



Programação Orientada a Objetos

Unidade 0 – Introdução à Linguagem Java com a IDE ECLIPSE



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
CPRE – CTFL – CTFL-AT
aparecidovfreitas@gmail.com



Programação Orientada a objetos

- Um programa OOP é um conjunto de **objetos** que se interagem (por meio de troca de mensagens) para a solução do problema!
- **Ações** em OOP sempre são executadas por agentes chamados de instâncias ou **objetos**.



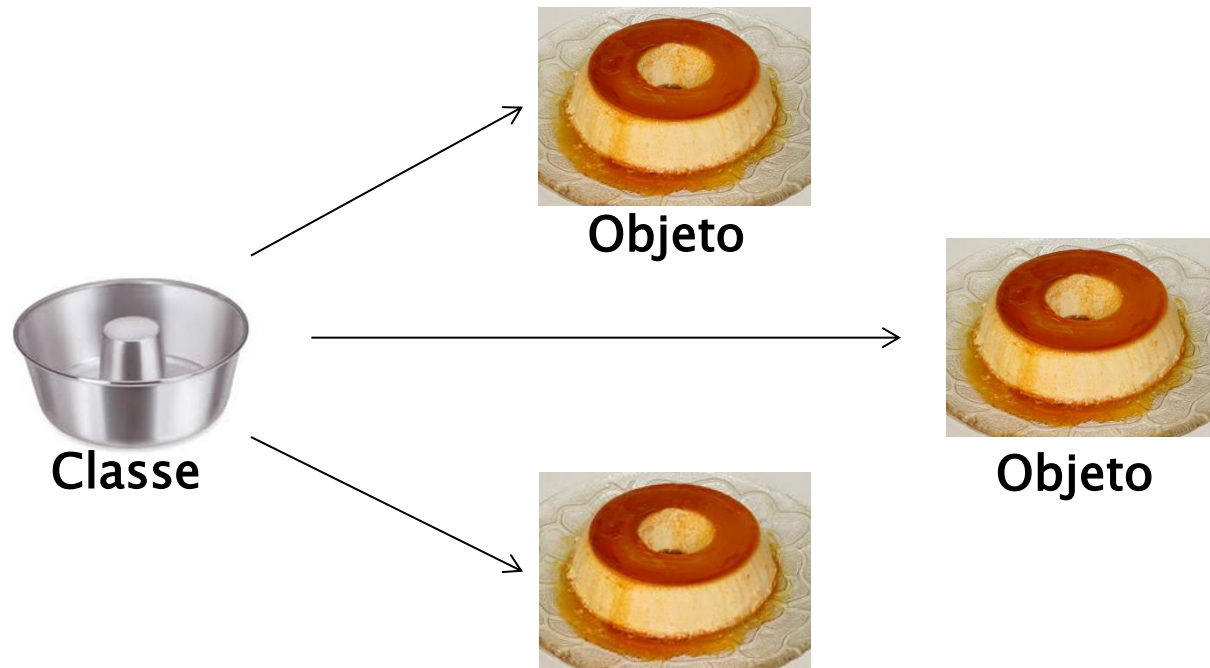
Como se cria um objeto num programa Java ?



Por meio de classes...

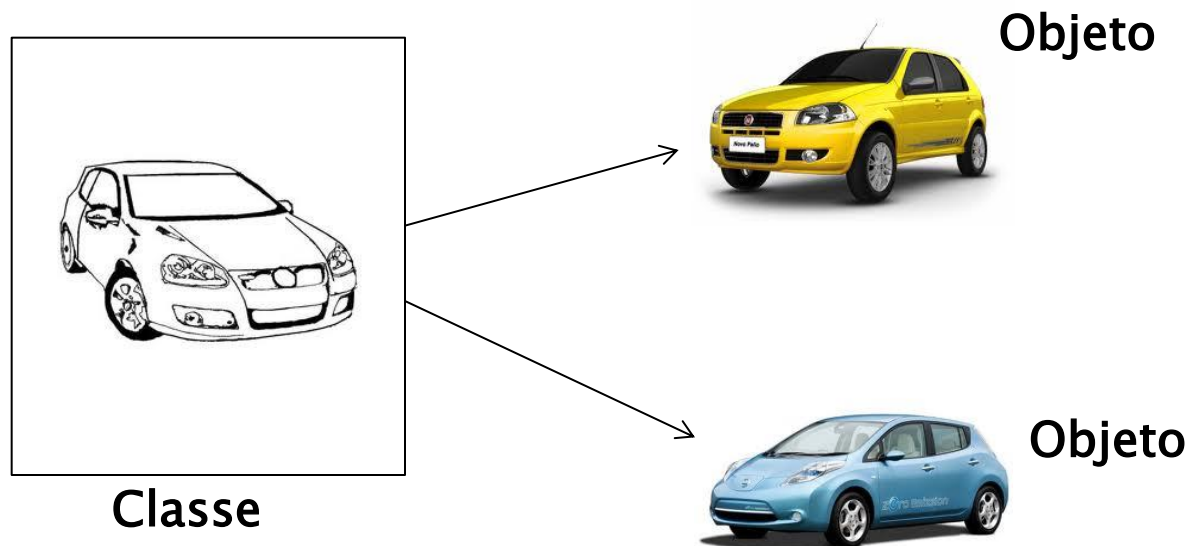


Uma classe é um modelo, uma especificação, um molde, a partir do qual criam-se objetos...



Classes

- ▣ Modelam objetos.
- ▣ Definem os atributos (propriedades) dos objetos e as suas funcionalidades.



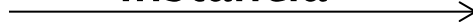
Instâncias

- ◆ Um objeto é uma instância de uma classe;
- ◆ Classes definem atributos (**dados**) do objeto e funcionalidades (**comportamentos**)



Classe

Instância



Objeto



Exemplo



**Atributos ou
Propriedades ou
Dados**

```
String Nome;  
int CodigoMatricula;
```

```
public Aluno (String n, int c) {  
    Nome = n;  
    CodigoMatricula = c;  
}  
  
public void Imprime() {  
    System.out.println("Nome = " + Nome);  
    System.out.println("Matricula = " + CodigoMatricula);  
}
```



Exemplo

Operações
ou Métodos
ou Funções

```
class Aluno {
```

```
String Nome;
```

```
int CodigoMatricula;
```

```
public Aluno (String n, int c) {
```

```
    Nome = n;
```

```
    CodigoMatricula = c;
```

```
}
```

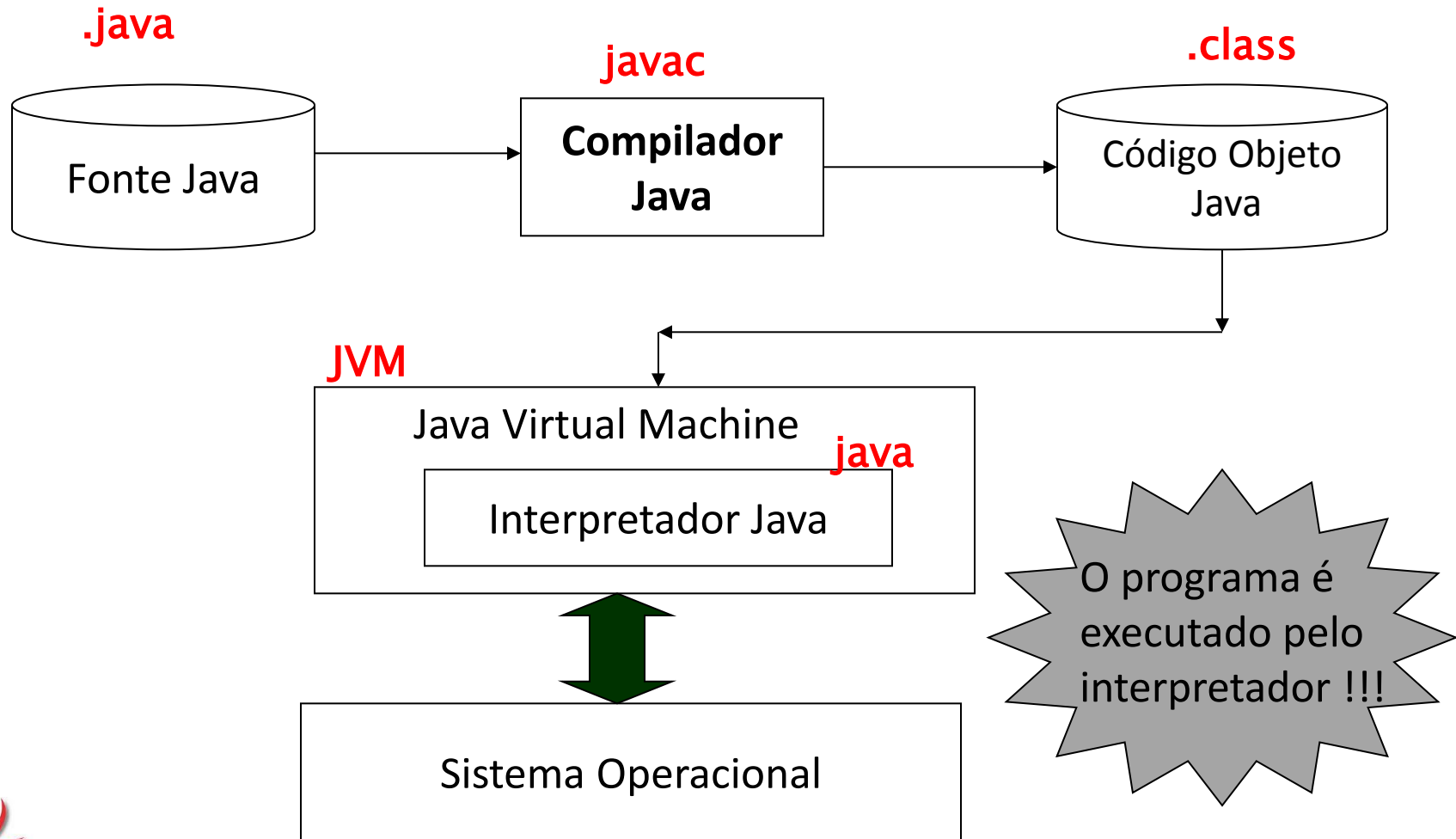
```
public void Imprime () {
```

```
    System.out.println("Nome = " + Nome);
```

```
    System.out.println("Matricula = " + CodigoMatricula);
```

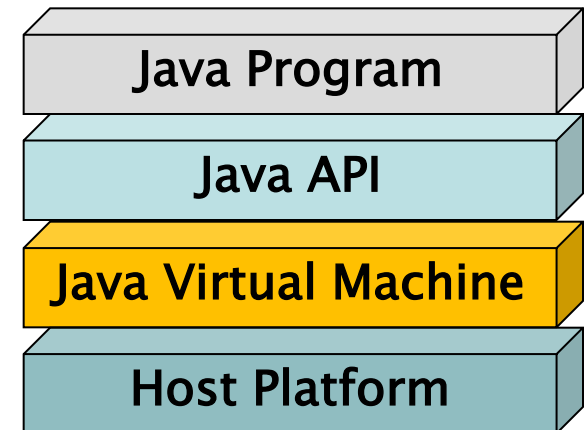


Ambiente Java



Plataforma Java

- Java além de linguagem é uma PLATAFORMA.
- Esta plataforma provê:
 - ❖ JVM – Java Virtual Machine
 - ❖ API – Application Program Interface



Um pequeno programa Java

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World...");  
    }  
}
```

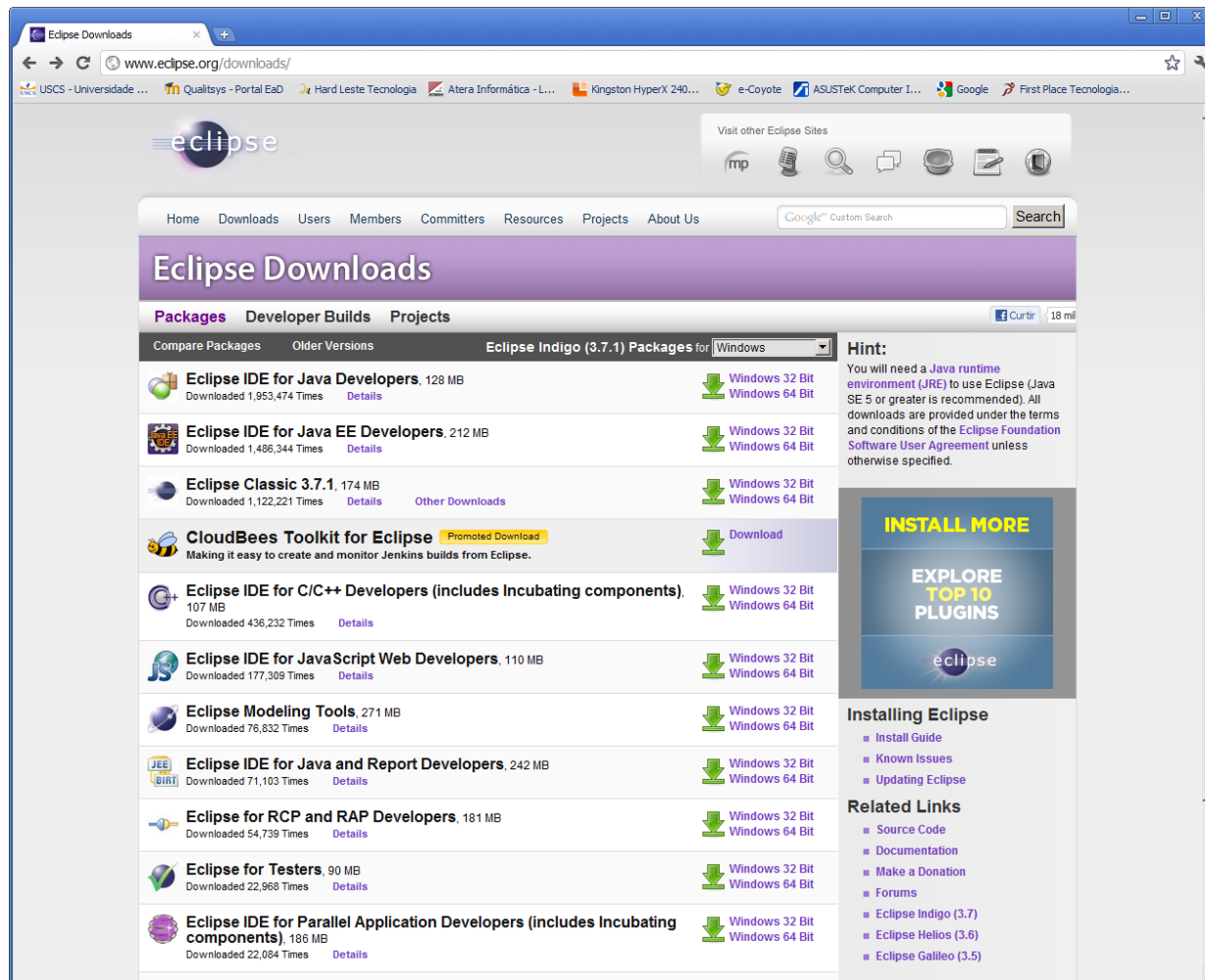


Desenvolvimento com IDE

- ❖ A IDE (Integrated Development Environment) mais empregada é **ECLIPSE**.
- ❖ **ECLIPSE** pode ser baixado a partir do endereço <http://www.eclipse.org>.
- ❖ **ECLIPSE** é escrito em Java.



Instalando Eclipse



The screenshot shows the Eclipse Downloads website in a web browser. The page has a navigation bar with links: Home, Downloads, Users, Members, Committers, Resources, Projects, and About Us. Below the navigation bar is a search bar and a "Visit other Eclipse Sites" section. The main content area is titled "Eclipse Downloads" and features a list of packages for Eclipse Indigo (3.7.1) on Windows. The packages include:

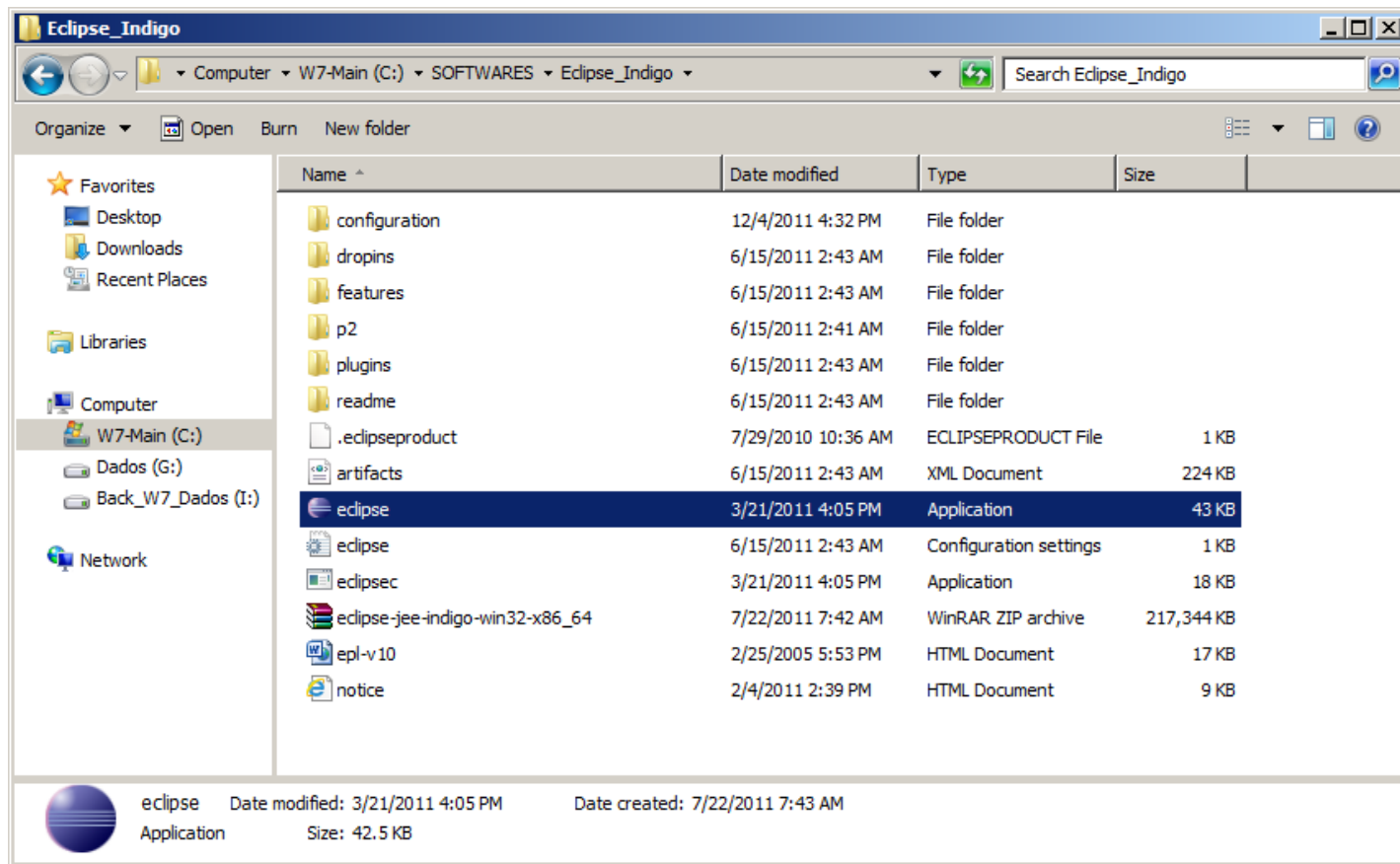
- Eclipse IDE for Java Developers, 128 MB
- Eclipse IDE for Java EE Developers, 212 MB
- Eclipse Classic 3.7.1, 174 MB
- CloudBees Toolkit for Eclipse (Promoted Download)
- Eclipse IDE for C/C++ Developers (includes Incubating components), 107 MB
- Eclipse IDE for JavaScript Web Developers, 110 MB
- Eclipse Modeling Tools, 271 MB
- Eclipse IDE for Java and Report Developers, 242 MB
- Eclipse for RCP and RAP Developers, 181 MB
- Eclipse for Testers, 90 MB
- Eclipse IDE for Parallel Application Developers (includes Incubating components), 186 MB

Each package entry shows the number of times it has been downloaded and provides a "Details" link. To the right of the package list is a "Hint" section and a "Related Links" section. The "Hint" section states: "You will need a Java runtime environment (JRE) to use Eclipse (Java SE 5 or greater is recommended). All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified." The "Related Links" section includes links to Source Code, Documentation, Make a Donation, Forums, Eclipse Indigo (3.7), Eclipse Helios (3.6), and Eclipse Galileo (3.5).



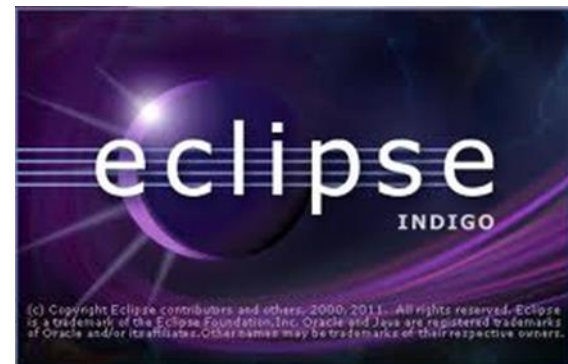
Executando o ECLIPSE

- ◆ Siga para o diretório de instalação do eclipse
- ◆ Execute o arquivo eclipse.exe conforme figura abaixo.



Workspace

- ◆ O Eclipse exige que seja informado um diretório que corresponde ao **Workspace**.
- ◆ Um **Workspace** é a área onde são armazenados os seus projetos do Eclipse.



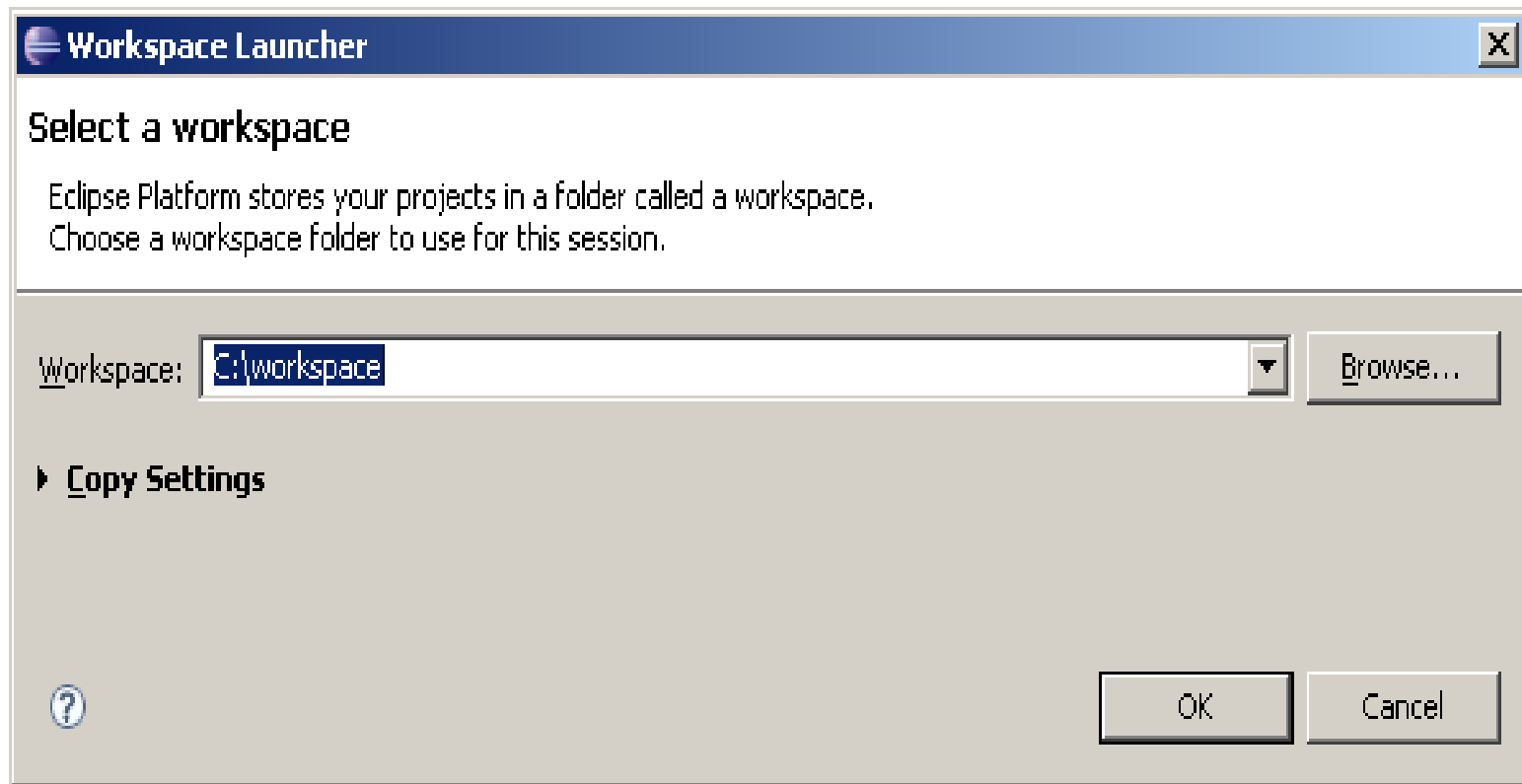
- ✦ Se for a primeira vez que o **workspace** escolhido for acessado, uma tela como a mostrada abaixo será exibida.
- ✦ Pode ser acessada pela função: **Help > Welcome**.



Pode fechá-la clicando no “X” na aba ao lado da palavra “Welcome”.



Escolha um diretório conforme a figura abaixo e clique em “Ok”.



O que é um Workspace ?

- ◆ Coleção de recursos que você usa para criar aplicações.

✓ Projects, folders, files.

- ◆ Solicitado quando o Eclipse é carregado.



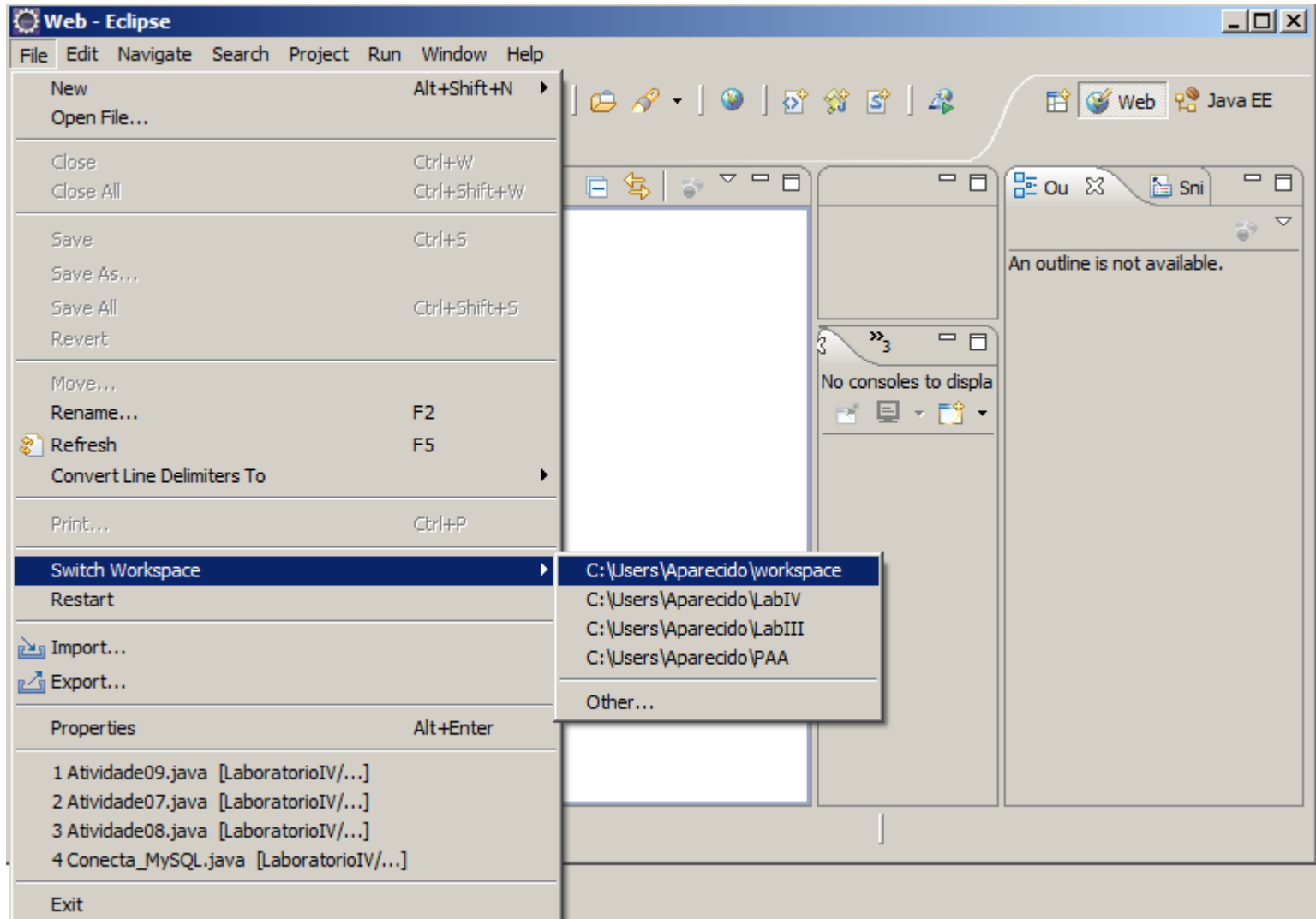
- ◆ Pode ser aberta por um único usuário de cada vez.



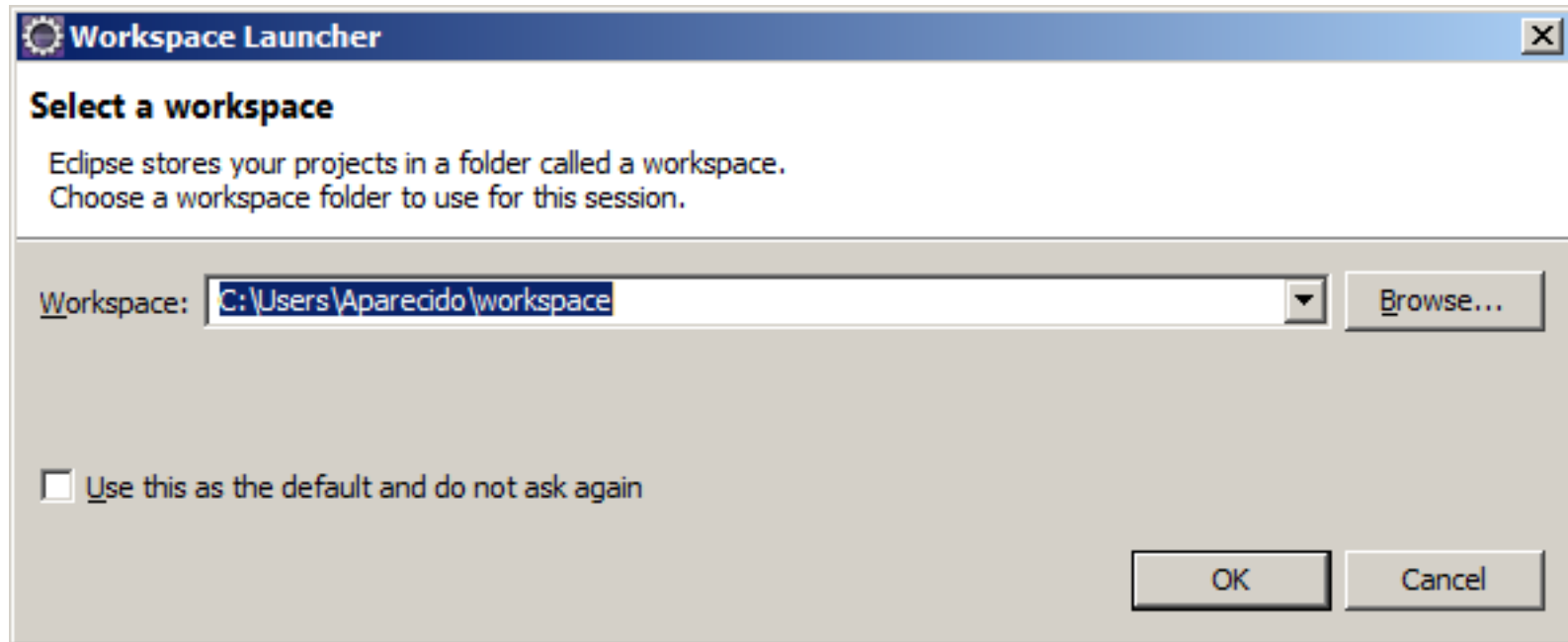
Exibindo a localização do Workspace

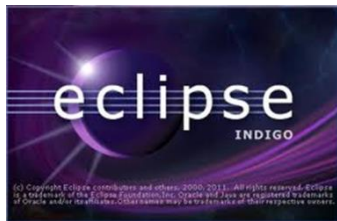


File > Switch Workspace



Exibindo a localização do Workspace



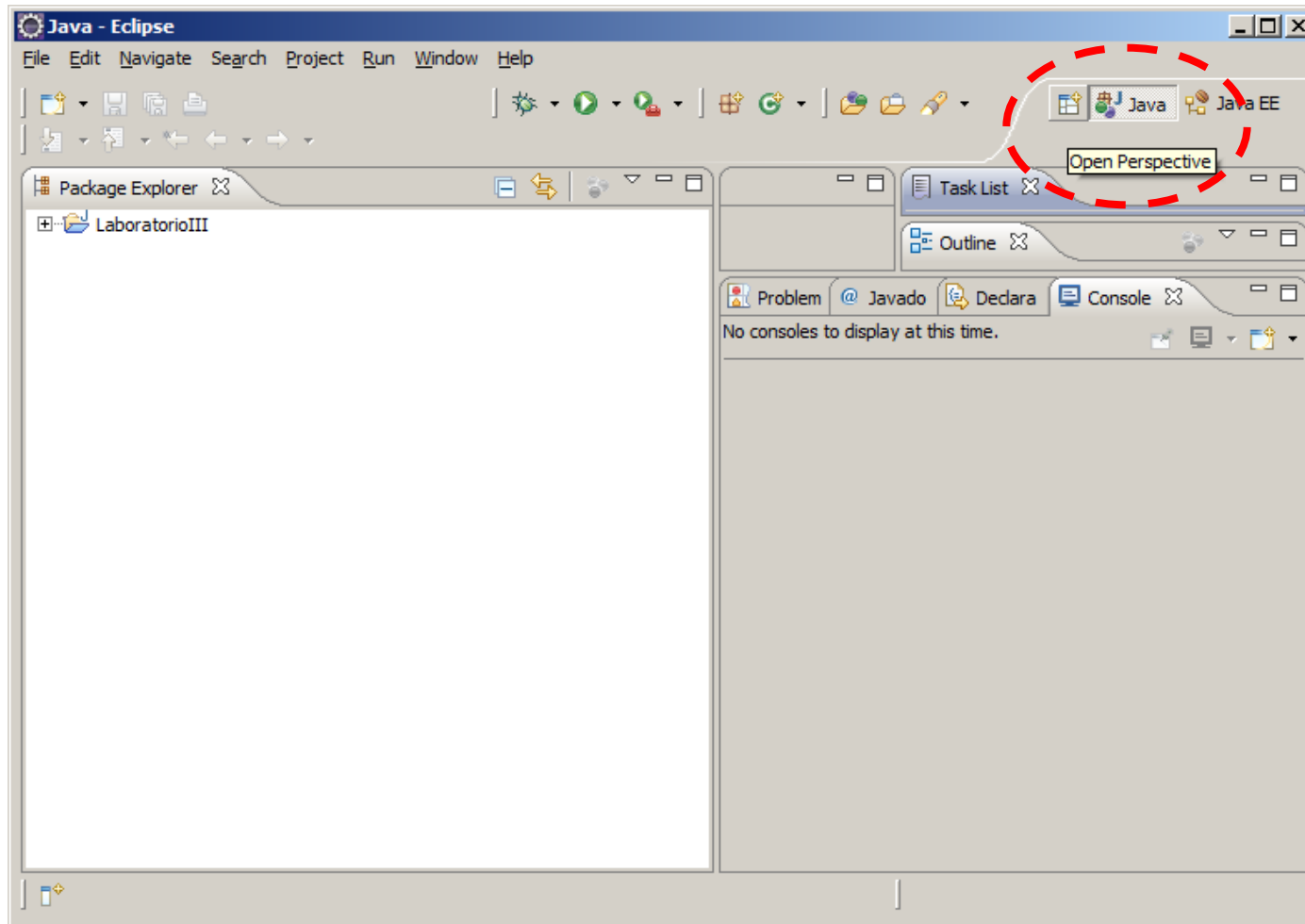


Perspective

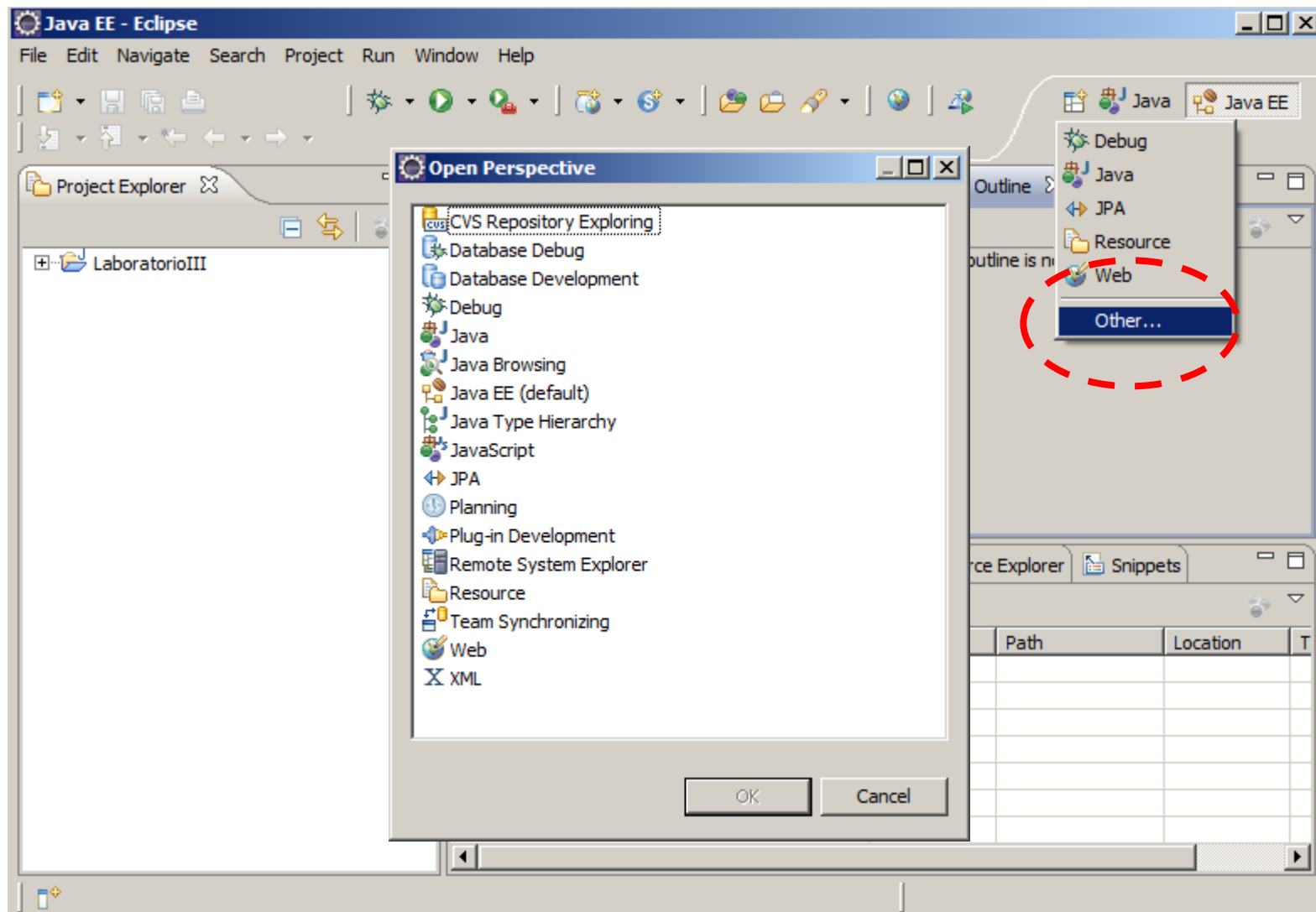
- ◆ Para ver algo “particular” em Eclipse, você deve primeiro abrir uma **perspective**.
- ◆ Uma **perspective** consiste de uma combinação de janelas e ferramentas que são adequadas a tarefas específicas.
- ◆ Para iniciar o desenvolvimento de programas Java, você deve primeiro abrir uma **perspective Java**.



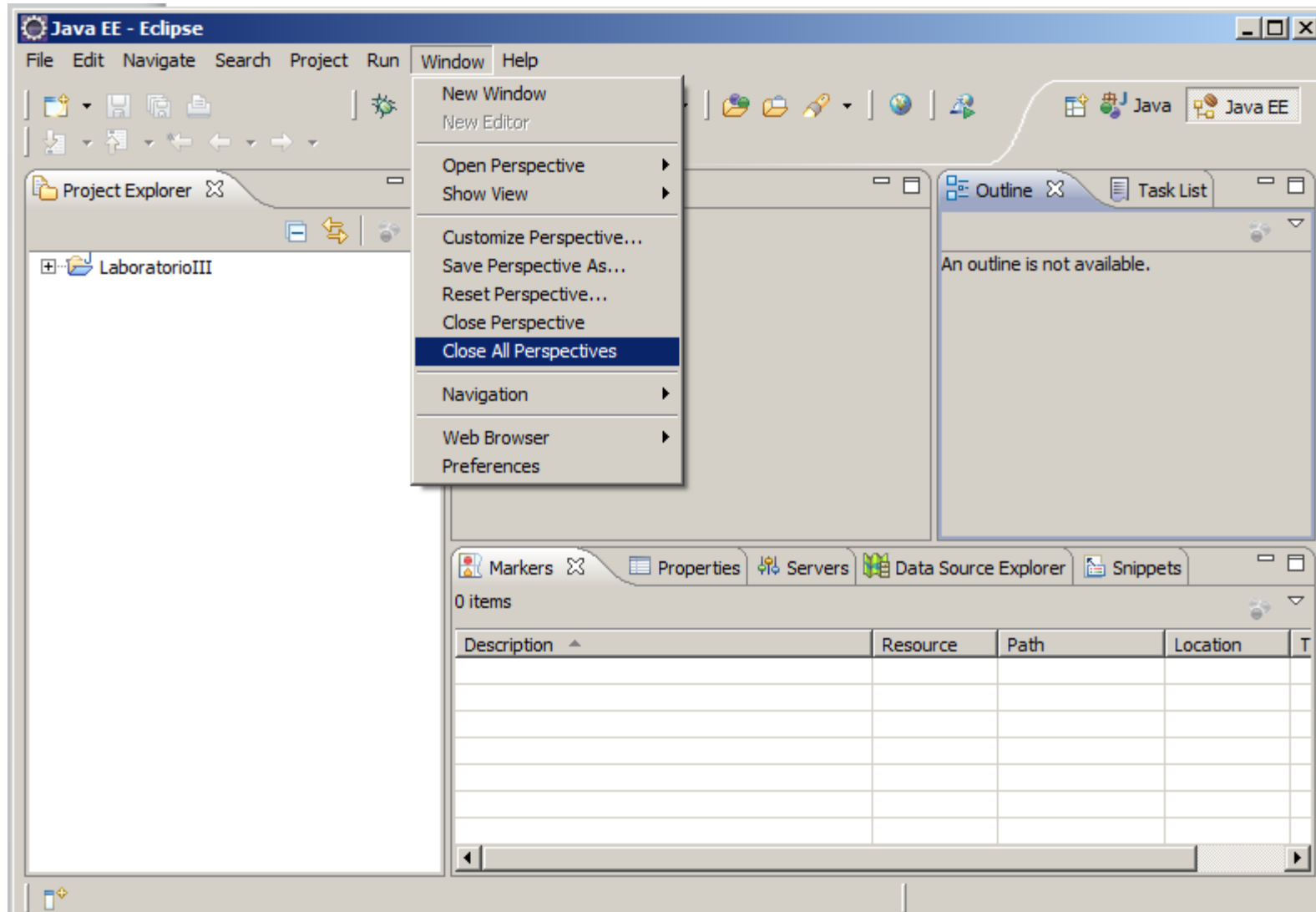
Perspectiva Java



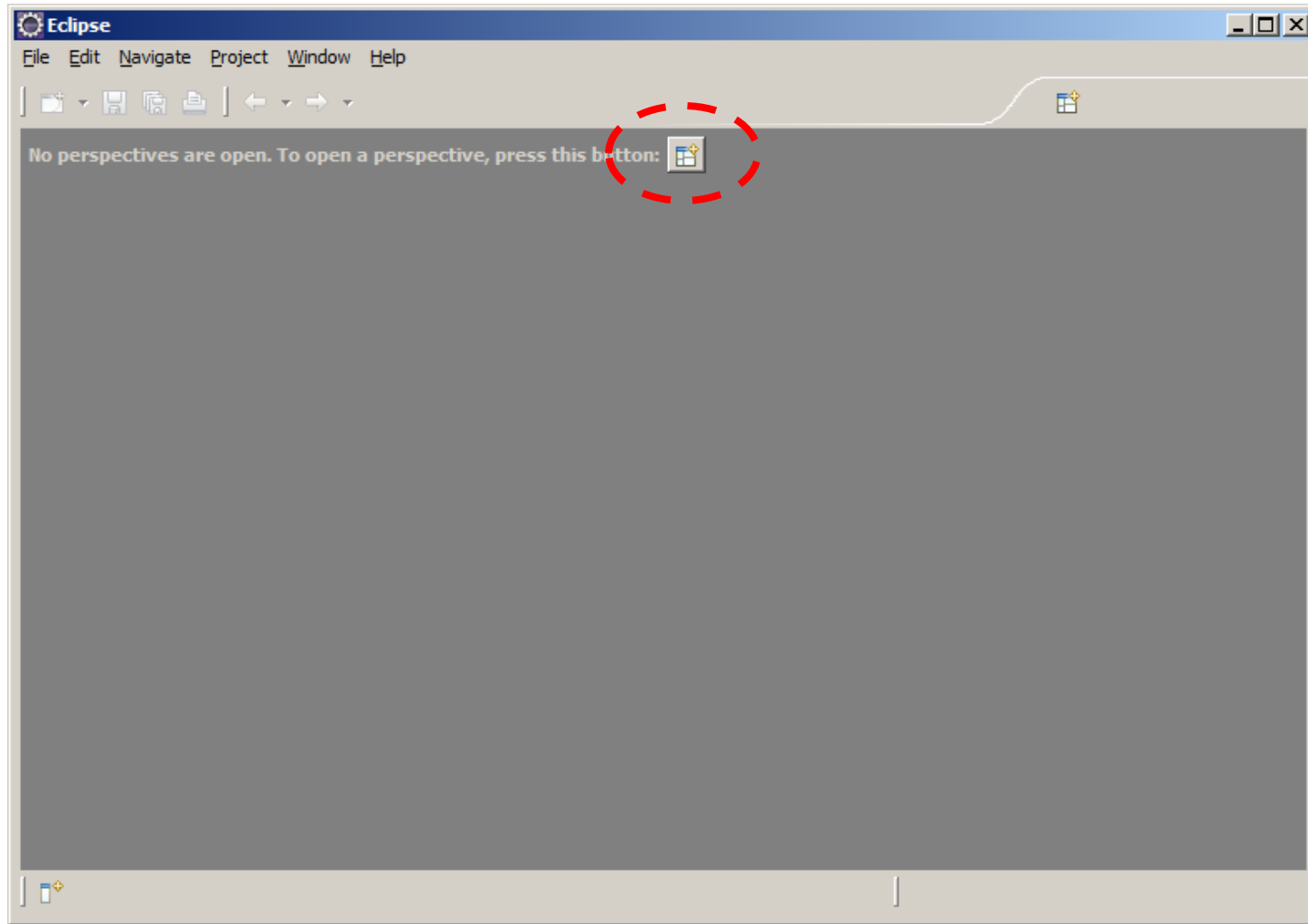
Perspectives



Fechando Perspective

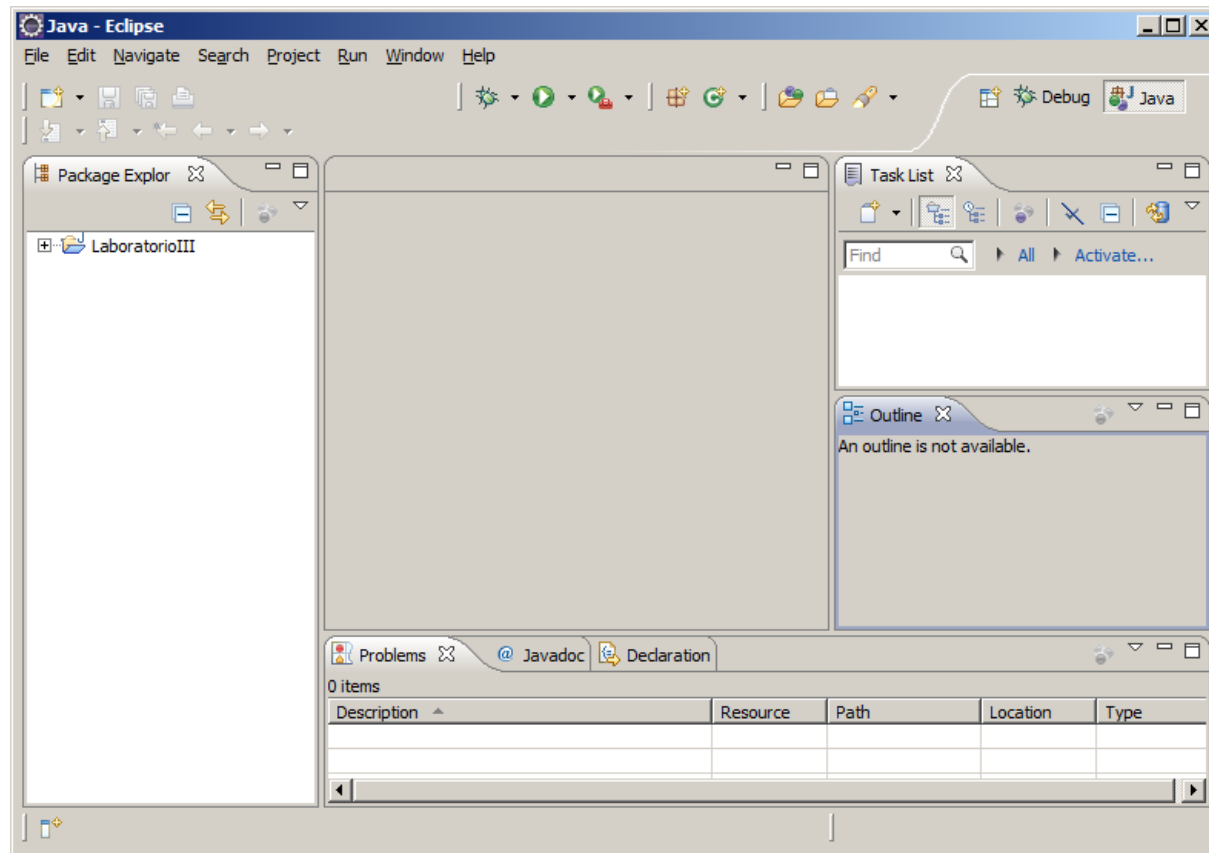


Fechando Perspective



Perspective Java

- Uma perspectiva Java exibe windows (Package Explorer, Hierarchy), menu items, e toolbar icons que são típicos para desenvolvimento Java.

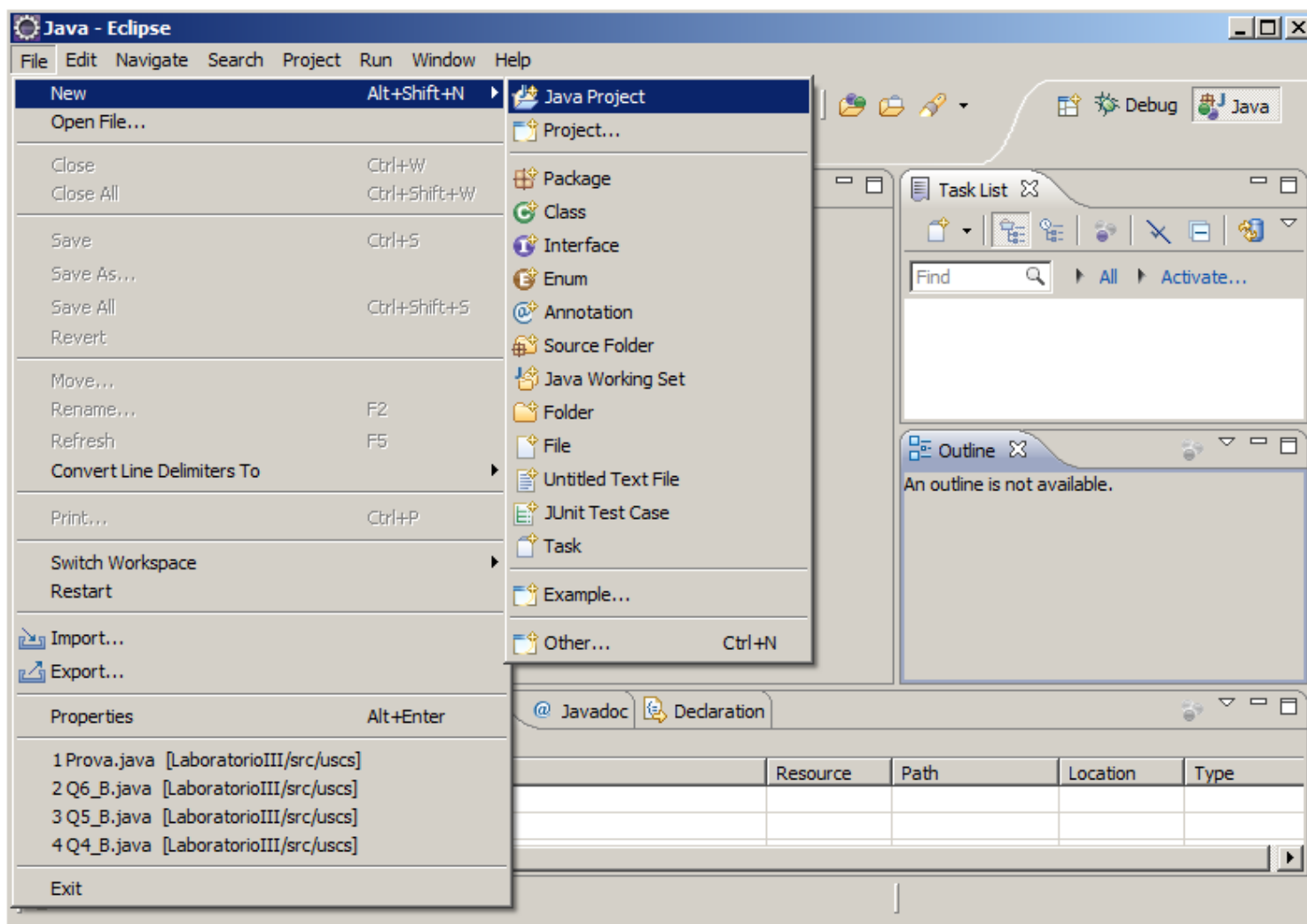


Criação de Projetos

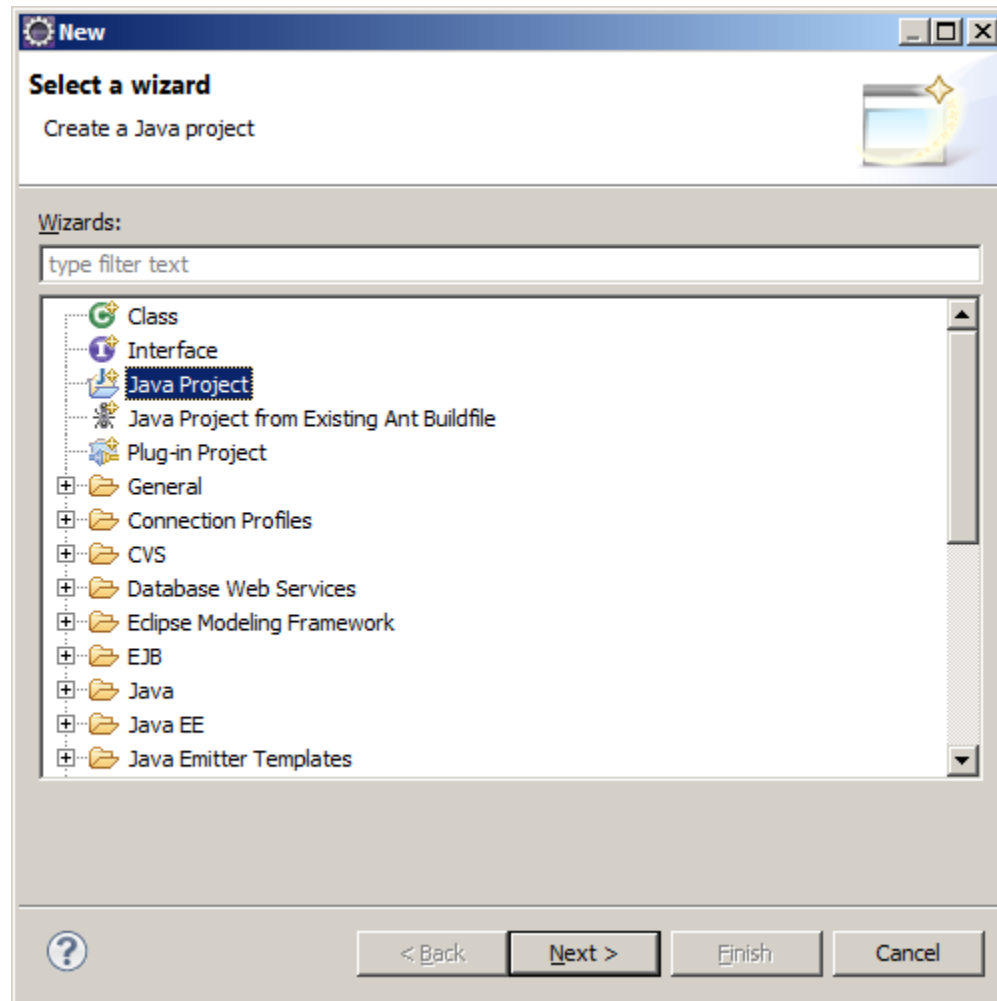
- ◆ Todos recursos são armazenados em projetos.
- ◆ Você pode criar projetos com diferentes estruturas, de acordo com a área de desenvolvimento, Java, Web, Enterprise JavaBeans (EJB), portal, outros.
- ◆ Você pode deletar projetos do workbench ou do workspace.
- ◆ O default é deletar somente do workbench.



Criação de um Projeto Java



Criação de um Projeto Java



Tecele Next



Criação de um Projeto Java

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: **Projeto_01**

☒ Use default location

Location: C:\Users\Aparecido\LabIII\Projeto_01 [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-1.6

☐ Use a project specific JRE: jre6 [Configure JREs...](#)

☐ Use default JRE (currently 'jre6')

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets

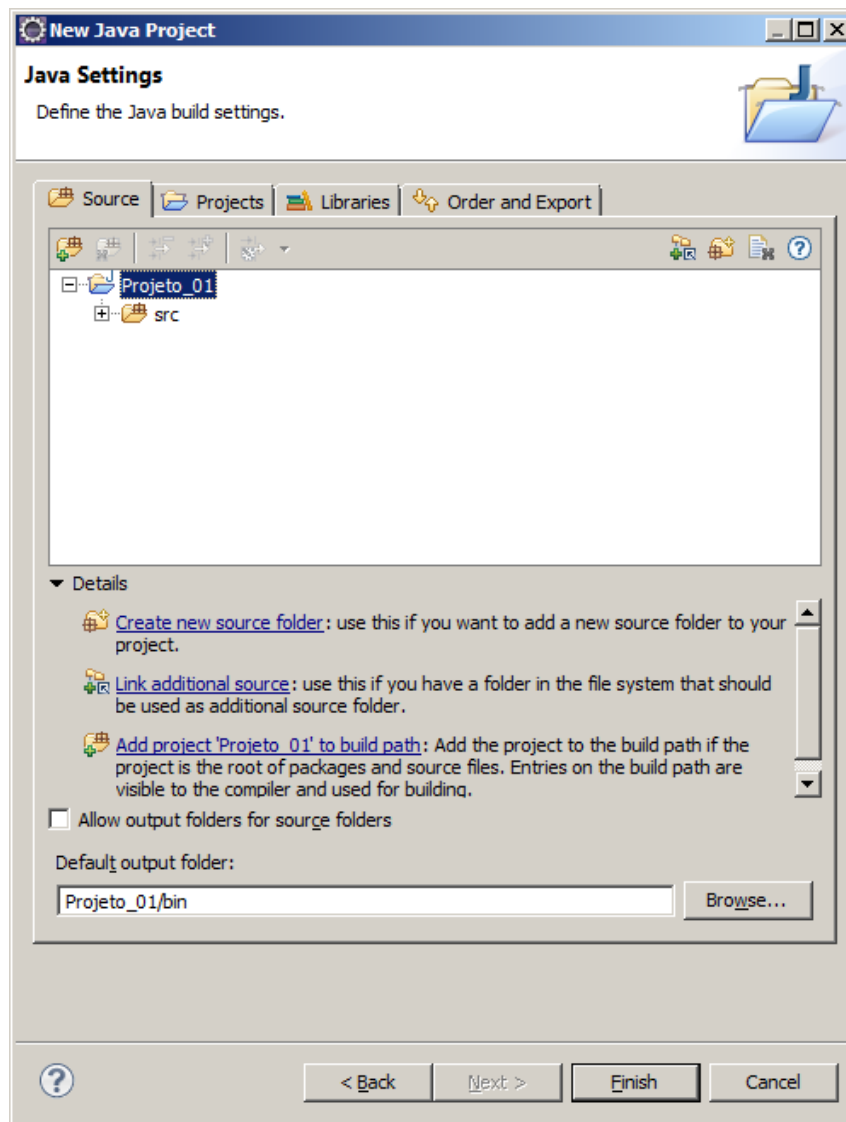
Working sets: [Select...](#)

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Defina **Projeto_01** como nome do projeto e tecle **Next**



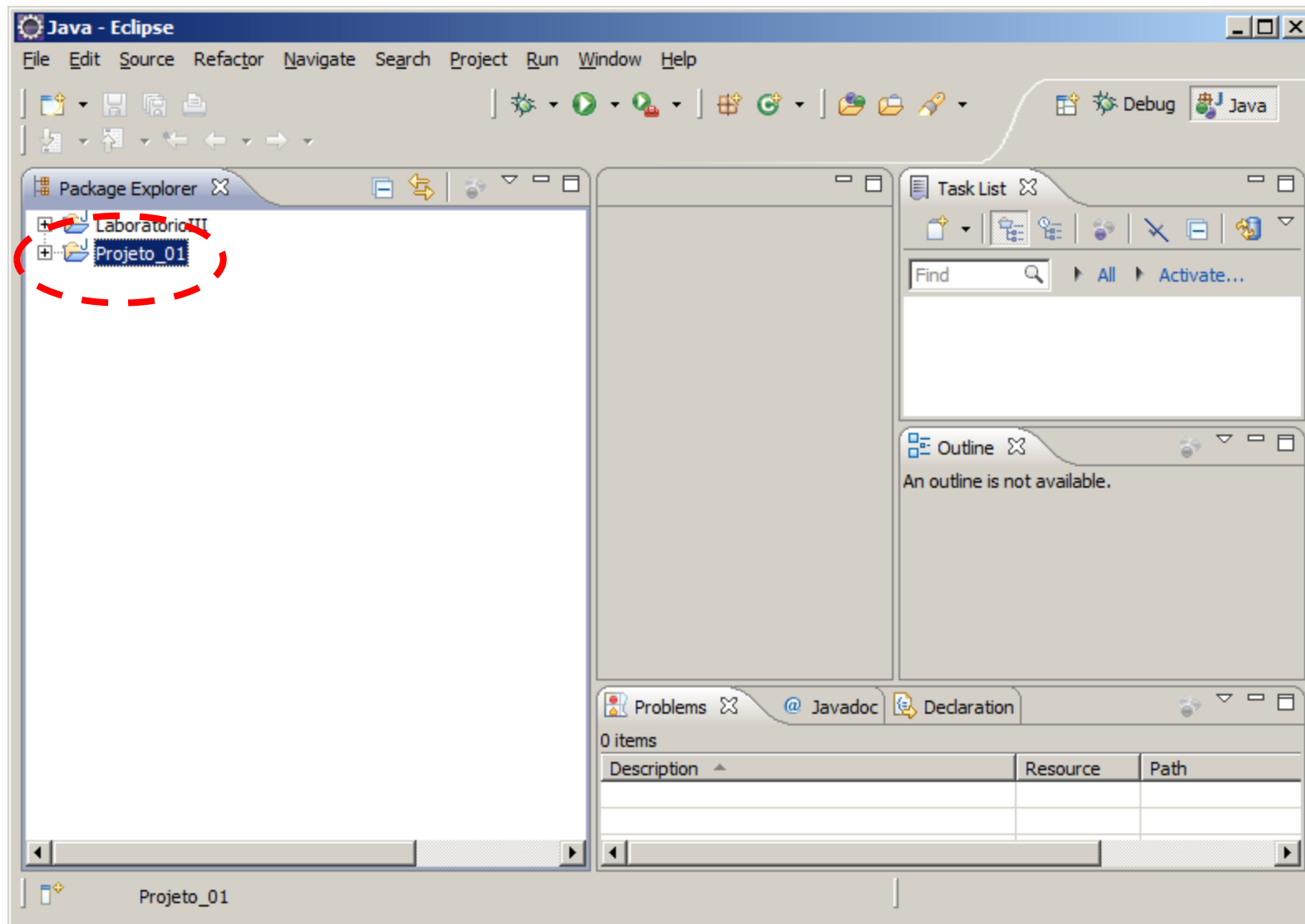
Criação de um Projeto Java



Tecele Finish

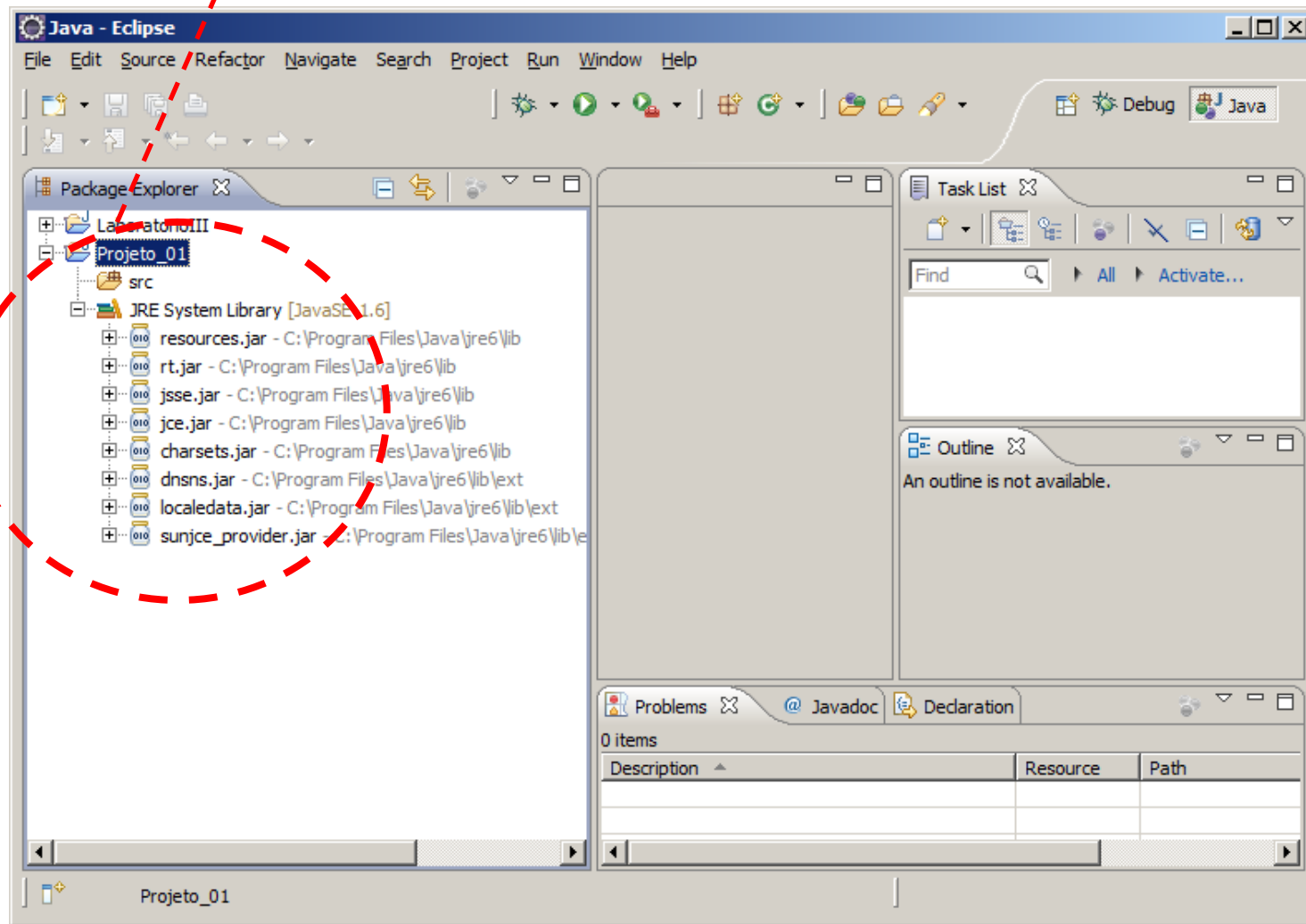


Criação de um Projeto Java

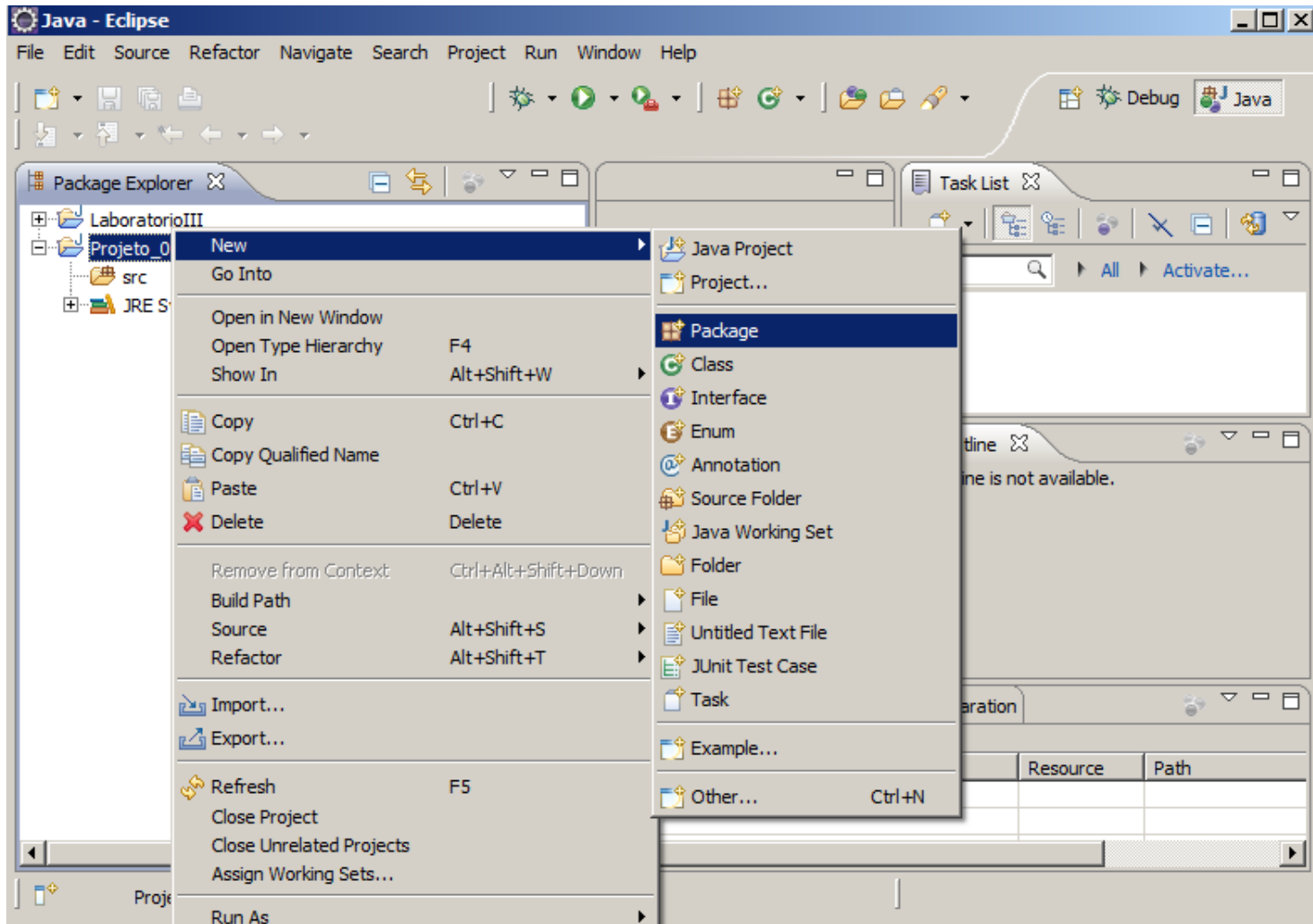


Criação de um Projeto Java

Package Explorer exibe o conteúdo do novo projeto, incluindo as bibliotecas do JRE.



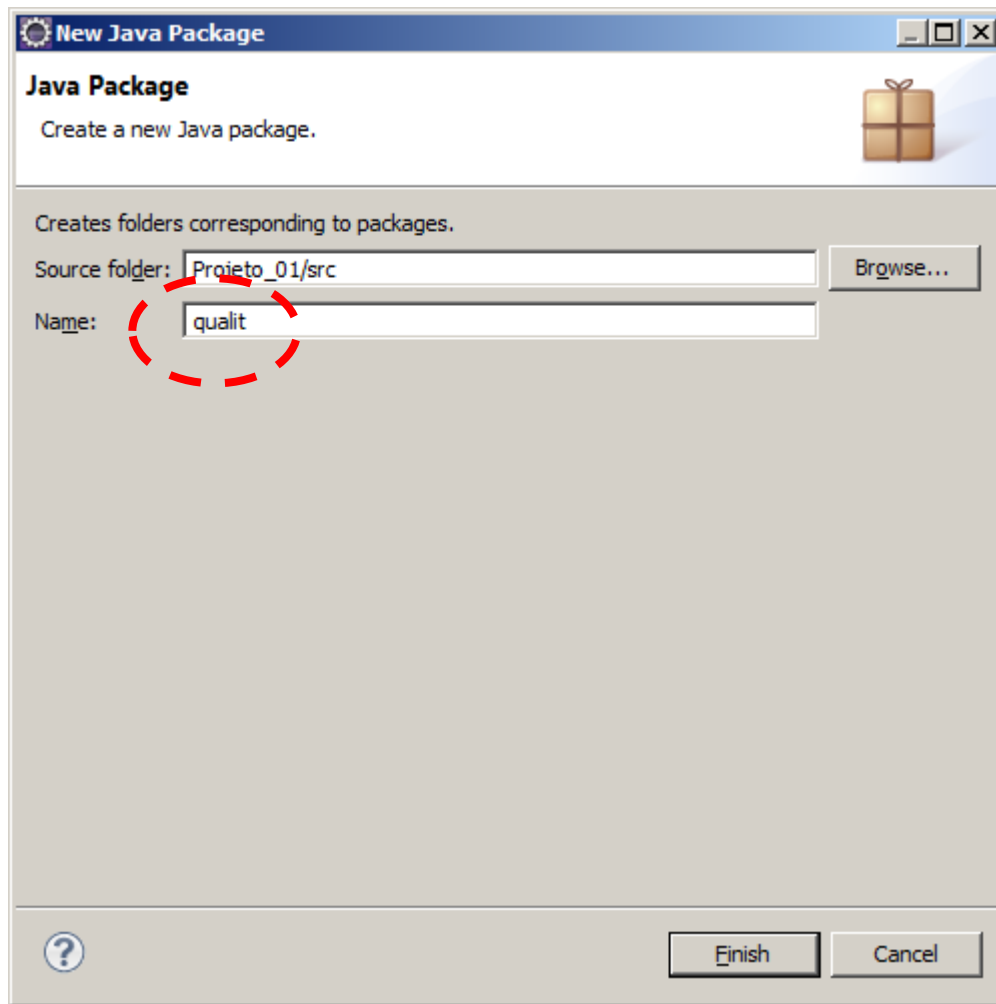
Criação de um package



Com o botão direito do mouse sob a pasta do projeto, clique em **New > Package**



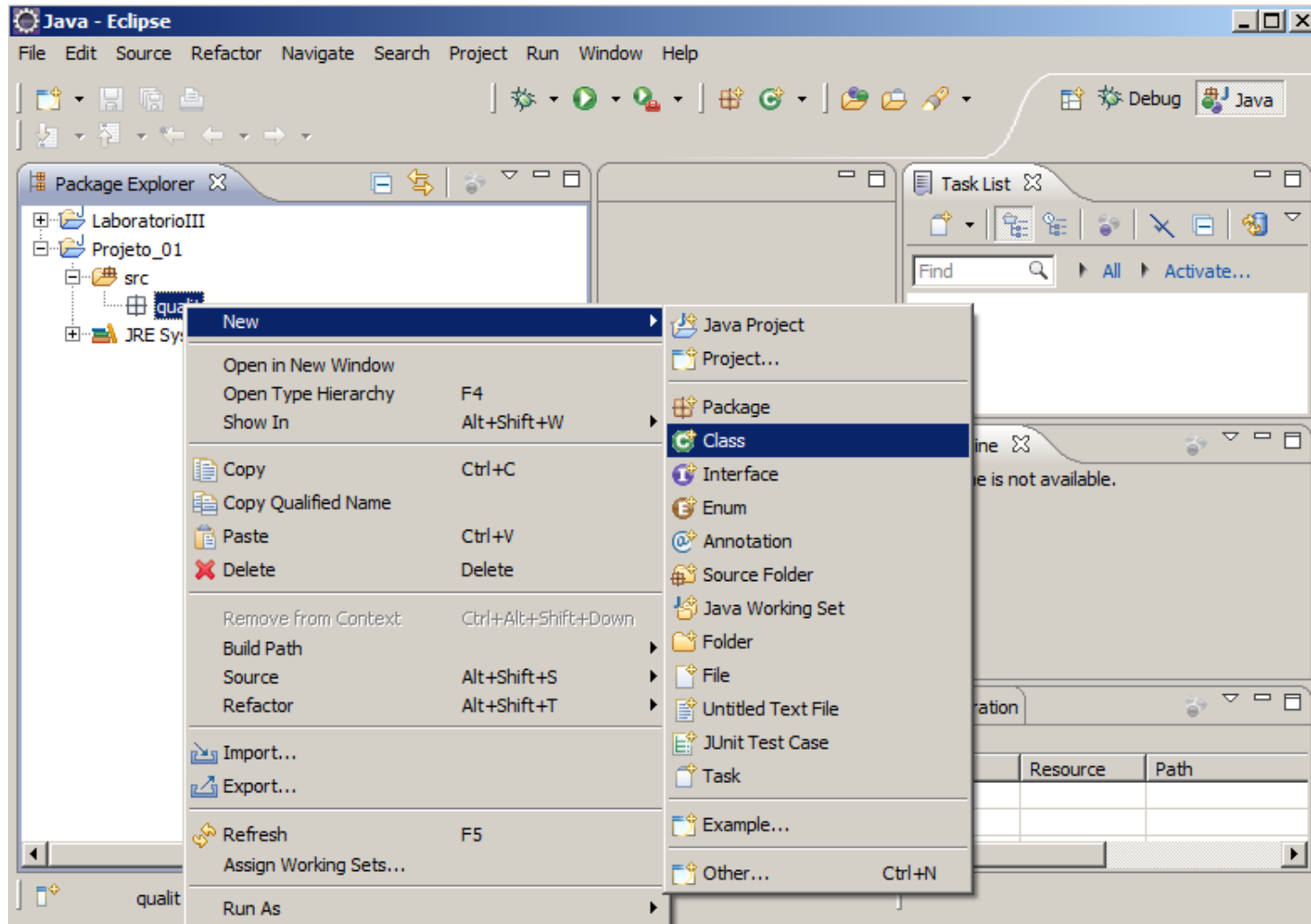
Criação de um package



Defina **qualit** com o nome do package e tecle **Finish**



Criação de uma classe



Com o botão direito do mouse sob o package **qualit** , clique em **New > Class**



Criação de uma classe

New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected

☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

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

☐ Constructors from superclass

☒ Inherited abstract methods

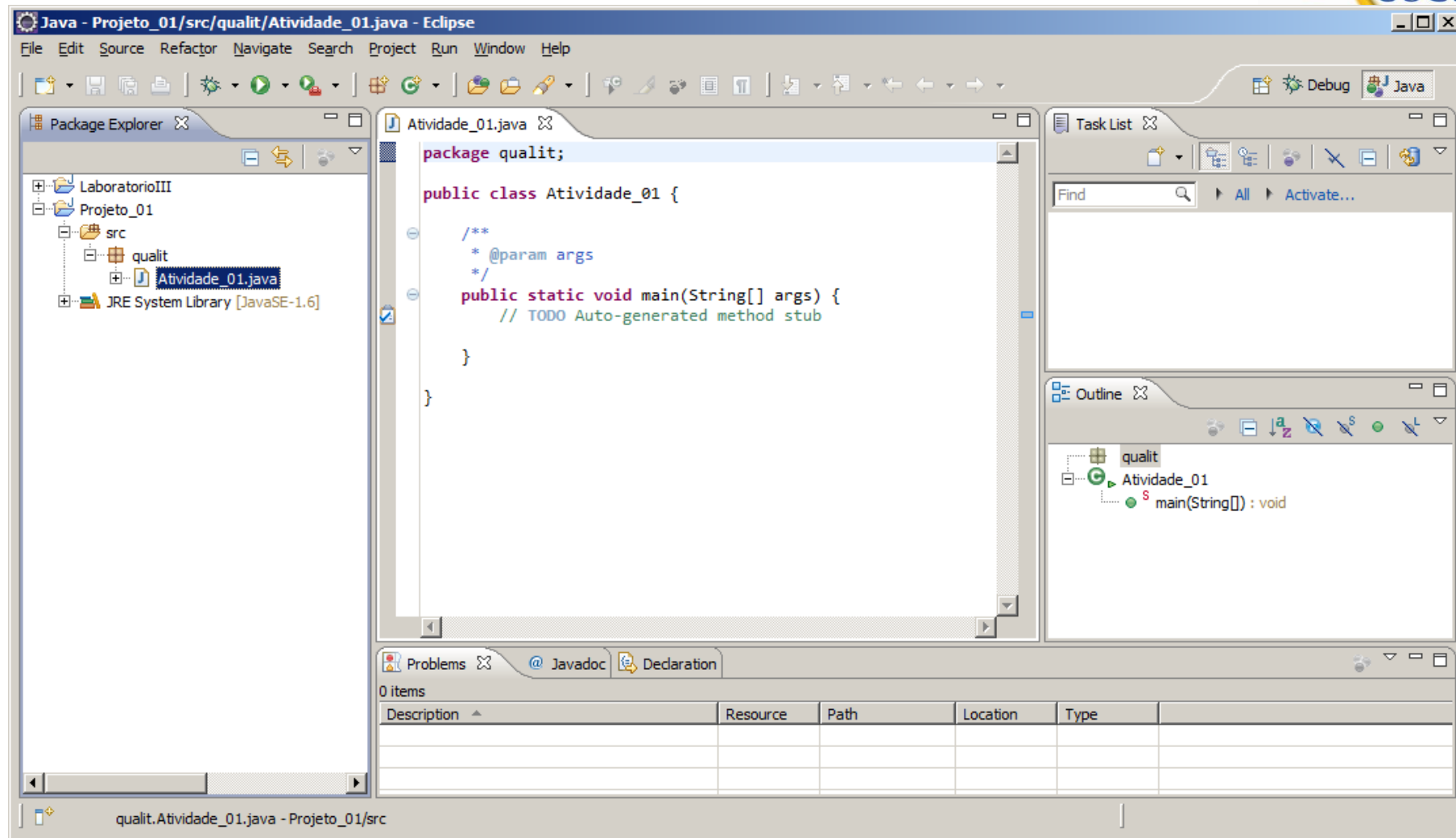
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Defina a classe como **Exercicio_01**, marque para gerar o método main, e tecle **Finish**



Editando o método main



Editando o método main



```
package qualit;

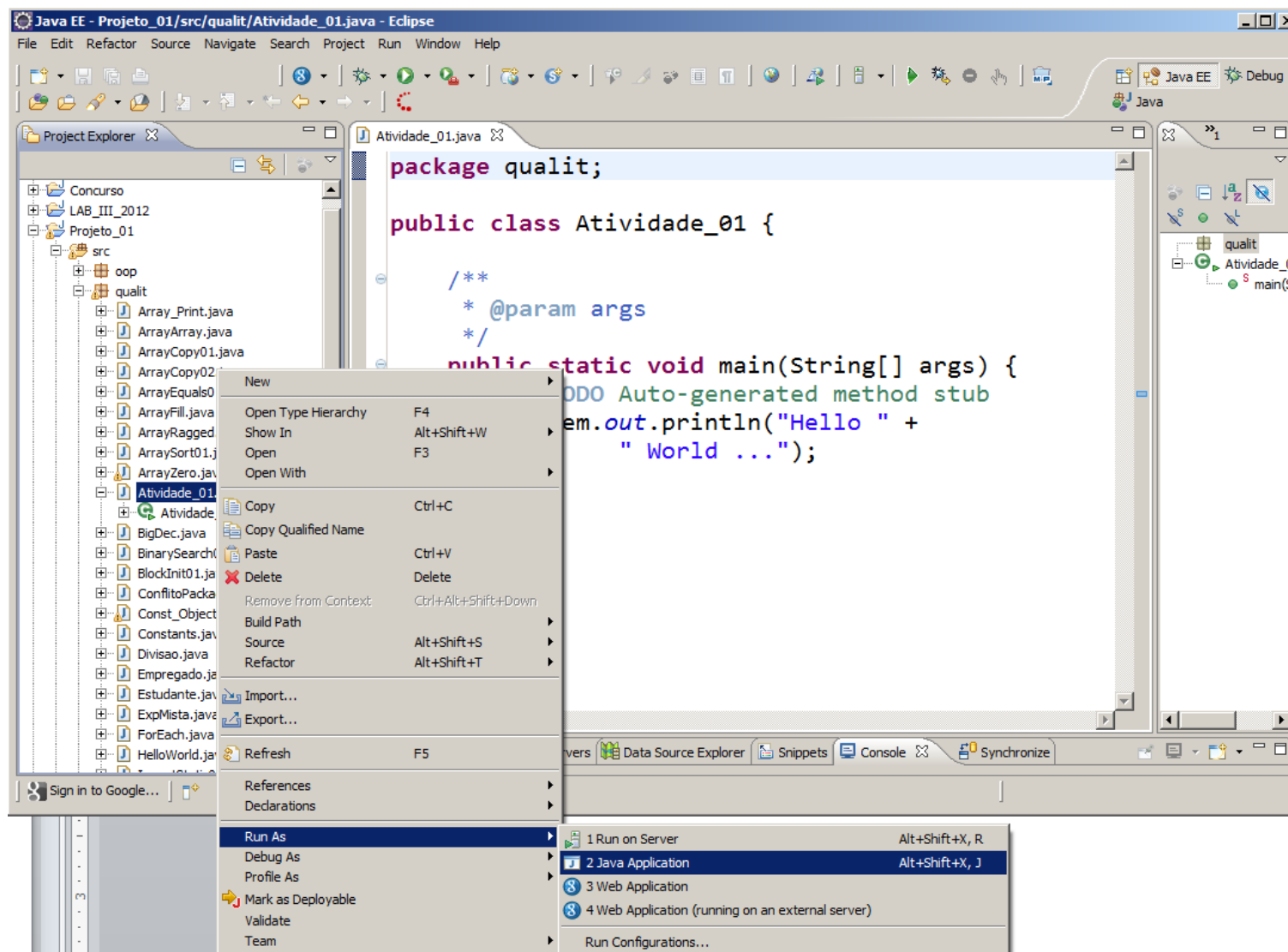
public class Atividade_01 {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello World...");
    }

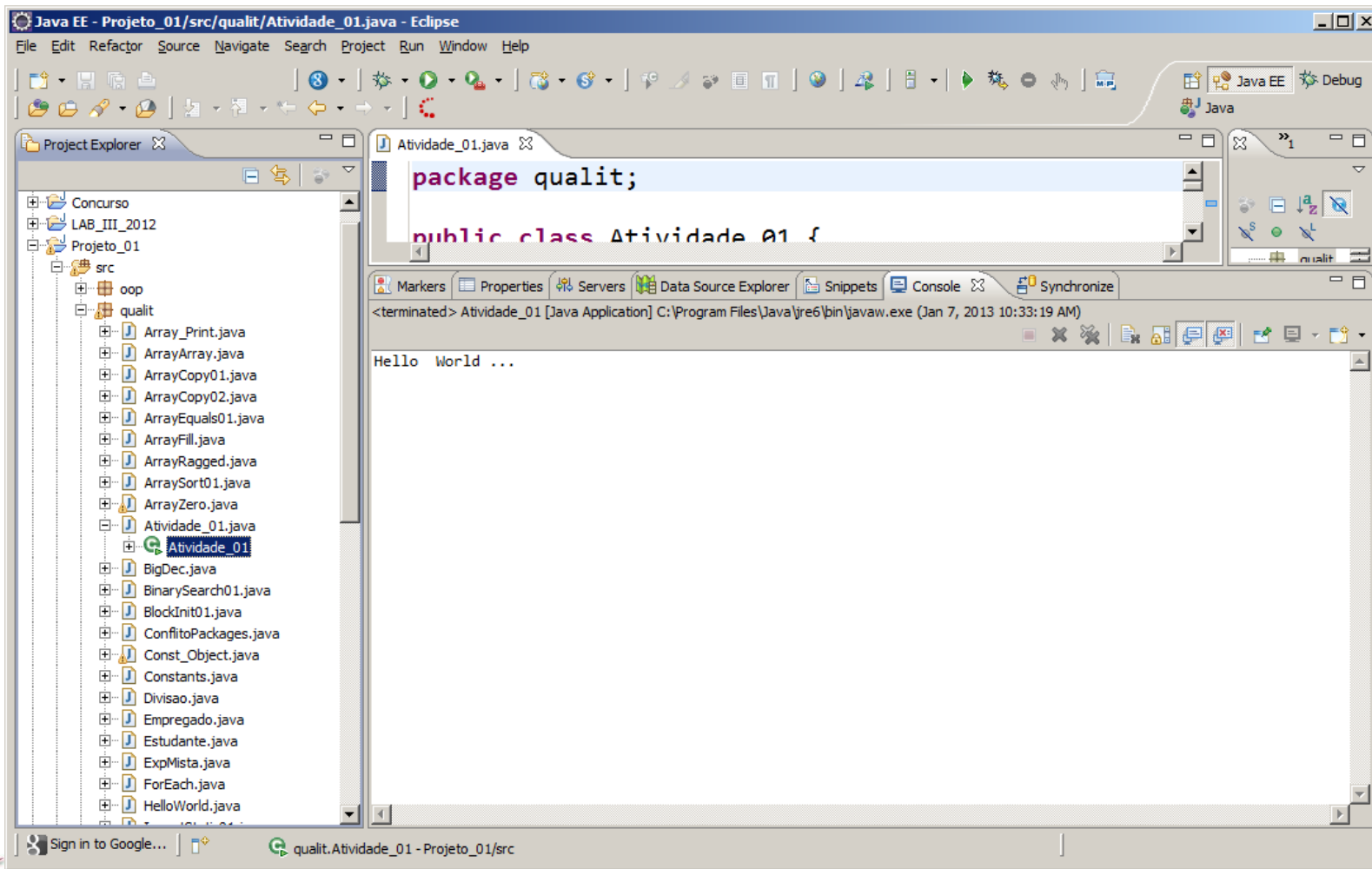
}
```



Executando a aplicação

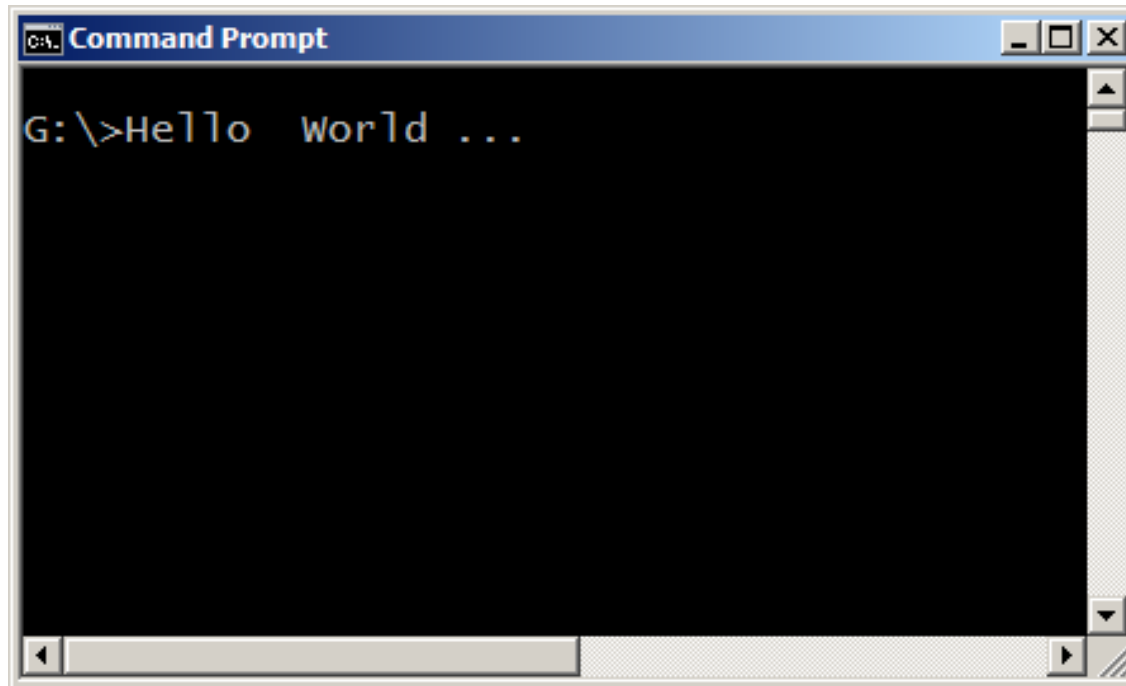


Executando a aplicação



Programas Java Console

- Recomendados para se iniciar o aprendizado da Linguagem.
- Interface **GUI** exige maior conhecimento de API's.
- A execução do método **println** direciona a saída para a console.



Um simples programa Java

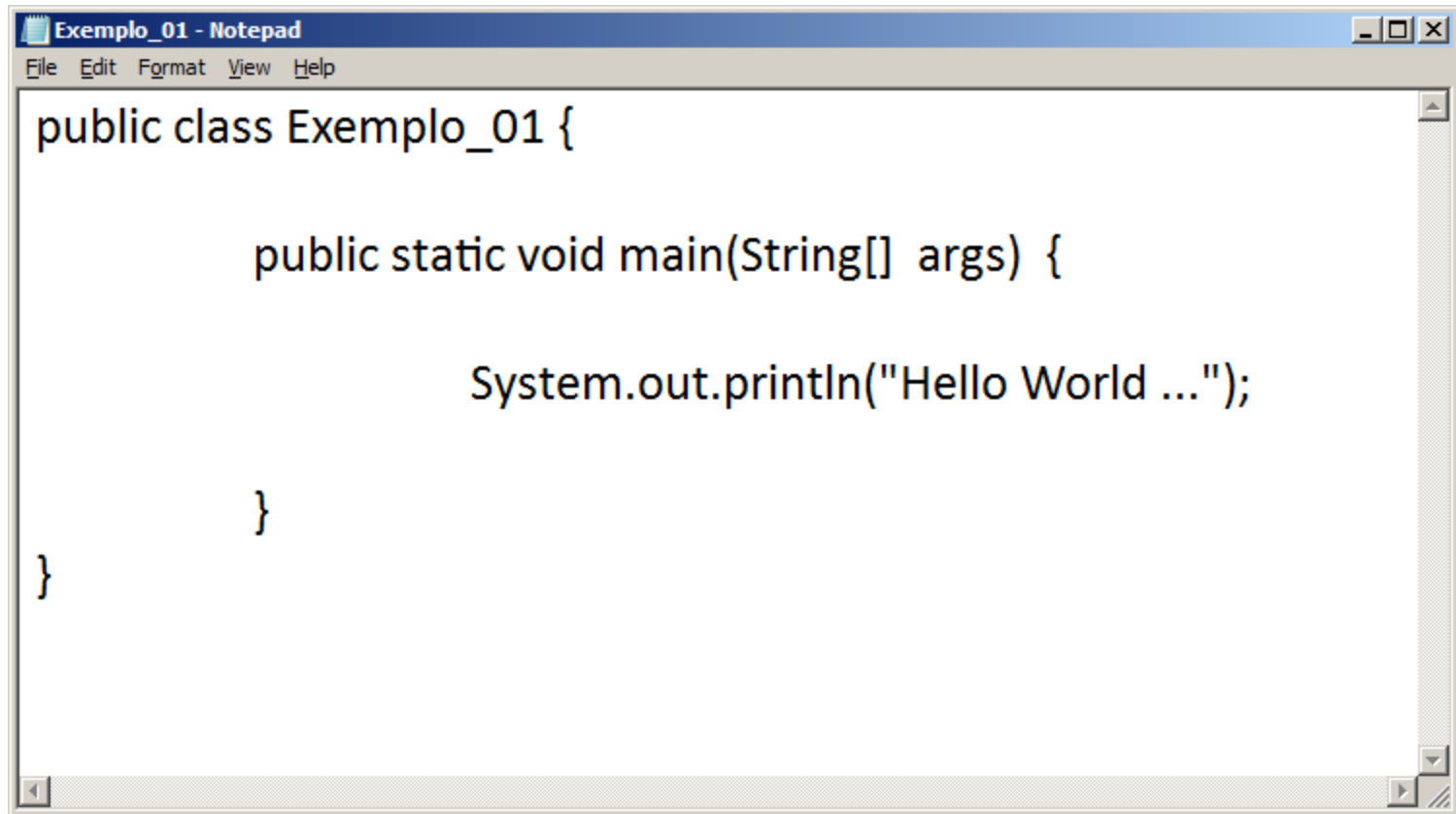
- Abra o prompt de comandos e execute o comando:

notepad Exemplo_01.java

```
Command Prompt
G:\QualitSys\Cursos_QUALIT\Cursos_Java_QUALIT\Java_Fundamentos_QUALIT\Atividades
>notepad Exemplo_01.java_
```



Editando com bloco de notas



```
Exemplo_01 - Notepad
File Edit Format View Help

public class Exemplo_01 {

    public static void main(String[] args) {

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

    }

}
```



As partes básicas do código

- A keyword **public** é chamada **modificador de acesso**.
- Um **modificador de acesso** determina como será a visibilidade de uma classe, atributo ou método a partir de outras classes ou métodos.

```
Exemplo_01 - Notepad
File Edit Format View Help

public class Exemplo_01 {

    public static void main(String[] args) {

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

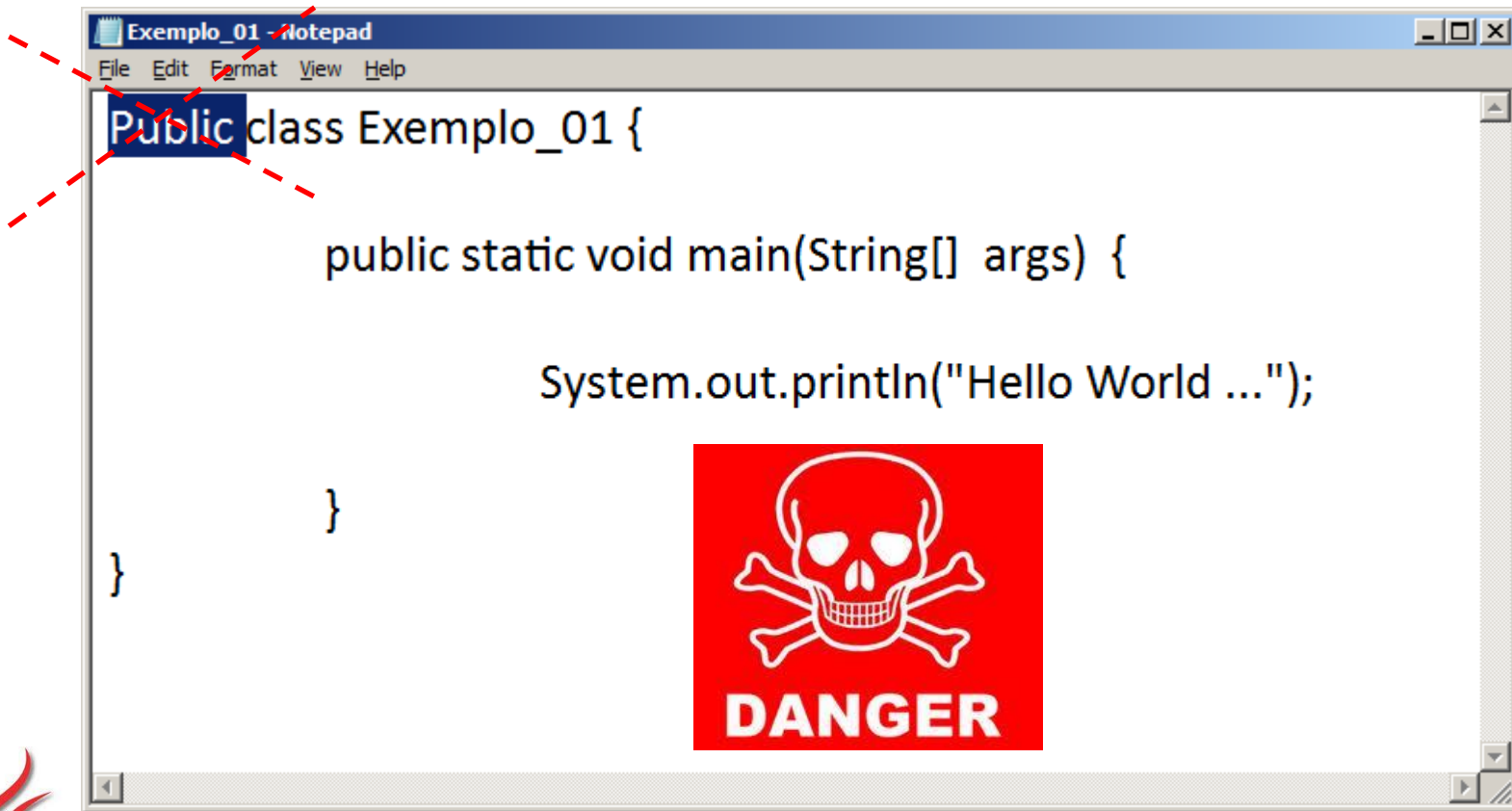
    }

}
```



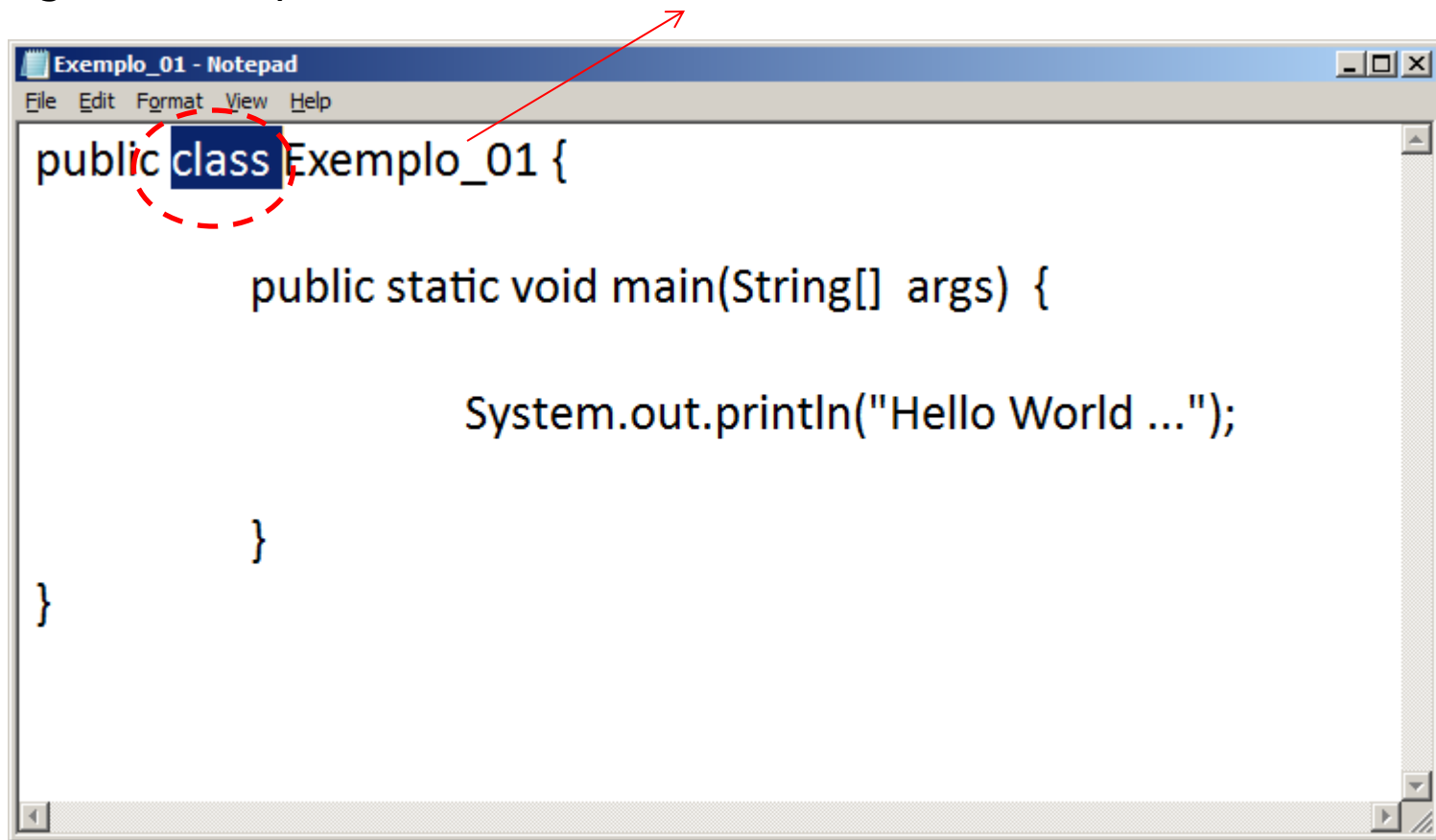
Java é Sensitive Case

- Assim, você deve codificar **public** e não ~~Public~~ !



A keyword class

- Contém a lógica e o comportamento da aplicação.
- Tudo em um programa Java está inserido dentro de uma classe.
- Seguindo a keyword **class** vem o nome da classe.



```
Exemplo_01 - Notepad
File Edit Format View Help
public class Exemplo_01 {
    public static void main(String[] args) {
        System.out.println("Hello World ...");
    }
}
```



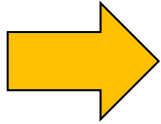
Regras para nomes

- Devem começar com uma **letra** , o sinal de **\$** ou o sinal de underscore **_** .
- Em seguida podem ter qualquer combinação de letras, \$, _ e dígitos.
- O tamanho do nome é ilimitado.
- Não podemos usar palavras reservadas para definir nomes.
- Convenção standard para nomes de classes: Primeiro caractere é maiúsculo.
Em caso de vários nomes, o caractere de início de cada nome é maiúsculo.
Esta convenção é chamada "**CamelCase**".



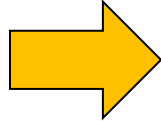
Exemplos de nomes

4juscs



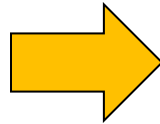
inválido, pois o primeiro caractere deve ser uma letra, ou sinal de dolar(\$) ou underscore (_)

_Step#4



inválido, pois o caractere # é inválido

_Sys_4\$



válido, apesar de estranho...



Palavras Reservadas (keywords)



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

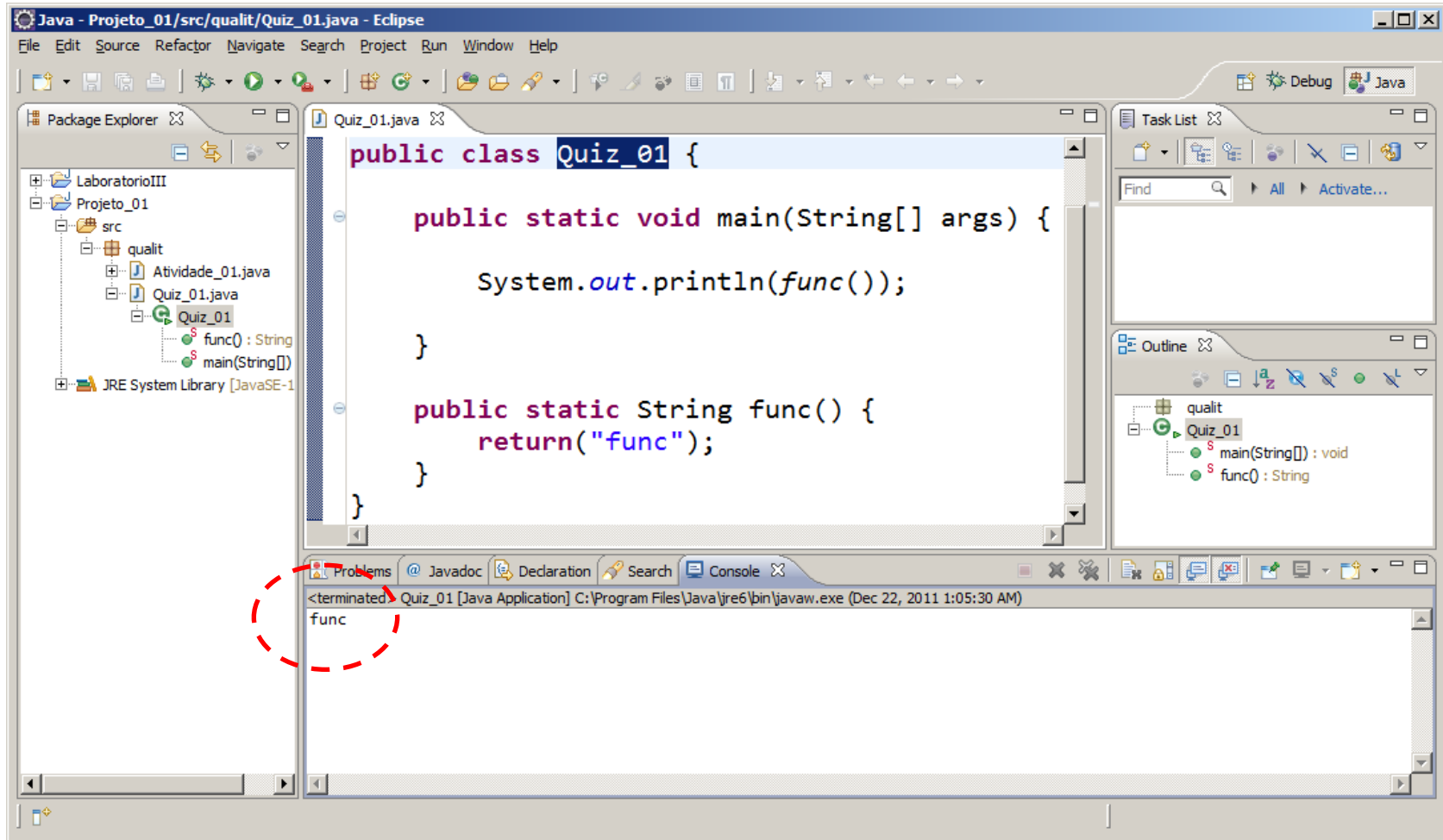


Este código compila ?

```
package qualit;  
  
public class Quiz_01 {  
  
    public static void main(String[] args) {  
  
        System.out.println(func());  
    }  
  
    public static String func() {  
        return("break");  
    }  
  
}
```

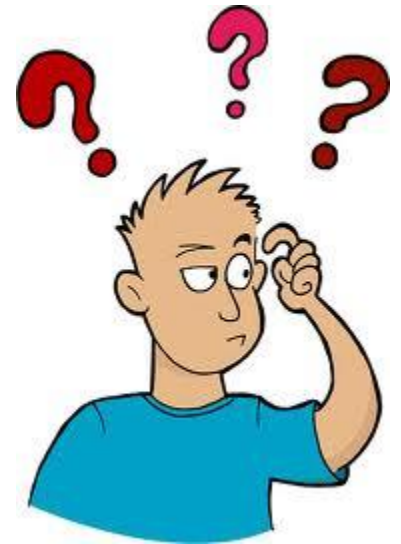


Sim...



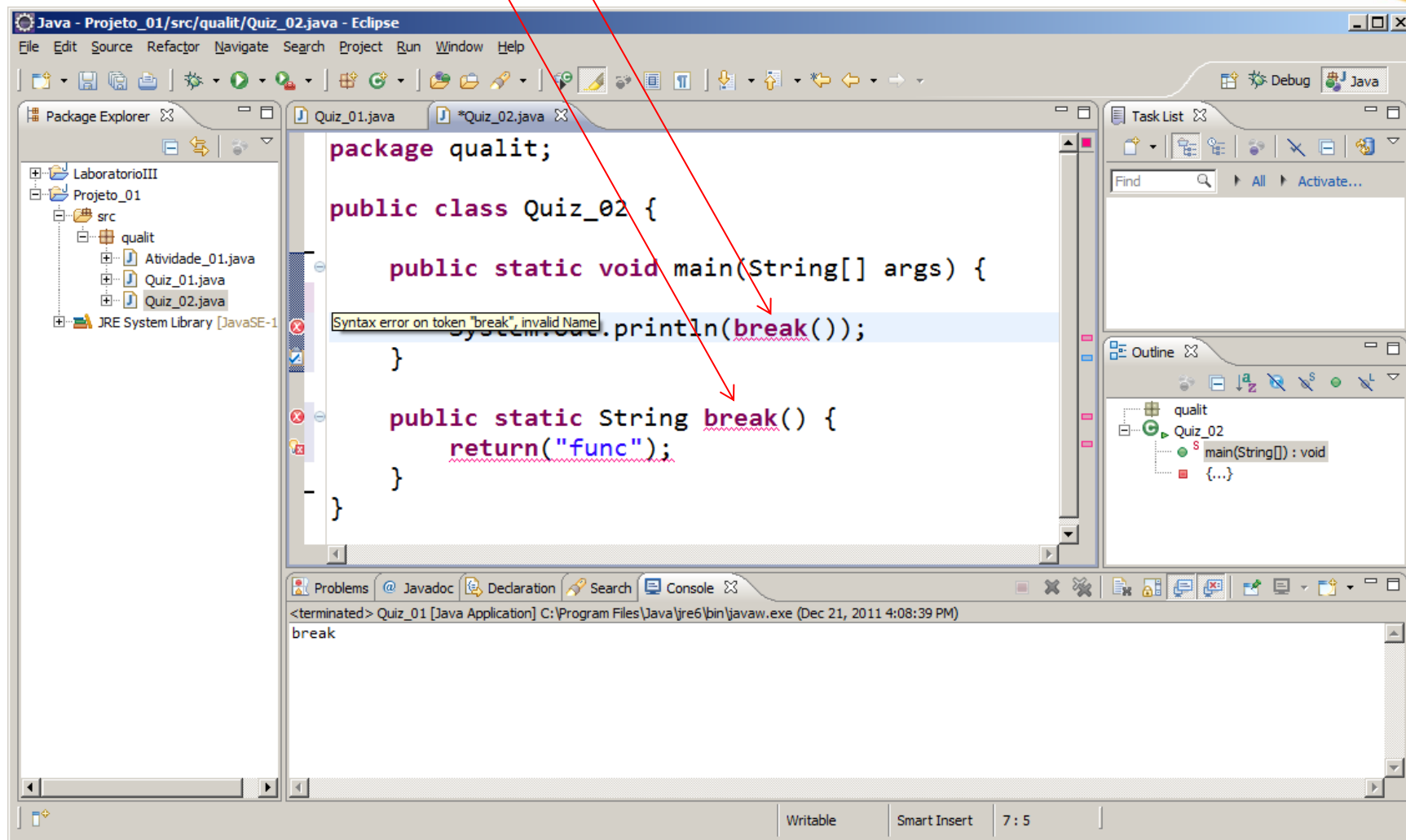
Este código compila ?

```
package qualit;  
  
public class Quiz_02 {  
  
    public static void main(String[] args) {  
  
        System.out.println(break());  
  
    }  
  
    public static String break() {  
        return("func");  
    }  
}
```





break é nome inválido...



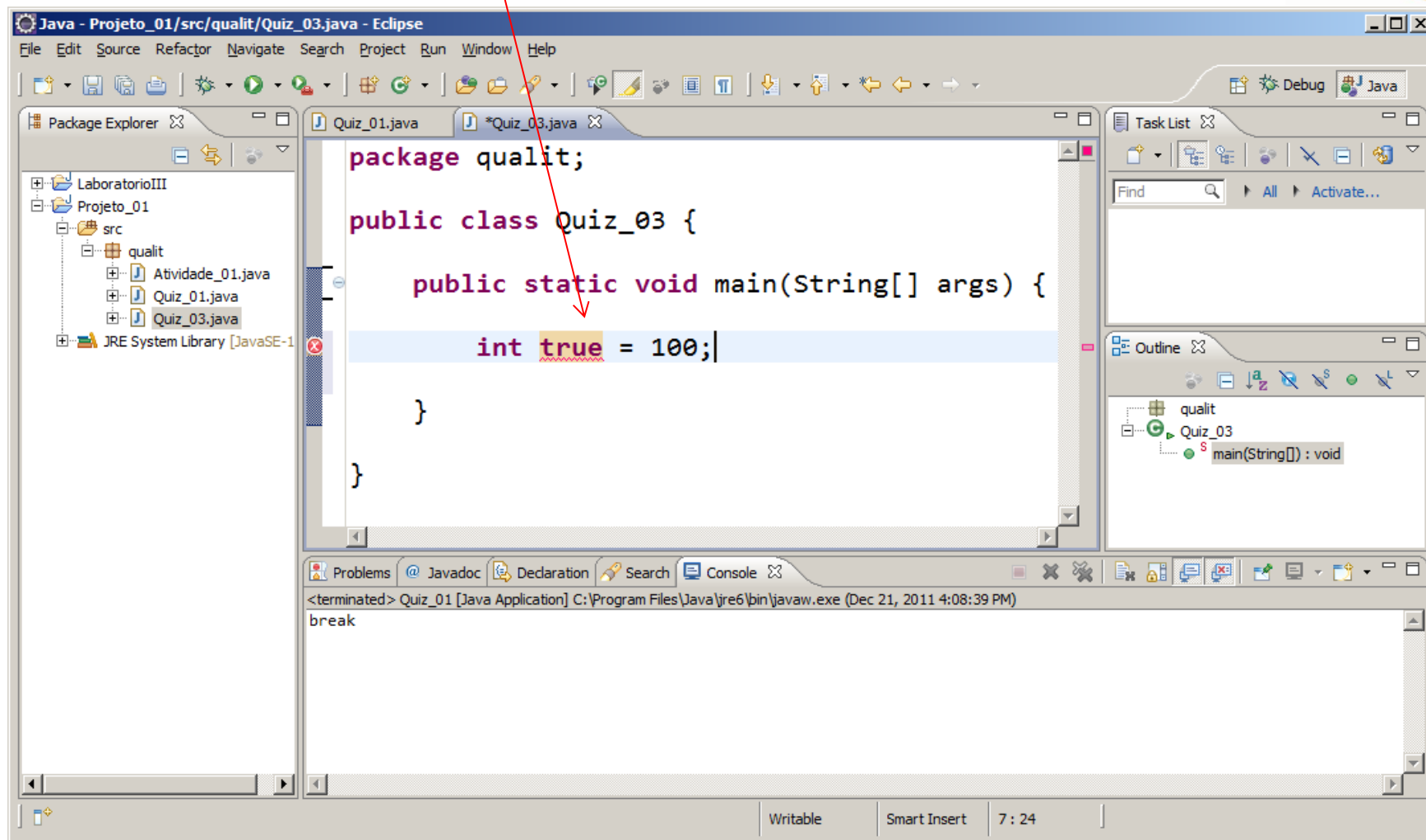
Este código compila ?

```
package qualit;  
  
public class Quiz_03 {  
  
    public static void main(String[] args) {  
  
        int true = 100;  
  
    }  
}
```





true é nome inválido...



O método main

- Toda aplicação Java deve ter um método **main**, declarado assim:

```
public class NomeDaClasse {  
  
    public static void main(String[] args) {  
  
        // statements do programa  
  
    }  
  
}
```

- Como qualquer método, o código deve ser definido entre **{ e }** (bloco).
- **void** indica que main não retorna um valor para o S.O. Para terminar o programa com um valor (**exit code**), use o método **System.exit** .



Comentários

- `//` ou `/*` e `*/`

```
public class Exemplo {  
  
    public static void main(String[] args) {  
  
        int i = 40; // comentario  
  
        /*          .....  
                   .....  
        */  
  
    }  
}
```

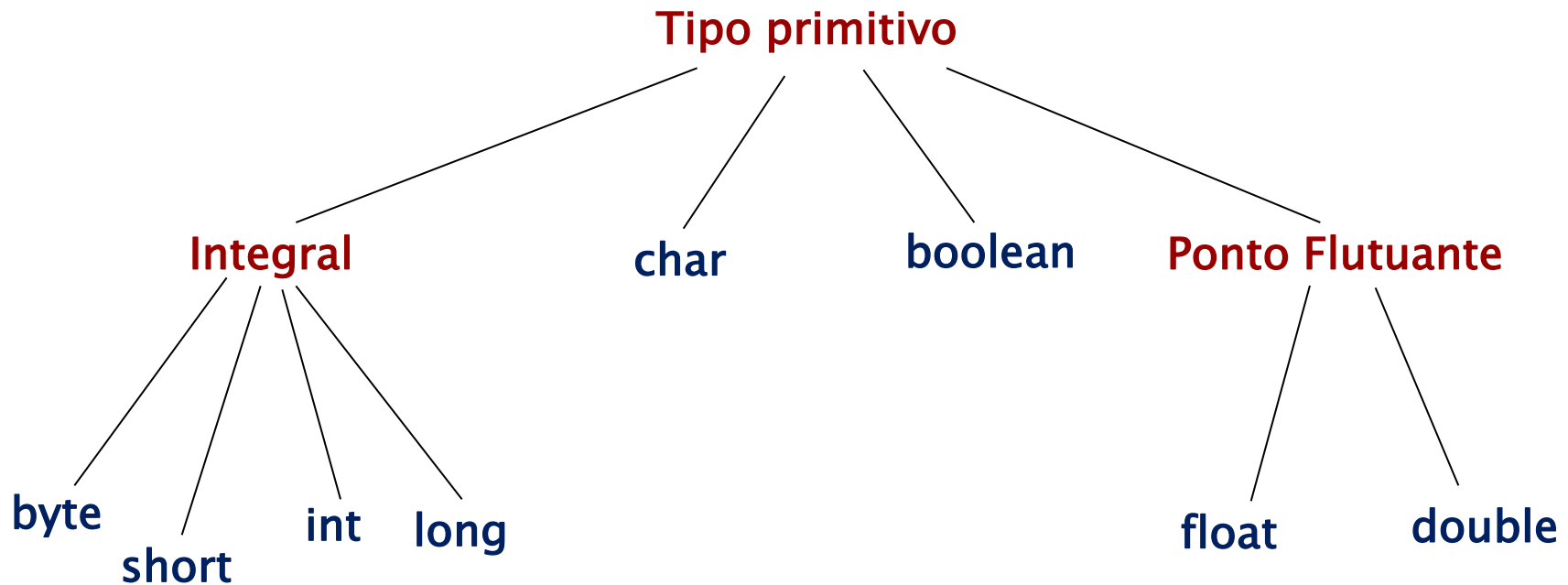


Só estou
esperando o
comentário



Tipos de Dados

- Java é **fortemente tipada**. Assim, toda variável deve ter um **tipo** declarado.
- Há oito tipos primitivos de dados.



Tipos inteiros

Tipo	Storage	Range (inclusive)
int	4 bytes	-2.147.483.648 a 2.147.483.647
short	2 bytes	-32.768 a 32.767
long	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
byte	1 byte	-128 a 127

- Tipos inteiros são números **sem** parte fracionária.
- O tipo **int** é o mais usado para fins práticos.
- Inteiros **long** têm o sufixo **L** (exemplo: 70L)
- Números hexadecimais têm o prefixo **0x** (exemplo: 0xFAB)
- Literais inteiros são **int** por default.



Tipos PontoFlutuante



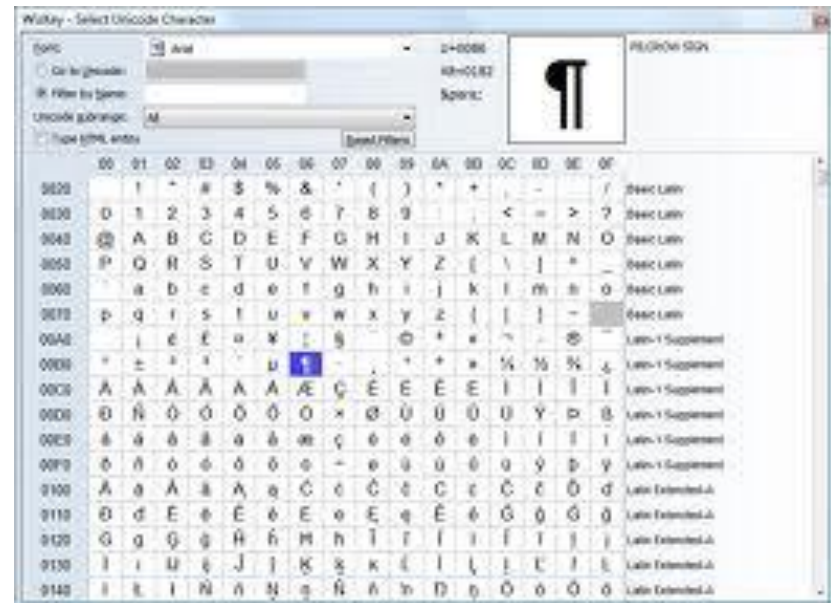
Tipo	Storage	Range (inclusive)
float	4 bytes	+ - 3.402823347 E+38F
double	8 bytes	+ - 1.79769313486231570E+308

- Tipos ponto flutuante são números **com** parte fracionária.
- O tipo **double** é o mais usado para fins práticos.
- Números **float** têm o sufixo **F** (exemplo: 9.67F)
- Opcionalmente, números double podem ter o sufixo D)
- Literais ponto flutuante são **double** por default.



O tipo char

- Usado para descrever caracteres isolados.
- Por exemplo, 'P' é uma constante caractere com o valor 80.
- 'P' é diferente de "P", um string contendo um único caractere.
- Literais char podem ser expressos por valores hexadecimais que representam código Unicode.
- Ex. `char x = '\u0080';`



Tipo boolean

- Tem dois valores: **false** e **true**.
- Usado para avaliação de condições lógicas.
- O compilador não permite a conversão entre valores inteiros e booleanos.

```
boolean x = true;
```

```
if (x = 0)
```



Constantes

- Em java, usamos a keyword **final** para denotar uma constante.
- Esta keyword indica que atribuímos um valor inicial a uma variável e este valor permanece constante durante a execução do programa.

```
package qualit;
```

```
public class Constants {
```

```
    public static void main(String[] args) {  
        double x=10.0;  
        double y=20.0;  
        System.out.println(x + y + DADO_CONSTANTE);  
    }
```

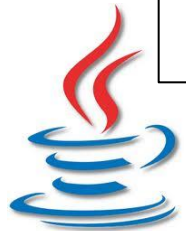
```
    public static final double DADO_CONSTANTE = 30.0;
```

```
}
```



Operadores

- `+ - * /` operadores aritméticos
- `++ --` operadores incremento e decremento
- `==` teste de igualdade
- `!=` teste de desigualdade
- `< <= > >=` testes de comparação
- `& &&` operador lógico and
- `| ||` operador lógico or
- `& | ^ ~` operadores bitwise para tipos inteiros
- `op=` `a+=b` `<=>` `a = a + b`
- `cond ? exp1 : exp2` operador ternário (?) Ex. `a < b ? a : b`



Operadores Bitwise

&	AND (1 se ambos forem 1)
	OR (0 se ambos forem 0)
^	XOR (0 se bits forem iguais)
~	COMPLEMENT (1 se 0; 0 se 1)



Operadores Bitwise

```
int i = 0xFFFF0;
```

```
int var = i & 0x00FF;
```

i	0xFFFF0	1111	1111	1111	0000
mascara	0x00FF	0000	0000	1111	1111
	i & 0x00FF	0000	0000	1111	0000



Tipos Enumerados

- Tem um número finito de valores nomeados.
- Por exemplo:

```
enum Tamanho { PEQUENO, MEDIO, GRANDE, EXTRA_GRANDE };
```

- Uma variável do tipo Tamanho pode somente armazenar valores listados na declaração de tipo ou o valor especial **null** que indica que a variável não tem elementos.



Tipos Enumerados



```
package qualit;  
  
public enum TAMANHO {  
  
    pequeno,  
    médio,  
    grande,  
    extra_grande  
  
}
```

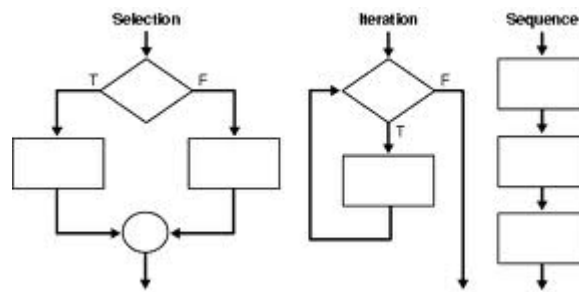


Tipos Enumerados



```
package qualit;  
  
public class TestaEnum {  
  
    public static void main(String[] args) {  
  
        TAMANHO x = TAMANHO.grande;  
  
        System.out.println (x);  
  
    }  
}
```





Estruturas de Controle



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Introdução

- Java, como qualquer linguagem, suporta comandos condicionais e loops para determinar o fluxo de controle.
- Os construtos de fluxo de controle em Java são semelhantes às linguagens C e C++.



Escopo de Blocos

- Um **bloco** corresponde a um conjunto de instruções java que são envolvidas por um par de { } .
- **Blocos** definem o **escopo** de suas variáveis.
- **Blocos** podem ser aninhados em outros blocos.
- Escopo de uma variável define a sua **visibilidade** no código.



Blocos aninhados

```
public static void main(String [] args) {  
  
    int a;  
  
    . . .  
  
    {  
  
        int b;  
  
        . . .  
  
    } // b é visível até aqui  
  
}
```



Comandos Condicionais



if (expressão)
 comando;

- expressão pode retornar um valor **true** ou **false**.
- Exemplo:

```
if (numero%2 != 0)  
    ++numero;
```



Comandos Condicionais

```
if (expressão)
{
    comando1;
    comando2;
    . . .
    comandon;
}
```



Comandos Condicionais

```
if (expressão)  
    comando1;  
    comando2;  
    ...  
comandon;
```



else

```
if (expressão)
{
    comandos;
}
else
{
    comandos;
}
```



Operadores Lógicos

&

AND lógico

&&

AND condicional (lazy)

|

OR lógico

||

OR condicional (lazy)

!

NOT



Operações AND

- Duas expressões devem ser ambas verdadeiras para que o resultado seja verdadeiro.

```
if (simbolo >= 'A' && simbolo <= 'Z')  
    System.out.println("simbolo eh maiusculo");
```



& e &&

- ◆ && não avaliará o operador da direita se o operador da esquerda for falso.
- ◆ & sempre avaliará os dois operandos.



Operações OR

- ◆ Se uma das expressões for **TRUE** então o resultado também será **TRUE**.

```
if (idade < 16 || idade >= 65)  
    Taxa *= 0.9;    // valor reduzido em 10%
```



| e ||

- ◆ || não avaliará o operador da direita se o operador da esquerda for **TRUE**.
- ◆ | sempre avaliará os dois operandos.



Operação NOT

- ◆ Se uma expressão for **TRUE** então **NOT** retorna **FALSE**.
- ◆ Se uma expressão for **FALSE** então **NOT** retorna **TRUE**.

```
if ( ! (idade >= 16 || idade < 65) )  
    Taxa *= 0.9;    // valor reduzido em 10%
```



for

```
for (exp_inicial; cond_loop; exp_increm)
{
    // comandos . . .
}
```



for

```
public static void main(String[ ] args) {  
    int valor = 10;  
    for (int i=0; i<20; i++) {  
        valor *= 3 ;  
        System.out.println ("Valor=" + valor);  
    }  
}
```



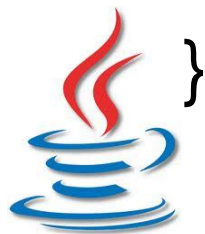
while

```
while (expressao)
{
    // comandos . . .
}
```



while

```
public static void main(String[ ] args)  {  
    int limite = 20;  
    int soma = 0;  
    int i = 1;  
    while (i <= limite)    {  
        soma = soma + i;  
        i = i + 1;  
        System.out.println ("Soma=" + soma);  
    }  
}
```



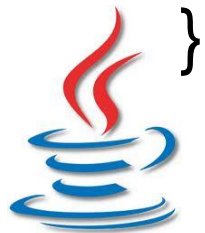
do while

```
do  
{  
    // comandos . . .  
} while (expressao);
```



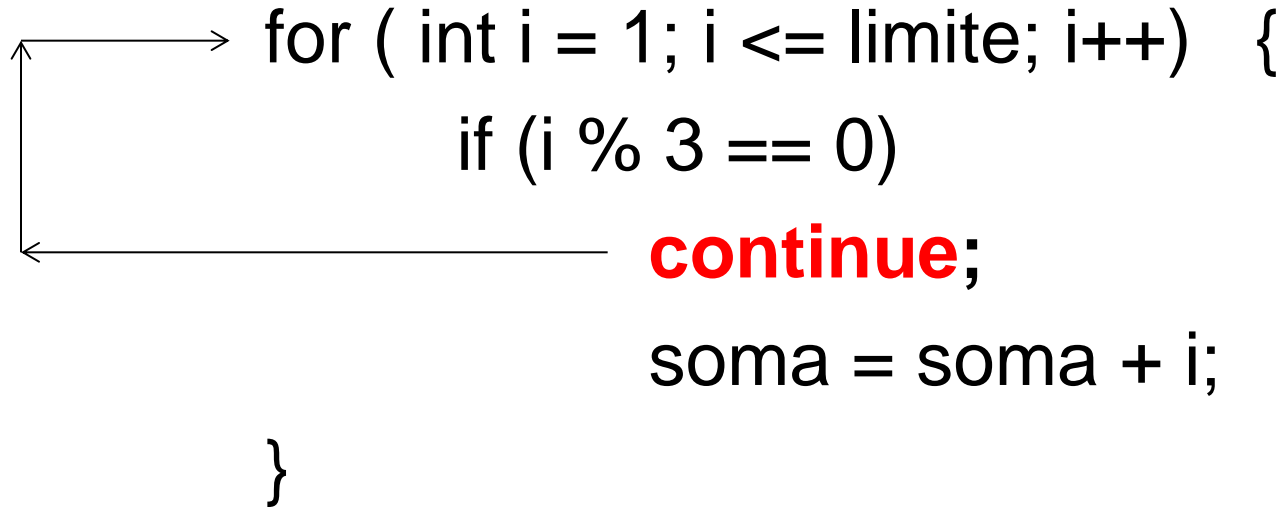
do while

```
public static void main(String[ ] args) {  
    int limite = 20;  
    int soma = 0;  
    int i = 1;  
    do {  
        soma = soma + i;  
        i = i + 1;  
        System.out.println ("Soma=" + soma);  
    } while (i <= limite);  
}
```



continue

```
→ for ( int i = 1; i <= limite; i++) {  
    if (i % 3 == 0)  
        continue;  
    soma = soma + i;  
}
```

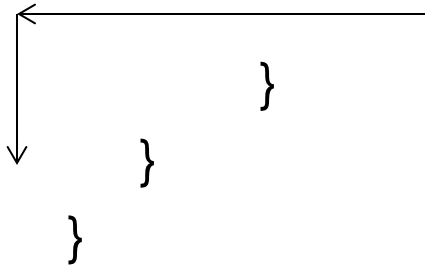


- Permite o salto para o início do loop.



break

```
int nValores = 200;  
boolean ehPrimo = true;  
for (int i = 2; i <= nValores; i++) {  
    ehPrimo = true;  
    for ( int j = 2; i <= j; j++) {  
        if (i % j == 0) {  
            ehPrimo = false;  
            break;
```



- Permite a quebra do loop



Arrays e Strings



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com

Introdução

- Um estagiário recebeu uma incumbência de escrever um programa Java que irá manipular **1000** números...



O que fazer ?

- ◆ Com os tipos básicos vistos nas unidades anteriores, cada identificador corresponde à uma única variável.
- ◆ Mas, como proceder para manusearmos um conjunto de valores do mesmo tipo ?
- ◆ Por exemplo: os primeiros 1000 números primos.



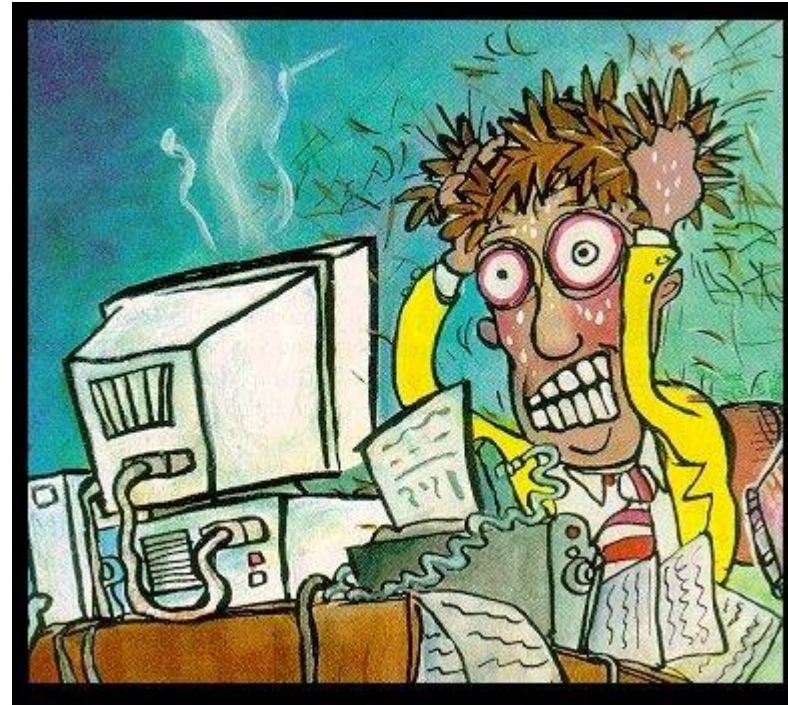
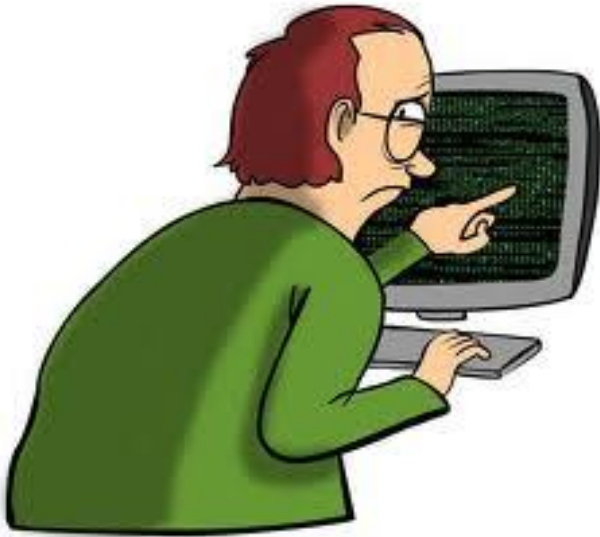
Uma alternativa...

- ◆ Criar 1000 variáveis, cada uma com um determinado nome...



Alternativa inviável ...

- ◆ O programa teria 1000 variáveis ...
- ◆ A tabela de símbolos certamente seria difícil de ser manipulada...



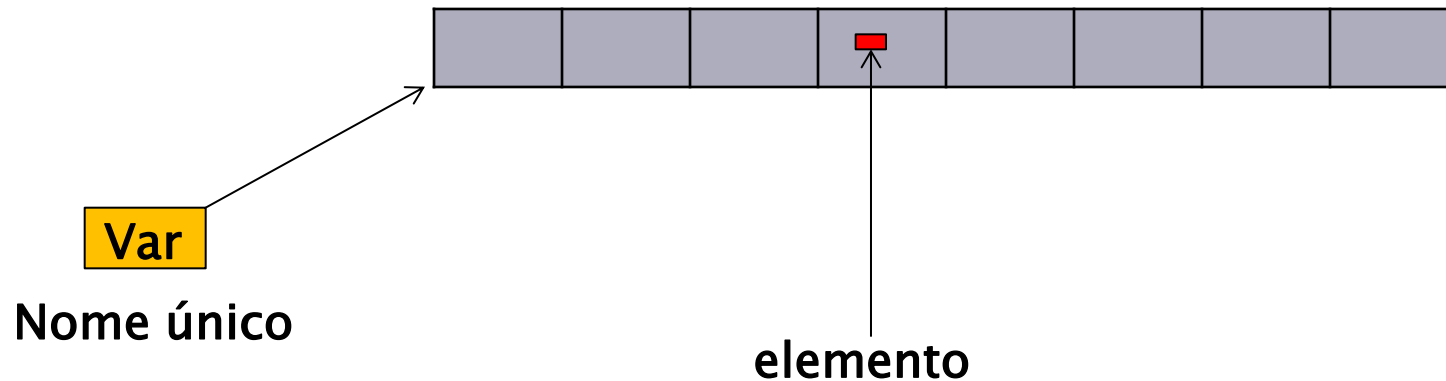
Outra alternativa...

◆ Empregar arrays ...



Arrays

- ◆ Um **array** é um conjunto de variáveis do mesmo tipo a qual atribuímos um nome único.
- ◆ Cada informação (dado) no **array** é chamada de **elemento** do **array**.



Arrays

- ❖ Para fazermos referência a um elemento de um array devemos usar o nome do array em conjunto com um número inteiro chamado **índice**.
- ❖ O primeiro elemento do array tem índice **0**, o segundo **1**, e assim por diante.

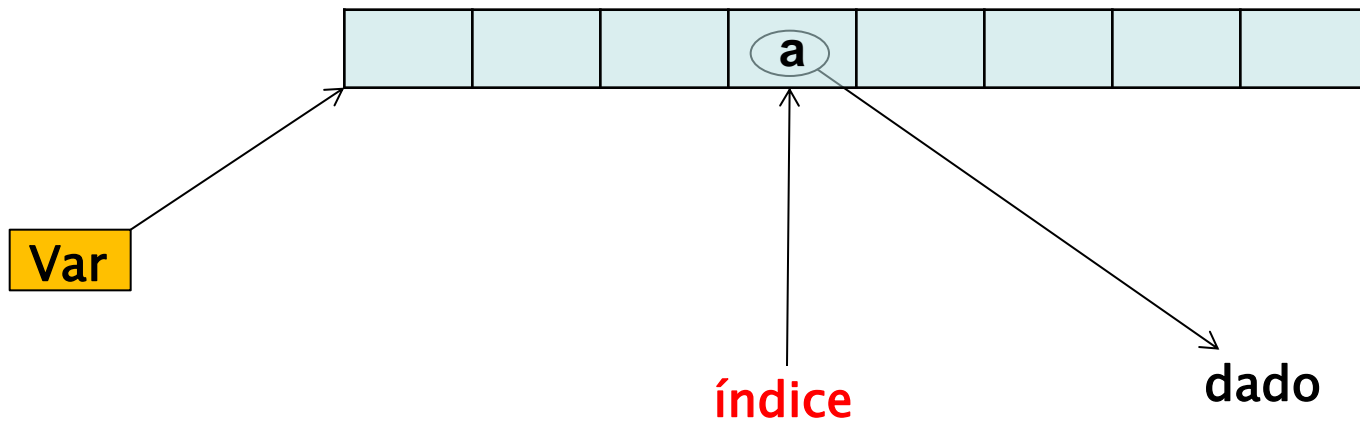


Índice é como o rótulo de uma caixa ...



Índice de um array

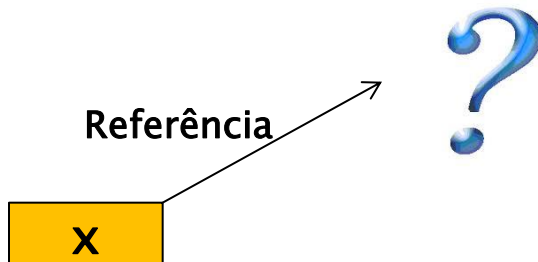
- ◆ Pode ser representado pela avaliação de uma expressão que deve resultar em um valor inteiro maior ou igual a zero.



Variáveis array

```
int [ ] x;
```

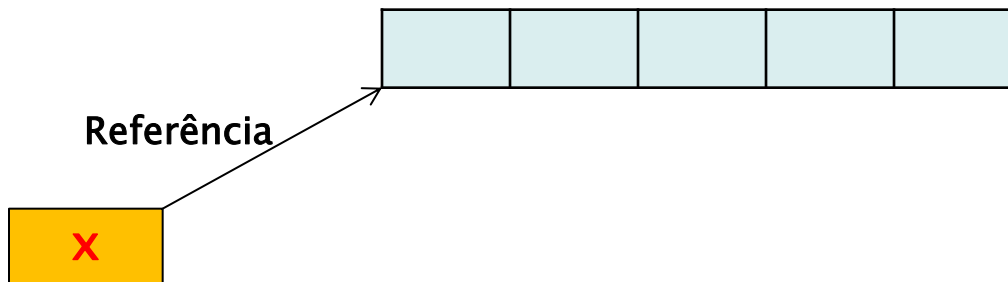
- ❖ A variável **x** corresponde a uma **referência** a um array de inteiros que ainda não foi criado.
- ❖ Portanto, neste ponto ainda não foi alocada memória para o array.



Definindo um array

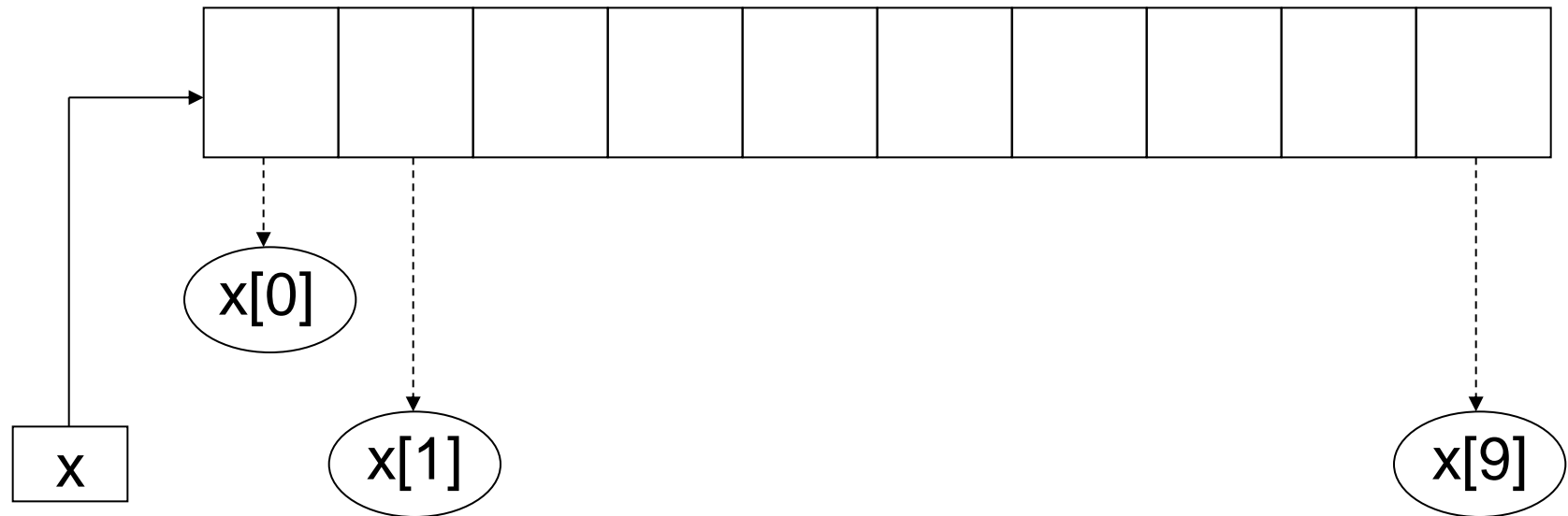
```
x = new int[5];
```

- ❖ O statement acima cria um array que irá armazenar 5 valores inteiros e grava uma referência ao array na variável **x**.
- ❖ A referência é simplesmente o endereço onde o array está na memória.



Acesso aos elementos do array

```
int[ ] x = new int[10];
```



Iniciando arrays

- ◆ Podemos inicializar um array explicitando os valores em tempo de declaração.
- ◆ Com este procedimento o tamanho do array e, consequente alocação de memória, é definido.

```
int [ ] x = {2,3,5,7,11,13,17} ;
```

ou

```
int[ ] x = new int [ ] { 2,3,5,7,11,13,17 } ;
```

O array acima tem 7 elementos inteiros.



Atribuição de arrays

```
int [ ] x = new int[100];
```

```
x[0] = 2;
```

```
x[1] = 3;
```

- ◆ Obs. Os demais itens do array são inicializados em **zero** (valor default)



Quantos arrays existem ?

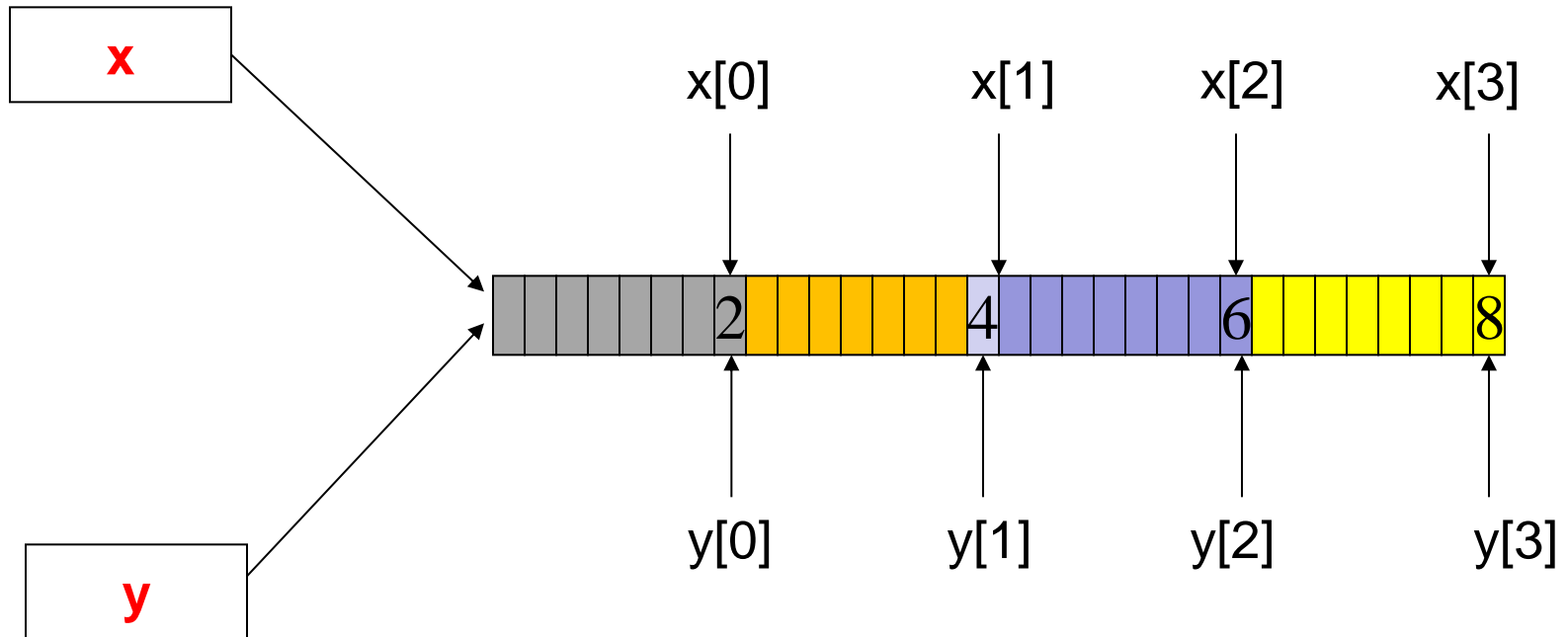
```
long [ ] x = {2L, 4L, 6L, 8L, 10L};  
long [ ] y = par;
```



Variáveis array

```
long [ ] x = {2L, 4L, 6L, 8L};
```

```
long [ ] y = par;
```



◆ Foram criadas duas variáveis array, porém temos apenas **um** array alocado em memória.



Populando arrays

```
double [ ] par = new double[50];  
for (int i=0; i < 50; i++)  
    par[i] = 100.0 *Math.random();
```

- ◆ Utilizamos elementos de array da mesma forma que usamos variáveis do mesmo tipo de dados.



Tamanho de um array

- ◆ Podemos nos referir ao tamanho de um array usando um membro de dados do objeto array chamado **length**.

```
double [ ] par = new double[50];  
double average = 0;  
for (int i=0; i < par.length; i++)  
    average += par[i];  
average /= par.length;
```



Atenção ...

- ❖ Se você definir um array com 50 elementos, as entradas do array variam de 0 a 49 (não de 1 a 50).

```
int [ ] x = new int [50];  
  
for (int i = 0; i < 50, i++) {  
    x[i] = i;  
}  
  
x[50] = 0; // erro...
```



Observação

- ◆ Podemos definir uma variável array de duas formas:

```
int [ ] x;
```

ou

```
int x[ ] ;
```



- ◆ A forma **int [] x;** é mais comumente utilizada...



Imprimindo os elementos do array



```
package qualit;  
  
public class Array_Print {  
  
    public static void main(String[] args) {  
        int[] x = new int[50];  
  
        for (int i=0; i< x.length; i++) {  
            x[i]=i;  
            System.out.println(x[i]);  
        }  
    }  
}
```



Array de caracteres

```
char [ ] mensagem;           //declara variavel
mensagem = new char [5];     // cria o array
mensagem[0] = 'a';
mensagem[1] = 'e';
mensagem[2] = 'i';
mensagem[3] = 'o';
mensagem[4] = 'u';
```

```
char [ ] mensagem = {'a', 'e', 'i', 'o', 'u'};
```



String

- ◆ É uma classe standard em Java a qual disponibiliza funcionalidades para o tratamento de listas de caracteres.
- ◆ Conceitualmente, strings são sequências de caracteres Unicode.
- ◆ Todo literal String (entre “ “) é uma instância da classe **String**.



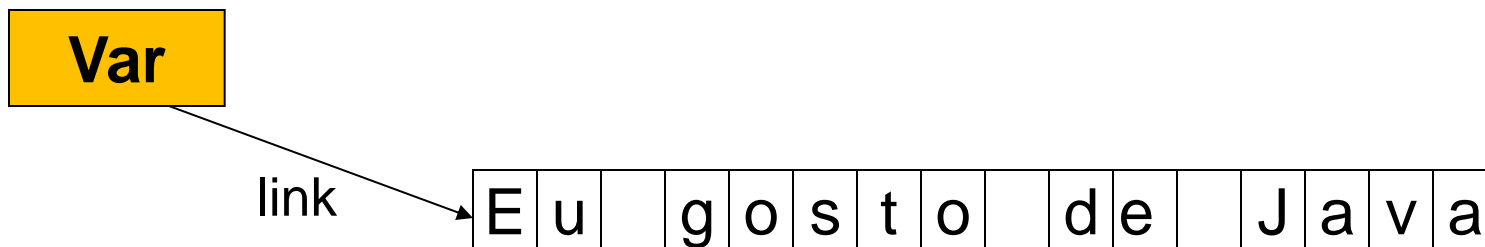
Literais String

- ◆ Correspondem à uma seqüência de caracteres delimitados por “ ”.
- ◆ Exemplo:
 “Eu gosto de estudar na USCS !”
- ◆ O exemplo acima é um objeto constante da classe **String** que o compilador cria para usarmos no programa.



Criação de Strings

String Var = “Eu gosto de Java”;



Entrando com dados pela console

- ❖ A classe **Scanner**, definida no package **java.util**, permite a leitura de dados a partir da “standard input stream”.



Obs. No eclipse, tecele <Ctrl> + <Shift> + O para import...

Entrando com dados pela console

```
import java.util.Scanner;

public class Scanner01 {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        System.out.print("Qual o seu nome ? ");
        String nome = in.nextLine();

        System.out.print("Quantos anos você tem ?");
        int idade = in.nextInt();

        System.out.println("Olá " + nome +
                           ", " + "você tem " + idade + " anos...");

    }
}
```



Convertendo argumentos String para inteiros

- ◆ main considera todos os argumentos **Strings**.
- ◆ Para avaliar um argumento como **int**, use **Integer.parseInt()**.

Integer.parseInt(variavel)

```
int x = Integer.parseInt(args[0]);
```

