

FIAP GRADUAÇÃO

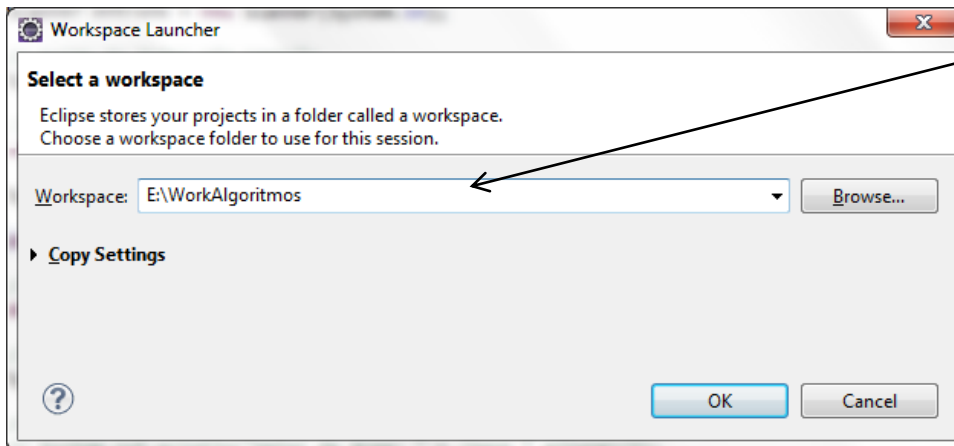
SISTEMA DE INFORMAÇÃO

LABORATÓRIO DE PROGRAMAÇÃO

PROF^a. EVELYN CID

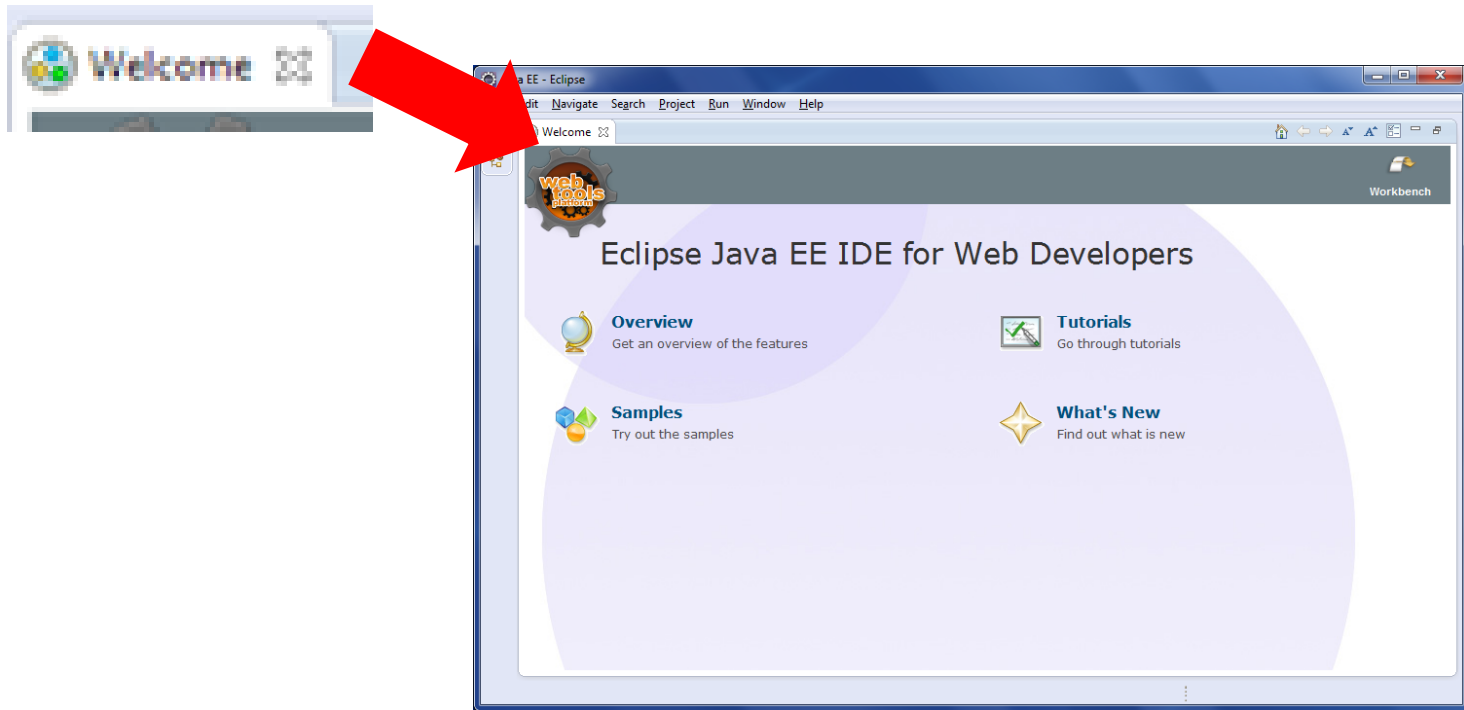
IDEs – Criação do Ambiente - Workspace

- É a área de trabalho onde armazenaremos os projetos, no nosso caso os exercícios desenvolvidos apoiados pelo Java aula a aula.
- Abrir o Eclipse e selecionar um local onde será criada a área de trabalho (Workspace).
 - Como sugestão, para melhor organização, crie a pasta: LabProgramação.



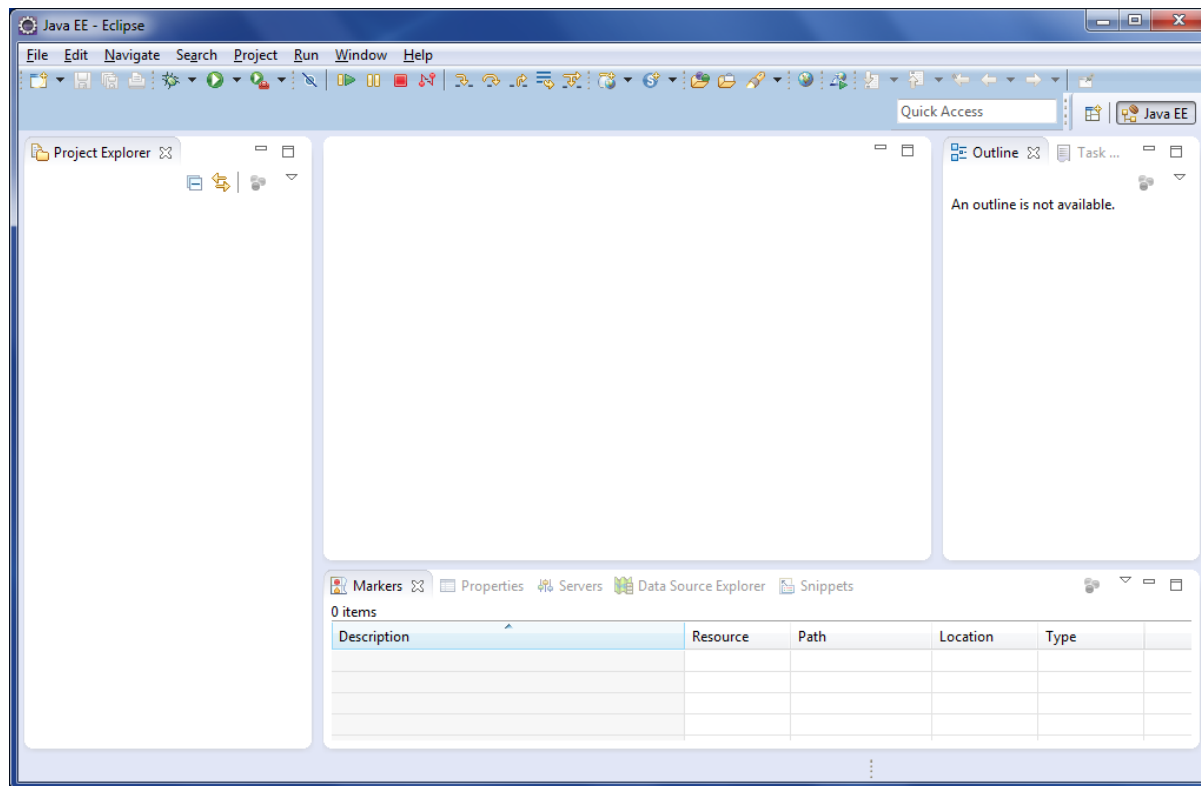
IDEs – Criação do Ambiente

Sempre que criada uma nova Workspace será exibida a janela abaixo, basta **fechá-la** para dar continuidade ao processo.

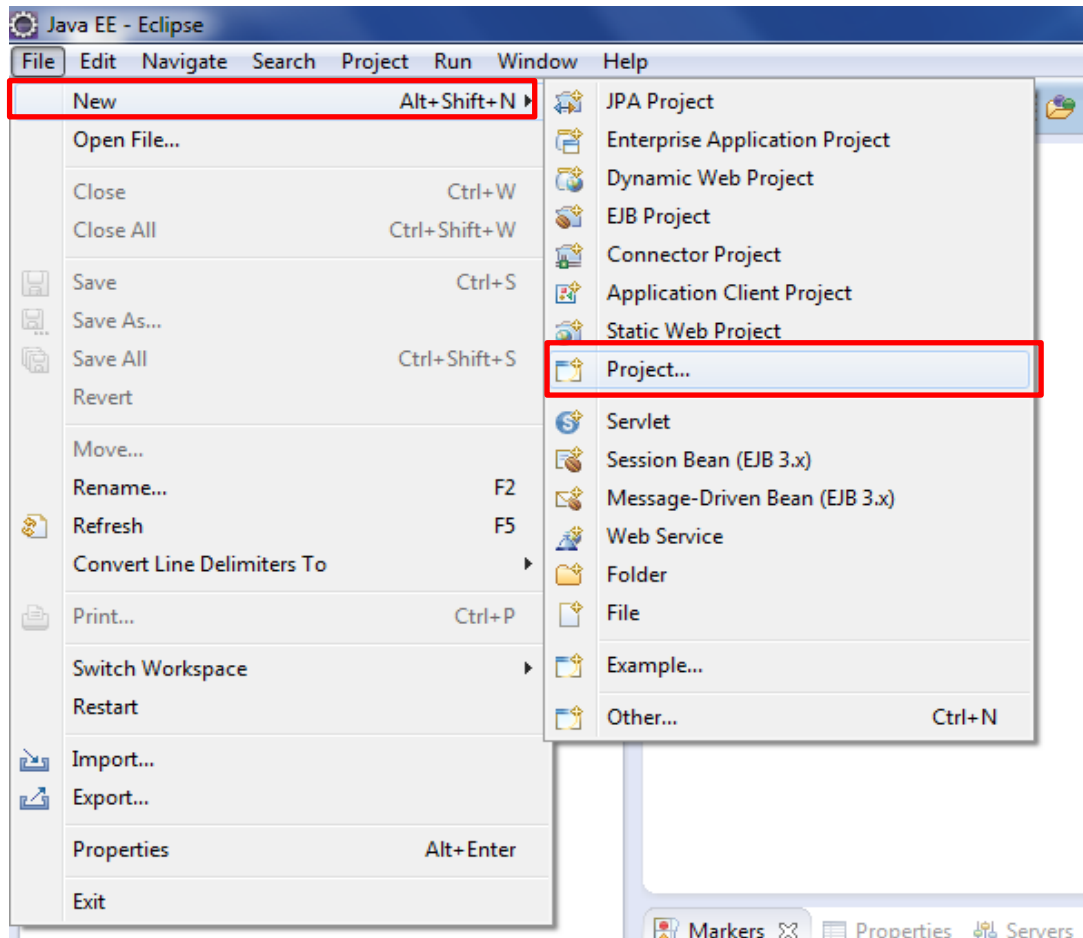


IDEs – Criação do Ambiente

Após fechada a janela anterior, será exibida a janela abaixo, onde um novo projeto será criado.



IDEs – Criação do Ambiente



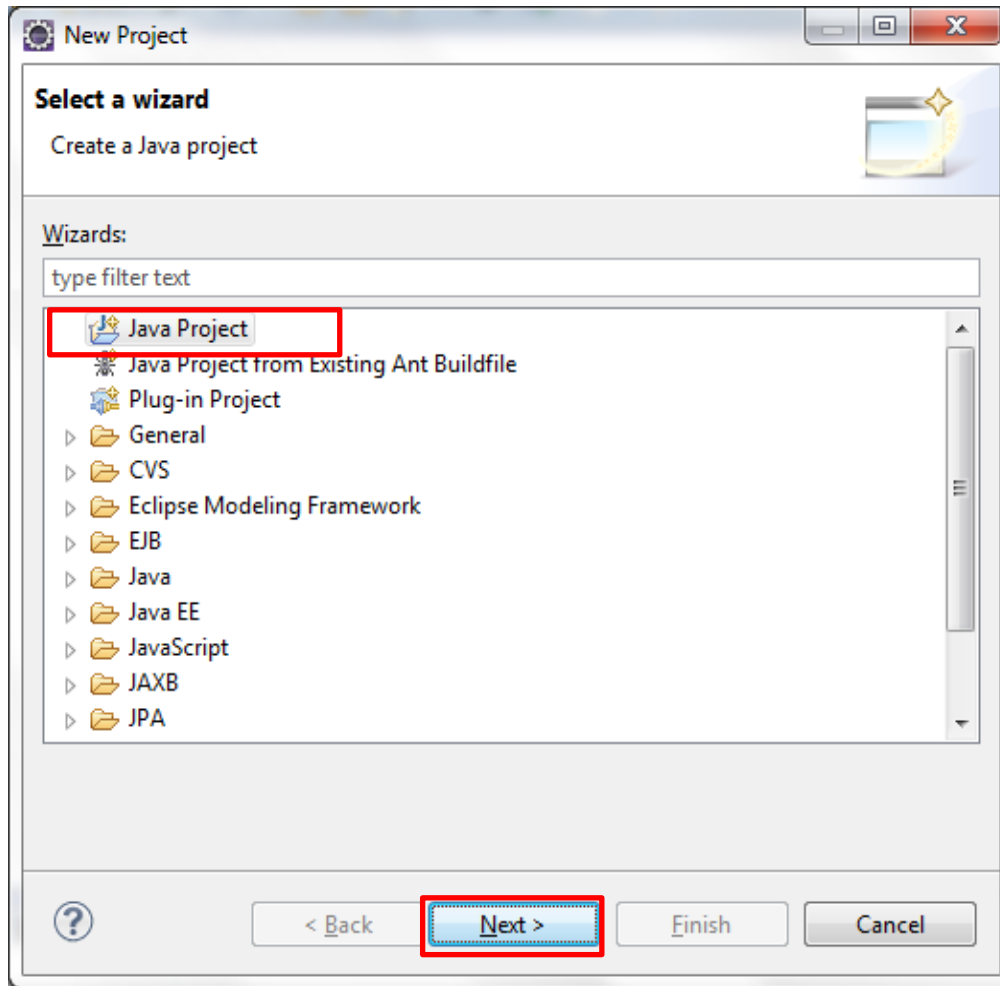
Criar um novo projeto.

Selecione menu **File**

Opção: **New**

A partir da opção New,
selecione a opção: **Project**

IDEs – Criação do Ambiente

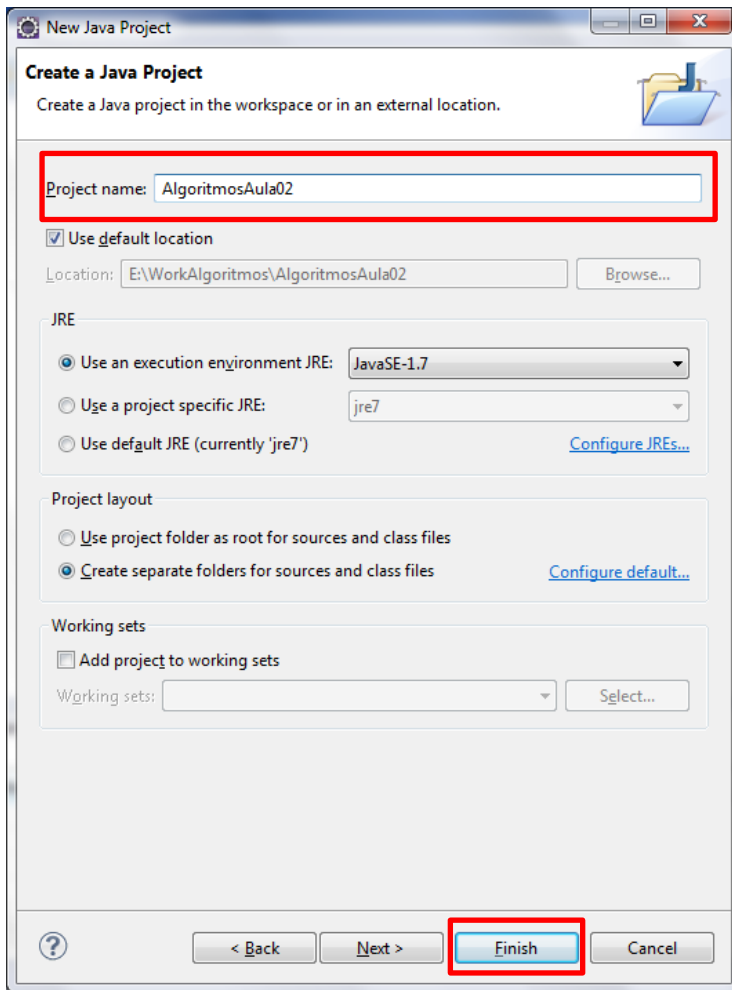


Selecione a opção:

Java Project

Clique em ***Next***

IDEs – Criação do Ambiente



Dar um nome ao Projeto.

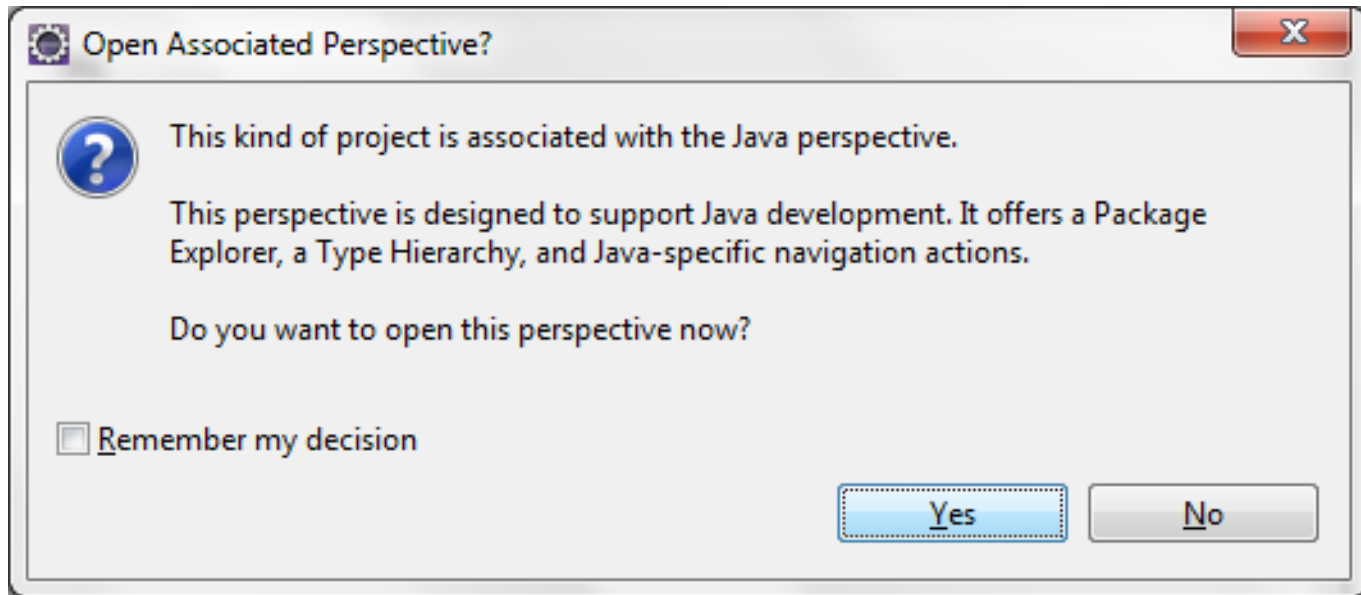
Sugestão para cada conjunto de aulas.

Nome do Projeto: **EstruturaRepetição**

Clique em **Finish**

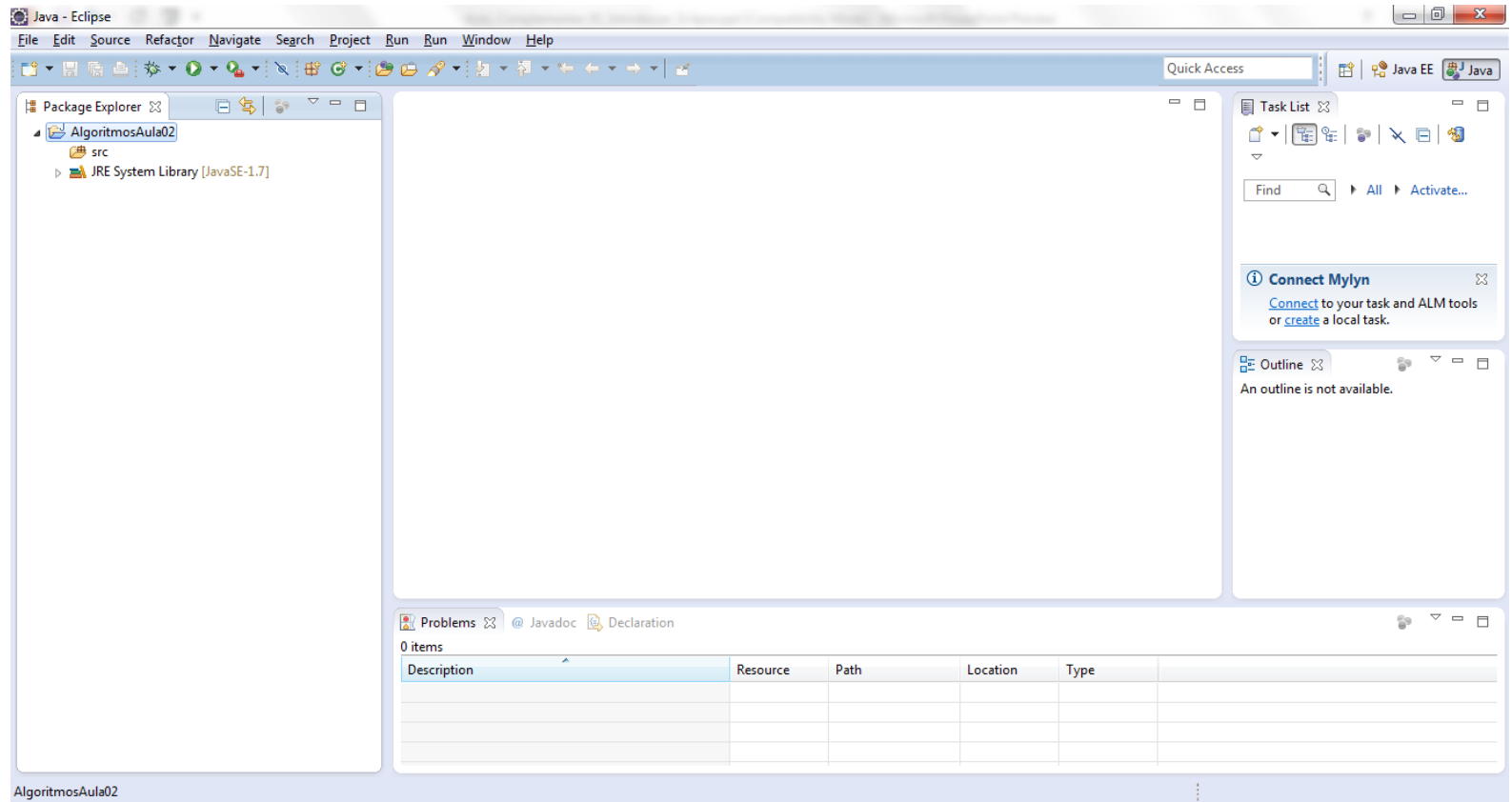
IDEs – Criação do Ambiente

Clique em **No**.

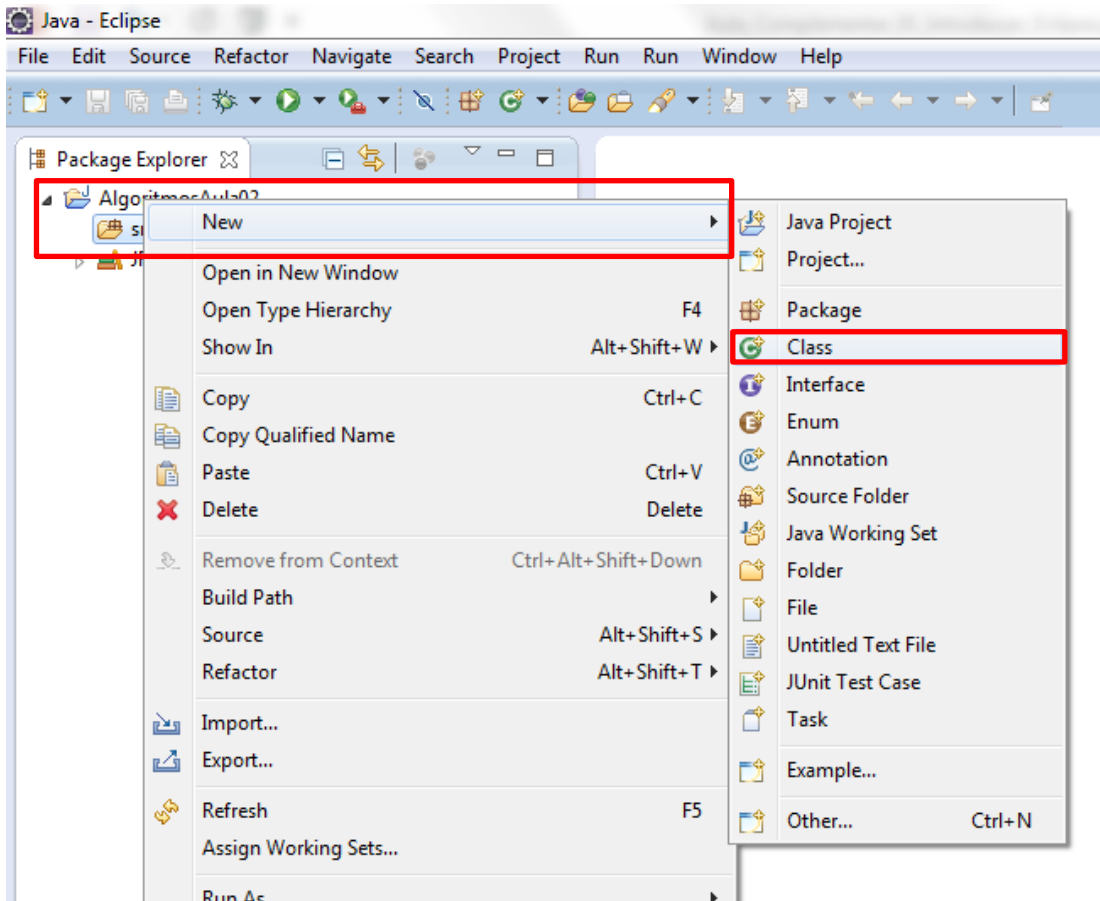


IDEs – Criação do Ambiente

Ambiente de desenvolvimento.



IDEs – Criação do Ambiente



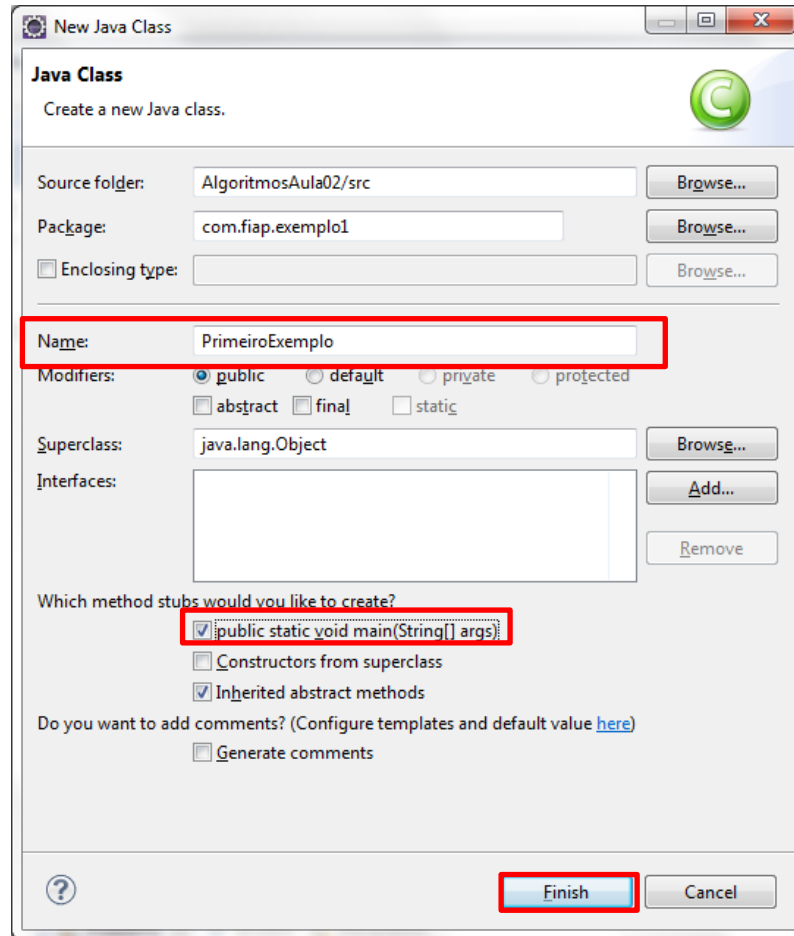
Criação da Classe.

Clique com o botão direito na pasta **src**

(source code = código fonte)

Selecione a opção **New** e em seguida na opção **Class**.

IDEs – Criação do Ambiente



Dar um nome a classe:

PrimeiroExemplo

Marque a opção:

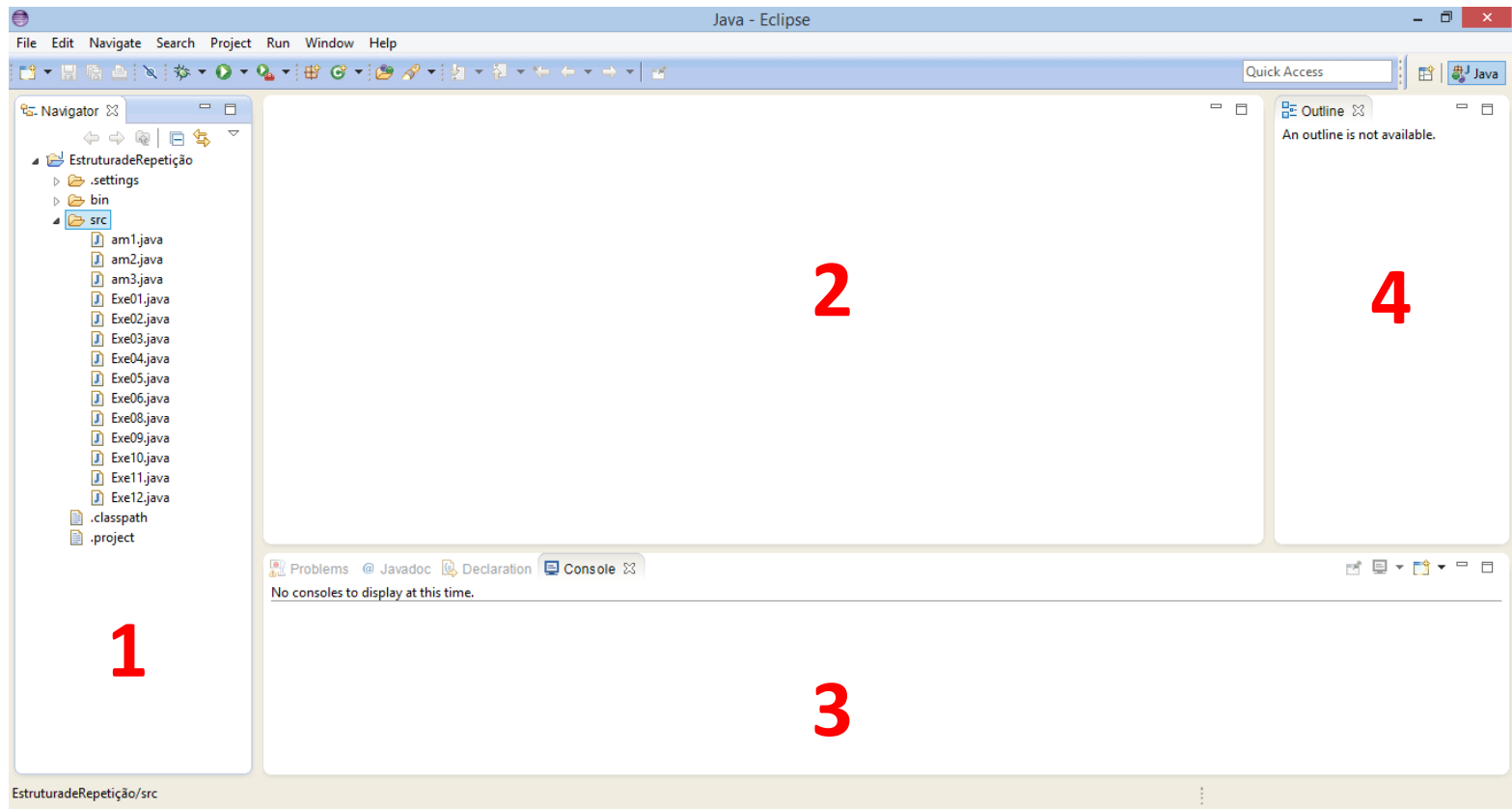
Public static void main(string[] args)

Clique em **Finish**

Sempre utilizar a convenção de nomenclatura de pacotes e classes.

IDEs – Criação do Ambiente

Ambiente configurado. A partir daqui já é possível iniciar os primeiros algoritmos em Java.



IDEs – Criação do Ambiente

1 - *Package Explorer*

O Package Explorer é um visualizador simples e elegante dos seus *projetos*. Um *projeto* representa toda a estrutura do seu programa, armazenando os arquivos-fonte (.java), os bytecodes (.class), as configurações gerais do ambiente para o projeto, eventuais arquivos de backup e outros arquivos inerentes ao escopo do programa (por exemplo, quaisquer possíveis imagens relacionadas ao projeto).

2 - *Editor de Texto*

O editor de textos do Eclipse, denota as palavras-chave de Java™ em letras destacadas para facilitar a leitura do código. Uma grande funcionalidade das principais IDEs atuais é a detecção de erros de compilação em tempo de implementação. O editor de textos do Eclipse, ao perceber um erro de sintaxe (e até alguns poucos erros de lógica), imediatamente marca em vermelho o trecho que ele supõe estar errado, além de indicar as possíveis causas do erro e sugerir algumas soluções.

IDEs – Criação do Ambiente

3 – Console

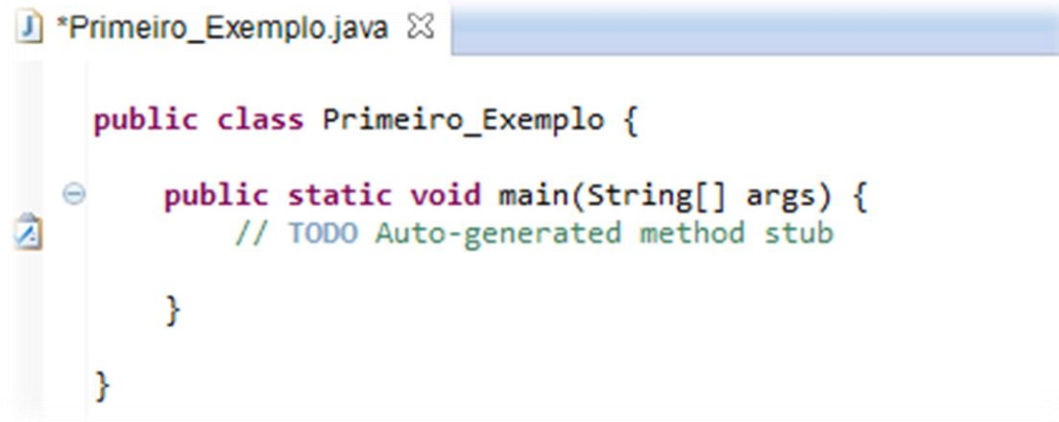
O Eclipse, assim como o BlueJ, oferece um terminal de console para a entrada de dados do teclado pela stream de entrada padrão e a saída de dados que o programa escreve na stream de saída padrão, exatamente como estamos habituados no BlueJ ou usando o JDK.

4 – Outline

A janela Outline funciona semelhantemente ao Package Explorer, sendo que voltada para a estrutura interna do seu arquivo .java - frequentemente a sua classe. Existem ícones diferentes para cada parte do arquivo.

Método Main

No código fonte de qualquer programa Java, temos uma classe principal. Dentro dessa principal, devemos ter o método main que é o método principal de qualquer programa Java e sem esse método o programa não funciona. Isso é por que é pelo método main que o programa inicia sua execução.



```
*Primeiro_Exemplo.java ✕  
  
public class Primeiro_Exemplo {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
    }  
}
```


Método Main

*Primeiro_Exemplo.java

```
public class Primeiro_Exemplo {
```

```
    public static void main(String[] args) {  
        // TODO Auto-generated method stub
```

```
    }
```

```
}
```

public indica que o método main pode ser chamado por outro objeto.

static indica que o método main é um método de classe.

void indica que o método main não retorna nenhum valor.

Comandos de Entrada e Saída - Java

Utilizaremos os métodos encapsulados abaixo para entrada e saída de dados pelo console:

System.in: utilizado para a entrada de informações.

System.out: utilizado para saída de informações.

Utilizamos os métodos `print()` ou `println()`:

print(): Exibe o conteúdo sem inserir uma nova linha;

println(): Exibe o conteúdo inserindo uma nova linha.

Para entrada de dados utilizaremos a **classe Scanner**, é uma classe para tratamento de entradas (System.in).

Atenção: Importar a classe Scanner: `import java.util.Scanner;`

Exemplos de comandos de entrada:

// Criação e instanciação da variável utilizada para entrada de dados

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

// Declaração da variável e atribuição de um valor informado pelo usuário via console

```
double base = entrada.nextDouble();
```

Comandos de Entrada - Java

Classe Scanner: `import java.util.Scanner;`

Exemplos de funções desta classe:

Função	Funcionalidade
<code>next()</code>	Aguarda uma entrada em formato string com uma única palavra.
<code>nextLine()</code>	Aguarda uma entrada em formato string com uma única ou várias palavras.
<code>nextInt()</code>	Aguarda uma entrada em formato inteiro.
<code>nextByte()</code>	Aguarda uma entrada em formato inteiro.
<code>nextLong()</code>	Aguarda uma entrada em formato inteiro.
<code>nextFloat()</code>	Aguarda uma entrada em formato número fracionário.
<code>nextDouble()</code>	Aguarda uma entrada em formato número fracionário.

Tipos de Dados

As *variáveis* são declaradas após a especificação de seus tipos. Os tipos de dados mais utilizados são: `int` (para números inteiros), `float` e `double` (para números reais), `char` (para um caractere), `String` (para vários caracteres) e `boolean` (para verdadeiro ou falso).

Tipo	Faixa de valores	Tamanho (aproximado)
<code>byte</code>	−128 a 127	8 bits
<code>char</code>	0 a 65.535	16 bits
<code>short</code>	−32.768 a 32.767	16 bits
<code>int</code>	−2.147.483.648 a 2.147.483.647	32 bits
<code>long</code>	−9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64 bits
<code>float</code>	-3.4×10^{-38} a 3.4×10^{38}	32 bits
<code>double</code>	-1.7×10^{-308} a 1.7×10^{308}	64 bits
<code>boolean</code>	true ou false	indefinido

Operadores e Funções Predefinidas

%	x % y	Obtém o resto da divisão de X por Y.
+=	x += y	Equivale a $X = X + Y$.
-=	x -= y	Equivale a $X = X - Y$.
*=	x *= y	Equivale a $X = X * Y$.
/=	x /= y	Equivale a $X = X / Y$.
%=	x %= y	Equivale a $X = X \% Y$.
++	x++	Equivale a $X = X + 1$.
++	y = ++x	Equivale a $X = X + 1$ e depois $Y = X$.
++	y = x++	Equivale a $Y = X$ e depois $X = X + 1$.
--	x--	Equivale a $X = X - 1$.
--	y = --x	Equivale a $X = X - 1$ e depois $Y = X$.
--	y = x--	Equivale a $Y = X$ e depois $X = X - 1$.

Comandos de Saída - Java

Exemplos de comandos de saída:

// Saída de dados via console

```
System.out.println("Informe o valor da base: ");
```

// Saída de dados via console contendo texto informativo e o resultado da área calculada

```
System.out.println("Valor da área: " + area);
```

Operador de Concatenação

Comandos de Entrada e Saída - Java

```
System.out.println("Digite o seu nome:");  
nome = entrada.next();
```

```
System.out.println("Digite sua idade: ");  
idade = entrada.nextInt();
```

```
System.out.println("Digite seu peso: ");  
peso = entrada.nextDouble();
```

```
System.out.println("Digite a cor dos olhos: ");  
corOlho = entrada.next().charAt(0);  
corOlho = Character.toUpperCase(corOlho);  
  
System.out.println("Digite cor dos olhos:");  
corOlhos = entrada.next().toLowerCase().charAt(0);
```

Converte para Maiúsculas

```
System.out.println("Digite cor dos olhos:");  
corOlhos = entrada.next().toLowerCase().charAt(0);
```

Converte para Minúsculas

O método `toUpperCase ()` converte uma string em letras maiúsculas.

Nota: O método `toUpperCase ()` não altera a string original .

Dica: Use o método `toLowerCase ()` para converter uma string em letras minúsculas.

Estrutura de Seleção

Simples

```
if (idade > 50) {
    // cont50=cont50+1;
    cont50++;
}
```

Composta

```
if (media>6 && numeroAula>40){
    msg= "Aprovado";
    contAprov++;
}else{
    msg= "Reprovado";
    contReprov++;
}
```

Encadeada

```
if (idade < 15) {
    tot1++;
} else {
    if (idade >= 16 && idade <= 30) {
        tot2++;
    } else {
        if (idade >= 31 && idade <= 45) {
            tot3++;
        } else {
            if (idade >= 46 && idade <= 60) {
                tot4++;
            } else {
                tot5++;
            }
        }
    }
}
```

Estrutura de Seleção

Escolha

```
switch (voto){  
  case 1:  
    cand1++;  
  break;  
  case 2:  
    cand2++;  
  break;  
  case 3:  
    cand3++;  
  break;  
  case 4:  
    cand4++;  
  break;  
  case 5:  
    cand5++;  
  break;  
  case 6:  
    cand6++;  
  break;  
}
```

```
switch (op){  
  case "A":  
    contA++;  
  break;  
  case "B":  
    contB++;  
  break;  
  case "C":  
    contC++;  
  break;  
  case "D":  
    contD++;  
  break;  
  case "E":  
    contE++;  
  break;  
  default:  
    System.out.println("Opção inválida");  
    contador--;  
}
```

Condição

Na linguagem JAVA, todas as condições devem estar entre parênteses.

Exemplos:

```
if (x == 3)
    System.out.println("Número igual a 3");
```

No exemplo anterior, existe apenas uma condição que, obrigatoriamente, deve estar entre parênteses.

```
if (X > 5 && X < 10)
    System.out.println("Número entre 5 e 10");
```

No exemplo anterior, existe mais de uma condição, as quais, obrigatoriamente, devem estar entre parênteses.

```
if ((X == 5 && Y == 2) || Y == 3)
    System.out.println("X é igual a 5 e Y é igual a 2, ou Y é igual a 3");
```

No exemplo anterior, existe mais de uma condição e mais de um tipo de operador lógico, portanto, além dos parênteses que envolvem todas as condições, devem existir ainda parênteses que indiquem a prioridade de execução das condições. Nesse exemplo, as condições com o operador &&, ou seja, (X == 5 && Y == 2), serão testadas, e seu resultado será testado com a condição || Y == 3.

Estrutura de Repetição

Para

```
for (int i = 0; i<10;i++){  
    System.out.println("Estrutura de Repetição Para em Java");  
}
```

Enquanto

```
int i=0;  
  
while(i<10){  
    System.out.println("Estrutura de Repetição Enquanto em Java");  
    i++;  
}
```

Repita

```
int i=0;  
do {  
    System.out.println("Estrutura de Repetição Repita");  
    i++;  
}while(i<10);
```

Aplicação no Java

1. Faça um programa que receba a idade, a altura e o peso de 20 pessoas. Calcule e mostre:
 - a) A quantidade de pessoas com idade superior a 50 anos:
 - b) A média das alturas das pessoas com idade entre 10 e 20 anos:
 - c) A porcentagem de pessoas com peso inferior a 40 quilos entre todas as pessoas analisadas:

Primeiro exemplo de algoritmo implementado em Java

1º

Insira a classe Scanner, no começo do código:

//Classe utilizada para entrada de dados via teclado

```
import java.util.Scanner;
```

Ou

CTRL+Shift+O → Realiza **IMPORT** automático dos pacotes dependentes

```
//Classe utilizadas para entrada de dados
import java.util.Scanner;
```

```
public class Exe01 {

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

Primeiro exemplo de algoritmo implementado em Java

2º

Crie as variável para entrada de dados embaixo do void main:

```
public class Exe01 {  
    public static void main(String[] args) {  
  
        // Criação da variável para entrada de dados  
        Scanner entrada = new Scanner (System.in);
```

Primeiro exemplo de algoritmo implementado em Java

3º

Declaração das variáveis e atribuição de valores iniciais.

```
int idade=0, contador=0, quantId=0, quantAl=0;  
double altura=0, peso=0, somaAltura=0, pesoMenor=0;
```


Primeiro exemplo de algoritmo implementado em Java

4º

Criar estrutura de Repetição e solicitar entradas:

```
while (contador<20) {  
  
    System.out.print("Digite idade:");  
    idade=entrada.nextInt();  
  
    System.out.print("Digite altura:");  
    altura=entrada.nextDouble();  
  
    System.out.print("Digite peso:");  
    peso=entrada.nextDouble();  
  
    //controle da repetição  
    contador++;  
}
```

Primeiro exemplo de algoritmo implementado em Java

5º

Criar estrutura de seleção:

```
//SELEÇÃO
//A)
if(idade>50) {
    cont50++;
}
//B)
if(idade>10 && idade<20) {
    contAlt++;
    somaAlt+=altura;
}
//C)
if(peso<40) {
    contPeso++;
}

} //Fecha Repetição
```

Primeiro exemplo de algoritmo implementado em Java

6º

Saídas:

```
System.out.println("Quantidade de pessoas com idade > 50:" + cont50);

if (contAlt == 0) {
    System.out.println("Média das alturas:0");
} else {
    System.out.println("Média das alturas:" + somaAlt / contAlt);
}

System.out.println("Porcentagem de peso inferior 40:"+(cont40/3)*100);

}

}
```

Classe Decimal Format (java.text.DecimalFormat)

A classe **DecimalFormat** é utilizada para realizar formatação de números decimais.

Podemos formatar números inteiros, decimais, notação científica, valores monetários, porcentagens.

Exemplo:

// Criação e instanciação da variável utilizada para receber o valor de um número à ser formatado

```
DecimalFormat numFormatado = new DecimalFormat("#,##0.00");
```

Onde: 0 → equivale a um dígito à ser exibido obrigatoriamente no display.

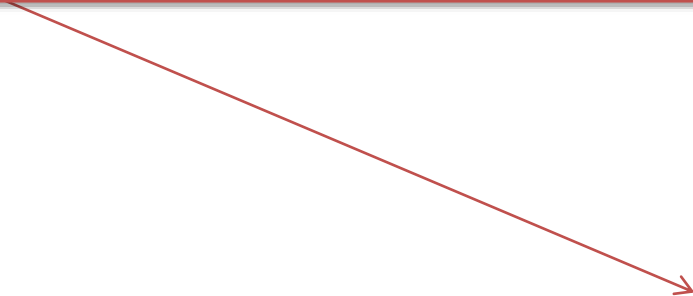
 # → equivale a um dígito à ser exibido opcionalmente, pois na ausência o zero será suprimido.

Atenção: Importar a classe DecimalFormat → import java.text.DecimalFormat;

Fonte: <http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html>, acessado em 13/02/2013.

Classe Decimal Format (java.text.DecimalFormat)

```
public static void main(String[] args) {  
    Scanner entrada = new Scanner (System.in);  
    DecimalFormat formataMoeda = new DecimalFormat("R$ ###0.00");  
  
    System.out.println("Total de transações à vista:" + formataMoeda.format(valorV));  
    System.out.println("Total de transações a prazo:" + formataMoeda.format(valorP));  
    System.out.println("Total de transações:" + formataMoeda.format(valorTotal));  
}
```



Condição

Comparação com campos String

```
if (categoria.equalsIgnoreCase("ST")){  
    imposto = 0.12;
```

Comparação com campos Numéricos

```
if (numDiarias==3){  
    desconto = 0;
```

Comparação com campos CHAR

```
if (corOlho=='A') {  
    quantAzul++;  
}
```

Validações

- Validação tipo de conta 1, 2 ou 3:

```
System.out.print("Digite o tipo de conta:");
tipo = entrada.nextInt();

while (tipo < 1 || tipo > 3) {
    System.out.print("Digite o tipo de conta corretamente:");
    tipo = entrada.nextInt();
}
```

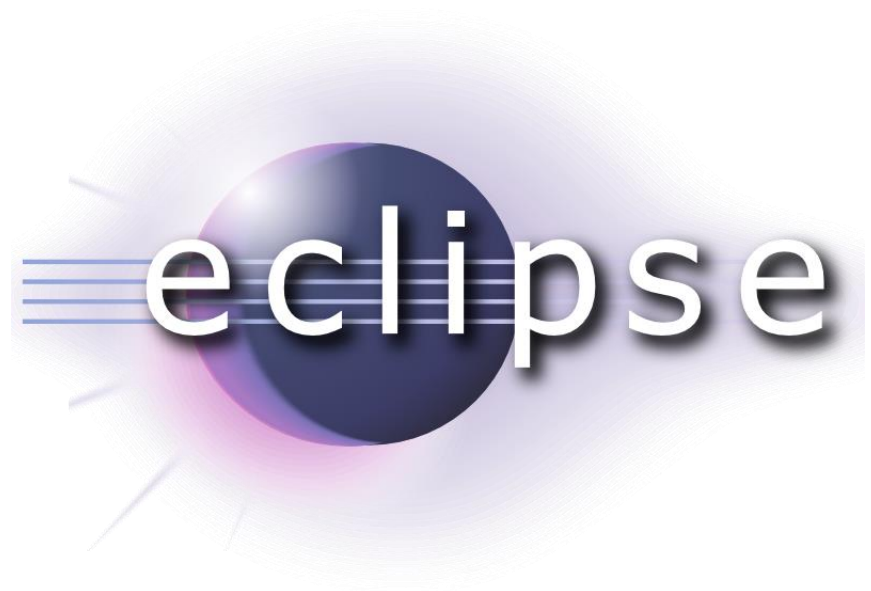
- Validação código transação V ou P:

```
System.out.print("Digite o código da transação:");
codigo=entrada.next();

while (!(codigo.equalsIgnoreCase("v")) && !(codigo.equalsIgnoreCase("P"))){
    System.out.print("Digite o código da transação corretamente:");
    codigo=entrada.next();
}
```

FERRAMENTA UTILIZADA

— <http://www.eclipse.org/downloads/>





Abaixo seguem não todos, mas as principais teclas de atalho encontradas no Eclipse IDE:

CTRL + Espaço: Auto-completa uma palavra, é o comando mais utilizado pois através dele você pode conhecer todos os métodos, classes ou comandos presentes da linguagem configurada em seu Eclipse, além de exibir toda documentação deles (Isso é quando existe documentação em um método ou classe de terceiros).

CTRL + Shift + S: Salva todos os arquivos, é mais produtivo do que realizar o comando **CTRL + S** que salva somente um único arquivo por vez.

CTRL + Shift + O: Busca e inclui todos os imports que serão utilizados no seu código. É muito útil, pois evita agente pesquisar import por import de cada framework instalado no projeto.

CTRL + Shift + F: Essa dica é para os programadores preguiçosos ou desorganizados que não consegue indentar o próprio código corretamente, esse comando formata todo seu código, respeitando todas as tabulações e espaços, tornando seu código limpo, organizado e compreensivo.

Alt + Shift + R: Para realizar esse comando primeiro selecione o nome de qualquer variável, método ou classe e depois realize o comando para renomear todas as ocorrências da classe, método ou variável que existir no projeto.


Alt - Shift + M: Extrai uma determinada porção de código selecionado para um novo método, ou seja, selecione um conjunto de linhas, realize este comando que irá surgir uma janela para configurar o nome do método e as regras desse método (**private**, **public**, **protected**), é muito útil, pois ele evita que dupliquemos códigos repetitivos.

Obs.: Esses comandos são os essenciais, existem muitos outros que não foram citados aqui, além do próprio Eclipse permitir a configuração de novos comandos, para fazer isso acesse o menu: **Window -> Preferences -> General Keys**.

| *Próxima aula estudaremos*

- ❑ Continuação através de exercícios.





Copyright © 2024 Prof^a. Evelyn Cid

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).