PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

Atividade: Conhecendo a Programação Orientada a Objetos
Tema: Usando uma nova IDE - Eclipse

INDICADORES ASSOCIADOS

2 – Analisa e avalia o funcionamento de computadores e periféricos em ambientes computacionais.

3 – Codifica programas computacionais utilizando lógica de programação e respeitando boas práticas de programação.

5 – Desenvolver capacidades linguísticas de modo, a saber, usar adequadamente a linguagem oral e escrita em diferentes situações e contextos.

6 – Conhecer o caráter do conhecimento científico aplicando a metodologia científica e utilizando redação acadêmica na realização de pesquisas, na escolha de métodos, técnicas e instrumentos de pesquisa.

8 – Utilizar estruturas de dados definindo-as e aplicando-as adequadamente nos programas.

11 – Desenvolver sistemas em diferentes segmentos.

19 – Projetar e desenvolver interfaces de sistemas utilizando modelos de desenvolvimento e respeitando normas de ergonomia de software.

21 – Projetar sistemas com base nos artefatos de análise utilizando ferramentas e linguagens de modelagem, respeitando padrões de projeto.

31 – Gestão das atividades.

32 – Cumprimento dos prazos.

Para a construção de soluções ligadas ao conceito de POO, faremos o uso da IDE Eclipse.

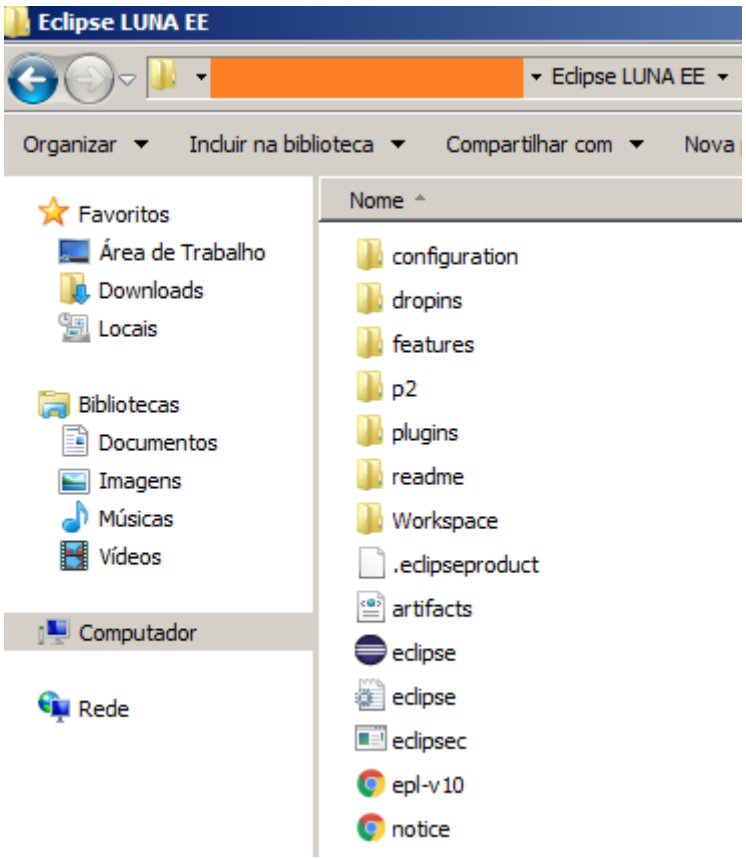
Por isso, esse material é fundamental para o seguimento de suas atividades.

INTRODUÇÃO AO ECLIPSE

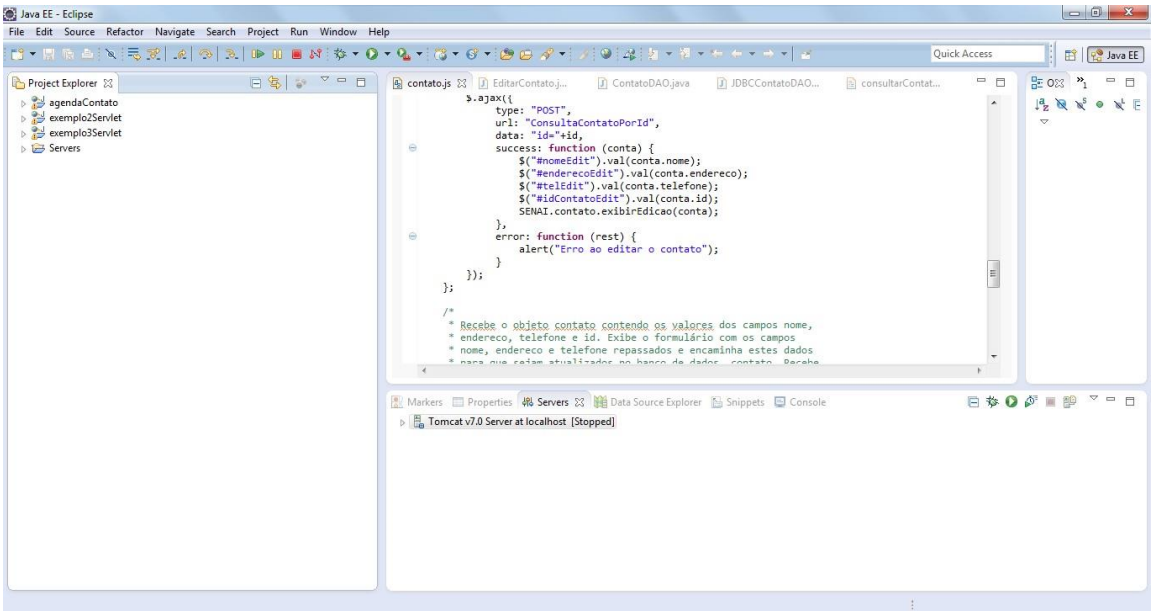
Como ferramenta de desenvolvimento, optamos trabalhar com o Eclipse que, por sua vez, trata-se de uma ferramenta muito boa e profissional. Essa permite a escrita dos programas em Java em um ambiente amigável e totalmente compatível com a linguagem.

O Eclipse não precisa ser instalado, mas sim copiado, para qualquer pasta de seu HD. Sugere-se que copie a pasta do Eclipse para a pasta Java, onde encontra-se o Kit Java de desenvolvimento.

O Eclipse possui diversas versões, para diferentes ocasiões e até linguagens, e elas podemser obtidas no endereço: http://www.eclipse.org/downloads/packages/. Nesta página, selecione “Eclipse IDE for Enterprise Java and Web Developers”, e faça o download para seu SO.



Como pode ver na imagem, usamos o Windows e a versão Eclipse **LUNA EE**. Porém, não se preocupe com isso, **baixe a versão mais atual** pois as diferençasbásicas serão mínimas. Segue uma imagem do Eclipse aberto:



Importante: para escrever programas Java utilizando o Eclipse como editor, deve primeiramente instalar o Kit de desenvolvimento Java pois ao executar o Eclipse este, automaticamente, procura pelo Kit Java — JDK, para fazer sua inicialização.

DESENVOLVIMENTO PROJETO INICIAL

Para que possa compreender como usar o ambiente e fazer seu (possivelmente) primeiro programa em Java, vamos criar um algoritmo que deve receber 10 números e depois exibir um menu para o usuário escolher o que quer ver entre:

- Todos os valores;
- Somente o primeiro valor;
- Somente o último valor;
- Mostrar somente os valores pares, separando por vírgula.

O programa deve também perguntar se o usuário deseja continuar executando ou encerrar.

Ressaltamos novamente que, apesar de ter sido desenvolvido em Java, este código não é orientado a objetos, ou seja, é utilizada a programação Imperativa no contexto procedural, pois nossa intenção é apenas a familiarização com a nova linguagem.

Para obtermos sucesso, siga os procedimentos para implementação e compreenda os recursos/instruções envolvidas.

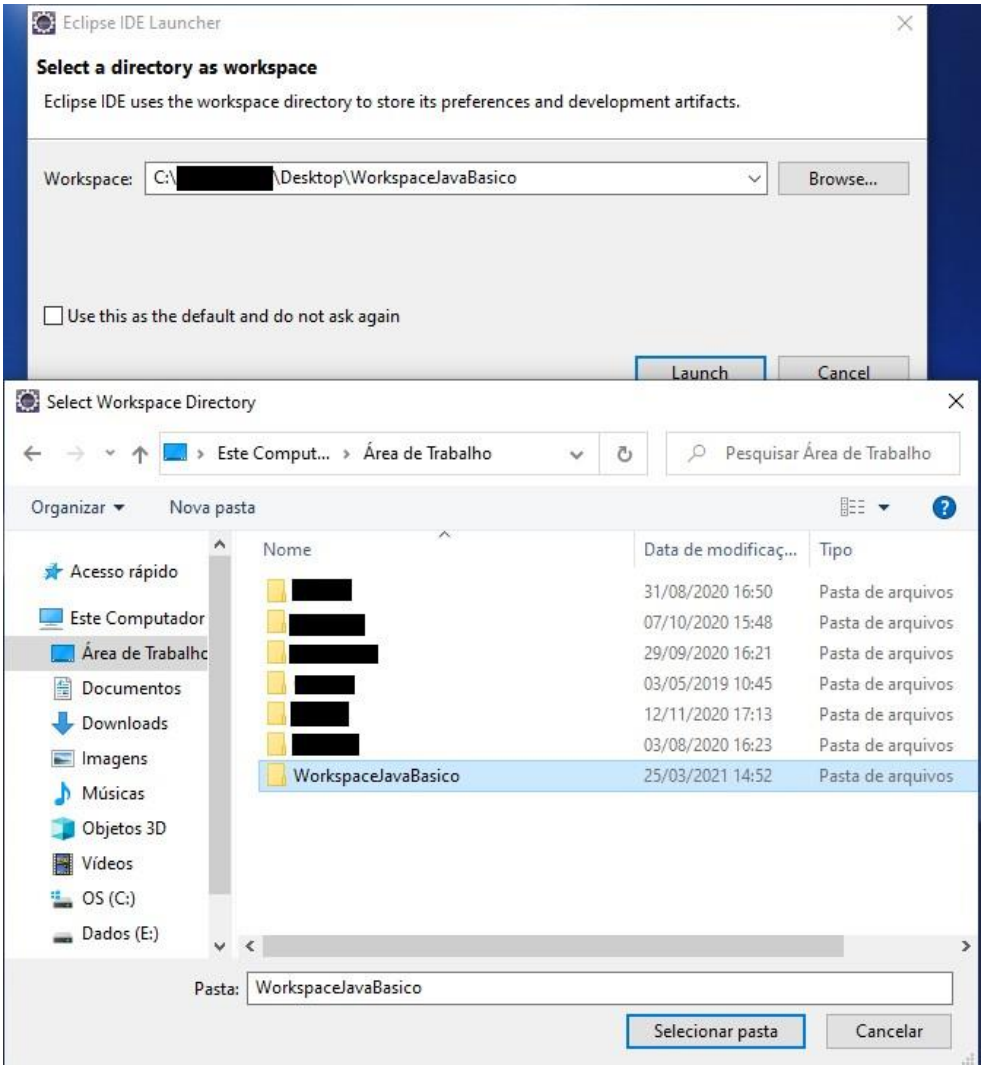
PROCEDIMENTOS

– Abra o Eclipse.

— Se estiver usando o Eclipse pela primeira vez em seu computador, você será solicitado a informar um local para sua **Workspace**. Ela nada mais é do que a pasta onde serão armazenados todos os seus projetos criados.

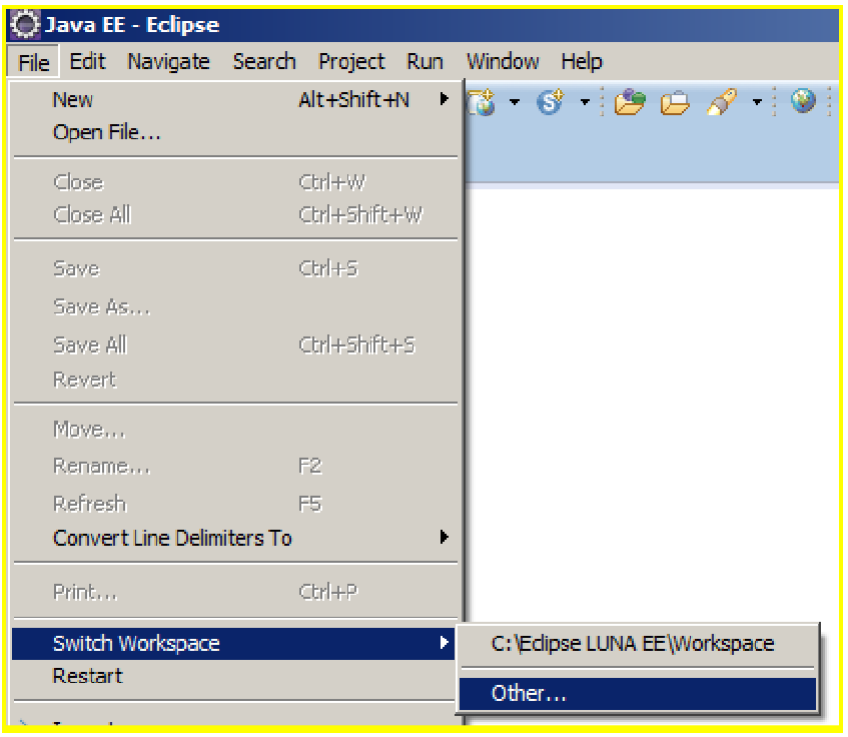
Faça o seguinte:

- Na área de trabalho, crie uma pasta chamada **WorkspaceJavaBasico**.
- Quando você abrir o Eclipse, na tela **Eclipse IDE Launcher**, você deverá clicar em **Browse** e selecionar como Workspace essa pasta que acabou de criar:



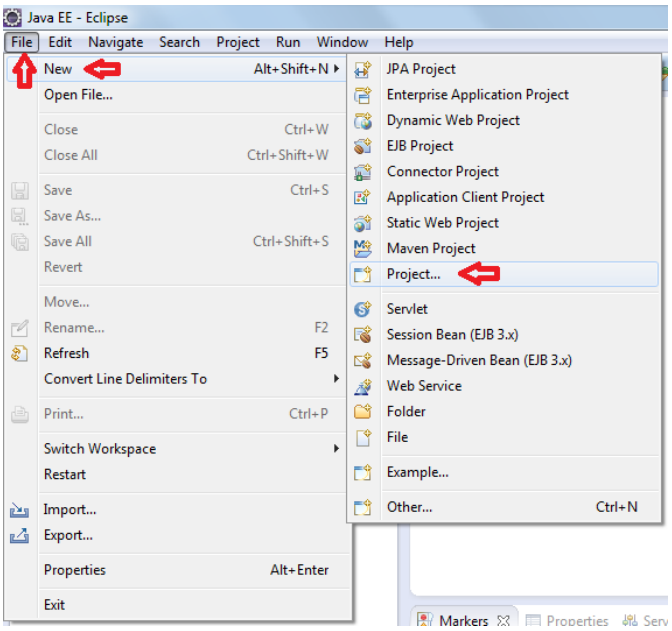
Observe que, por padrão, o Eclipse define como Workspace outra pasta. Altere para a sua e clique em **Launch**!

Tome cuidado! *Certifique-se de estar usando a* ***SUA*** *Workspace.* Caso você abra sem querer a Workspace padrão ou alguma outra que estiver "setada" quando abrir o Eclipse, não se preocupe, basta trocá-la em: File -> Switch Workspace-> Other, onde novamente a tela "Workspace Launcher" será aberta e você poderá setar a sua Workspace corretamente:

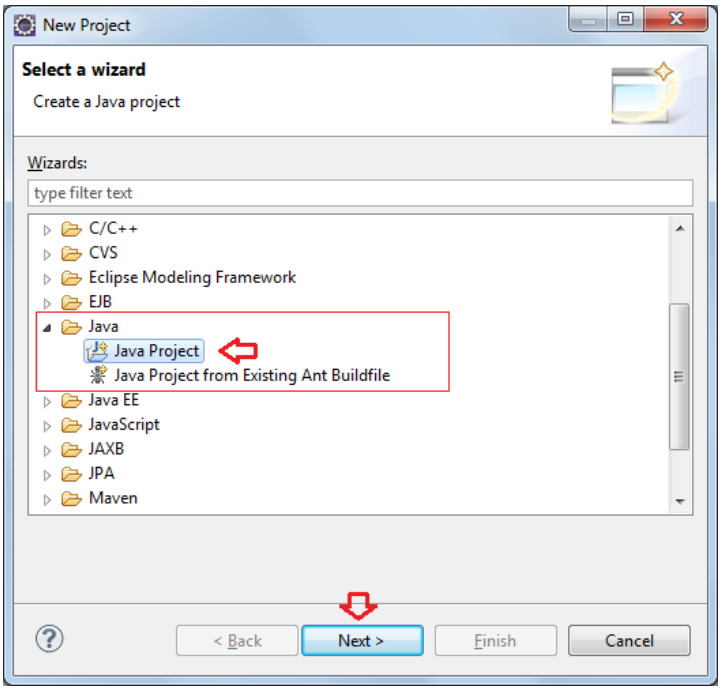


1 Crie um novo Projeto Java chamado **IntroducaoJava**. Para isso, selecione

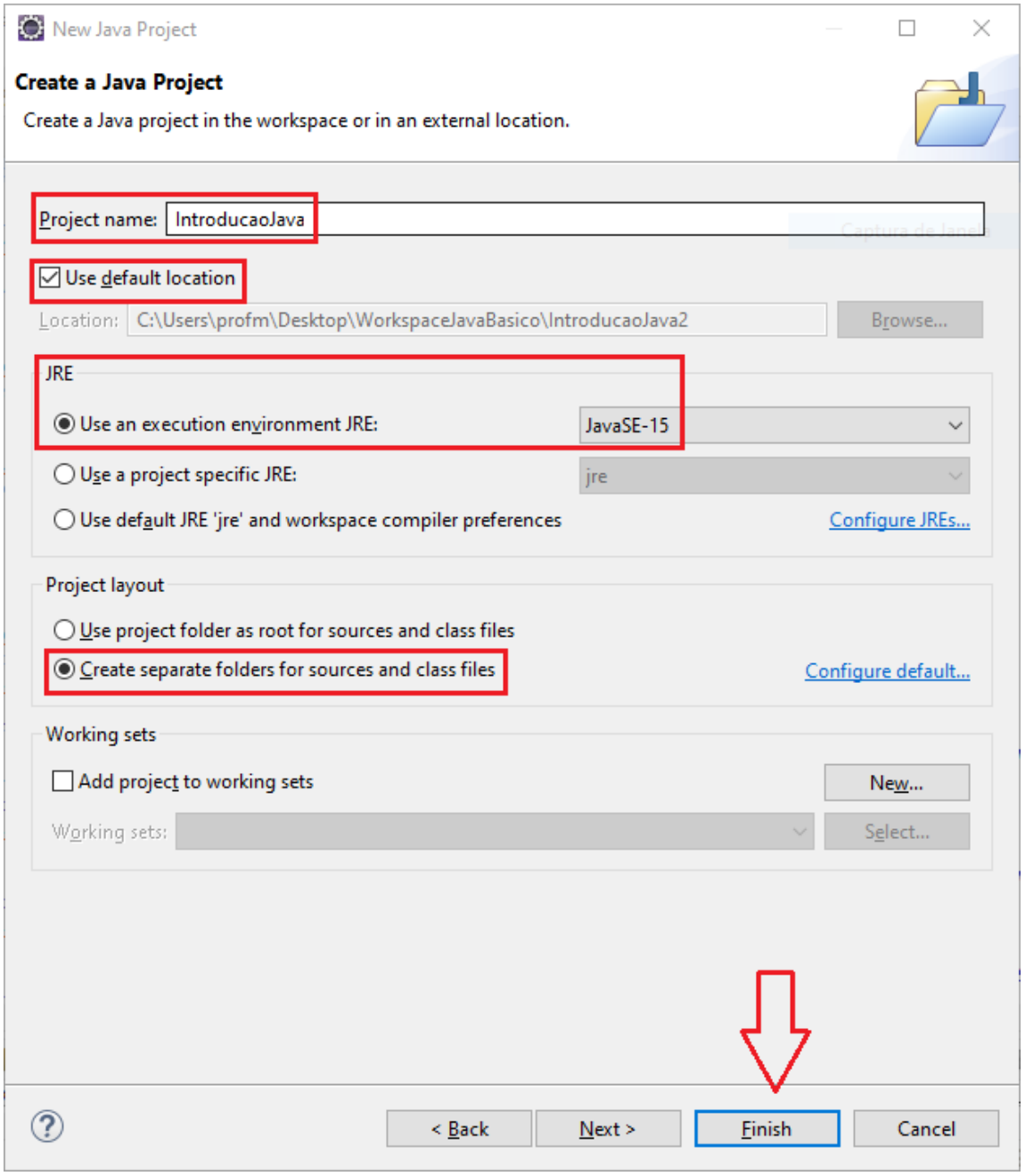
File >> New >> Project



2 Na sequência aparecerá a tela New Project, localize a pasta Java, abra sua estrutura, selecione a pasta Java Project e clique no botão Next, conforme mostrado a seguir:



A seguir será mostrada a tela para a criação do Projeto:



Note que em **Project Name**, deve informar o nome do projeto. Como sugerimos, identifique-o com o nome de **IntroducaoJava.** Note que como sugere o Java, a primeira letra de cada palavra no nome de um projeto inicia com letra **MAIÚSCULA**.

Particularidades

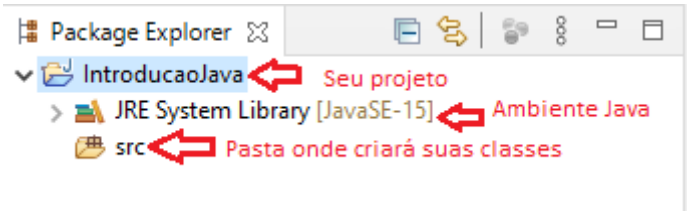
➔ Deixaremos a opção **"Use default location"** marcada. Isto significa que desejamos que este projeto seja criado na **Workspace** inicialmente determinada, ou seja, a sua;

Em JRE (Java Runtime Environment – Ambiente Java de Execução), deixamos o padrão fornecido pelo **Eclipse**, significando que desejamos executar nossa aplicação no ambiente padrão. Os valores podem variar, devido à versão instalada do JDK, mas deixe o que apareceu como padrão;

➔ E, por fim, em **Project layout**, layout do projeto ou estrutura do projeto, estamos configurando para que nossas classes possam ser criadas na pasta de origem **src**, possibilitando assim uma maior e melhor organização das mesmas.

Após clicar no botão **Finish**, pode ser solicitada a criação de um arquivo **module-info.java. NÃO CRIE-O!**

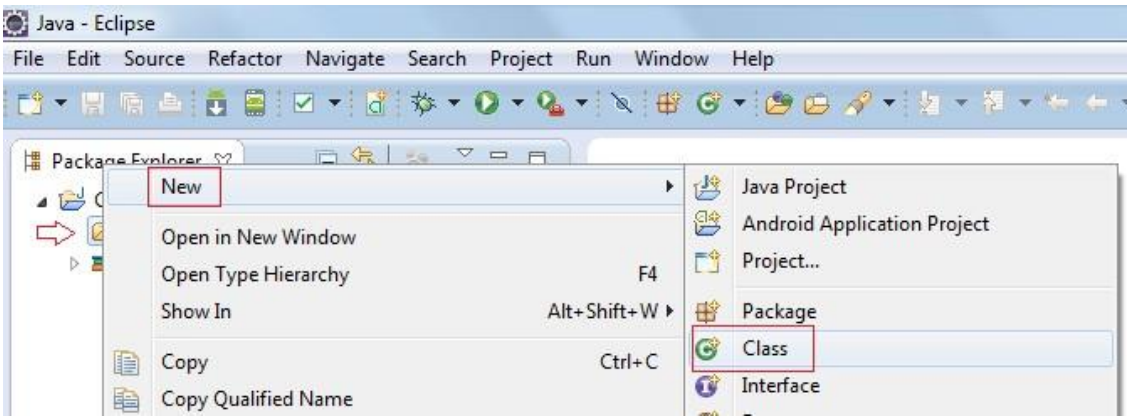
Ao término, obterá como resultado:



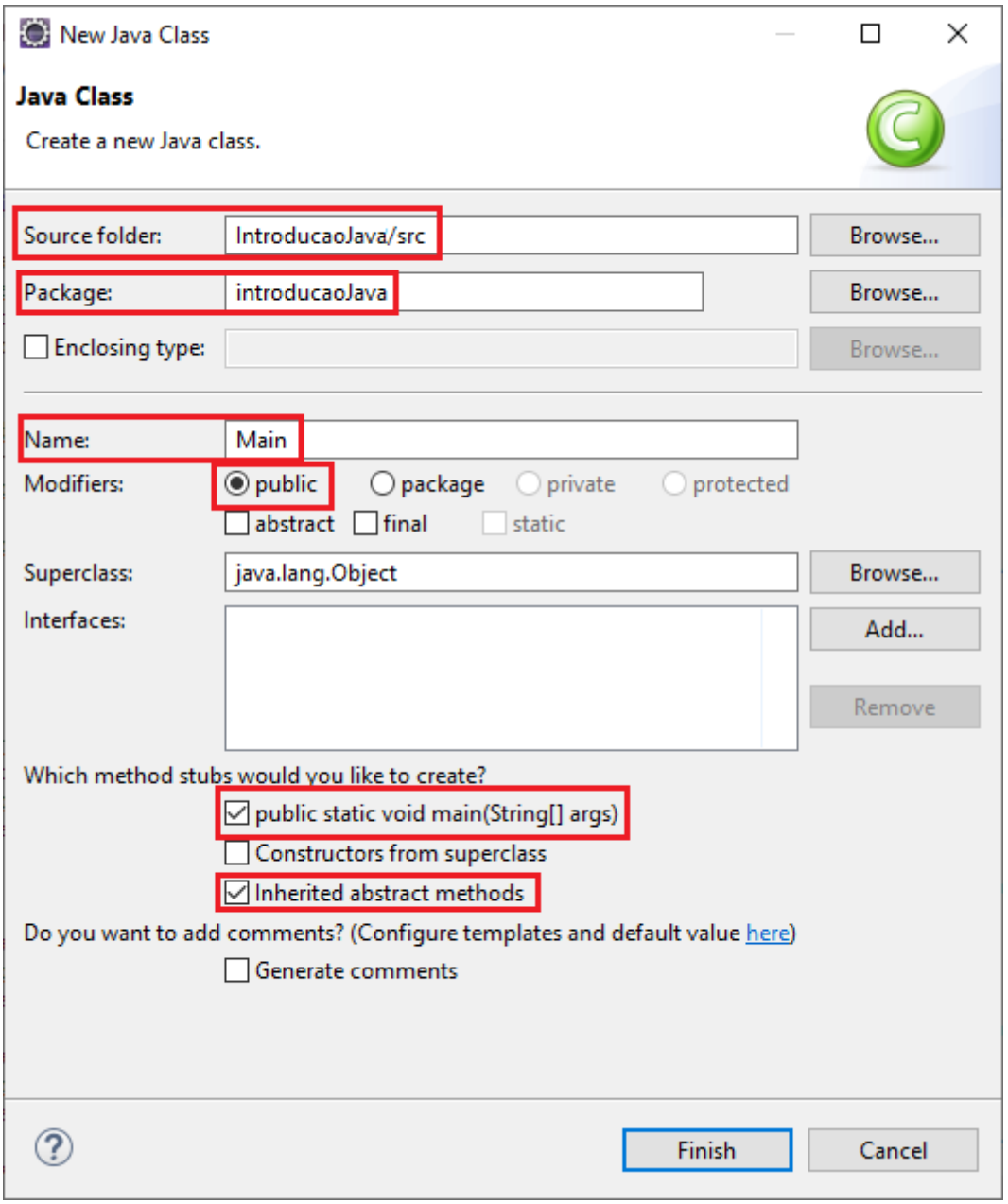
Observação final: toda aplicação **Java** é um projeto.

1 – Agora que já temos nosso projeto, podemos partir para o desenvolvimento de nossa aplicação. Você também deve lembrar que todo código em **Java** deveestar dentro de uma **classe**, porém não se preocupe ainda com o conceito desta, mais adiante detalharemos. Em **Java**, existe uma classe **principal**, conhecida como "**main**", pois é aquela onde declaramos o método "**main**", o ponto de partida de sua aplicação, ou seja, é desta classe que sua aplicaçãoserá **executada**, portanto, quando desenvolvemos uma aplicação **Java** devemos escrever nosso código principal na classe principal "**main**". Então vamos criá-la identificando-a com o nome de **Main**, já que nomes de classes devem, por **convenção Java**, serem iniciados com letras **MAIÚSCULAS**.

Mantendo a pasta **src** selecionada, clique com o botão direito do mouse e na sequência escolha a opção **New >> Class** como mostra a figura abaixo:



A seguir será mostrada a tela **Java Class** para que possamos determinar a configuração da classe que desejamos criar. Siga o sugerido:



Particularidades

➔ **Source folder** - indica a pasta em sua workspace onde a classe será salva. Mantenha o padrão para evitarmos problemas ao fazer seus backups;

➔ **Package** – falaremos um pouco mais sobre estes mais adiante, portanto não se preocupe, apenas saiba que o nome de um pacote deve iniciar com letras minúsculas, então repita o nome do projeto, apenas trocando o I de "**Introducao**" para um i minúsculo;

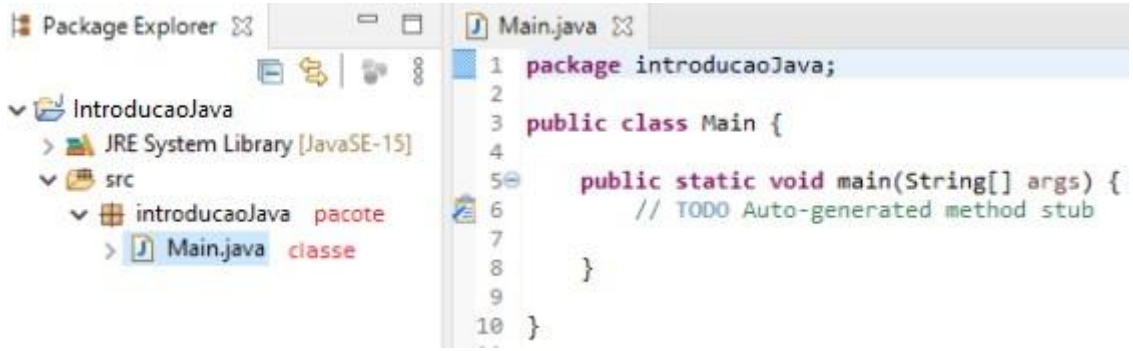
➔ **Name** – identificar o nome da classe, no caso **Main**;

➔ **Modifiers: public (Modificadores: público)** — deixamos selecionado significando que nossa classe "**main**" tem acesso público, ou seja, qualquer outra classe deste ou de outro projeto pode acessá-la. Por hora os conhecimentos sobre modificadores podem não ser muito compreendidos e relevantes por você, mas no futuro será muito importante;

➔ public static void main (String [] args) — selecionamos esta opção significando que esta classe é a principal de nossa aplicação pois estamos inserindo o método "**main**". Mais adiante detalharemos mais sobre métodos;

Inherited abstract methods — como nos foi fornecido este item selecionado, por hora vamos deixá-lo, mas não tem efeito nenhum em nossa aplicação. Mais adiante falaremos sobre Herança e daí as coisas começarão a fazer sentido.

Após clicar no botão **Finish**, obterá como resultado:



Particularidades

➔ **Declarando a classe** - a **parte interna da classe, onde escrevemos os códigos,** deve ficar entre **chaves**, da mesma forma que em outras linguagens que utilizam OO como **C/C++**, **PHP** e outras.

public class Main {//Início da classe MainParte interna da classe
} //Término da classe Main

➔ **O método "main"** – indica que a classe é a principal ou a primeira a ser executada, caso existam outras classes na aplicação.

A declaração **public static void** main (String [] args) significa em detalhes:

public — este método é de acesso público, podendo ser acessado por outras classes;

static – estático, indica que este método ficará armazenado na memória **RAM** do computador enquanto o programa for executado e, portanto, tudo que está dentro dele poderá ser acessado pelo programa a qualquer momento. Isto também é assunto para mais tarde;
void – este método não retorna nada para ninguém, apenas é executado.

(String [ ] args) – significa que se uma aplicação externa acessar estaclasse ela poderá passar informações no formato **array** do tipo **String**chamado **args**. Neste momento, também não é muito relevante, só por curiosidade.

public static void main (String[] args) {//Início do método main

```
    //Aqui escrevemos o código da nossa aplicação
```

} //Término do método main

2 – Já temos o projeto, nossa classe principal **Main** também, e agora vamos, efetivamente, iniciar a escrita dos códigos de nossa aplicação. Cabe aqui uma explicação importante: não estamos preocupados neste momento com interfaces bem elaboradas, tratamento de erros e tampouco com navegabilidade em programação, nosso grande objetivo a partir deste momento é a fundamentação da linguagem Java. Dito isto, vamos iniciar nossa aplicação.
IMPORTANTE: Junto a esse documento, você obteve acesso ao "**Tradutor VisuAlg - Java**". Esteja com ele sempre em mãos para tirar suas dúvidas sobre como escrever os comandos que aprendeu no VisuAlg na nova linguagem.
A aplicação deve solicitar ao usuário que este realize a entrada de dados de um número, para poder descobrir se ele é par ou ímpar e mostrar sua tabuada. Para isso, na classe Main, mais precisamente no método *main*, implemente o código das linhas destacadas em amarelo:

Main.java

```java
package introducaoJava;

//importando classe JOptionPane para usarmos os paineis de diálogo
import javax.swing.JOptionPane;

public class Main {

    public static void main(String[] args) {

        int[] valores = new int[10];
        //receber e guardar os 10 números
        for (int i=0; i<10; i++) {
            valores[i] = Integer.parseInt(JOptionPane.showInputDialog("Informe o "+(i+1)+"º valor: "));
        }

    }

}
```

Temos 2 novas seções, então vamos a elas.

Linhas 3 e 4: o comentário já ajuda a entendê-la, né? Aprofundando um pouco, o **Swing** é um toolkit GUI (Graphical User Interface), ou seja, um kit de ferramentas para melhorar a interface de uso da linguagem. Vamos usá-lo inicialmente (de modo equivalente ao **escreva()** e **leia()** do VisuAlg) de maneira simples, através da classe **JOptionPane**.

Linhas 10 a 14: Aqui recebemos os 10 valores e guardamos em um vetor, mas como muitas coisas são diferentes do VisuAlg, vamos falar detalhadamente sobre cada linha:

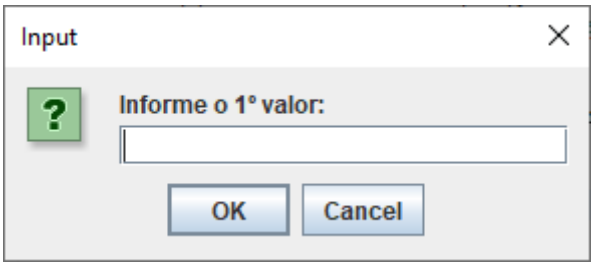
- **10**: criamos um vetor do tipo int chamado valores, com 10 posições. É importante destacar que **a primeira posição de um vetor é sempre 0 no Java,** então as posições válidas desse vetor são de 0 a 9**;**
- **12**: iniciamos um for (para-faça), cuja variável contadora é chamada **i** e começa de 0, que deve repetir seu código enquanto **i** for menor do que 10 (pois o vetor vai de 0 a 9), e o incremento em **i** a cada repetição é de 1 (i++). Explicando melhor os 3 parâmetros:
  - O **primeiro** equivale ao "*variável* de *valorinicio*" do VisuAlg, ou seja, define a **variável** que será usada e o seu **valor inicial** (permitindo ainda que ela seja **declarada** nesse momento, como pode ser visto pelo **int** ao lado da variável **i**);
  - O **segundo** é o mais diferente, pois ele pede uma **condição de entrada/repetição**, e **enquanto a condição for verdadeira, o for é executado**, então, no caso acima, enquanto i for menor que 10, o for é repetido;
  - O **último** contém o **incremento** ("passo" do VisuAlg), que no caso acima é

de 1 (**i++** é o mesmo que **i=i+1**), mas poderia ser usada qualquer conta (por exemplo, **i=i*3** para ir multiplicando por 3);

- **13**: usamos pela primeira vez a **JOptionPane**, mais especificamente seu método **showInputDialog** (Mostrar diálogo de entrada), que cria uma **janela** com o **texto** enviado como parâmetro **e um campo** digitável para o usuário inserir um valor - equivalente ao LEIA() do Visualg. Além disso, é executado o método **parseInt** da classe **Integer** para **transformar** o valor digitado em um valor do **tipo int (inteiro)**, que é o tipo do array (é o equivalente ao vetor, mas acostume-se com esse nome) que criamos para receber esse dado, o **valores**. Falando nele, como indicador da posição onde o valor deve ser guardado, usamos **i**, pois a cada repetição ela terá um valor diferente de nosso array. Por fim, veja que para atribuir um valor a uma variável, se utiliza o sinal "=" ao invés de "<-";
- **14**: fechamos as chaves do for.

Importante: por padrão o showInputDialog recebe os valores digitados pelo usuário como String e os retorna no mesmo formato, então caso a informação seja de outro tipo é necessário usar um método para convertê-la.

Por favor, salve a aplicação e execute para testar se está tudo OK! Como resultado deverá obter:



Pergunta: consegue explicar o (i+1) da linha 13?

É lógico que como resultado se obterá esta entrada de dados 10 vezes, pois foi isto que solicitamos em nosso código com o vetor e o *for*.

Com esta implementação, o usuário já consegue fornecer os números para os processos posteriores.

3 — Continuando a implementação de nossa aplicação **Main**, vamos agora, ainda dentro do método **main** e abaixo do código recém implementado, implementar as instruções para *exibir o menu ao usuário e criar a possibilidade*

de encerrar o programa. Implemente o código das linhas destacadas em amarelo a seguir, tendo muita atenção com a sua posição em relação aos códigos antigos.

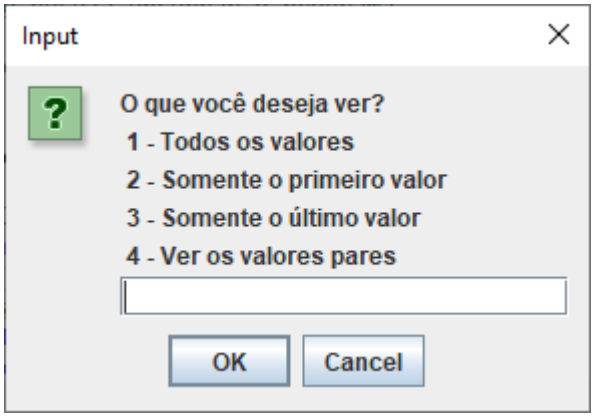
Main.java

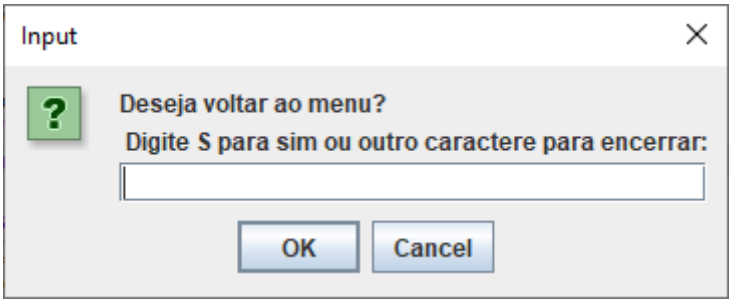
```
public class Main {

    public static void main(String[] args) {

        int[] valores = new int[10];
        //receber e guardar os 10 números
        for (int i=0; i<10; i++) {
            valores[i] = Integer.parseInt(JOptionPane.showInputDialog("Informe o "+(i+1)+"º valor: "));
        }

        //criando variável para armazenar se usuário deseja encerrar o programa
        String repetir;
        //início da estrutura de repetição para o menu
        do {
            //solicitando opção do menu
            int opcao = Integer.parseInt(JOptionPane.showInputDialog("O que você deseja ver?"
                    + "\n 1 - Todos os valores"
                    + "\n 2 - Somente o primeiro valor"
                    + "\n 3 - Somente o último valor"
                    + "\n 4 - Ver os valores pares"));
            //solicitando se usuário deseja voltar ao menu
            repetir = JOptionPane.showInputDialog("Deseja voltar ao menu? "
                    + "\n Digite S para sim ou outro caractere para encerrar:");
        //fim da estrutura de repetição para o menu
        }while (repetir.equals("S"));
    }

}
```

Vamos apenas explicar alguns pontos mais importantes que você deve ter notado, mas talvez não tenha entendido.

- O **“\n”** que aparece em alguns textos representa uma quebra de linha (um enter, digamos assim);
- Veja que a **String repetir** foi criada FORA do *do-while*, isso tem a ver com a visibilidade das variáveis no Java: uma variável pode ser criada em qualquer local de seu sistema, porém só pode ser vista dentro dos {} onde foi criada. Sendo assim, se ela fosse criada dentro do *do-while*, ela não seria visível na condição do *while* (pois ela fica fora das chaves da estrutura...);
- No *while*, veja o uso do *equals()*, um método para comparação de variáveis e valores do tipo String.

Agora teste sua aplicação para verificar se todas as rotinas estão funcionando. Além de ter que informar os 10 números, você deve ver essas duas caixas de diálogo:





Na primeira, como não criamos a lógica do menu, contanto que digite um valor, não terá problema. Já a segunda, você já pode preencher testando as situações: com um S, a caixa do menu deve aparecer novamente, mas com outro caractere o programa deve ser encerrado. Ah, “S” é diferente de “s”: para situações de comparação como essa o Java é **case-sensitive** (não sabe o que é isso? pesquise, por favor...).

Importante: não prossiga caso isso não aconteça. Verifique o código desde o teste anterior e corrija o necessário, pois algo está errado...

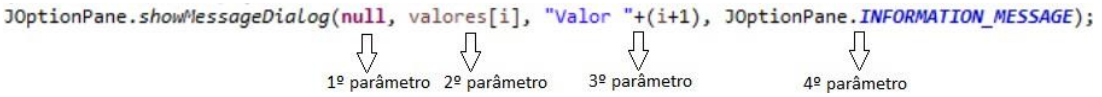
4 Agora vamos desenvolver a programação do menu em si. Para isso, acrescente em seu código as linhas na imagem cujos números estão marcados em amarelo, entre as linhas de código destacadas (aumentar o zoom deve ser necessário para que veja bem o código):

```
                + "\n 3 - Somente o último valor"
                + "\n 4 - Ver os valores pares"));

        //programação do menu
        switch (opcao) {
        //Mostrando todos os valores
        case 1:
            for (int i=0; i<10; i++) {
                JOptionPane.showMessageDialog(null, valores[i], "Valor "+(i+1), JOptionPane.INFORMATION_MESSAGE);
            }
        break;
        //Mostrando somente o primeiro valor
        case 2:
            JOptionPane.showMessageDialog(null, valores[0], "Primeiro valor", JOptionPane.INFORMATION_MESSAGE);
        break;
        //Mostrando somente o último valor
        case 3:
            JOptionPane.showMessageDialog(null, valores[9], "Último valor", JOptionPane.INFORMATION_MESSAGE);
        break;
        //Mostrando somente os valores pares
        case 4:
            String valoresPares = "";
            for (int i=0; i<10; i++) {
                if (valores[i]%2==0) {
                    if (valoresPares!="") {
                        valoresPares += ", ";
                    }
                    valoresPares += valores[i];
                }
            }
            JOptionPane.showMessageDialog(null, valoresPares, "Valores Pares", JOptionPane.INFORMATION_MESSAGE);
        break;
        //caso a opção seja inválida
        default:
            JOptionPane.showMessageDialog(null, "Opção inválida", "Erro", JOptionPane.WARNING_MESSAGE);
        break;
        }

        //solicitando se usuário deseja voltar ao menu
        repetir = JOptionPane.showInputDialog("Deseja voltar ao menu? "
                + "\n Digite S para sim ou outro caractere para encerrar:");
```

Veja que, como no VisuAlg, usamos o switch-case (escolha-caso) para o menu. Cada caso é uma das opções:

- **case 1:** O primeiro tem uma repetição de 0 a 9 pra mostrar cada valor separadamente com o método da **JOptionPane** chamado **showMessageDialog** (Mostrar diálogo de mensagem), que usamos para mostrar uma **mensagem com um botão de OK.** De seus **4 parâmetros**, vamos nos preocupar agora com o **segundo**, que consiste na **mensagem** que você deseja mostrar, e o **terceiro**, que é o **título** da caixa de diálogo;

Apenas exemplificando a ordem dos parâmetros no showMessageDialog():



- **case 2:** O segundo mostra apenas o primeiro valor, ou seja, o que está na posição 0;
- **case 3:** O terceiro mostra apenas o último valor, ou seja, o que está na

posição 9 (lembre-se: o vetor tem 10 posições, e como sempre começado 0, sua última posição é a 9);

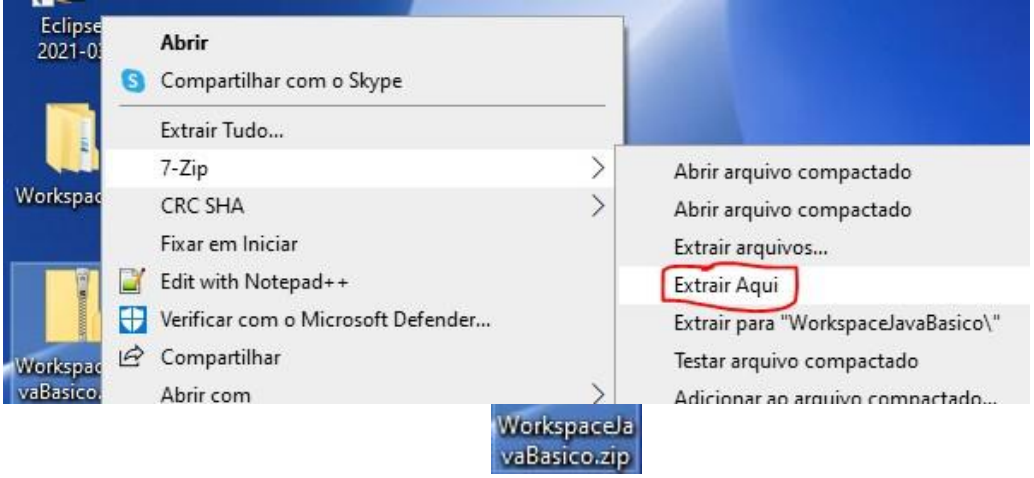
- **case 4:** Na quarta opção, **concatenamos** a variável **valoresPares** para ir **adicionando os novos resultados** nela mesma se (if) ela for par (veja que fazemos igual ao VisuAlg, usando o % para calcular o resto da divisão do valor por 2), separando todos por vírgula se já houver algum valor na variável (o **+=, das linhas 49 e 51,** é um resumo de, no primeiro caso, **valoresPares = valoresPares + ", "**), até formar a lista completa, exibida após o *for;*
- No *default*, programamos uma mensagem de erro, pois esse código será alcançado caso nenhum **case** seja realizado. Equivalente ao **outrocaso** do VisuAlg.

Agora sim, teste sua aplicação para verificar se todas as rotinas estão funcionando (é, é pra verificar cada opção do menu, inclusive a mensagem de erro, por favor). Agora vem mais uma pergunta: Percebeu alguma(s) possibilidade(s) de melhorias nesta solução? Se sim, discuta conosco na validação!

SALVANDO OS PROJETOS JAVA

Para salvar esse projeto, como ele já está dentro de sua Workspace, basta **fechar o Eclipse**, **compactar** a pasta da Workspace e salvá-la no Drive.

Veja que usamos o **programa 7zip**, pois ele é levinho e gratuito (nos computadores do SENAI, já temos ele instalado, mas para instalar em sua casa, pode baixá-lo aqui: https://www.7-zip.org/download.html), e o formato de compactação **zip**. O programa não é o mais relevante, se tiver outro, ok, mas o formato zip é crucial, não use outros, ok? Caso esteja utilizando seu notebook pessoal, siga o mesmo processo diariamente, para efetuar um backup (muitos alunos já perderam seus trabalhos por problemas no notebook, não queremos



que aconteça com você). O arquivo gerado será similar a esse:

Feito isso, é só adicioná-lo ao seu Drive de estudante do SENAI. Quando for usar novamente o material compactado, basta que você efetue o *download* do arquivo no Drive, descompacte-o (conforme imagem abaixo) Atenção: use o **EXTRAIR AQUI**, não use as outras opções de extração).

Você verá que o Windows criará uma pasta “normal” com o mesmo nome da compactada (no exemplo, **WorkspaceJavaBasico**). Abra o Eclipse e indique tal pasta como a sua Workspace na tela **Eclipse IDE Launcher**:

Para os exercícios de Java, daqui para frente:

Utilize essa mesma Workspace; para cada **lista de exercício** crie um novo **projeto** (assim como você criou o projeto IntroducaoJavaTabuadaJava); e para cada **exercício**, uma nova **classe** (assim como criou a classe Main). Nomeie os exercícios das listas como “Exercício01”, “Exercício02” para facilitar a validação.

Veja o exemplo: