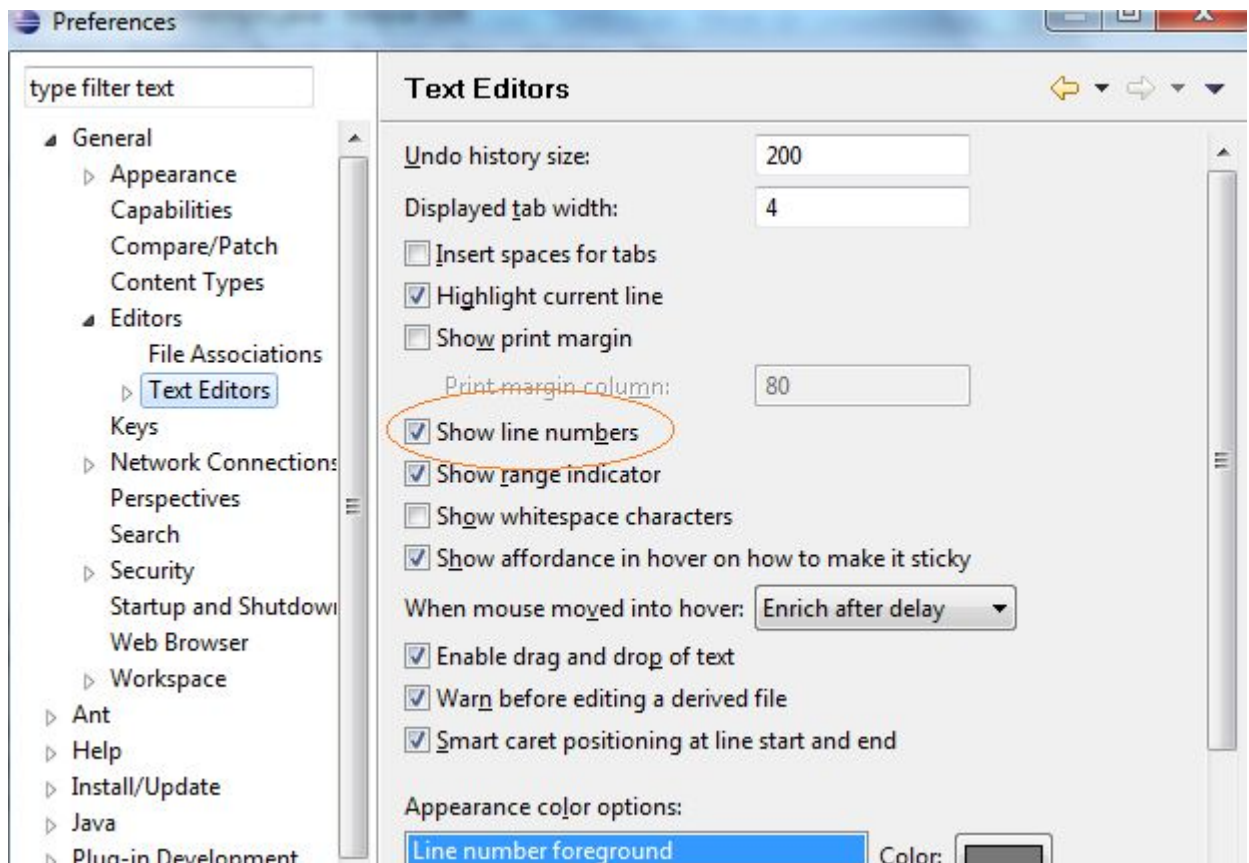
☐ Constructors from superclass

☒ Inherited abstract methods

Deixe marcada a opção `public static void main(String[] args)` e perceba a opção “`public`” marcada. Lembrando que temos que ter obrigatoriamente uma classe pública para executar o método `main`. Pronto, após clicar em `Finish` teremos criado a nossa primeira classe em Java. Agora nós temos uma classe pública chamada `PrimeiroExemplo.java` com o seu método `main` obrigatório.

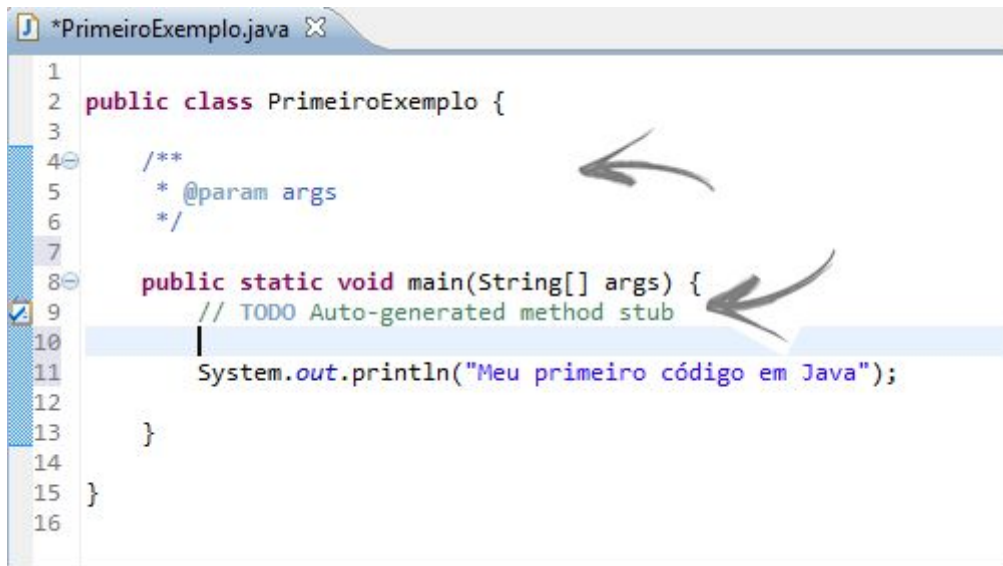
```
PrimeiroExemplo.java X
1
2 public class PrimeiroExemplo {
3
4     /**
5      * @param args
6      */
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10    }
11
12 }
13
```

Para efeito de melhor visualizar o código, vamos exibir as linhas numeradas no Eclipse. Segue o passo a passo: Menu Window, Preferences, General, Editors, Text Editors e marque a opção Show line numbers.




Finalmente vamos ao que interessa: **PROGRAMAÇÃO** 😄.  
Vamos digitar um código para mostrar uma mensagem na tela e compilá-lo. Digite o código conforme figura abaixo:

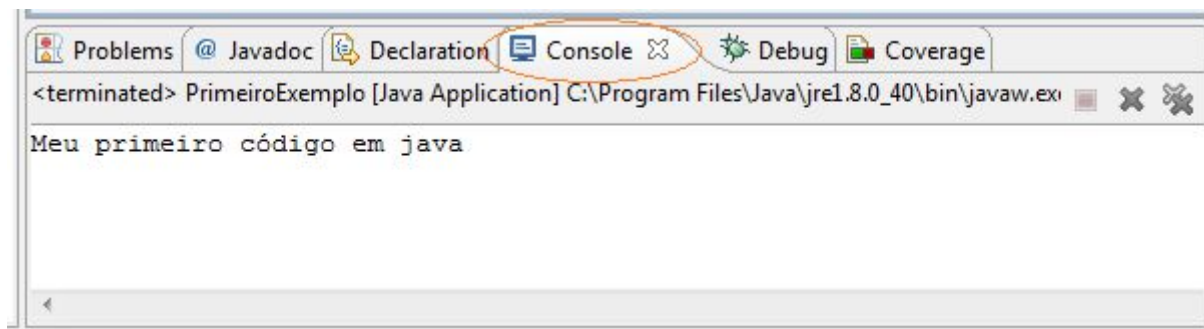




```
1
2 public class PrimeiroExemplo {
3
4     /**
5      * @param args
6      */
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10
11         System.out.println("Meu primeiro código em Java");
12
13     }
14
15 }
16
```

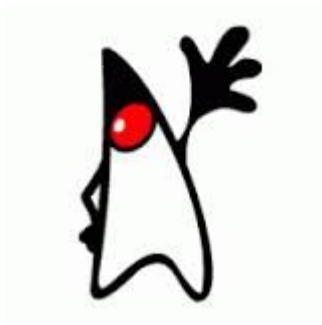
*Estas linhas com barras e asteriscos são apenas comentários, não interferem no código no momento, você pode até apagá-las se quiser. Falaremos sobre comentários mais adiante.*

Comando digitado agora tecla CTRL+S para salvar. Após salvar o arquivo vamos executá-lo. Tecla F11 ou clique em  na parte superior da tela do Eclipse. Agora na view Console teremos a execução da nossa primeira aplicação Java.



A tela mostra o resultado da execução da nossa aplicação.

Utilizamos o comando `System.out.println()` para exibir a frase “Meu primeiro código em Java” na tela. Perceba a utilização de aspas dentro do parêntese, que é onde o texto deve ficar. O que estiver dentro dessas aspas o Java vai entender como apenas texto.



**Dica:**

- Digite **syso**, tecla **CTRL + espaço** e depois **Enter** você terá toda a linha de exibição na tela `"System.out.println()"`.

- Dentro da classe pública você pode digitar **main**, depois **CTRL + espaço**, **Enter**. Você terá automaticamente toda a linha obrigatória da classe pública `"public static void main (String[] args)"`

### Módulo III –

#### Tipos de dados, Utilizando variáveis, comentários....

Agora que já estamos com o nosso ambiente de desenvolvimento pronto e testado, vamos fazer uma pausa muito importante para vermos alguns conceitos de POO (Programação Orientada a Objetos).

Para se criar uma classe no Java utilizamos a palavra reservada **class** como fizemos no exemplo anterior:

```
1
2 public class PrimeiroExemplo {
3
4     /**
5      * @param args
6      */
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10    }
11
```



`public` é o modificador (veremos mais adiante), `class` é a palavra reservada para criar uma classe e as chaves delimitam o escopo, o corpo da classe, que é onde nosso código ficará.

Para criarmos um objeto desta classe utilizamos a palavra reservada **NEW**, veremos adiante uma lista com essas palavras reservadas.

## :: Tipos de dados

Sempre que formos utilizar uma variável devemos declarar o seu tipo de dado. Iremos ver agora os tipos de dados existentes no Java. São eles:

- boolean: armazena um valor lógico **true** (verdadeiro) ou **false** (falso).

Não se pode converter um valor lógico em outro tipo de dado;

- byte: compreende valores de -128 a 127. Muito usado para economizar espaço em disco.

- short: compreende valores de -32768 a 32767;

- int: tipo inteiro, suporta números de 32 bits. Este tipo de dado é padrão para operações com números inteiros em Java.;

- long: suporta números de 64 bits;

- float: armazena números de ponto flutuante (dízimas) com 32 bits. Para definirmos um valor com esse tipo de dado devemos atribuir-lhe um “F” ou “f” no final. Exemplo: float num = 33.45F;

- double: armazena números de ponto flutuante (dízimas) com 64 bits. Para definirmos um valor com esse tipo de dado devemos atribuir-lhe um “D” ou “d” no final. Exemplo: Double num = 33.45D;

- char: criado para armazenar um caracter Unicode. Exemplo: char c1 = ‘B’ ;

- String: armazena texto. Obs: String é um tipo mais especial, veremos mais detalhes à frente. Detalhe: O tipo `String` não é bem

um tipo primitivo, na verdade é uma classe inteira com vários métodos. Veremos sobre a classe String mais adiante.

### Caracteres de escape:

'\n': nova linha

'\r': retorno do carro

'\t': tabulação

```
1
2 public class TiposDeDados {
3     public static void main(String[] args) {
4
5         boolean bo = true; //armazena um valor lógico true (verdadeiro) ou false (falso).
6
7         byte by = -122; //compreende valores de -128 a 127. Muito usado para economizar espaço em disco.
8
9         short sh = 30654; //compreende valores de -32768 a 32767
10
11         int i = 200; //tipo inteiro, suporta números de 32 bits. Este tipo de dado é padrão para
12                     //operações com números inteiros em Java;
13
14         long lo = 53676; //suporta números de 64 bits
15
16         float fl = 55.60F; //armazena números de ponto flutuante (dízimas) com 32 bits. Para definirmos
17                             //um valor com esse tipo de dado devemos atribuir-lhe um "F" ou "f" no final.
18
19         double db = 60.42D; //armazena números de ponto flutuante (dízimas) com 64 bits. Para definirmos
20                             //um valor com esse tipo de dado devemos atribuir-lhe um "D" ou "d" no final.
21
22         char ch = 'T'; //criado para armazenar um caracter Unicode;
23
24         String st = "Java na Prática"; //armazena texto. Obs: String é um tipo mais especial, veremos mais
25                                     //detalhes à frente
26     }
27
28 }
29
```

### :: Utilizando variáveis

Declaração de variáveis: para declarar uma variável no Java devemos indicar o tipo de dado e um nome para a variável. Exemplo:

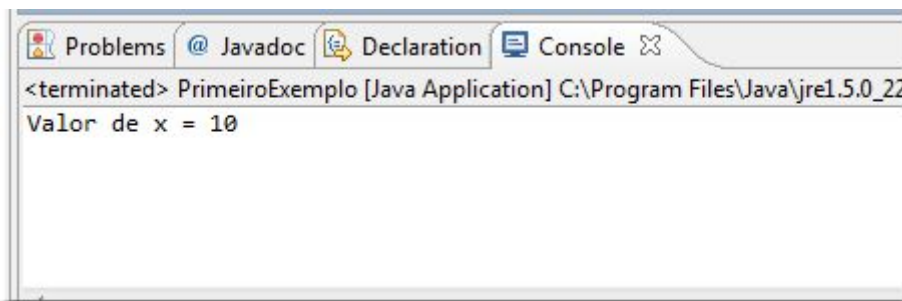
```

2 public class PrimeiroExemplo {
3
4     public static void main(String[] args) {
5
6         int x; //declaração da variável x do tipo inteiro
7         x = 10; //atribuindo o valor 10 a esta variável
8
9         /* Este é um comentário
10            de mais de uma linha */
11
12         System.out.println("valor de x =" + x);
13     }
14 }

```

Neste exemplo declaramos uma variável “x” do tipo int e atribuímos o valor 10 a esta variável.

Perceba a utilização de “//” para realizar um comentário de uma linha e o “/\* \*/” para comentário de mais de uma linha. Comentários são ignorados pelo Java, não interferem no código, sua utilização serve apenas para anotações. Utilizamos também o comando `System.out.println()` para exibir o valor da variável x, que no caso é 10. Tecle Ctrl+s para salvar, execute este código teclando F11 e veja o resultado.



Dentro do comando `System.out.println()` temos entre aspas o texto “valor de x=” concatenado com a variável x. A concatenação utiliza o símbolo “+”. Não confundir com o símbolo de soma que todos conhecemos, não tem nada a ver. Concatenação é uma junção. Nesse caso, uma junção do texto e do valor da variável x que é 10. Também podemos declarar uma variável já atribuindo um valor inicial na hora da declaração.

```

2 public class PrimeiroExemplo {
3
4     public static void main(String[] args) {
5
6         int x = 10; //declarando uma variável já atribuindo um valor
7
8
9         System.out.println("valor de x =" + x);
10    }
11 }

```

Lembre-se de executar o código (tecla F11) para visualizar o resultado.

Podemos atribuir o valor desta variável a outra variável. Exemplo:

```

int x = 10; //declarando uma variável já atribuindo um valor
int y = x + 5;

System.out.println("valor de x =" + x);
System.out.println("valor de y =" + y);

```

Neste caso estamos declarando uma variável x, do tipo int com valor inicial 10, uma variável y, do tipo int com valor inicial x + 5, ou seja, y valerá 15.

Podemos também alterar o valor de uma variável no meio do caminho. Exemplo:

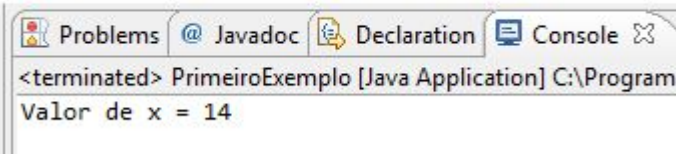
```

public static void main(String[] args) {

    int x=10;
    x = x + 4;

    System.out.println("Valor de x = " + x)
}

```



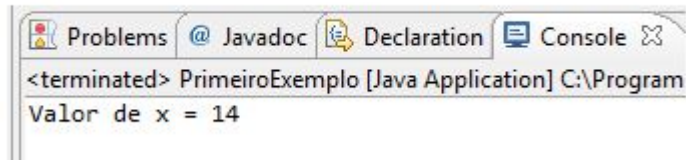
Agora declaramos uma variável x, do tipo int, com valor inicial 10; depois atribuímos a x o valor x + 4, ou seja, agora x vale 14. Também podemos escrever o exemplo acima de uma outra forma. Exemplo:

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

    int x=10;
    x += 4;

    System.out.println("Valor de x = " + x);

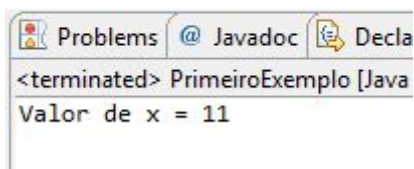
}
```



Lemos a operação acima da seguinte maneira: x recebe x + 4, ou seja, o x ainda valerá 14.

No Java também existe os operadores de incremento (++) e decremento (--). Exemplo:

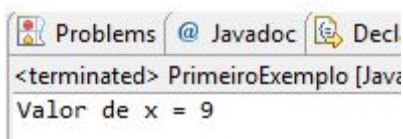
```
int x=10;
x ++;
```



Aqui nós utilizamos o operador de incremento “++” e lemos da seguinte forma: x recebe x + 1, ou seja, o x agora valerá 11.

Agora veremos o operador de decremento:

```
int x=10;
x --;
```



Aqui nós utilizamos o operador de decremento -- e lemos da seguinte forma: x recebe x - 1, ou seja, o x agora valerá 9.

### :: Escopo das variáveis

No Java, diferente de outras linguagens, pode-se declarar variáveis a qualquer momento e em qualquer lugar do código, mas dependendo de onde você as declarou ela irá valer de determinado ponto a outro do código.

O escopo de uma variável é o nome dado ao trecho de código em que aquela variável existe e pode ser acessada. Exemplo:

```
8      int contador;  
9      public void variaveis(){  
10         int contador;  
11     }
```

Na linha 8 declaramos uma variável de escopo da classe, ou seja, a classe inteira e quem estiver dentro dela poderá utilizar esta variável contador;

Na linha 10 declaramos uma variável de escopo do método (também chamada de **variável local**), ou seja, esta variável contador só existe dentro do método variaveis(), só pode ser usada dentro dele;



Observe que apesar das duas variáveis de nosso exemplo terem o mesmo nome (contador), elas são completamente diferentes. Uma pode ser usada pela classe inteira e a outra só existe dentro do método variaveis().

- **Variáveis de instância** (linha 8) são campos do objeto, declarados fora de qualquer método e inicializados quando é instanciado (criado) um objeto da classe. Também são chamadas de campos ou atributos.

- **Variáveis locais** (linha 10) são as declaradas dentro de um método. Seu ciclo de vida está relacionado ao seu escopo (no máximo o do método) e elas sempre ficam numa área de memória chamada de *stack*. Antes de tentar utilizá-las é necessário inicializá-las explicitamente, em uma parte atingível do código (fora de um if, por exemplo), senão o compilador retornará um erro. Variáveis locais também são chamadas de variáveis de *stack*, temporárias, automáticas ou de método.

- Vamos desenvolver um programa simples onde criamos variáveis com seus respectivos atributos e tipos de dado e exibimos no console do Eclipse os seus valores iniciais.

```

1 public class PrimeiroExemplo {
2
3     public static void main(String[] args) {
4
5         int qtdNotas = 0;
6         double numeroDouble = 10.5D;
7         char letra = 'B';
8         float numeroFloat = 4.6F;
9         String nome = "Curso Java na Prática";
10
11         System.out.println(qtdNotas);
12         System.out.println(numeroDouble);
13         System.out.println(letra);
14         System.out.println(numeroFloat);
15         System.out.println(nome);
16     }
17 }

```

Explicando o código acima:

1 – criamos nossa classe pública com o seu método main obrigatório:

```

public class PrimeiroExemplo {

    public static void main(String[] args) {

```

2 – declaramos nossas variáveis (atributos) com : tipo de dado, nome do atributo e seu valor inicial respectivamente. Lembrando que não é obrigatório atribuir um valor inicial a estas variáveis.

```

int qtdNotas = 0;
double numeroDouble = 10.5D;
char letra = 'B';
float numeroFloat = 4.6F;
String nome = "Curso Java na Prática";

```

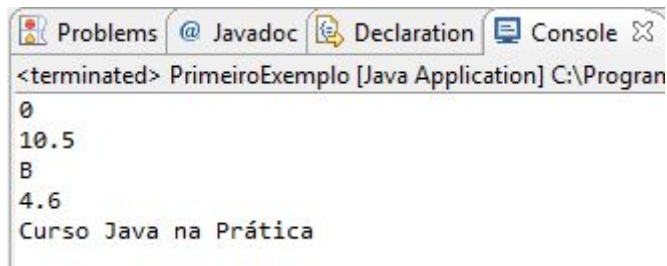
3 – Exibimos os valores iniciais das variáveis no console através do comando `System.out.println()`. Não esqueça de utilizar o `syso`, tecla `CTRL + espaço` e depois `Enter`, para facilitar:

```

System.out.println(qtdNotas);
System.out.println(numeroDouble);
System.out.println(letra);
System.out.println(numeroFloat);
System.out.println(nome);

```

4 – Teclando `F11` nós vamos executar o código e obteremos o resultado na view Console:



The screenshot shows a Java IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of a Java application named 'PrimeiroExemplo'. The output consists of five lines: an empty line, '10.5', 'B', '4.6', and 'Curso Java na Prática'.

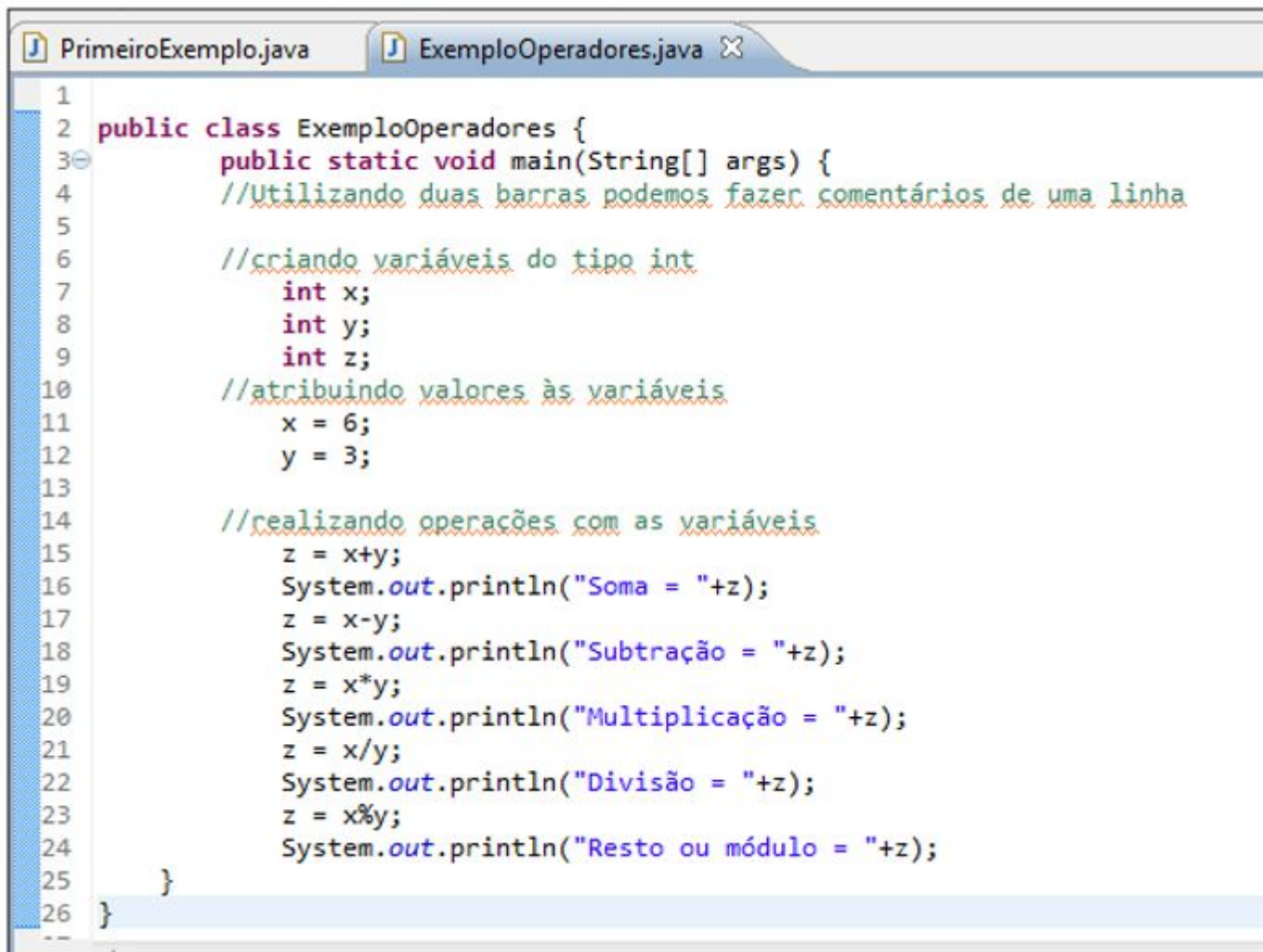
```
<terminated> PrimeiroExemplo [Java Application] C:\Program
0
10.5
B
4.6
Curso Java na Prática
```

## :: Operadores

- Aritméticos: + (soma), - (subtração), \* (multiplicação), / (divisão), % (resto da divisão ou módulo)
- Igualdade e Desigualdade: == (igual), !=(diferente).
- Relacionais: >, <, >=, <=.
- Lógicos: && (e), || (ou).

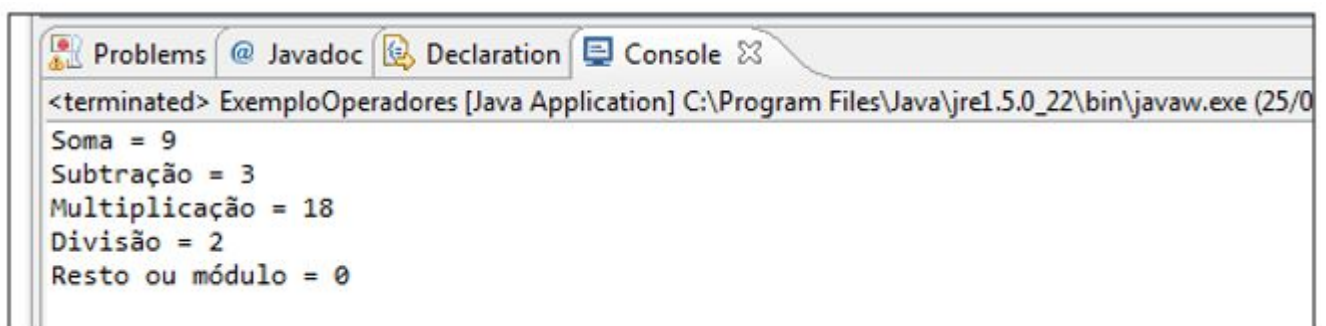
Exemplo de código utilizando operadores e comentários. Crie uma classe chamada ExemploOperadores e digite o seguinte código:





```
1
2 public class ExemploOperadores {
3     public static void main(String[] args) {
4         //Utilizando duas barras podemos fazer comentários de uma linha
5
6         //criando variáveis do tipo int
7         int x;
8         int y;
9         int z;
10        //atribuindo valores às variáveis
11        x = 6;
12        y = 3;
13
14        //realizando operações com as variáveis
15        z = x+y;
16        System.out.println("Soma = "+z);
17        z = x-y;
18        System.out.println("Subtração = "+z);
19        z = x*y;
20        System.out.println("Multiplicação = "+z);
21        z = x/y;
22        System.out.println("Divisão = "+z);
23        z = x%y;
24        System.out.println("Resto ou módulo = "+z);
25    }
26 }
```

Pressione F11 para executar o código e teremos o seguinte resultado no console do Eclipse:



```
<terminated> ExemploOperadores [Java Application] C:\Program Files\Java\jre1.5.0_22\bin\javaw.exe (25/0)
Soma = 9
Subtração = 3
Multiplicação = 18
Divisão = 2
Resto ou módulo = 0
```

**Obs.**

Lembre-se que o sinal de “+” pode ter dois comportamentos distintos: a soma e a **concatenação**. Na linha 15 ele soma os valores de x e y. Já nas linhas 16, 18, 20, 22 e 24 este sinal tem comportamento de concatenação, junção.

## :: Controle de Fluxo

- Instrução **if else** (se então): podemos ler como “se, condição verdadeira faça, senão faça outra coisa”, ou seja, o bloco de instrução só será executado se uma certa condição for satisfeita. Exemplo:

```
public class PrimeiroExemplo {  
    public static void main(String[] args) {  
        int dia = 0;  
        dia = 7;  
        if (dia == 7){  
            System.out.println("O dia é "+dia);  
        }else{  
            System.out.println("O dia não é 7");  
        }  
    }  
}
```

<terminated> PrimeiroExem  
O dia é 7

No exemplo acima a variável *dia* começa valendo 0, depois o seu valor é alterado para 7; se a variável *dia* for igual a 7 tem-se a mensagem “o dia é 7”; se a variável *dia* não for igual a 7 tem-se a mensagem “o dia não é 7”.

Faça um teste também alterando o valor da variável *dia* para, por exemplo, 5. Perceba também que cada bloco de instrução começa e termina com suas respectivas chaves `{ }` e dentro do `if` o operador de igualdade é `“==”` (nesse caso seria um operador de comparação).

Utilizando vários blocos de controle de fluxo ou condição. Exemplo:

```

public class PrimeiroExemplo {

    public static void main(String[] args) {

        int num = 10;

        if (num > 10) {
            System.out.println("maior que 10");
        } else
        if (num < 10){
            System.out.println("menor que 10");
        } else {
            System.out.println("igual a 10");
        }
    }
}

```

O resultado desta operação será a mensagem “igual a 10”. A variável *num* começa com o valor 10, em seguida o java testa se *num* é maior que 10, depois testa se *num* é menor que 10 e finalmente verificado que *num* não é maior e nem menor que 10, logo *num* será igual a 10.

Faça o teste com outros valores para *num* e você pode também salvar esta classe com outro nome por exemplo “ControleFluxo”. Menu File, Save As, digite o nome, Ok.

- Instrução **Switch** : esta instrução testa somente variáveis que possam ser convertidas para inteiro. Depois de testada a variável devemos trabalhar com blocos “case” dentro do switch. Exemplo:

```

public static void main(String[] args) {

    int num = 5;

    switch (num){

        case 10:
            System.out.println("O valor é 10");
            break;
        case 5:
            System.out.println("O valor é 5");
            break;
        case 0:
            System.out.println("O valor é 0");
            break;
    }
}

```

No exemplo acima o Switch verifica o valor da variável *num*, dependendo do valor que lhe foi atribuído um bloco *case* referente será executado, nesse caso o resultado da aplicação será a mensagem “o valor é 5”. O switch é muito usado para se evitar vários blocos IF else, com os blocos case fica mais prático. Podemos ler o exemplo acima da seguinte forma: caso *num* igual a 10, caso *num* igual a 5, caso *num* igual a 0.

Perceba a utilização em cada case da palavra reservada “break”, ela serve para executar o case referente ao resultado da variável testada e logo depois sair do Switch, se você *não utilizar o break*, todos os cases serão executados. Teste este código no seu Eclipse atribuindo outros valores à variável *num*.

Outro exemplo da instrução Switch:

```
2 public class Teste2SwitchCase {
3
4     public static void main(String[] args) {
5
6         int mes = 20;
7
8         switch (mes){
9             case 1 :
10                System.out.println ("Janeiro");
11                break;
12             case 2 :
13                System.out.println ("Fevereiro");
14                break;
15             case 3 :
16                System.out.println ("Março");
17                break;
18             case 4 :
19                System.out.println ("Abril");
20                break;
21
22             case 5 :
23                System.out.println ("Maio");
24                break;
25             case 6 :
26                System.out.println ("Junho");
27                break;
28             case 7 :
29                System.out.println ("Julho");
30                break;
31             case 8 :
32                System.out.println ("Agosto");
33                break;
34             case 9 :
```

```

34         System.out.println ("Setembro");
35         break;
36         case 10 :
37             System.out.println ("Outubro");
38             break;
39         case 11 :
40             System.out.println ("Novembro");
41             break;
42         case 12 :
43             System.out.println ("Dezembro");
44             break;
45         default :
46             System.out.println ("Mês inválido");
47     }
48 }
49 }

```

No exemplo acima o Switch verifica o valor da variável *mes*, dependendo do valor que lhe foi atribuído um bloco *case* referente será executado, nesse caso o resultado da aplicação será a mensagem “mês inválido”, se colocarmos a variável *mes* com o valor 10, por exemplo, a mensagem será “Outubro” e assim por diante. Atenção na linha 45, caso a variável *mes* tenha um valor não encontrado em todos os blocos *case*, então será exibido um valor padrão (default).

Instrução **While** (enquanto): o bloco será executado *enquanto* a condição for verdadeira. Perceba a diferença entre o **If else** e o **While**. No primeiro o bloco será executado somente SE a condição for verdadeira; no segundo o bloco será executado ENQUANTO a condição for verdadeira. Exemplo:

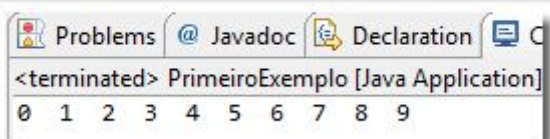
```

public static void main(String[] args) {

    int x = 0;

    while (x <10) {
        System.out.print(x+" ");
        x++;
    }
}

```



Neste exemplo declaramos uma variável *x*, do tipo *int*, com valor inicial 0. Em seguida o Java exibe na tela o valor de *x*, que

começou com 0, depois acrescenta 1 a este valor utilizando o operador de incremento ++.

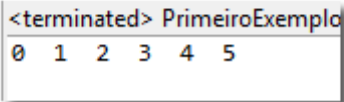
Resumindo: o valor de x será exibido e acrescentado 1 ENQUANTO x for menor que 10. Outra coisa para se notar é que neste código removemos o “\n” (linha) do final do comando “System.out.print”, por isso tudo foi exibido na horizontal. Outra coisa foi a concatenação da variável x com espaço em branco entre aspas. Digite este código no Eclipse e tecle F11 para testar.

Com o While também podemos utilizar o “break” para sair do bloco. Exemplo:

```
int x = 0;

while (x < 10) {
    System.out.print(x+ " ");

    if (x == 5){
        break;
    }
    x++;
}
```



<terminated> PrimeiroExemplo						
0	1	2	3	4	5	

Neste exemplo assim que o Java entra no bloco while ele exibe o valor atual do x (que no caso é 0), testa se x é igual a 5, se x for igual a 5 interrompemos o bloco utilizando o break, se ainda não for igual a 5 ele é incrementado com 1 e volta para o início do laço. Se executar este código o resultado final de x será 5. Teste este código no Eclipse.

Outro exemplo do While. Este código verifica quando um número 0 a 70 é ímpar e quando ele é divisível por 4. Perceba que agora vamos misturar While com If Else.



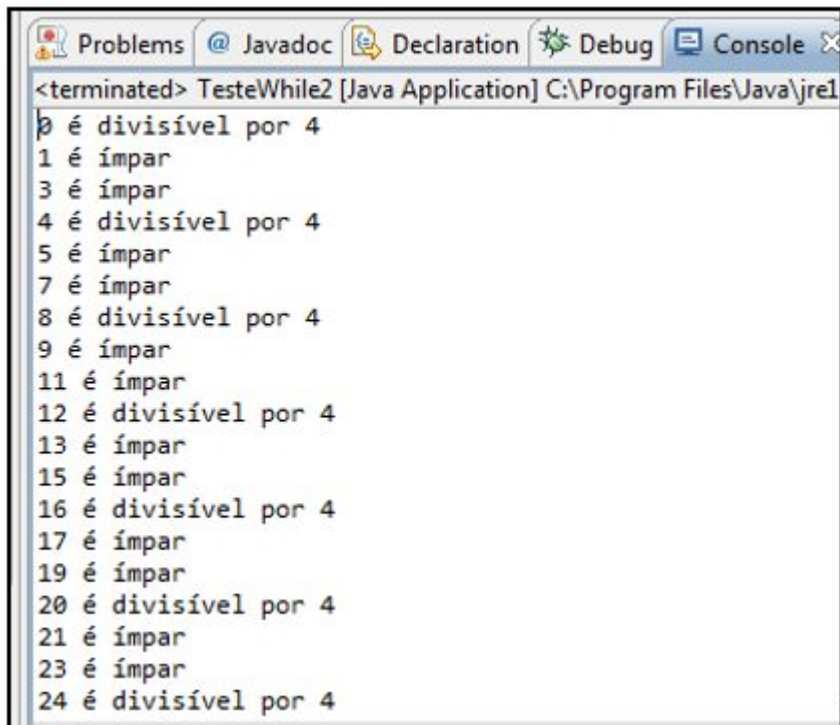
```

1
2 public class TesteWhile2 {
3
4     public static void main(String[] args) {
5
6         int num = 0;
7
8         while (num <= 70){
9
10            if (num % 4 == 0){
11                System.out.println(num+" é divisível por 4");
12            }
13            if (num % 2 !=0){
14                System.out.println(num+" é ímpar");
15            }
16            num++;
17        }//fim do while
18    }// fim do método main
19
20 }

```

- Linha 6: variável num recebe 0 como valor inicial;
- Linha 8: estrutura While será executada enquanto num for menor ou igual a 70, ou seja, em 71 ele para e finaliza o laço;
- Linha 10: comando If testa se o módulo (% , resto da divisão) da divisão por num é igual a 0, se for = a 0, significa que o número é divisível por 4, se for diferente de 0, então ele passa adiante para a linha 13;
- Linha 13: comando If testa se o módulo da divisão por num é diferente de 0, se for, o número é ímpar;
- Linha 16: num é incrementado em uma unidade com o sinal ++;

Executando este código:



```
<terminated> TesteWhile2 [Java Application] C:\Program Files\Java\jre1
0 é divisível por 4
1 é ímpar
3 é ímpar
4 é divisível por 4
5 é ímpar
7 é ímpar
8 é divisível por 4
9 é ímpar
11 é ímpar
12 é divisível por 4
13 é ímpar
15 é ímpar
16 é divisível por 4
17 é ímpar
19 é ímpar
20 é divisível por 4
21 é ímpar
23 é ímpar
24 é divisível por 4
```

- Instrução For (para): o bloco será executado até que atinja o valor recebido pela variável. Exemplo:

```
public static void main(String[] args) {  
    for (int x = 0; x < 10; x++){  
        System.out.print(x + " ");  
    }  
}
```

```
<terminated> PrimeiroExemplo  
0 1 2 3 4 5 6 7 8 9
```

Perceba que dentro dos parênteses do For temos 3 passos importantes separados por ponto e vírgula, são eles respectivamente: declaração da variável a ser testada com atribuição de um valor inicial (int x=0); o teste efetivo desta variável (x < 10); o seu incremento (x++).

No código acima a variável x é do tipo int e recebe 0 como valor inicial; depois definisse até quando o laço será executado (no caso, até x menor que 10, ou seja, 9); e depois incrementa x, que começou com 0, em uma unidade utilizando o operador de incremento ++. No exemplo acima o Java exibirá na tela o valor de x, incrementando uma unidade, enquanto x for menor que 10.



Teste este código no eclipse para vê-lo funcionando. Se quiser o resultado apresentado na vertical, basta acrescentar o "ln" ao final do comando `System.out.println`.

Outro exemplo do For. Neste código temos dois em um: o primeiro código exibe a soma dos primeiros 30 números inteiros; o segundo código exibe a tabuada de um número.

```
2 public class TesteFor2 {
3
4     public static void main(String[] args) {
5
6         System.out.println("Soma dos números");
7         int soma=0;
8         for (int x = 0; x <= 30; x++){
9             soma = soma + x;
10        }//final for
11        System.out.println(soma);
12
13        //-----
14        System.out.println();
15        System.out.println("Tabuada de um número");
16
17        int numTabuada = 6;
18        for (int y = 0; y <= 10; y++){
19            System.out.println(y * numTabuada);
20        }
21    }
22
23 }
```

- Linha 7: variável soma recebe 0 como valor inicial;
- Linha 8: instrução for que inicializa uma variável x com valor inicial 0, executa o código até que x seja menor ou igual a 30 e incrementa (++) x em uma unidade;
- Linha 9: variável soma recebe o valor de soma mais o valor de x;
- Linha 11: exibe na tela o valor final de soma;
- Linha 14: linha em branco com o comando `System.out.println()`;
- Linha 17: variável numTabuada recebe 6 como valor inicial;

- Linha 18: instrução for que inicializa uma variável y com valor inicial 0, executa o código até que y seja menor ou igual a 10 e incrementa (++) y em uma unidade;

- Linha 19: exibe na tela o valor atual de y multiplicado por numTabuada;

## **Módulo IV – Métodos, Objetos, Características da P.O.O. ....**

### **:: Trabalhando com métodos e Objetos**

Como vimos antes neste curso, os métodos irão representar uma ação e seu nome começa com letra minúscula por convenção. Neste exemplo teremos um programa que irá exibir os números ímpares de um certo intervalo e também teremos vários conceitos de orientação a objetos. Você irá gostar deste tópico! Antes, mais um pouco de teoria.

### **:: Características da Orientação a Objetos:**

- Reutilização de código: o programador muitas vezes usa o mesmo código em várias ocasiões, isso diminui muito o tempo e trabalho de programação. No Java podemos reutilizar o mesmo código várias vezes;

- Concentração de tarefas: cada objeto realiza a sua tarefa. Por exemplo, você tem uma classe que só realiza cálculo de Imposto de Renda, então toda vez que o governo fizer uma alteração na base de cálculo, só esta classe será alterada, deixando todo o restante do projeto imutável;

- Encapsulamento: haverá casos na nossa programação que não é interessante exibir todo o nosso código, nós vamos ocultar (encapsular) determinadas áreas do código. Usando como exemplo uma analogia clássica na Orientação a Objetos: todo mundo sabe utilizar um liquidificador ligando, aumentando a velocidade, encaixando o copo, mas não sabemos como funciona a parte elétrica dele, não é interessante para o fabricante que se mexa nesta parte então ele (o fabricante) à encapsula.

- Herança: muitas vezes um determinado indivíduo (objeto) herdará características de outros. Por exemplo: um carro Gol e um Golf, ambos são carros diferentes, mas todos tem portas, motor, pneus, bancos... são atributos herdados de outros carros;

- Polimorfismo: um objeto irá se comportar de várias formas diferentes.

#### :: Pacotes, Classes, Métodos, Objetos e Atributos

- Pacote (package) é um conjunto de classes. Ex: java.util é um pacote que contém várias classes como Date, SimpleDateFormat e muitas outras.

- Uma classe é um tipo de dados que representa uma estrutura e é definida pelo programador. Por exemplo, criar uma classe que somente realiza o cálculo do FGTS, outra que só calcula o Imposto de Renda.

- Um objeto representa um elemento desta classe.

- Método é a maneira com que uma classe irá realizar ações por exemplo soma(), subtração(). Lembrando que sempre haverá parênteses após o nome do método.

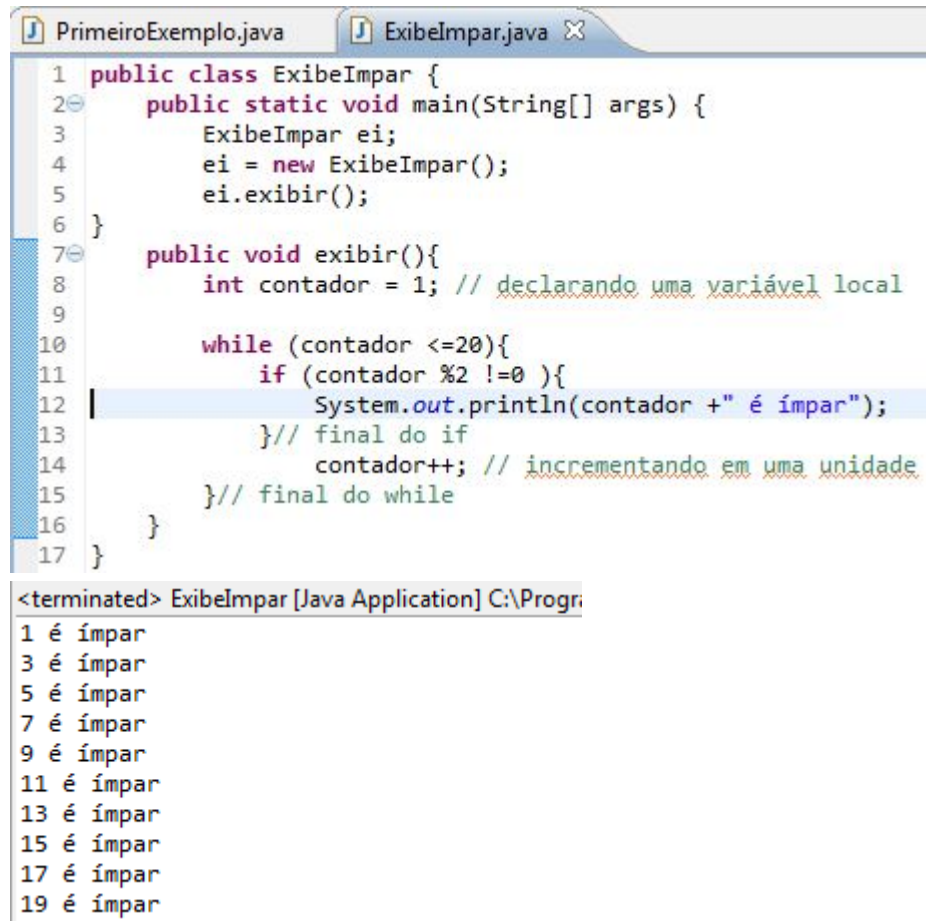
- Atributos (campos) são as características desse objeto. Por exemplo, o campo idade, endereço, cep etc.

- O escopo (podemos entender como o corpo do código) de uma classe é delimitado por chaves “ { } “. Todo o código da classe estará dentro destas chaves.

Por exemplo, fazendo uma analogia com a planta de uma casa. A planta da casa seria a nossa classe; as casas que nós criamos a partir desta planta seriam os objetos; as características dessa casa (tamanho, cômodos, cor, altura) seriam os atributos, ou seja, a partir de uma classe eu posso criar vários objetos com vários atributos.

*Também podemos dizer que um objeto é uma **instância** de uma classe. A casa que criamos a partir da planta da casa é um objeto, que também pode ser chamado de **instância** (acostume-se com este termo).*

Vamos ao exemplo. Primeiro crie uma classe chamada ExibeImpar.java, já vimos isso no começo do curso. File, New, Class, ExibeImpar.java. Não esqueça de deixar marcada a opção ☒ public static void main(String[] args) Digite o código abaixo:



```
1 public class ExibeImpar {
2     public static void main(String[] args) {
3         ExibeImpar ei;
4         ei = new ExibeImpar();
5         ei.exibir();
6     }
7     public void exibir(){
8         int contador = 1; // declarando uma variável local
9
10        while (contador <=20){
11            if (contador %2 !=0 ){
12                System.out.println(contador + " é ímpar");
13            } // final do if
14            contador++; // incrementando em uma unidade
15        } // final do while
16    }
17 }
```

<terminated> ExibeImpar [Java Application] C:\Progr

1 é ímpar  
3 é ímpar  
5 é ímpar  
7 é ímpar  
9 é ímpar  
11 é ímpar  
13 é ímpar  
15 é ímpar  
17 é ímpar  
19 é ímpar

Neste código aplicamos vários conceitos explicados até aqui, então leia este passo a passo pelo menos 3 vezes para entender tudo perfeitamente. Explicando:

- Linha 1: criamos uma classe (class) pública (public) com o nome ExibeImpar (seguimos a convenção Java para nomes de classe, quando tem mais de uma palavra a primeira letra de cada palavra é maiúscula) e lembrando que o nome do arquivo deve ser o mesmo da classe pública;
- Linha 7: muita atenção agora. Nesta linha nós criamos um **Método** (representa uma ação) público (public), sem valor de retorno (void indica que o método não retorna nenhum valor), com o nome exibir().



*Perceba que abrimos e fechamos um parêntese para identificar um método, isso é uma regra e não uma convenção;*

- Linha 8: nós criamos uma variável local (só pode ser utilizada dentro do método) chamada *contador* e atribuímos um valor inicial 1, este valor será utilizado pelo bloco while;
- Linha 10: abrimos um bloco `while` que será executado até que *contador* seja menor ou igual a 20, lembrando que a variável *contador* já está com o valor 1;
- Linha 11: abrimos um laço IF (perceba que agora temos um laço IF dentro de um `while`) que testa se o resto da divisão (% , resto da divisão, também chamado de módulo) de *contador* por 2 é diferente de 0. Se for diferente de zero é porque o valor de *contador* é ímpar;
- Linha 12: exibe na tela o valor ímpar de *contador* com o comando `System.out.println()`;
- Linha 13: fecha o escopo do bloco IF;
- Linha 14: incrementa *contador* em uma unidade com o operador de incremento. Lembre-se que o laço `while` só será finalizado quando *contador* for menor ou igual a 20, enquanto não atingir este valor ele continuará executando;
- Linha 15: fecha o escopo do laço `while`;
- Linha 2: criamos o nosso método `main` (obrigatório para executar o programa);
- Linha 3: importante. Declaramos uma variável chamada de *ei* do tipo classe `ExibImpar`. Esta variável *ei* é que será utilizada para acessar o método da classe;

- Linha 4: utilizando a palavra reservada **NEW**, criamos (instanciamos) um objeto da classe `ExibImpar` e o atribuímos à variável `ei`, ou seja, `ei` é uma variável de objeto (instância da classe).

Instanciamos esta variável para podermos acessar o método `exibir()`. Mais tarde iremos aprender a fazer toda essa operação de criação de objetos em uma só linha.

- Linha 5: utilizamos a instância da classe para acessar o método `exibir()` e executar o que está dentro dele.

Digite e execute este código no seu Eclipse.

Você pode alterar este código para exibir os números pares, basta que o resto da divisão por 2 seja “`==0`” e não “`!=0`”.

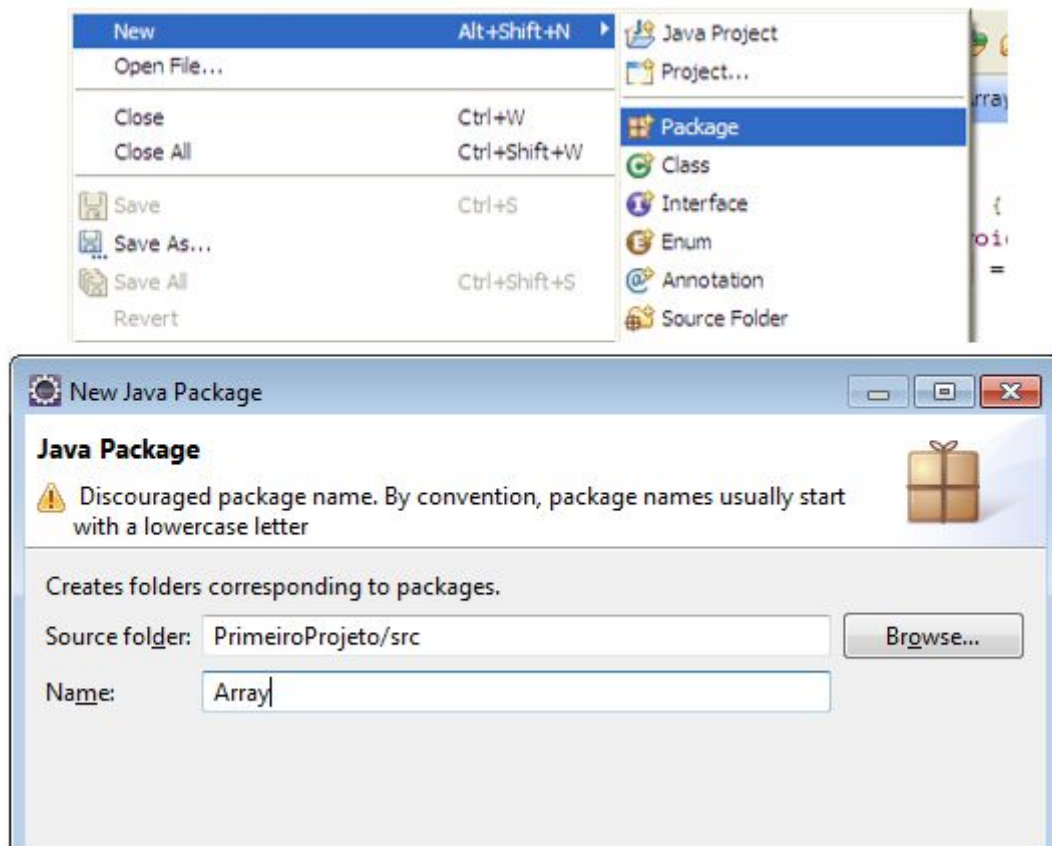
## **Módulo V - Pacotes, praticando com exemplos mais rebuscados....**

### **:: Pacotes**

O código fonte de uma aplicação real é formado por uma quantidade grande de arquivos, conforme essa quantidade cresce surge a necessidade de algum tipo de organização para poder encontrar os arquivos mais rapidamente quando for necessário modificá-los.

A ideia para organizar logicamente os arquivos de uma aplicação é bem simples e muito parecida com o que um usuário de computador já está acostumado a fazer. Os arquivos são separados em pastas e subpastas ou diretórios e subdiretórios.

Na terminologia do Java, dizemos que as classes são organizadas em **pacotes**, também conhecidos como bibliotecas ou API. O primeiro passo, para colocar uma classe em um determinado pacote, é utilizar o comando `Package` no código fonte. Por exemplo, vamos criar um pacote chamado `Array` seguindo os passos abaixo: Dentro do Eclipse: File, New, Package



Depois de colocarmos o nome basta clicar Finish. Todas as classes que forem criadas dentro deste pacote terão a palavra reservada *package*. Depois de criado o pacote, crie uma classe chamada Array1 (botão direito no pacote Array, New, Class). Perceba na figura abaixo que agora estamos realmente dentro do pacote Array.



**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

```

1 package Array;
2
3 public class Array1 {
4     public static void main(String[] args) {
5         int[] vetor = new int [10];
6         int i = 0;
7     }
8 }

```

Isso é muito bom para organizar todas as classes com um grau de afinidade em um mesmo pacote. Digamos que você tenha várias classes que realizam cálculos matemáticos, você pode organizá-las em um único pacote chamado Cálculos. Detalhe, quando você deseja utilizar uma classe de determinado pacote você deve utilizar a palavra reservada `import` “nome do pacote”. Exemplo:

```

1 package Strings_E_Datas;
2
3 import java.text.*;

```

No exemplo acima estamos utilizando a classe `Java.text` do pacote `Strings_E_Datas`.

Neste trecho de código estamos avisando ao Java que a nossa classe pertence ao pacote `Strings_E_Datas`, para manter organizado neste pacote `Strings_E_Datas` poderíamos ter somente classes referentes à manipulação de Strings e Datas, se alguém quiser

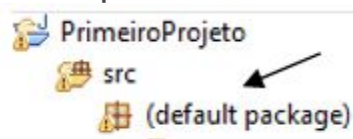


usar essas classes terá que importá-las com o comando *import*. Na Linha 3 temos algo parecido com isso: estamos importando todas as classes `java.text(` indicamos isso com o `*`) do pacote `String_E_Datas`.

No Java temos um pacote padrão que a maioria das classes utiliza, é o pacote `Java.lang`. Esse pacote é automaticamente importado pelo compilador do Java em qualquer programa. O Java disponibiliza na Internet gratuitamente milhares de API's (pacotes) prontas para uso. Por exemplo, mais a frente, vamos utilizar API's com dezenas de classes e métodos específicos para tratar Datas, Strings e Cálculos matemáticos.

:: **Praticando** – Prepare-se que vamos treinar e aprender muito mais agora. Faça todos esses exemplos, pratique.

Agora que já temos alguns fundamentos do Java e da Orientação a Objetos, vamos praticar mais um pouco e aproveitar para aprender novas funções do Java, na prática, nos códigos abaixo. Os próximos exemplos devem ficar em nosso pacote padrão mesmo,



o default

1 – Criar uma classe com o nome `Ordena` onde o usuário entra com dois valores e o sistema os exibe no console ordenando-os do menor para o maior:

```

1 import java.util.*;
2 public class Ordena {
3     public static void main(String[] args) {
4         Scanner infor = new Scanner (System.in);
5         int x, y, z;
6
7         System.out.print("digite o valor de X = ");
8         x = infor.nextInt();
9         System.out.print("digite o valor de Y= ");
10        y = infor.nextInt();
11
12        if (x > y){
13            System.out.println(y);
14            System.out.println(x);
15        }else {
16            System.out.println(x);
17            System.out.println(y);

```

Problems @ Javadoc Declaration C

<terminated> Ordena [Java Application] C:\Progra

digite o valor de X = 8

digite o valor de Y= 2

2

8

Explicando o código:

- Linha 1: através do comando `import` nós chamamos o pacote (API do Java), e suas classes, `java.util.*`, podemos ler como “importe todas as classes do pacote `java.util`”. Neste momento podemos utilizar todas as classes e métodos deste pacote. Acostume-se a conhecer as principais API’s do Java, são muito úteis para o programador não ter que “reinventar a roda”, já está tudo prontinho e testado para ser utilizado;
- Linha 2: criamos nossa classe pública obrigatória;
- Linha 3: criamos o método `main` (principal e obrigatório para a execução do código);
- Linha 4: novidade. Instanciamos um objeto da classe `Scanner` e o atribuímos à variável `infor`. A classe `Scanner` pertence ao pacote `Java.util` e basicamente a utilizaremos para entrada ou saída de dados. No nosso caso, entrada de dados pelo teclado;
- Linha 5: declaramos as variáveis `x,y,z` do tipo `int`;
- Linha 7: exibimos o texto “digite o valor de X= “ para indicar ao usuário que ele deve entrar com o valor de `x`;
- Linha 8: importante. Utilizamos a variável de instância (ou variável de objeto) `infor` para acessar o método `nextInt()` da classe `Scanner` do

pacote `Java.util`. O método `nextInt()` aguarda um valor inteiro vindo do teclado que o usuário deverá digitar;



*Se o usuário for digitar um `Double`, utiliza-se `nextDouble()`, se o usuário for digitar um `float`, utiliza-se `nextFloat()`, etc.*

Linhas 9 e 10 é o mesmo raciocínio das linhas 7 e 8 só que para a variável `Y`;

Linha 12: testa se o valor de `x` é maior que o valor de `y`, ambos valores informados pelo usuário. Se `x` for maior que `y` então se executam as Linhas 13 e 14, ou seja, exibe primeiro o valor de `y` e depois o valor de `x`. Se `x` for menor que `y` então se executam as Linhas 16 e 17, ou seja, exibe primeiro o valor de `x` e depois o valor de `y`;

Digite e execute o código acima.

2- Crie uma classe `AreaCirculo` para determinar a área de um círculo (área =  $\pi \times \text{raio}^2$ ): Atenção na linha 10.

```
1 public class AreaCirculo {
2     public static void main(String[] args) {
3
4         int raio = 8;
5         double area;
6         double pi = Math.PI;
7         double potencia = Math.pow(raio, 2);
8
9         area = pi * potencia;
10        System.out.println("Área do círculo= " + (float)area);
11
12    }
13 }
14
```

```
<terminated> AreaCirculo [Java Applet]
Área do círculo= 201.06194
```

- Linha 4: variável `raio`, do tipo `int`, com valor inicial 8;
- Linha 5: variável `área` do tipo `double` sem valor inicial.
- Linha 6: variável `pi` que irá armazenar o valor de  $\pi$  . Para

obtermos o valor do  $\pi$  utilizamos o método PI da classe **Math** (atenção para esta classe, ela é muito útil para cálculos matemáticos);



Utilizamos neste exemplo a classe **Math** sem precisarmos dar um import do pacote. Isso porque esta é uma classe do pacote *Java.lang* (já mencionado acima) que é automaticamente importado pelo Java sem precisar explicitar isso com o comando *import*. É o único pacote do Java com essas características, os outros devem ser importados normalmente.

- Linha 7: variável potencia do tipo double, onde vamos armazenar o valor do raio elevado ao quadrado. Note que utilizamos o método **pow** (potência) da classe **Math** do pacote *Java.lang* (importado automaticamente pelo Java);

- Linha 9: a variável *area* recebe o valor do  $\pi$  e o multiplica pelo valor da variável potencia;

- Linha 10: exibe o resultado no console. Detalhe: perceba essa parte do código `+"(float)area);` Que raios esse float dentro dos parênteses está fazendo? Ele está fazendo um Casting de Tipos Primitivos, é uma “conversão” de um tipo em outro. Por exemplo, tipo *int* em *Double*, tipo *float* em *int*, etc. Veremos um exemplo logo abaixo após a tela com outros métodos da classe **Math**.

Digite e execute o código acima.

## Módulo VI – Mais métodos da classe **Math**, Casting de tipos primitivo....

**Mais alguns métodos importantes da classe *Math*, pratique:**

Crie uma classe chamada *ClasseMath*

```

2 public class ClasseMath {
3     public static void main(String[] args) {
4         System.out.println("Eleva 4 ao cubo: " + Math.pow(4,3));
5         System.out.println("Raiz quadrada de 225: " + Math.sqrt(225));
6         System.out.println("Logaritmo de 128: " + Math.log(128));
7         System.out.println("Valor absoluto: " + Math.abs(-3));
8         System.out.println("Arredonda para cima: " + Math.ceil(3.5));
9         System.out.println("Arredonda para baixo: " + Math.floor(3.5));
10        System.out.println("Arredonda: " + Math.round(4.6));
11        System.out.println("Valor randômico entre 0 e 1: " + Math.random());
12        System.out.println("Maior valor: " + Math.max(10, 2));
13        System.out.println("Menor valor: " + Math.min(10,2));
14    }
15 }

```

```

<terminated> ClasseMath [Java Application] C:\Prog
Eleva 4 ao cubo: 64.0
Raiz quadrada de 225: 15.0
Logaritmo de 128: 4.852030263919617
Valor absoluto: 3
Arredonda para cima: 4.0
Arredonda para baixo: 3.0
Arredonda: 5
Valor randômico entre 0 e 1: 0.16559407
Maior valor: 10
Menor valor: 2

```

:: Pausa para conhecer Casting (implícito e explícito) de tipos primitivos: É como se fosse uma “conversão” de um tipo primitivo para o outro.

Exemplo:

```

1 public class TesteCasting {
2
3     public static void main(String[] args) {
4
5         int x = 10; //declaração normal de uma variável do tipo int
6         double y = x; //valor de x convertido para double, casting implícito
7         long z = 10L; //por padrão o 10 é um int, depois convertido para long
8
9         float w = 10.0F; //por padrão 10.0 é um double, depois convertido para float
10        int r = (int)14.5; //casting explícito, converte double para int truncando
11                               //o valor decimal
12        byte b = 127; // o tipo byte só vai até 127, mais do que isso não aceita
13
14        System.out.println(x+", "+y+", "+z+", "+w+", "+r+", "+b);
15    }

```

```

<terminated> Variaveis [Java Applic
10, 10.0 10, 10.0, 14, 127

```

### Casting Implícito

- Linha 5: uma declaração normal da variável *x* sendo do tipo *int*;
- Linha 6: uma variável *y* do tipo *double* recebendo a variável *x* que

é do tipo int. Neste momento o valor 10 que antes era int, agora se tornou double.

- Linha 7: por padrão o número 10 é um int, então temos que informar que queremos que ele seja long, fazemos isso com a letra L no final;

- Linha 9: por padrão o número 10.0 é um Double, então temos que informar que queremos que ele seja float, fazemos isso com a letra F no final;

#### Casting Explícito

- Linha 10: 14.5 por padrão é um Double, queremos realizar o casting para int, antecedemos o valor 14.5 com (int). Detalhe: o valor decimal é truncado, a variável r ficará com o valor 14. Foi o que fizemos no código 2, no exemplo da Área do Círculo.

3 – Crie um programa que soma 5 números inteiros:

```
2 public class SomaCincoInteiros {
3     public static void main(String[] args) {
4         int contador = 1;
5         int valor = 0;
6
7         while (contador <= 5){
8             valor = valor + contador;
9             contador ++;
10        }
11        System.out.println("Soma dos 5= "+valor);
12    }
13 }
```

<terminated> SomaCincoInteiros [J  
Soma dos 5= 15

- Linha 4: criamos uma variável contador, do tipo int, que recebe o valor inicial 1;

- Linha 5: criamos uma variável valor, do tipo int, que recebe o valor inicial 0;

- Linha 7: um laço WHILE que executará o seu código até que a variável contador atinja um valor menor ou igual a 5;

- Linha 8: a variável valor recebe valor + contador;



- Linha 9: incrementamos o valor de contador em uma unidade utilizando o operador de incremento (++);
- Linha 11: exibimos na tela o resultado da variável valor;

4 – Crie um programa onde o usuário entra com um ano e o sistema identifica se é bissexto ou não:

```

1  import java.util.*;
2
3  public class AnoBissexto {
4      public static void main(String[] args) {
5          Scanner infor = new Scanner(System.in);
6          int ano;
7          System.out.print("Informe o ano: ");
8          ano = infor.nextInt();
9
10         if (ano % 4 == 0 && ano % 100 != 0 || ano % 400 == 0){
11             System.out.println(ano + " é bissexto");
12         }else{
13             System.out.println(ano + " não é bissexto");
14         }
15     }

```

Relembrando alguns operadores:

% módulo ou resto da divisão  
 == sinal de igualdade em uma estrutura condicional  
 && "e"  
 || "ou"

Problems @ Javadoc Declaration  
 <terminated> AnoBissexto [Java Application]  
 Informe o ano: 2017  
 2017 não é bissexto

Linha 5: utilizando a palavra reservada *NEW*, criamos (instanciamos) um objeto da classe *Scanner* e o atribuímos à variável *infor*, ou seja, *infor* é uma variável de objeto (instância da classe). Instanciamos esta variável para podermos acessar o método *next.Int()* na Linha 8. Utilizamos a variável de instância (ou variável de classe) *infor* para acessar o método *nextInt()* da classe *Scanner* do pacote *Java.util*;

Linha 6: criamos uma variável *ano*, do tipo *int*;

Linha 8: a variável *ano* recebe um valor do tipo *int* informado pelo usuário;



(repetindo: variável *entra* acessando o método `nextInt()` da classe `Scanner` do pacote (API) `Java.util`)

Linha 10: com o IF testamos se o ano informado pelo usuário é bissexto;

5 - Crie um programa onde o usuário entra com o valor da sua dívida e o sistema verifica a quantidade de parcelas:

```
1 import java.util.*;
2 public class Divida {
3     public static void main(String[] args) {
4         Scanner entra = new Scanner(System.in);
5         double divida, dividaFinal;
6         float juros;
7         int qtdMeses;
8
9         System.out.println("Valor da divida: ");
10        divida = entra.nextDouble();
11        System.out.println("Número de parcelas: ");
12        qtdMeses = entra.nextInt();
13
14        System.out.println("Juros de (%): ");
15        juros = entra.nextFloat();
16        juros = (float) (juros*(divida/100));
17        dividaFinal = qtdMeses + juros + divida;
18        System.out.println("Total a pagar com juros: "+(float)dividaFinal);
19    }
```

```
<terminated> Divida [Java Application] C:\Program File
Valor da divida:
1200
Número de parcelas:
10
Juros de (%):
10
Total a pagar com juros: 1330.0
```

- Linha 1: importamos todas as classes do pacote `Java.util`;
- Linha 4: criamos uma variável de classe *entra* que irá receber um valor fornecido pelo usuário;
- Linha 5: criamos duas variáveis *divida* e *dividaFinal* do tipo `Double`;
- Linha 6: variável *juros* do tipo `float`;
- Linha 7: variável *qtdMeses* do tipo `int`;

- Linha 10: a variável *divida* recebe um valor do tipo double informado pelo usuário;
- Linha 12: a variável *qtdMeses* recebe um valor do tipo int informado pelo usuário;
- Linha 15: a variável *juros* recebe um valor do tipo float informado pelo usuário;
- Linha 16: a variável *juros* recebe o cálculo dos juros da divida e também utilizamos o casting explícito para que o valor final seja um float;
- Linha 17: a variável *dividaFinal* receberá o cálculo da dívida;
- Linha 18: exibe na tela o valor da dívida realizando o casting explícito do valor para float.

6: Crie um programa onde o usuário entra com um número e o sistema informa todos os valores da tabuada deste número:

```

1 import java.util.Scanner;
2 public class Tabuada {
3     public static void main(String[] args) {
4         Scanner x = new Scanner(System.in);
5         int contador=1, result=0;
6         System.out.println("Entre com o número tabuada:");
7         int nt = x.nextInt();
8
9         while(contador<=10){
10             result=nt*contador;
11             System.out.println(nt+" X "+contador+" = "+result);
12             contador++;
13         }
14     }
15 }
16

```

```

Entre com o número tabuada:
9
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90

```

Linha 4: Utilizamos a variável de classe (ou variável de instância) *x* para acessar o método `nextInt()` da classe `Scanner` do pacote `Java.util`;

Linha 5: variável *contador* com valor inicial 1 e variável *result* com valor inicial 0, ambas do tipo `int`;

Linha 7: variável *nt*, do tipo `int`, que irá receber um número inteiro informado pelo usuário;

Linha 9: laço `WHILE` que executará o seu código até que a variável *contador* seja menor ou igual a 10;

Linha 10: atribui-se o valor da multiplicação de *nt* por *contador* à variável *result*;

Linha 12: incrementamos em uma unidade a variável contador utilizando o operador de incremento (++);

### :: A Classe STRING

Comentamos rapidamente sobre a classe String na parte de declaração de tipos primitivos e vamos agora aprofundar o assunto que é muito importante. Variáveis do tipo String guardam referências a objetos, e não um valor, como acontece com os tipos primitivos. Assim como vimos vários métodos da classe Match, também veremos vários métodos da classe String.

Uma string irá armazenar um conjunto de caracteres como por exemplo nome, endereço, cidade, etc. Temos duas maneiras para declarar uma String e dependendo do modo de declaração o resultado retornado poderá ser diferente. Vamos entender melhor no exemplo. Crie uma classe chamada ClasseString:

```
2 public class ClasseString {
3     public static void main(String[] args) {
4
5         String y1 = "abc";
6         String y2 = "abc";
7
8         if (y1 == y2){
9             System.out.println("Variáveis y1 e y2 conteúdos iguais");
10        }else{
11            System.out.println("Variáveis y1 e y2 conteúdos diferentes");
12        }
13
14        String x1 = new String("abc");
15        String x2 = new String("abc");
16
17        if (x1 == x2){
18            System.out.println("Variáveis x1 e x2 são iguais");
19        }else{
20            System.out.println("Variáveis x1 e x2 são diferentes");
21        }
22
23        if (x1.equals(x2)){
24            System.out.println("Objetos x são iguais");
25        }else{
26            System.out.println("Objetos x são diferentes");
27        }
28    }
```

Problems Javadoc Declaration Console

<terminated> ClasseString [Java Application] C:\Program Files

Variáveis y1 e y2 conteúdos iguais  
Variáveis x1 e x2 são diferentes  
Objetos x são iguais

Linha 2: Criamos a classe ClasseString;

Linhas 5 e 6: Declaramos duas variáveis  $y1$  e  $y2$  de forma **literal** (é a forma mais simples e mais utilizada), bem semelhante à declaração de qualquer variável de tipo primitivo como já vimos várias vezes. Perceba que ambas tem o mesmo conteúdo;

Linha 8: com o comando `IF` iremos comparar (com o sinal `==`) os conteúdos das variáveis para saber se são iguais. Neste caso sim, eles são iguais como mostra o resultado da execução;

Linha 14 e 15: a segunda maneira de declarar uma String.

Instanciamos dois objetos `x1` e `x2` da classe String com conteúdos iguais.

Linha 17: novamente com o comando `IF` iremos comparar os conteúdos das variáveis para saber se são iguais. Neste caso perceba que agora o resultado do programa deu “diferente” mesmo os conteúdos das variáveis sendo idênticos. A pergunta

inevitável:  (porque?)

O Java reserva uma área chamada **String Constant Pool** (pool constante de string). Sempre que uma nova variável na forma literal (como nas linhas 5 e 6) é criada, o Java verifica o pool (área de memória) para confirmar se existe uma variável string idêntica a que foi criada, que no nosso exemplo sim, elas são iguais.

Na forma literal utilizando o operador `“==”` o Java compara os endereços de memória, já na segunda forma de declaração, com o objeto instanciado, o Java compara os conteúdos com o auxílio do método `equals()` (método este da classe String) como na linha 23.

Resumindo: na linha 17 não comparamos o conteúdo, comparamos os objetos, área de memória diferentes. Já na linha 8 comparamos o conteúdo. Mesma área de memória. É a forma mais utilizada.

- Linha 23: com o comando IF comparamos, com o auxílio do método equals(), da classe String, os dois objetos x1 e x2. Agora sim o resultado será “igual”, objetos com conteúdos idênticos, conforme podemos verificar na figura da execução do programa.

Digite e execute este código no seu Eclipse.

**Mais alguns métodos importantes da classe String:**

```
2 public class TesteString {
3     public static void main(String[] args) {
4         //utilizando o modo literal que é o mais usado.
5         String nome="Antonio";
6         String sobrenome = "Marcos";
7
8         System.out.println("Concatenando:"+ nome + sobrenome);
9         System.out.println("Tudo maiúscula: "+nome.toUpperCase());
10        System.out.println("Tudo minúscula: "+sobrenome.toLowerCase());
11        System.out.println("Retorna o caractere da posição: "+nome.charAt(4));
12        System.out.println("Retorna a quantidade de caracteres: "+sobrenome.length());
13        System.out.println("Recebe um int e retorna a partir da posição: "+sobrenome.substring(3));
14        System.out.println("Recebe um char e retorna a posição da primeira ocorrência: "+nome.indexOf('t'));
15        System.out.println("Verifica se termina com o valor especificado: "+nome.endsWith("nio"));
16        System.out.println("Verifica se começa com o valor especificado: "+nome.startsWith("cos"));
17    }
18 }
```

```
<terminated> TesteString [Java Application] C:\Program Files\Java\jre1.5.0_22\bin\java
Concatenando:AntonioMarcos
Tudo maiúscula: ANTONIO
Tudo minúscula: marcos
Retorna o caractere da posição: n
Retorna a quantidade de caracteres: 6
Recebe um int e retorna a partir da posição: cos
Recebe um char e retorna a posição da primeira ocorrência: 2
Verifica se termina com o valor especificado: true
Verifica se começa com o valor especificado: false
```

**:: Palavras chaves ou reservadas**

São aquelas palavras que só podem ser utilizadas pelo Java. Por exemplo você não pode chamar uma variável com o nome “byte”, você terá que usar outro nome.



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

### :: Referência This:

É permitido que uma variável local tenha o mesmo nome de uma variável de instância. Neste caso, utilize a referência `this` para acessar a variável de instância.

Se de dentro do método `variáveis()` quisermos referenciar(chamar) a variável `contador` do escopo da classe, utilizamos a palavra reservada `this`. Exemplo:

```

8      int contador;
9      public void variaveis(){
10         int contador;
11         this.contador = 10;
12     }
13

```

Entendendo: perceba que dentro do método `variáveis` nós continuamos tendo duas variáveis totalmente diferentes. Na linha 10 declaramos uma variável local `contador` (só funciona dentro do método `variáveis()` ).

Na linha 11 estamos utilizando a palavra reservada `this` para acessar a variável de instância `contador` da linha 8.

### :: Falando mais sobre Métodos

Um método é um procedimento ou função que foi criada dentro de uma classe. Um Método é um bloco de instruções que vai gerar uma ação e só pode ser criado dentro de uma classe. Vamos ao exemplo e explicação em cima dele:



```

2 public class Lampada {
3     boolean ligada;
4
5     void ligar(){
6         this.ligada=true;
7     }
8
9     void desligar(){
10        this.ligada= false;
11    }
12
13    void imprimir(){
14        if (this.ligada==true){
15            System.out.println("Lâmpada ligada");
16        }else{
17            System.out.println("Lâmpada desligada");
18        }
19    }
20 }

```

Essa parte do nosso curso é, mais uma vez, muito importante, atenção. Neste exemplo vamos criar uma classe Lâmpada onde teremos dois estados: ligada e desligada definidos por métodos. Lembrando que esta classe não pode ser executada, perceba que não há o método main obrigatório.

Linha 3: variável *ligada* do tipo boolean (valor true ou false). Esta variável irá guardar o estado da lâmpada: se ligada *true*, se desligada *false*;

Linha 5: Neste método (um método representa uma ação) iremos “ligar” a lâmpada atribuindo o valor true a variável *ligada*. Criamos o método *ligar()* - perceba a obrigatoriedade dos parênteses no final do método – a palavra reservada *void* indica que o método não terá valor de retorno . Já dentro do método nós acessamos a variável de classe com a palavra reservada *this* e atribuímos o valor true a ela. Lembrando que essa variável *ligada* poderá ser usada pela classe inteira e quem estiver dentro dela, sendo acessada com o auxílio do *this*;



Para criar um método nós utilizamos o que chamamos de assinatura do método. Onde identificamos se haverá retorno de algum valor com a palavra reservada *void*, damos um nome ao método seguido de parênteses, abrimos e fechamos chaves que é onde o código do método ficará (o escopo).

Linha 9: criamos o método *desligar()* sem valor de retorno (*void*).

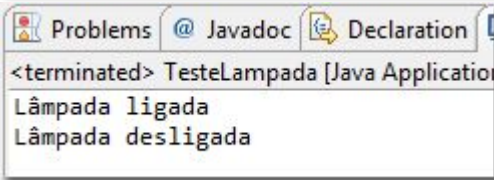
Atribuímos o valor *false* à variável de classe *ligada*;

Linha 13: criamos um método *imprimir()* que irá nos mostrar na tela o estado atual da lâmpada, utilizando o *IF* que já aprendemos anteriormente no curso. Se a variável *ligada* receber *true* aparecerá a mensagem “lâmpada ligada”, se receber *false* aparecerá a mensagem “lâmpada desligada”.

Agora estamos realmente pensando “orientado a objetos”. Criamos uma classe Lâmpada que é especializada em alterar/exibir o estado atual da lâmpada com seus métodos. Poderíamos por exemplo criar uma outra classe especializada em somente verificar a voltagem da lâmpada e colocarmos outros métodos dentro desta classe.

Agora criaremos a classe principal onde teremos nosso método obrigatório *main* e também iremos instanciar um objeto para chamar esses métodos *ligar()*, *desligar()* e *imprimir()* que criamos. Crie uma classe chamada *TesteLampada*.

```
2 public class TesteLampada {
3     public static void main(String[] args) {
4         Lampada lamp = new Lampada();
5
6         lamp.ligar();
7         lamp.imprimir();
8         lamp.desligar();
9         lamp.imprimir();
10    }
11 }
```



The screenshot shows the IDE interface. The top part displays the code for the `TesteLampada` class. Below the code, there is a tab labeled "Problems" and a tab labeled "Javadoc". Below these tabs, there is a text area showing the output of the program: "`<terminated> TesteLampada [Java Application]`", "Lâmpada ligada", and "Lâmpada desligada".

- Linha 3: declaramos o nosso método principal *main*. É ele que o Java irá procurar para executar definitivamente o código;

- Linha 4: importante. Em uma só linha utilizando a palavra reservada **NEW**, criamos (instanciamos) um objeto da classe

Lâmpada e o atribuímos à variável *lamp*, ou seja, *lamp* é uma variável de objeto (instância da classe Lâmpada). Trazendo isso para o mundo real podemos dizer que criamos uma lâmpada, mas podemos criar quantas lâmpadas quisermos como *lamp2*, *lamp3*, etc;

- Linha 6: agora que criamos uma lâmpada (instanciamos um objeto da classe Lâmpada, estou repetindo bastante porque é importante mesmo), vamos utilizar a variável *lamp* para acessar os métodos da classe Lâmpada. Nesta linha 6 nós utilizamos a variável *lamp*, seguida de ponto, para acessar o método *ligar()*, ou seja, nesse momento a variável *ligada* da classe Lâmpada recebeu o valor *true*;

- Linha 7: utilizamos a variável *lamp*, seguida de ponto, para acessar o método *imprimir()*, para verificar o estado atual da variável *lamp* (que no nosso caso está ligada com o valor *true*);

- Linha 8: desligamos a lâmpada, ou seja, chamamos o método *desligar()* onde se atribui *false* à variável *ligada* da classe Lâmpada;

- Linha 9: utilizamos a variável *lamp*, seguida de ponto, para acessar o método *imprimir()*, para verificar o estado atual da variável *lamp* (que no caso agora está desligada, com o valor *false*).

Perceba que nosso programa completo é formado por dois arquivos: **Lampada** e **TesteLampada**.

- Herança: muitas vezes um determinado indivíduo (objeto) herdará características de outros. Por exemplo: um carro Gol e um Golf, ambos são carros diferentes, mas todos tem portas, motor, pneus, bancos... são atributos herdados de outros carros;

- Polimorfismo: um objeto irá se comportar de várias formas diferentes.

## :: Modificadores

Métodos e variáveis de instância são chamados de "membros" e podem utilizar modificadores de acesso, ao contrário de variáveis locais. Porém, se a classe em si não estiver visível para outra

classe, seus membros também não estarão visíveis, independentemente de qualquer modificador utilizado neles. Os modificadores de acesso aplicáveis são:

- **public**: qualquer classe pode acessar
- **private**: não pode ser acessado por nenhuma outra classe.

Campos com esse modificador só podem ser vistos/acessados pelos membros da classe que ele se encontra. Este modificador é responsável por um dos conceitos mais importantes da orientação a objetos, o **Encapsulamento**. Quando você utiliza este modificador você está encapsulando (pode-se entender o termo como “escondendo/ocultando”) o membro da classe: variável ou método;

- **default** (não declarando explicitamente nenhum dos outros modificadores o Java entende que você está utilizando o default): acessível somente por classes do mesmo pacote.

- **protected**: acessível por classes do mesmo pacote ou através de herança. Os membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

Como de costume, exemplo. Anteriormente criamos a classe `TesteLampada` que chamava três métodos da classe `Lampada`: `ligar()`, `desligar()` e `imprimir()`. Caso estes métodos estivessem com o modificador **private**, este acesso não seria possível, pois tais métodos só podem ser acessados/vistos pela sua própria classe `Lâmpada`.

Classe pública.

```
public class Lampada {  
    boolean ligada;  
  
    private void ligar(){  
        this.ligada=true;  
    }  
  
    private void desligar(){  
        this.ligada= false;  
    }  
  
    private void imprimir(){  
        if (this.ligada==true){  
            System.out.println("Lâmpada ligada");  
        }else{  
            System.out.println("Lâmpada desligada");  
        }  
    }  
}
```

- Os três métodos da classe Lâmpada encontram-se com o modificador **private**, com isso somente métodos desta classe podem acessá-los, outras classes não poderão fazê-lo. Isto é o **Encapsulamento** trabalhando, Orientação a Objeto na veia. Se estes métodos estivessem declarados com **public** por exemplo, qualquer classe estaria liberada para acessá-los.



*Apenas uma classe (não estou me referindo a métodos) definida como public pode existir em um arquivo Java e o nome deste arquivo deve ser o mesmo da classe definida como public. No exemplo acima temos a classe Lâmpada (pública) dentro do arquivo Lâmpada.java, ou seja, para um arquivo Lâmpada.java só pode existir uma e somente uma classe pública.*

Teste o código acima e perceba o erro que irá acontecer na classe TesteLampada. Os métodos não poderão mais ser acessados devido a utilização do modificador private.

```

1
2 public class TesteLampada {
3     public static void main(String[] args) {
4         Lampada lamp = new Lampada();
5
6         lamp.ligar();
7
8         lamp.desligar();
9
10        lamp.imprimir();
11    }
12 }
13
14

```

Perceba que nenhum dos métodos pôde ser acessado justamente por estarem encapsulados pelo modificador `private` na classe `Lampada`. Se você voltar na classe `Lampada` e retirar esses modificadores `private`, tudo voltará ao normal e os métodos poderão agora ser acessados. Faça esse teste para visualizar o funcionamento.

### :: Métodos get e set

Como visto anteriormente, o **Encapsulamento** "protege/oculta" os atributos ou métodos dentro de uma classe, portanto devemos prover meios para acessar tais membros quando eles são privados, ou seja, quando possuem o modificador *private*. O que torna isso possível é a criação de métodos.

Em programação orientada a objetos, esses métodos são chamados de métodos **getters** e **setters**, pois eles provêem acesso aos **atributos** da classe declarados como *private* e, geralmente, se iniciam com `get` ou `set`.

Na verdade, não há nada de diferente entre os métodos comuns e os métodos assessores (como são chamados) `getters` e `setters`. A única importância está no fato da orientação a objeto. Pois, sempre que formos acessar um membro privado em Java usaremos `get` ou `set`, é uma convenção muito utilizada. Uma das vantagens é que os `getters` e `setters` protegem os atributos de algum acesso indevido de uma outra classe.

Antes de mais nada, é uma má prática de programação sair criando `getters` e `setters` para todos os atributos de uma classe que estão com o modificador `private`. Só os utilize quando realmente

necessários. É uma boa prática de programação declarar como privados os atributos quando necessário e criar seus métodos setters e getters para acessá-los de fora da classe.

#### - Set

Nomeamos um método com `set` toda vez que este método for alterar algum campo (atributo) de uma classe, ou seja, se não criarmos um método assessor `set` para algum atributo, isso quer dizer que este atributo não deve ser alterado.

Portanto, como o valor de um atributo da classe será modificado não é necessário que este método retorne nenhum valor, por isso, os métodos setters são `void`. Porém, obrigatoriamente, eles têm que receber um argumento que será o novo valor do campo. Calma J.

#### - Get (que também pode ser chamado de “is”)

Nomeamos um método com `get` toda vez que este método for verificar algum campo ou atributo de uma classe. Como este método irá verificar um valor, ele sempre terá um retorno como `String`, `int`, `float`, etc. Mas não terá nenhum argumento.

Já sei, já sei ... EXEMPLO: continuando com nossa classe `Lampada`.

```
2 public class Lampada {  
3     private boolean ligada;  
4  
5     public void setLigar(boolean lig){  
6         this.ligada=lig;  
7     }  
8  
9     public void setDesligar(boolean deslig){  
10        this.ligada= deslig;  
11    }  
12  
13    public boolean getImprimir(){  
14        return this.ligada;  
15    }  
}
```

- Linha 2: a classe pública obrigatória e com o mesmo nome do arquivo Java;

- Linha 3: declaração da variável `ligada`, do tipo `boolean` (verdadeiro ou falso) e com o modificador `private`, ou seja, somente pode ser acessada dentro da classe `Lâmpada`;



- Linha 5: método **setLigar** para iniciar um valor na variável de classe *ligada*. Considerações: perceba que o método set irá receber um valor (parâmetro), por isso a declaração da variável *lig* dentro dos ( ). Outra coisa a se notar é que como o método set terá que receber um valor, então o seu tipo é declarado como **void**;

- Linha 6: a variável de classe ligada (lembra do this ?) recebe o valor passado como parâmetro na variável *lig*;

- Linha 9: semelhante ao método setLigar. O método **setDesligar** recebe um valor para a variável deslig e por isso também vai ser declarado como **void**;

- Linha 10: a variável de classe ligada (o this novamente) recebe o valor passado como parâmetro na variável *deslig*;



Note as variáveis *lig* e *deslig*, elas estão declaradas dentro dos ( ) dos métodos setters. Elas são chamadas **variáveis de método**, pois só podem ser utilizadas dentro deles, dos seus respectivos métodos.

- Linha 13: método **getImprimir** (ou **isImprimir**, também é aceito normalmente) que irá retornar o valor atual da variável de classe *ligada*. Perceba que este método é diferente, os ( ) estão vazios, ou seja, ele não recebe nenhum valor, não recebe nenhum parâmetro, logo, ele não deve ser declarado como **void**, deve ser declarado com o tipo de dado da variável privada que estamos tratando, no nosso caso **boolean**, linha 3;

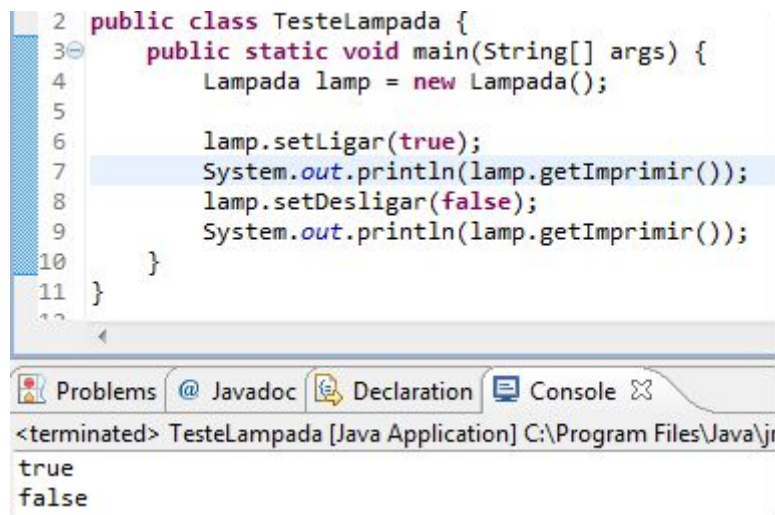
- Linha 14: retorna para o Java o valor atual da variável de classe *ligada*. Note a utilização da palavra reservada **return** (atenção, isso é muito importante);

Estamos realmente pensando Orientado a Objetos. Criamos uma classe *Lampada* especializada em acender e apagar a nossa lâmpada, encapsulamos (*private*) nossa variável de classe *ligada* e utilizamos métodos **getters** e **setters** para podermos acessá-la

posteriormente. Agora precisamos de uma classe com o método `main` obrigatório para podermos utilizar estes métodos da classe `Lâmpada` fora dela.

As coisas começam a se encaixar. Continuando:

```
2 public class TesteLampada {
3     public static void main(String[] args) {
4         Lampada lamp = new Lampada();
5
6         lamp.setLigar(true);
7         System.out.println(lamp.getImprimir());
8         lamp.setDesligar(false);
9         System.out.println(lamp.getImprimir());
10    }
11 }
```

The screenshot shows the Eclipse IDE with a Java file named 'TesteLampada.java'. The code is as follows:

```
2 public class TesteLampada {
3     public static void main(String[] args) {
4         Lampada lamp = new Lampada();
5
6         lamp.setLigar(true);
7         System.out.println(lamp.getImprimir());
8         lamp.setDesligar(false);
9         System.out.println(lamp.getImprimir());
10    }
11 }
```

The IDE's interface includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, showing the output of the program: 

```
<terminated> TesteLampada [Java Application] C:\Program Files\Java\jre
true
false
```

- Linha 2: criamos a classe pública `TesteLampada` que será responsável por acessar a classe `Lâmpada` e seus métodos assessores **setters** e **getters**;
- Linha 3: método `main` obrigatório para executar a classe;
- Linha 4: instanciamos um objeto (com a palavra reservada `new`) da classe `Lâmpada` e o atribuímos à variável `lamp`;
- Linha 6: com o auxílio da variável `lamp` acessamos o método `setLigar` da classe `Lâmpada` e passamos como parâmetro o valor **true** dentro dos parênteses;
- Linha 7: através do método `getImprimir` (ou `isImprimir`) exibimos o estado atual da `Lâmpada`, que no caso é verdadeiro/ligada;
- Linhas 8 e 9: é a mesma coisa, só que agora passamos o valor **false** como parâmetro para o método `setDesligar`, a lâmpada está desligada. Teste o código acima.



**Dica:** assim que você declarar uma variável com o modificador *private*, o Eclipse já sabe que devem ser criados os `setters` e `getters` correspondentes, então no Eclipse temos atalho para fazer isso mais rapidamente. Logo depois de declarada a variável com o modificador *private* tecle `CTRL + espaço` (veja figura) e você verá uma lista onde

as últimas opções são os métodos set e get (is) já prontinhos, basta clicar e o Eclipse faz o resto.

