UNIP
UNIVERSIDADE PAULISTA

Interativa

Linguagem de Programação Orientada a Objetos

Autor: Prof. Ricardo da Costa Veras
Colaboradoras: Profa. Vanessa Santos Lessa
Profa. Larissa Rodrigues Damiani

Professor conteudista: Ricardo da Costa Veras

Mestre em Engenharia da Informação pela Universidade Federal do ABC – UFABC (2018). Especialista em Orientação a Objetos pela Faculdade de Informática e Administração Paulista – Fiap (2004). Graduado em Engenharia Elétrica com ênfase em Eletrônica pela Escola de Engenharia Mauá (1994). Desde 2009 é professor da Universidade Paulista – UNIP e vem ministrando aulas para os cursos de Ciência da Computação, Sistemas de Informação e Engenharia (básico). Tem experiência na área de Tecnologia da Informação desde 1995, tendo trabalhado com consultoria e prestação de serviços de análise e desenvolvimento de sistemas em diversas empresas.

Dados Internacionais de Catalogação na Publicação (CIP)

V476l Veras, Ricardo da Costa.

Linguagem de Programação Orientada a Objetos / Ricardo da Costa Veras. – São Paulo: Editora Sol, 2022.

172 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Variáveis. 2. Polimorfismo. 3. Modificador. I. Título.

CDU 681.3.062

U514.16 – 22

Sumário

Linguagem de Programação Orientada a Objetos

Unidade III

Unidade IV

APRESENTAÇÃO

A linguagem de programação apresentada neste livro-texto segue o paradigma de orientação a objetos (OO), o qual se baseia na criação de entidades de programação que representam elementos do mundo real (uma pessoa, um cliente, uma empresa, uma área de uma empresa, um produto de uma empresa etc.).

Alguns exemplos de linguagens de programação que seguem o paradigma de OO são: Java, C#, C++, PHP, Ruby, Python, entre outras.

Neste livro-texto, focaremos o estudo da OO com base na linguagem de programação **Java**, que é totalmente voltada a esse paradigma. Essa linguagem vem mantendo desde 2001 sua posição entre as três primeiras linguagens mais utilizadas por programadores e pelas empresas, segundo o Índice Tiobe (TIOBE, s.d.).

Aqui, pretende-se organizar o aprendizado do paradigma OO e da linguagem Java de forma progressiva e sequencial a fim de proporcionar um processo didático eficiente e prazeroso. A intenção deste curso não é apresentar todos os conceitos existentes sobre a Linguagem Java, mas, sim, os seus principais conceitos básicos, juntamente aos conceitos básicos do paradigma OO.

Sem dúvida nenhuma, o constante crescimento da tecnologia e de seus recursos reflete na linguagem Java. Sempre, ao longo de nossa vida profissional, precisamos ficar atentos à sua evolução, sendo constante o seu estudo e aprendizado.

Este livro-texto está dividido em quatro unidades, cujo conteúdo segue a seguinte estruturação:

- **Unidade I**: serão apresentados os elementos básicos de orientação a objetos (**classe**, **objeto**, **atributo** e **método**) e a estrutura lógica oferecida pela linguagem Java (**tipagem**, **estrutura condicional** e de **repetição**).

- **Unidade II**: serão apresentados recursos específicos de orientação a objetos, como o **encapsulamento**, a **herança** e o **polimorfismo**, assim como sua base de entendimento – os modificadores **public**, **private**, **protected** e **(default)** e os **métodos construtores** –, recursos esses que compõem os pilares desse paradigma.

- **Unidade III**: será explicado o **polimorfismo de classes**, além de apresentadas algumas ferramentas da linguagem relacionadas ao controle da estruturação do sistema, que alteram o comportamento dos elementos trabalhados na sua codificação (os modificadores **static**, **final** e **abstract** e as **interfaces**).

- **Unidade IV**: serão estudados recursos de tratamento de exceções, como os erros em tempo de execução de programas (**exception**), assim como será estudado o conceito de multitarefa ou de processamento paralelo (**thread**).

O empenho no estudo desta disciplina possibilitará ao estudante adquirir uma base sólida de conhecimentos que o permitirá evoluir para um estudo continuado. Além do embasamento teórico apresentado, é importante que o estudante dedique boa parte de seus estudos aos exercícios práticos sugeridos ao longo deste material, pois só assim garantirá seu real aprendizado.

Persista em seus estudos. Empenhe-se em adquirir conhecimentos além deste livro-texto, como os que são sugeridos ao longo da leitura. Essa será a base de seu crescimento profissional. Aproveite bem este curso e garanta o seu sucesso.

Bons estudos!

INTRODUÇÃO

O paradigma orientado a objetos (OO) vem sendo vastamente divulgado e utilizado desde a década de 1990 (por volta de 1993) a partir do surgimento da internet. No entanto, seu conceito remete a épocas anteriores, desde os anos 1960.

Tendo sido idealizado por matemáticos e estudiosos da tecnologia da informação, sendo um de seus criadores o estadunidense Alan Kay, o paradigma OO surgiu da evolução dos processos de programação, acompanhando o progresso tecnológico, a necessidade de organização da codificação e a evolução visionária dos procedimentos e entidades de programação.

Na época, o caos enfrentado pelos programadores na análise de centenas de milhares de linhas de código em sequência pedia um novo paradigma, ou seja, uma nova concepção de estruturação que o processo de design (codificação de sistemas) deveria respeitar, a fim de acompanhar de forma ágil o progresso tecnológico, além de proporcionar um melhor entendimento do sistema criado, agilizando suas futuras implementações ou manutenções.

A linguagem Java foi desenvolvida a partir da ideia de ser multiplataforma. Para que isso fosse possível, seus desenvolvedores criaram uma máquina virtual (um software JVM – Java Virtual Machine) para cada plataforma, sendo ela um software que roda programas criados na linguagem Java, de modo que qualquer programa feito naquela linguagem (em qualquer uma das plataformas) pudesse rodar em outra plataforma sem ocorrer problemas em sua funcionalidade, bastando que ali estivesse instalada a máquina virtual Java.

No entanto, essa JVM não pode existir sozinha. Ela faz parte de um Java Runtime Environment (JRE), que é todo o ambiente de execução da linguagem Java, o qual contém a JVM, toda a biblioteca de classes Java e o coletor de lixo (garbage collector), que permite ao programador gerar códigos sem se preocupar em ficar eliminando elementos da memória ao longo do funcionamento do programa à medida que eles não são mais utilizados.

Assim, para que um programa Java funcione em um dispositivo, basta que ele tenha instalado o JRE adequado, de acordo com a plataforma, pois sem ele não é possível acionar e rodar qualquer programa criado nessa linguagem.

Se o leitor for um curioso da programação, vale a pena fazer o download do JDK (Java Development Kit), um kit de desenvolvimento que, além do JRE e, consequentemente, da JVM, possui todo o código fonte do Java (os arquivos com extensão .java) e o JavaDoc (documentação básica de todas as classes existentes na biblioteca, podendo esta ser acessada de modo off-line), além de algumas outras ferramentas de programação.

Em 1991, uma equipe liderada por James Gosling (com a parceria de Patrick Naughton e Mike Sheridan) participou de um projeto pela empresa Sun Microsystems com a intenção de criar uma tecnologia de dispositivos digitais que seria capaz de unir equipamentos eletrônicos diversos (como a televisão, entre outros aparelhos) com o computador. Dessa nova tecnologia, nasceu uma linguagem

de programação cujo nome era **Oak**, que inicialmente seria embutida em dispositivos de controles remotos de televisão, além da própria televisão, e que permitiria gerar solicitações de controle da programação de TV, de controle de vídeo e de controle da própria televisão. Essa tecnologia, apesar de visionária, veio cedo demais e não conseguiu atingir o mercado tecnológico da época, pois exigiria a construção de uma infraestrutura (redes de servidores e de cabos de comunicação) economicamente inviável para sua comercialização.

Porém, com o surgimento da internet, a mesma equipe enxergou uma oportunidade de lançar aquela tecnologia, já que os meios físicos e toda a infraestrutura de comunicação estavam se expandindo naturalmente com a própria internet. Foi então que, com a liberação da internet para o uso das empresas e a expansão do uso por particulares, os criadores da linguagem Oak, cujo nome não poderia ser reutilizado para esta nova linguagem, criaram outra linguagem, à qual deram o nome de **Java**, totalmente adaptada àquela nova tecnologia, a internet.

Assim, em 1995 lançaram a primeira versão do Java, com características inovadoras para a época: uma linguagem inteiramente orientada a objetos, simples, dinâmica, segura, multitarefa, multiplataforma e que não dependeria de outras linguagens.

Antes daquela época, a internet era um meio extremamente versátil de comunicação, porém restrita apenas a informações, não possibilitando a interação direta dos usuários (por exemplo, para fazer compras ou solicitar serviços). Com a linguagem Java, construiu-se uma série de programas (applets) que eram transportados pela internet e que rodavam de forma local nos computadores, dando às páginas uma funcionalidade dinâmica e inovadora e que poderia ser utilizada por qualquer um dos usuários da internet.

A facilidade de criação de sistemas diversos com aquela linguagem, sua versatilidade de acompanhar as novas tecnologias, além do fato de funcionar em diversas plataformas, fizeram crescer o interesse pela sua utilização entre os desenvolvedores e as empresas.

A partir de então, e ao longo dos anos, a Sun Microsystems lançou diversas versões da linguagem, acompanhando as necessidades e as tecnologias crescentes. Em 2007, a Sun acabou por tornar a linguagem Java totalmente open-source (de código aberto), ou seja, permitiu que qualquer um pudesse ter acesso a todo código fonte (original) da linguagem, podendo inclusive tornar-se "parceiro" na sua construção e evolução.

Entre 2009 e 2010, a Sun Microsystems foi adquirida pela empresa Oracle Corporation, a qual ficou responsável por divulgar, acompanhar e controlar as necessidades de crescimento da linguagem Java.

Ao final de 2014, mais de 9 milhões de desenvolvedores por todo o mundo já programavam em Java, transformando essa linguagem numa das mais utilizadas.

Unidade I

1 INTRODUÇÃO À ORIENTAÇÃO A OBJETOS

O paradigma orientado a objetos foi criado e pensado com base em três conceitos fundamentais, considerados os pilares da orientação a objetos:

- herança;
- encapsulamento;
- polimorfismo.

Esses três conceitos serão vistos com mais detalhes ao longo deste livro-texto, após vermos conceitos mais básicos da linguagem. Basicamente, a herança é um recurso que permite o reaproveitamento de código, o que remete ao conceito de generalização e especialização; o encapsulamento protege a utilização inapropriada dos atributos de uma classe, de modo a possibilitar o controle de seus valores por meio dos seus métodos; e o polimorfismo é um recurso em que os elementos podem se comportar de formas diferentes dependendo de como são estruturados (e dependendo da necessidade do programa).

A orientação a objetos é um paradigma de linguagem de programação que tem como princípio a utilização de elementos que representam entidades do "mundo real", ou seja, elementos existentes numa situação real (uma questão, um problema, uma solução etc.).

Como exemplo disso, poderíamos citar a construção de um sistema para auxiliar nas atividades de controle de RH ou de finanças de uma empresa, no qual são criadas classes que representam as entidades existentes naquele controle, como o funcionário, a empresa, os usuários, a gerência, a diretoria etc. Outro exemplo poderia ser um sistema de compra e venda de produtos em que são criadas classes que representam os produtos, o vendedor, o cliente, o pagamento, o banco etc. Um último exemplo: se o sistema criado for um jogo de corrida de automóveis, as classes poderiam representar os carros, a pista, o usuário/jogador etc.

Percebam que as entidades representadas são elementos ou algo que encontramos e com que convivemos no dia a dia, na vida profissional, nas necessidades diárias ou no entretenimento, e no programa nós criamos uma representação dessas entidades em forma de classes. Essa é a capacidade de abstração da linguagem orientada a objetos, ou seja, trazer para o programa uma representação dos elementos reais com os quais estamos trabalhando.

Um objeto é um elemento criado **em memória** (na memória RAM) a partir do modelo da classe, de forma a possuir todas as suas características. Já a classe é o **molde** para criarmos os objetos. Como

exemplo, poderíamos pensar que uma classe Pessoa poderia ser a ideia de pessoa em que se definem as possíveis características que ela poderá possuir e as possíveis ações que poderão ser acionadas. Assim, esse modelo deverá permitir a construção de um objeto que poderá ter, por exemplo, um nome, uma idade, um CPF, uma altura, assim como ações que poderão ser acionadas, como cadastrar, contatar, pagar, contratar etc. Dessa forma, o objeto que efetivamente será gerado a partir daquela classe possuirá todas as suas características, mas com valores definidos, tais como o nome **João Batista**, a idade de **30 anos**, o CPF **123.456.789-10**, a altura de **1,60 m** etc., além de permitir acionar qualquer das ações predeterminadas para a classe que o gerou.

Quando dizemos que o programa é orientado a objetos, estamos especificando que apesar de programarmos pensando nos modelos (nas classes) daqueles elementos que queremos trabalhar, devemos sempre pensar que quando o sistema estiver sendo utilizado, esses elementos existirão na memória RAM na forma de **objetos** que representam as entidades reais, ou seja, é um programa criado com base em classes, mas orientado a funcionar com base em objetos na memória.

Neste curso, para que possamos construir nossos exemplos, iremos utilizar um software específico, a IDE do Eclipse, que nos possibilitará colocar em prática todos os exemplos e aprendizados contidos neste livro-texto.

1.1 A IDE do Eclipse

Uma IDE (**i**ntegrated **d**evelopment **e**nvironment – ambiente integrado de desenvolvimento) é um software que fornece um ambiente integrado para se trabalhar com o desenvolvimento de outros softwares, de acordo com a linguagem de programação com que se trabalha. Geralmente esses ambientes integram as áreas de texto, onde escrevemos as linhas de código do nosso programa; as facilidades de identificação dos termos necessários ao código descrito; a geração automática dos códigos compilados (quando necessários); a possível execução rápida do programa criado; a fácil verificação dos erros que cometemos ao codificar um programa; além de facilitadores de programação que nos ajudam a gerar elementos específicos em nossos programas.

Para executar todos os exemplos e exercícios aqui apresentados, precisamos instalar o Java Runtime Environment (JRE) e a IDE do Eclipse, que é um software livre cujo download pode ser feito no endereço https://www.eclipse.org/downloads/. Os processos de instalação são detalhados a seguir:

Instalando o Java

Para este curso, seu micro deve ter instalada a JRE (J2SE) do Java.

Para baixar o arquivo de instalação mais recente, digite em alguma página de busca da internet o seguinte texto:

java jre j2se

Procure o link da Oracle com o texto:

Java 2 Platform, Standard Edition (J2SE) <número_versão> - Oracle

Procure pelo link equivalente a:

Java 2 Platform, Standard Edition version <número_versão>

Após clicar nesse link, na página que abrir, clique no link equivalente à última versão do Java SE <release>, o qual abrirá uma página com diversos links para diversas plataformas (Linux, MacOS e Windows).

Selecione a plataforma de seu micro (existe um link para Windows com o texto Windows x64 Installer, que baixará um arquivo executável de instalação do Java). Esse é um arquivo que instala a JDK que contém a JRE necessária.

Execute o arquivo baixado, instalando o J2SE.

Instalando o Eclipse

Agora vamos instalar a IDE, que nos possibilitará gerar projetos de sistemas em Java. No caso, vamos instalar o Eclipse, que é uma IDE criada inicialmente pela IBM e hoje pertence a uma comunidade de empresas e programadores, a qual, por sua vez, pertence a um projeto continuado pela Eclipse Foundation, uma associação europeia que presta serviços e dá o suporte para auxiliar ao projeto de divulgação, desenvolvimento e elaboração das novas versões dessa linguagem.

Para instalar a versão 64 bits do Eclipse, basta entrar no link a seguir:

https://www.eclipse.org/downloads/

A partir desse link, baixa-se um arquivo executável *.exe.

Na página desse link, clique no botão Download x86_64.

Após abrir a nova página, clique no botão Download.

Após o término do download, deve-se executar o arquivo de instalação.

Para instalar a versão 32 bits do Eclipse (para micros mais antigos), basta entrar na página a seguir referente ao Eclipse Luna, uma versão estável para micro 32 bits dessa IDE:

https://www.eclipse.org/downloads/packages/release/luna/sr2

É necessário clicar no link 32-bit do Eclipse IDE for Java Developers (veja a imagem a seguir). Com esse link, baixa-se um arquivo compactado *.zip.

Figura 1 – Trecho da página de download de arquivos relacionado ao Eclipse na sua versão Luna

Nesse último caso, faz-se o download de um arquivo compactado, que deve ser descompactado no diretório raiz de seu micro (c:\).

Em qualquer uma das instalações, será gerado um diretório eclipse na raiz do micro (c:\eclipse), dentro do qual existe o arquivo executável do Eclipse (eclipse.exe).

O arquivo Eclipse.exe é o arquivo de inicialização do aplicativo Eclipse. Se você instalou uma versão atual, ao ser acionado, já é aberta a área de programação:

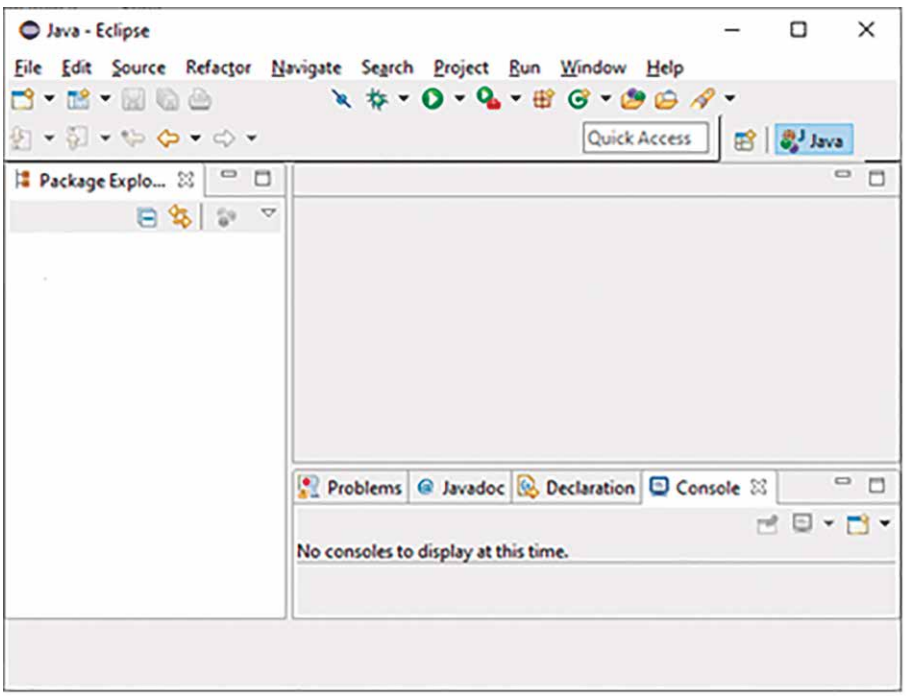

Figura 2 – Tela com a visão de programação (perspectiva) Java do Eclipse – no caso, a tela do Eclipse Luna, cuja disposição dos quadros é semelhante à de outras versões

Caso você inicie o Eclipse na sua versão Luna, ele abre em sua tela de Welcome (veja a imagem a seguir).

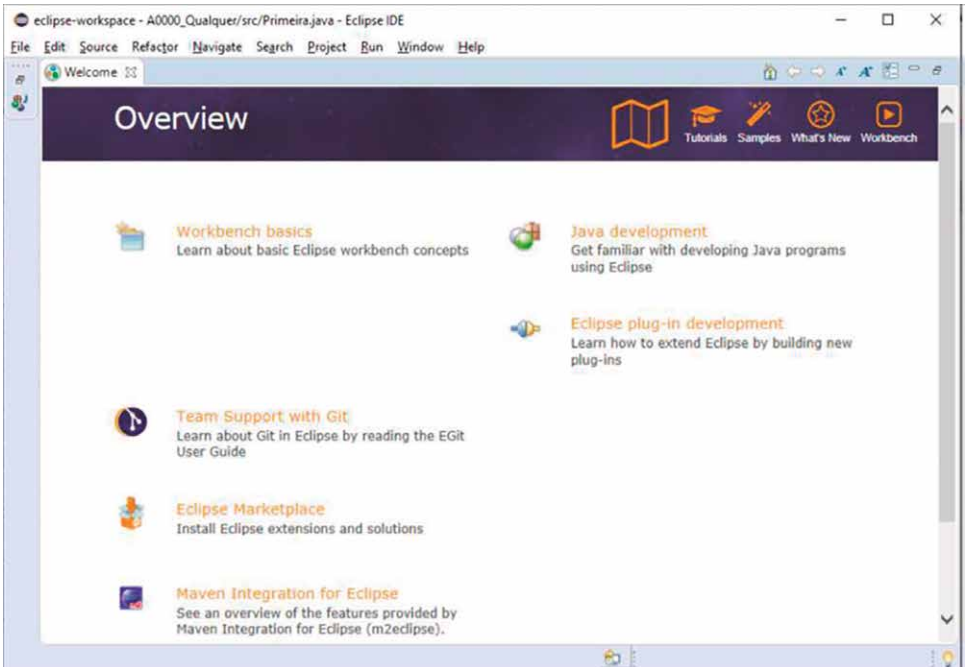

Figura 3 – Tela de Welcome do Eclipse Luna

Neste caso, para abrir o ambiente de programação, deve-se clicar no ícone Workbench, que fica no canto superior direito da tela:

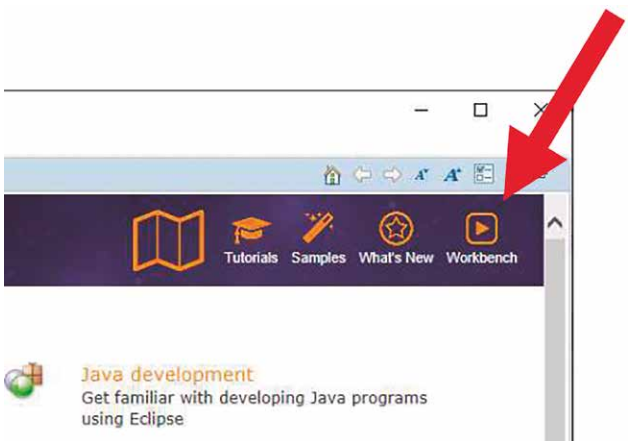

Figura 4 – Canto superior direito da tela de Welcome do Eclipse Luna com o acesso à área de programação (link Workbench)

Sempre que neste livro-texto for dado um exemplo de código, devemos criá-lo em uma classe gerada no Eclipse, a partir do pacote src de um projeto Java (pacote raiz das classes de um projeto Java). Algumas classes deverão ser criadas com o método main; outras, sem esse método. Essa necessidade estará especificada no exemplo ou no exercício proposto.

1.2 O método main

O método main é o método responsável por acionar um programa Java. Na maioria dos casos, esse método precisa existir em apenas uma das classes do sistema construído (apenas naquela que inicialmente acionará o programa).

Uma vez acionado o método main, a partir dele podemos acionar diversos outros métodos pertencentes a uma infinidade de outras classes.

Sua sintaxe, dentro de uma classe, será:

```
public class ClasseA {
   public static void main(String[] args) {
                        // ...linhas de código
   }
}
```

Ou, ainda:

```
public class Calculo {
   public static void main(String[] args) {
     double x = 7.45 + 2.91;
     int y = -47 + 82;
     System.out.println(x + " :: " + y);
   }
}
```

Para executar o método main de uma classe no Eclipse, basta abrir o código da classe na tela central do Eclipse e clicar no menu:

Run – Run As – Java Application

1.3 Classes, objetos, atributos e métodos

Classes são modelos a partir dos quais serão construídos os objetos em memória, quando efetivamente o sistema desenvolvido for utilizado.

Uma classe possui:

- **atributos**: elementos que definem a classe, em que podem ser inseridos os valores que a caracterizam;

- **métodos**: elementos que podem ser acionados ou não e que definem as possíveis ações que podem ser executadas a partir dos objetos gerados com aquela classe.

Na UML (Unified Modeling Language – Linguagem de Modelagem Unificada), uma classe é representada da seguinte forma:

Figura 5 – Representação de uma classe pela UML

Perceba que a representação mostra também a ordem na qual sempre devemos dispor os elementos em uma classe. Internamente, primeiro devemos sempre elencar todos os seus atributos, e só depois descrevermos os métodos que a classe possuirá.

A classe a seguir (ClasseQualquer) é apenas um exemplo de como deve ser a disponibilidade dos elementos em uma classe:

```
public class ClasseQualquer {
   // ...Primeiro os atributos
   public int atributo01;
   public int atributo02;
   public int atributoN;
   // ...Depois os métodos
   public void metodo01() {
                    // ...linhas de código
   }
   public void metodo02() {
                    // ...linhas de código
   }
   public void metodoN() {
                    // ...linhas de código
   }
}
```

1.4 Algumas convenções de programação em Java

Toda linguagem de programação possui algumas regras que são geradas não pelo criador da linguagem (o que estaria embutido no interpretador, ou compilador, daquela linguagem), mas pelos desenvolvedores que trabalham com aquela linguagem (os profissionais que desenvolvem sistemas).

A linguagem Java possui algumas convenções que **devem** ser seguidas em sua codificação. A seguir estão algumas regras iniciais:

- toda **classe** deve ter um nome que **inicia** com **Letra Maiúscula**;
- todo **método** deve ter um nome que **inicia** com **letra minúscula**;
- todo **atributo** deve ter um nome que **inicia** com **letra minúscula**;
- toda **constante** deve ter um nome **totalmente descrito em CAIXA ALTA** (todas as letras em **MAIÚSCULO**).

Além disto, todo **nome** que identifica uma entidade do Java, seja uma classe, um método, um atributo, uma variável local ou uma constante, deve seguir, de modo geral, as seguintes regras:

- **não** deve iniciar com número;
- **não** deve conter acentuação;
- **não** deve conter espaço;
- **não** deve conter símbolo, com exceção do underline (_);
- **não** deve ser uma palavra reservada da linguagem.

Observação

Essas últimas cinco regras estão embutidas na compilação da linguagem Java, de modo que, se não forem seguidas, é gerado um erro de compilação, e o programa fica impedido de ser executado.

Exemplo de **nomes de variáveis** que não podem ser utilizados:

- **1oAno**, pois contém número no início;
- **Nome do Aluno**, pois contém espaços internos ao nome;
- **MédiaFinal**, pois contém acentuação;
- **Codigo#Principal!**, pois contém símbolos;
- **Double**, pois é uma palavra reservada (que define um elemento na linguagem Java).

1.5 Meu primeiro projeto Java

Sempre que se cria um programa (uma classe) Java, deve-se fazê-lo em um projeto Java, seja ele já existente ou novo. Em nossos exemplos, criaremos classes a partir da raiz do projeto (o pacote ou diretório src dos projetos do Eclipse).

Vamos criar o nosso primeiro programa Java: a classe OlaMundo.

Nosso objetivo será criar uma classe (um programa Java) que, ao ser acionada, imprime na Tela da Console o texto Olá Mundo!.

Para isso, vamos **abrir o Eclipse** e gerar um novo projeto: o Projeto LPOO, nos seguintes passos:

1 – Clicar em: **File – New – Project**.

2 – Em seguida, na tela que aparece, selecionar a pasta Java, clicando no símbolo (>) naquela opção, e selecionar **Java Project**. Em seguida, deve-se clicar em **Next**>.

3 – Inserir um nome para o projeto (por exemplo, LPOO) no campo **Project name**: e clicar em **Finish** (veja na figura a seguir o Package Explorer com o projeto gerado).

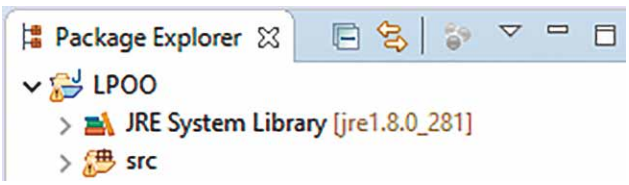

Figura 6 – Projeto LPOO no Eclipse

Agora, vamos criar a classe OlaMundo nos seguintes passos:

1 – Clicar no **botão direito do mouse** sobre a package (pacote) src, mostrada na figura a seguir:

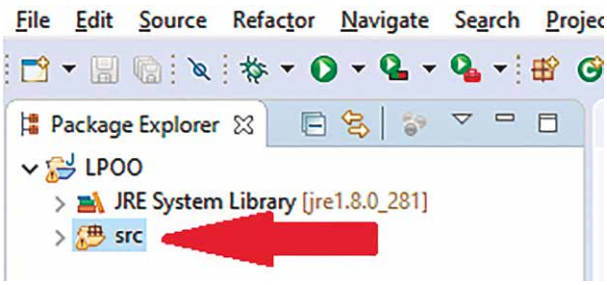

Figura 7 – Pacote src do projeto LPOO

2 – Selecionar **New – Class**.

3 – Na tela que se abre, no campo **Name:**, inserir o nome da classe (por exemplo, OlaMundo – note que o nome deve seguir as regras de nome de variáveis, assim como as regras de nomes de classes, como visto anteriormente). Perceba que o arquivo criado após se gerar uma classe sempre terá exatamente o mesmo nome da classe, seguido da extensão .java (no exemplo, OlaMundo.java).

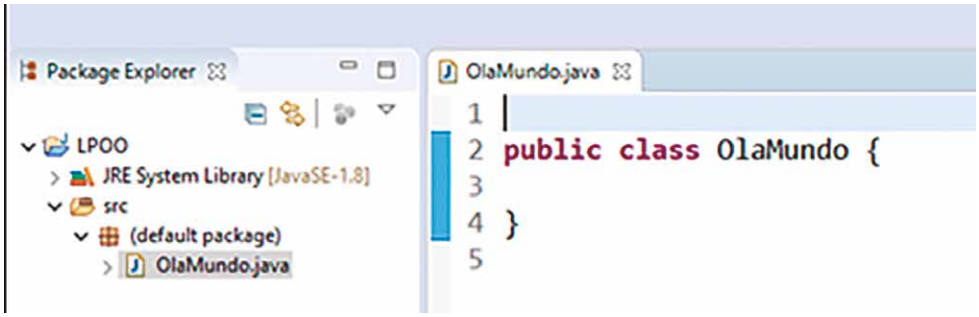

Figura 8 – Arquivo e classe OlaMundo

4 – Na área de edição de código, completar o programa para que fique com os seguintes comandos:

```
public class OlaMundo {
   public static void main(String[] args) {
     System.out.println("Olá Mundo !!");
   }
}
```

Observe o ponto-e-vírgula ao final da linha de comando de impressão.

5 – Para rodar o código criado, clicar com o botão direito do mouse sobre o código (na própria área de edição de código) e selecionar a opção **Run As – Java Application**, de forma que, após o programa ser rodado, aparecerá na área da console a frase desejada (Olá Mundo !!):

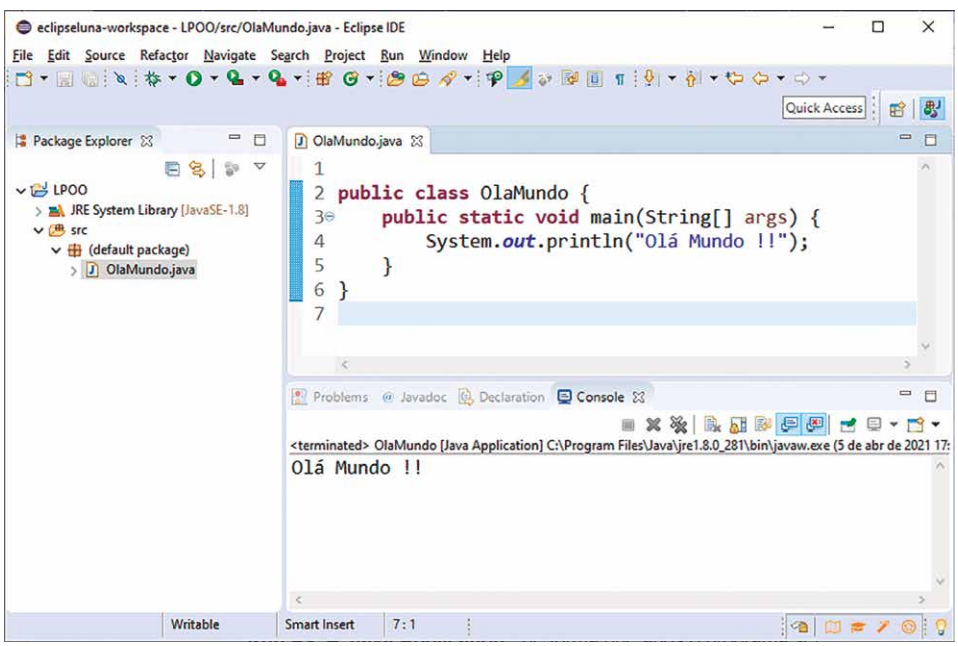

Figura 9 – Classe OlaMundo e o resultado de seu acionamento

1.6 Os elementos e suas sintaxes

Classe

Uma classe é um modelo para gerarmos os objetos na memória quando utilizamos nosso sistema. A declaração de uma classe define a sua estrutura básica, e sua sintaxe deve ser:

```
public class NomeDaClasse {
   //...
}
```

Mais à frente explicaremos o significado da palavra reservada **public** para a linguagem Java. A palavra reservada **class** indica que o elemento será uma classe. O nome da classe deve seguir as regras de convenção.

Objeto e sua instanciação

Os objetos são as classes em memória (na memória RAM), gerados quando o sistema está sendo utilizado pelo usuário. Para que um objeto seja criado em memória, utilizamos a seguinte sintaxe:

```
NomeDaClasse nomeDoObjeto = new NomeDaClasse();
```

Gerar um objeto em memória é **instanciar uma classe**. Neste caso, o nome do objeto é a varável que identificará aquele objeto representando a classe em memória.

A partir de uma mesma classe, podemos instanciar diversos objetos em memória.

Atributo

Um atributo é uma característica que define uma classe. Todo atributo deve ser declarado dentro da classe (entre as "chaves" da classe) seguindo a sintaxe a seguir:

```
modificadores tipoDoAtributo nomeDoAtributo;
```

Ou, ainda:

```
modificadores tipoDoAtributo nomeDoAtributo = valorInicial;
```

A diferença entre essas duas declarações está apenas no fato de se ter ou não um valor inicial para o atributo.

Exemplo:

```
public int quantidadeProduto;
protected String nomeEmpresa;
// ... ou ainda ...
public int quantidadeEstoque = 0;
private String nomeSistema = "Sistema de Compras";
```

Observação

Os modificadores deste exemplo (public, protected ou private) serão explicados mais à frente.

Método:

Um método é uma ação que pode ser acionada após se instanciar a classe. Em outros paradigmas, um método pode ser comparado à uma função ou uma sub. A sintaxe geral de declaração de um método é a seguinte:

```
"...
modificadores tipoDeRetorno nomeDoMetodo(parâmetros) {...}
..."
```

Os parâmetros de um método são os dados (ou valores) que o método deverá receber ao ser acionado e que serão processados pelo método. A sintaxe dos parâmetros deve ser a seguinte:

```
(tipo1 parametro1, tipo2 parametro2, ..., tipoN parametroN)
```

Percebam que um método pode receber N parâmetros (separados por vírgula), e para cada um deles devemos definir seu tipo, mesmo que sejam o mesmo tipo.

Para o método, as chaves **{...}** definem seu bloco e englobarão o código a ser executado quando o método for acionado.

O termo tipoDeRetorno define o tipo do dado que será retornado (devolvido como resposta) pelo método. Em geral, quando um método retorna um valor, ao chamarmos o método, ele deve ser guardado, ou ainda recebido, por uma variável. Quando um método não retornar valor algum, então esse termo deve ser void. Da mesma forma, se um método retornar algum valor, então ele deverá ter definido expressamente o valor de retorno.

O termo modificadores, tanto dos atributos quanto dos métodos, indica qual o modificador de acesso e/ou o modificador de comportamento que o elemento terá na sua declaração (eles serão melhor explicados mais à frente), os quais podem ser:

- Modificadores de acesso:
  - public;
  - private;
  - protected;
  - (default).
- Modificadores de comportamento:
  - static (este não é utilizado para classes);
  - final;
  - abstract (este não é utilizado para atributos).

O termo tipoDoAtributo deve sempre ser definido para qualquer variável utilizada em um programa Java. Ele indica o tipo de valores que a variável poderá receber. A linguagem Java é **fortemente tipada**, o que significa que para podermos utilizar uma variável em um código, primeiro precisamos declarar seu tipo, que será imutável ao longo daquele bloco de código.

O tipo String na linguagem Java não é um tipo primitivo, mas sim uma classe. Devemos declarar variáveis do tipo String da seguinte forma:

String nomeDaVariavel = "valor texto inicial da variável";

Perceba que o tipo String é escrito com a primeira letra maiúscula (pois é uma classe) e que o valor literal de sua variável deve ser sempre colocado entre aspas duplas ("txt").

Vejamos o exemplo a seguir, com a classe SomaValores, cujos métodos proporcionam duas formas de adição de valores para tipos diferentes:

```
public class SomaValores {
   // - atributos da Classe:
   public int resultInt;
   public double resultDouble;
   // - métodos da Classe:
   public int somarInteiros(int a, int b) {
     // Este método recebe dois valores inteiros
     // ...e retorna o resultado inteiro da soma dos dois valores
     // ...além de guardar a soma no atributo "resultInt".
     int result = a + b;
```

```
    resultInt = result;
    return result;
  }
  public double somarReais(double a, double b) {
    // Este método recebe dois valores reais
    // ...e retorna o resultado real da soma dos dois valores
    // ...além de guardar a soma no atributo "resultDouble".
    double result = a + b;
    resultDouble = result;
    return result;
  }
  public void mostrarResultados() {
    // Este método inicialmente monta um texto (uma String)
    String txt = "------------------------\n";
    txt += "Resultado da soma de Inteiros: " + resultInt + "\n";
    txt += "Resultado da soma de Reais: " + resultDouble + "\n";
    // ...e ao final mostra este texto montado na Console.
    System.out.println(txt);
  }
  // Abaixo, o método "main" (que será o 1o a ser acionado)
  // ..e que acionará os outros métodos desta Classe
  public static void main(String[] args) {
    // ...instanciando a Classe SomaValores
    SomaValores sv = new SomaValores();
    //..- somando os valores inteiros: 7 + 15
    //..- trazendo o resultado para a variável "vi"
    int vi = sv.somarInteiros(7, 15);
    //..- e mostrando este resultado na Console
    System.out.println(vi);
    //..- somando os valores reais: 3.6 + 9.2
    //..- trazendo o resultado para a variável "vd"
    double vd = sv.somarReais(3.6, 9.2);
    //..- e mostrando este resultado na Console
    System.out.println(vd);
    //-----------------------
    //Abaixo vamos acionar o método "mostraResultados()"
    //...do objeto "sv" (que representa a Classe SomaValores)
    //...e que mostra organizadamente os valores dos atributos
    sv.mostrarResultados();
  }
}
```

Os elementos em verde do código que acabamos de apresentar (antecedidos por //) são **comentários de programa** e servem para explicar as linhas de código. O texto nele descrito não será executado, de forma que não existirá no arquivo compilado (arquivo.class de execução).

Ao acionarmos essa classe a partir de seu método main, teremos o seguinte resultado na tela da console:

```
22

12.799999999999999

-----------------------

Resultado da soma de Inteiros: 22

Resultado da soma de Reais: 12.799999999999999
```

Saiba mais

Saiba mais sobre métodos sem retorno e métodos com retorno de valores no capítulo 6, itens 6.1 e 6.2, de:

FURGERI, S. *Java 7*: ensino didático. 2. ed. São Paulo: Érica, 2012.

Exemplo de aplicação

Um professor de programação criou um texto representando uma classe e distribuiu o arquivo entre os alunos para que eles criassem a classe em uma IDE. Quando os alunos copiaram o texto com o código da classe, e depois foram compilá-lo, a IDE acusou nele um erro de compilação, ou seja, existe nesse programa um erro conceitual que impede que ele possa ser executado. O programa distribuído pelo professor foi:

```
public class ClasseA {
   public String x = "teste";
   public double y;
   public void metodo01(int a, int b) {
     System.out.print(a + " : " + b);
   }
   public int metodo02(String s) {
     x = s;
     int a = 4;
     return a;
```

```
    }
    public String metodo03(String z) {
      String s = x + z;
      System.out.print(s);
    }
}
```

Qual foi o erro encontrado pelo compilador que o impediu de executar o programa?

A) No método metodo01(...) não se pode mostrar simultaneamente os valores de a e b, pois ambos são de tipos diferentes.

B) No método metodo02(...) o programador deveria ter declarado a variável x.

C) Na declaração de atributos da classe, faltou atribuir um valor à variável y.

D) No corpo do método metodo03(...) há um comando que soma duas strings, e não se pode somar strings a não ser que tenham valores numéricos.

E) No método metodo03(...) faltou o retorno de um valor.

Resposta correta: alternativa E.

Resolução

A) Alternativa incorreta.

Justificativa: os valores de a e de b, apesar de serem de tipos diferentes, estão sendo utilizados por um método de impressão na console que permite a mistura de vários tipos de informações, e, portanto, esse fato não caracteriza um erro.

B) Alternativa incorreta.

Justificativa: se observarmos bem, a variável x utilizada no método metodo02(...) representa na verdade o atributo x que já foi declarado no início da classe; portanto, esse comando está correto.

C) Alternativa incorreta.

Justificativa: vimos que os atributos podem ser declarados com ou sem um valor inicial, podendo ser dado a ele um valor ao longo da utilização do sistema, seja por um método da própria classe ou por métodos de outras classes. Sua declaração pode ter ou não uma atribuição, de forma que não caracteriza um erro.

D) Alternativa incorreta.

Justificativa: no corpo do método método03(...) a variável z recebe um valor que é resultado de uma concatenação de strings, ou seja, esse símbolo de adição significa que o valor de uma variável do tipo string está sendo concatenada (unida) ao valor da outra variável do tipo string e, portanto, não caracteriza um erro.

E) Alternativa correta.

Justificativa: se observarmos a declaração do método metodo03(...), ele possui um tipo de retorno que deve ser um valor do tipo string, e em seu corpo está faltando a declaração de retorno de valor, o qual é representado pela palavra reservada return, o que no caso deveria retornar um valor do tipo string.

2 VARIÁVEIS, ESTRUTURAS DE CONTROLE E ORGANIZAÇÃO DO PROJETO

Estruturas de controle são as estruturas vinculadas à lógica de programação. Neste item, vamos ver como devemos trabalhar com variáveis em Java (lembrando que a linguagem Java é uma linguagem fortemente tipada) e como devem ser descritas as estruturas condicionais e as estruturas de repetição.

2.1 Tipos de variáveis

Variáveis são espaços em memória reservados para que possamos guardar valores, a fim de podermos processá-los. Vimos que toda variável possui um nome (determinando a sua identificação), um tipo (que depende do valor a se trabalhar) e um valor.

Em Java, toda variável deve ser declarada antes de ser efetivamente utilizada, para que o compilador saiba como e o quanto ele deve reservar de memória a fim de poder trabalhar com a informação que será processada.

Como a linguagem Java é uma linguagem totalmente voltada ao paradigma de orientação a objetos, as variáveis podem ser utilizadas como atributos (mantendo um comportamento de variável global na classe), ou como variáveis locais, existindo apenas no bloco onde foi definida.

A declaração de um atributo já foi explicada anteriormente neste material. Já as variáveis locais devem ser declaradas com a seguinte sintaxe:

```
tipoDaVariavel nomeVariavel;
```

Ou, ainda:

```
tipoDaVariavel nomeVariavel = valor;
```

Observação

Essa segunda forma de declaração faz com que tenhamos um valor inicial para a variável.

Perceba que as variáveis locais não possuem o termo de modificadores na sua declaração.

O tipo da variável definirá o tipo da informação que ela conterá, assim como o tamanho do espaço em memória a ser reservado para isso.

A linguagem Java trabalha com diversos tipos de dados. São eles:

- Tipos numéricos inteiros:
  - **byte**: números de -128 a 127;
  - **short**: números de -32.768 a 32.767;
  - **int**: números de -2.147.483.648 a 2.147.483.647;
  - **long**: números de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
- Tipos numéricos reais (de ponto flutuante):
  - **float**: ótima precisão com números de até 7 casas decimais (0.1234567...);
  - **double**: ótima precisão com números de até 15 casas decimais (0.123456789012345...).

Observação

Um valor literal de número real deve ser colocado na forma inglesa, ou seja, usando ponto para separar o valor inteiro do valor decimal.

Exemplo:

```
double variavel1 = 3.25;
double variavel2 = 1.47e2; // representando 1,47 * 10^2
```

- Tipo lógico:
  - **boolean**: para guardar um valor lógico (true ou false).

- Tipo alfanumérico:

  – **char**: para guardar um valor de caractere. Por exemplo: 't', ou 'x', ou '%'...

Percebam que o valor literal de uma variável do tipo char deve ser sempre colocado entre aspas simples, como em 'c'.

Observação

Uma vez declarada uma variável, ela não poderá sofrer alteração de seu tipo ao longo do bloco em que se encontra.

2.2 Strings

Na linguagem Java, as strings são classes (não fazem parte do grupo dos tipos primitivos) e possuem uma forma de declaração específica que permite que uma variável seja declarada de forma equivalente a uma variável primitiva.

Declaração na forma de classe:

```
String str1 = new String();
```

Declaração na forma equivalente ao de uma variável primitiva:

```
String str1 = "Esta é uma String...";
```

Percebam que, na declaração em que damos um valor literal à variável do tipo string, o valor deve ser descrito sempre entre aspas duplas.

Para que possamos concatenar (unir) duas strings, utilizamos o símbolo de adição + (mais).

Exemplo:

```
String str01 = "abc";
String str02 = "def";
String str03 = str01 + str02;
// Uma String em Java aceita concatenação recursiva
str03 += "ghi";
System.out.println(str03); // imprime: abcdefghi
```

Veremos adiante alguns métodos utilizados no tratamento de strings. Por enquanto devemos entender que uma string é um conjunto de caracteres caracterizando uma palavra ou uma frase.

Observação

Diferentemente de outras linguagens (como Python ou C#), na linguagem Java uma string não é um array (uma matriz) de caracteres.

Exemplo de aplicação

Uma empresa recebe, de tempos em tempos, um consultor de informática que pertence a uma empresa terceirizada. Como esse consultor normalmente tem que acessar determinados programas do servidor, a empresa decidiu fazer uma implementação de controle em um sistema já em funcionamento, para que seja inserida a figura de um usuário extra com acessos limitados. Como o sistema da empresa foi feito na linguagem Java, o consultor pretende criar uma classe chamada UsuarioExtra contendo atributos para as seguintes características:

- login do usuário;
- senha do usuário;
- valor cobrado por hora;
- quantidade de horas contratada (horas exatas, sem levar em consideração os minutos, segundos e demais submedidas);
- área cliente (cada área da empresa é considerada um cliente de TI e é representada por uma letra apenas);
- usuário administrador (valor lógico que indica se esse usuário deve ter ou não acesso a áreas restritas do servidor).

Qual dos itens a seguir mostra, **respectivamente**, a tipagem mais adequada para cada um dos atributos descritos anteriormente?

A) string, boolean, int, int, string, string.

B) char, char, int, double, string, boolean.

C) string, string, double, int, char, boolean.

D) string, string, double, double, double, char.

E) string, char, double, int, string, int.

Resposta correta: alternativa C.

Resolução

De acordo com o enunciado, a classe contém atributos que devem caracterizar a classe, e cada um desses atributos deve ser declarado com um tipo, que definirá a informação que nele pode ser guardada.

Segue então uma análise dos atributos um a um:

- O login do usuário é a sua identificação, geralmente representada por uma palavra que identifica o usuário e que, portanto, é uma informação alfanumérica, e deve ser representada por uma variável do tipo string.

- A senha do usuário pode ser composta por letras, números e símbolos; portanto, a variável que a representa deve ser do tipo string.

- O valor cobrado por hora costuma ser um valor monetário; portanto, deve ser representado por uma variável do tipo double (valor numérico real).

- A quantidade de horas contratada, por ser uma "hora exata" (de acordo com o enunciado), não deve levar em consideração as subdivisões de horários; portanto, deve ser representada por uma variável do tipo numérico inteiro.

- A área cliente (por exemplo, as áreas internas de uma empresa, como RH, financeiro, jurídico etc.), segundo o enunciado, devem ser representadas por uma única letra. Neste caso, o sistema aceitaria uma representação por uma variável do tipo char ou string.

- A informação usuário administrador, segundo o enunciado, possui um valor lógico true (verdadeiro) ou false (falso), indicando se o usuário pode ou não acessar áreas restritas do servidor; portanto, deve ser representada por uma variável do tipo boolean.

Assim, teremos:

A) Alternativa incorreta.

Justificativa: a senha do usuário não pode ser representada por uma variável do tipo boolean. Além disso, não é adequado representar um valor monetário por um tipo inteiro nem uma variável lógica como string.

B) Alternativa incorreta.

Justificativa: nem o login, nem a senha do usuário podem ser representados por uma variável do tipo char, já que este tipo aceita apenas um caractere de informação. Além disso, não é adequado representar um valor monetário por um tipo inteiro.

C) Alternativa correta.

Justificativa: essa sequência de tipagens está de acordo com a explicação dada anteriormente. Quanto à última variável (que representa a área cliente), poderia até ter um valor do tipo string, mas o mais adequado, já que ela só receberá uma letra (segundo o enunciado), seria uma variável do tipo char.

D) Alternativa incorreta.

Justificativa: a quantidade de horas contratadas está especificando horas exatas sem os minutos e segundos; portanto, deve ser um valor inteiro. Não há como representar uma área da empresa com um valor do tipo double, e a última variável, por ter um valor lógico, deve ser representada por uma variável do tipo boolean.

E) Alternativa incorreta.

Justificativa: não é possível representar uma senha por uma variável do tipo char, e os dois últimos tipos não são adequados para as variáveis que representam.

2.3 Estruturas condicionais

As estruturas condicionais são consideradas estruturas de decisão.

No Java existem basicamente duas estruturas condicionais:

- a estrutura if – else if – else;
- a estrutura switch – case.

2.3.1 if – else if – else

Na linguagem Java, essa estrutura segue a seguinte sintaxe:

Na sua forma simples

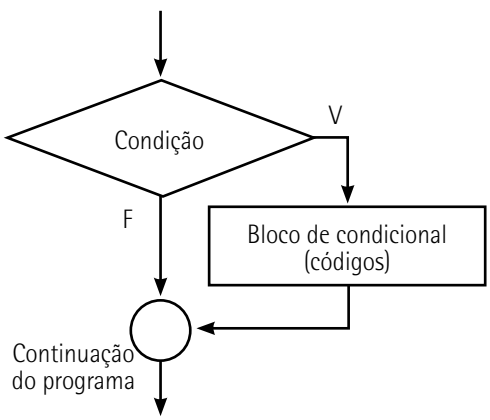

Figura 10 – Fluxograma da estrutura condicional simples

```
if (condicao) {
   <bloco da condicional>
}
```

A condição que define essa estrutura é uma expressão de valor lógico (uma comparação que pode ser verdadeira ou falsa), e o bloco da condicional representa o conjunto de código que somente será executado se a condição (da declaração da estrutura if) for verdadeira.

Percebam que o <bloco da condicional> está entre as chaves {...} que definem a estrutura, de forma que todo código que estiver dentro dessas chaves fará parte da estrutura.

Exemplo:

```
"...
if (nota < 6) {
   System.out.println("Nota considerada baixa");
}
..."
```

Neste exemplo, a frase **Nota considerada baixa** somente será impressa na tela da console se a variável **nota** contiver um valor menor que 6. Do contrário, nada será impresso.

Na sua forma composta

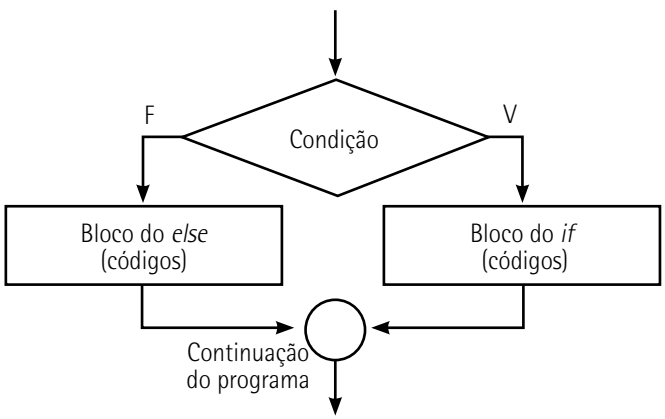

Figura 11 – Fluxograma da estrutura condicional composta

```
if (condicao) {
   <bloco do if>
} else {
   <bloco do else>
}
```

O bloco do if representa o conjunto de código que somente será executado se a condição (da declaração da estrutura if) for verdadeira e o bloco do else representar o conjunto de código que somente será executado se a condição (da declaração da estrutura if) for falsa.

Nesta estrutura, apenas um dos blocos será executado: ou o bloco do if, ou o bloco do else. Assim, após a execução de qualquer um dos blocos, o sistema irá executar os códigos que estiverem além do final da estrutura condicional (fora dela).

Exemplo:

```
"...
if (nota < 6) {
   System.out.println("Nota considerada baixa.");
} else {
   System.out.println("Boa nota.");
}
..."
```

Neste exemplo, a frase Nota considerada baixa. somente será impressa na tela da console se a variável nota contiver um valor menor que 6. De modo contrário (neste exemplo), o sistema imprimirá a frase Nota boa.

Na sua forma encadeada

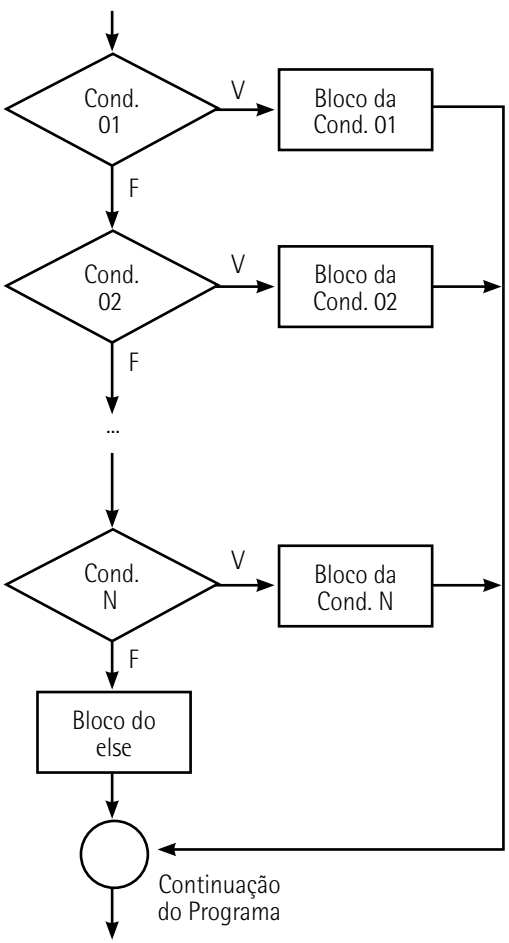

Figura 12 – Fluxograma da estrutura condicional encadeada

```
if (condicao_01) {
  <bloco da condição 01>
} else if (condicao_02) {
  <bloco da condição 02>
...
} else if (condicao_N) {
  <bloco da condição N>
} else {
  <bloco do else>
}
```

O bloco da condição 01 somente será executado se a condição_01 for verdadeira, o bloco da condição 02 somente será executado se a condição_01 for falsa e a condição_02 for verdadeira, e assim por diante, sendo que o bloco do else somente será executado se todas as condições anteriores da mesma estrutura forem falsas.

Nesta estrutura, apenas um dos blocos será executado, de forma que após a sua execução o sistema sairá da estrutura condicional, continuando a executar os códigos que estiverem além do seu final (fora dela).

Exemplo:

```
"...
if (nota < 3) {
   System.out.println("Nota de reprovação.");
} else if (nota < 5) {
   System.out.println("Nota baixa. Deve melhorar.");
} else if (nota < 7) {
   System.out.println("Tente uma nota mais alta.");
} else {
   System.out.println("Excelente nota.");
}
..."
```

Neste exemplo, apenas uma das frases será impressa na tela da console. Percebam a sequência dos valores de comparação, que seguem uma ordem crescente, possibilitando que o programa possa dar respostas adequadas, dependendo do valor da variável nota.

2.3.2 switch – case

A estrutura switch é uma estrutura condicional que, a partir do valor de uma variável, executa um dos seus blocos case definidos, aquele no qual se apresenta um valor exatamente igual ao da variável de controle.

Essa estrutura equivale a uma estrutura if – else if – else, em que a comparação feita nas suas condições seja a de igualdade.

A sintaxe desta estrutura é a seguinte:

```
switch (variavel) {
   case valor_01:
     <bloco do valor 01>
     break;
   case valor_02:
     <bloco do valor 02>
     break;
   ...
   case valor_N:
     <bloco do valor N>
     break;
   default:
     <bloco do default>
     break;
}
```

O bloco do valor 01 somente será executado se o valor da variável, definida na declaração (switch) da estrutura, for exatamente igual ao valor 01 definido na declaração do seu case. O bloco do valor 02 somente será executado se o valor da variável, definida na declaração (switch) da estrutura, for exatamente igual ao valor 02 definido na declaração do seu case, e assim por diante. O bloco default somente será executado se o valor da variável definida na declaração da estrutura for diferente a qualquer um dos valores declarados nos cases daquela estrutura.

A variável de comparação declarada no termo switch desta estrutura pode ter apenas um dos seguintes tipos: byte, short, char, int, string, além de algumas wrapper classes (algumas classes que representam tipos primitivos, como as classes character, byte, short e integer).

Um bom exemplo de utilização de uma estrutura switch – case é a geração de menus de opção, em que o usuário poderia selecionar, dentre uma lista de possíveis opções (na forma de números inteiros), a opção de ação que ele pretende realizar.

Além deste exemplo, pode-se utilizar a estrutura switch – case em processos automáticos que envolvem a leitura de arquivos externos (geralmente arquivos textos), em que há alguns identificadores (caracteres ou conjunto de caracteres) nesse texto que remetem o sistema a determinadas ações específicas dependendo do caractere lido naquela sequência.

Observação

Uma estrutura case pode ser, em qualquer ocasião, substituída por uma estrutura if.

No entanto, nem toda estrutura if pode ser substituída por uma estrutura case, como aquelas em que a lógica de comparação não é a lógica de igualdade.

Exemplo de aplicação da estrutura switch – case:

```
import javax.swing.JOptionPane;
public class TesteSwitch {
   public static void main(String[] args) {
     int x = Integer.parseInt(
              JOptionPane.showInputDialog("Digite o valor de x:"));
     switch(x) {
     case 1:
              System.out.println("O valor de x é igual a 1");
              break;
     case 2:
              System.out.println("O valor de x é igual a 2");
              break;
     case 3:
              System.out.println("O valor de x é igual a 3");
              break;
     default:
              System.out.println("O valor de x é maior que 3");
              System.out.println("...ou menor que 1");
              break;
     }
   }
}
```

2.4 Estruturas de repetição

As estruturas de repetição são estruturas que repetem a execução de determinado bloco (conjunto) de códigos, de acordo com uma regra pré-definida, dependendo da estrutura utilizada.

A cada uma das execuções do bloco definido pela estrutura de repetição, chamamos de **iteração**.

Lembrete

Atenção aos conceitos:

- **iteração** é uma única execução completa do bloco definido pela estrutura de repetição;
- **interação** é a ação mútua ou compartilhada entre duas ou mais entidades (por exemplo, entre duas pessoas, ou entre uma pessoa e o computador etc.).

2.4.1 for

A estrutura for, em Java, é uma estrutura de repetição utilizada quando se sabe exatamente a quantidade de repetições a serem executadas.

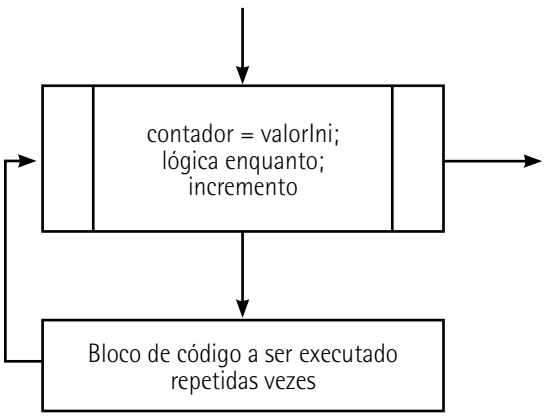

Figura 13 – Fluxograma da estrutura for

Essa estrutura é controlada por uma variável de valor inteiro, a qual chamamos de variável contadora (ou simplesmente contador). Essa variável terá um valor inicial, de forma que a lógica de repetição (enquanto) define o valor final da variável contadora, além do incremento que acontecerá ao final de cada iteração.

A sintaxe dessa estrutura é a seguinte:

```
for (<início>; <condição>; <variação>) {
    <bloco de código a ser executado repetidas vezas>
}
```

Onde:

- <início> = atribuição de um valor inicial à variável contadora.
- <condição> = condição (valor lógico) da repetição em relação à variável contadora.
- <variação> = variação do valor da variável contadora.

Exemplo:

```
for (int x = 1; x <= 10; x++) {
   System.out.println(x);
}
// o exemplo acima imprime na tela da console
// ...os números inteiros de 1 a 10
```

Arrays

A estrutura for é também muito útil quando precisamos trabalhar com matrizes (arrays) de dados.

Em Java, um array é uma variável que pode conter mais de um valor, desde que todos os valores daquele mesmo array tenham o mesmo tipo de informação.

Para gerarmos um array de informações, utilizamos a seguinte sintaxe:

```
tipoArray[] var = new tipoArray[tamanho];
```

A variável var será um array de dados que terão (todos) o tipo definido em tipoArray, e que terá a quantidade de valores definida em tamanho.

Os valores de um array são organizados em posições definidas por índices (números inteiros sequenciais que começam sempre no valor 0). Assim, para ler ou alterar um valor de uma determinada posição do array, basta indicar o índice daquele valor no array com colchetes, na seguinte sintaxe: array[índice].

Exemplo:

```
int[] a = new int[3];
a[0] = 4;
a[1] = -15;
a[2] = 72;
System.out.println(a[1]);
// imprimirá o valor -15
```

No exemplo, a variável a será um array de dados do tipo int com três valores, sendo que na primeira posição (no índice 0) estará o valor 4, na segunda posição (no índice 1) estará o valor -15 e na terceira posição (no índice 2) estará o valor 72.

Observação

Em informática, os índices que indicam a posição do valor em um conjunto de valores são sempre números inteiros, começam sempre no valor 0 e são sempre sequenciais.

É possível se gerar um array de dados, já diretamente (ou explicitamente) com seus valores, utilizando a seguinte sintaxe:

```
double[] b = {1.2, 7.3, -0.8, 4};
```

No exemplo, a variável b foi gerada diretamente como um array de dados do tipo double, com quatro valores numéricos, ou seja, contendo o tamanho 4. Assim, para que possamos acessar cada um dos valores do array, podemos nos utilizar da estrutura for, já que ela é controlada por uma variável contadora, que pode representar cada um dos índices, indicando a posição de cada um dos valores de um array.

Vejamos o exemplo a seguir, que acessa os valores de dois arrays utilizando a estrutura for de repetição:

```
int[] a = new int[3];
a[0] = 4;
a[1] = -15;
a[2] = 72;
double[] b = {1.2, 7.3, -0.8, 4};
// Acessando cada um dos arrays acima (a e b)
// utilizando a estrutura for
for(int x = 0; x < a.length; x++) {
   System.out.println(a[x]);
}
for(int x = 0; x < b.length; x++) {
   System.out.println(b[x]);
}
```

Lembrete

Um array é indexado por números inteiros que sempre iniciarão no valor 0 e sempre serão sequenciais, de modo que nunca existirão lacunas na sua sequência numérica.

2.4.2 while

A estrutura while é uma estrutura de repetição controlada por uma condição, de forma que o bloco definido pela estrutura será executado repetidas vezes enquanto a condição verificada na declaração for verdadeira.

A sintaxe dessa estrutura é a seguinte:

```
while (<condição de repetição>) {
   <bloco de código a ser executado repetidas vezas>
}
Exemplo:
int x = 1;
while (x <= 10) {
   System.out.println(x);
   x++;
}
// assim como o exemplo dado para a estrutura "for",
// o exemplo acima também imprimirá na tela da console
// ...os números inteiros de 1 a 10
```

Percebe-se no exemplo que o valor da variável de controle (x) é acrescido de uma unidade ao final de cada iteração. No entanto, o comando que definiu esse acréscimo (**x++;**) teve que ser adicionado dentro do bloco da estrutura.

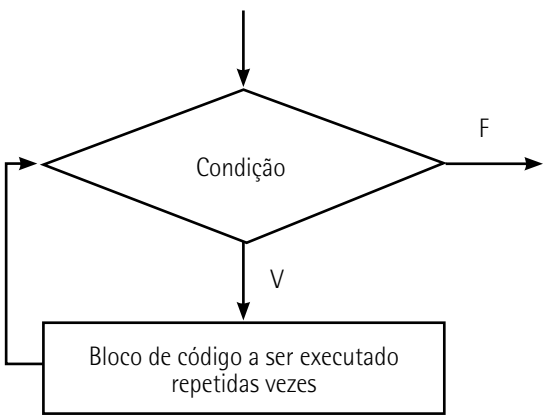

Figura 14 – Fluxograma da estrutura while

Nessa estrutura, devemos sempre tomar cuidado para que a condição definida na sua declaração se torne falsa em algum momento, a fim de que cessem as iterações, saindo da execução repetida do bloco da estrutura. Se isso não ocorrer (se a condição nunca se tornar falsa), ocorre o que chamamos de looping infinito (ou laço infinito), caso em que o sistema nunca finalizará a execução das repetições (as iterações).

2.4.3 do – while

A estrutura do – while é uma estrutura de repetição controlada por uma condição posterior à execução dos códigos de dentro de seu bloco, de forma que, assim como a estrutura while, o bloco definido pela estrutura será executado repetidas vezes enquanto a condição verificada na declaração for verdadeira.

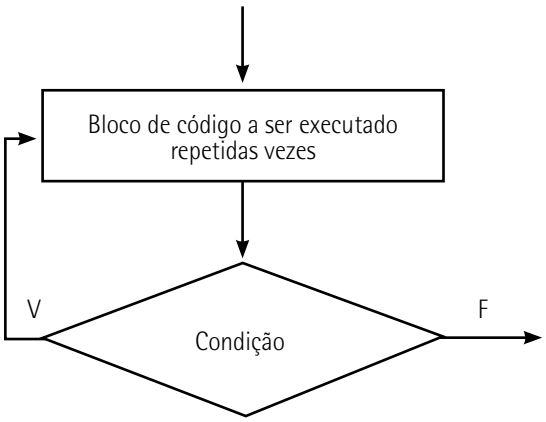

Figura 15 – Fluxograma da estrutura do – while

A diferença dessa estrutura em comparação com a estrutura while é que neste caso a condição é verificada no final da estrutura (ao final da iteração). Efetivamente, a única diferença funcional entre elas pode ser percebida sobre a situação em que a condição seja inicialmente falsa. Neste caso, com a estrutura while, o bloco de códigos interno à estrutura não será executado, mas com a estrutura do – while, esse mesmo bloco interno à estrutura, nessas mesmas condições, será executado ao menos uma vez.

A sintaxe dessa estrutura é a seguinte:

```
do {
   <bloco de código a ser executado repetidas vezas>
} while (<condição de repetição>);
```

Observe que a condição está sendo verificada ao final do bloco e que sua linha de definição termina com ponto-e-vírgula (;).

Também nessa estrutura, devemos sempre tomar cuidado para que a condição definida ao final de sua declaração se torne falsa em algum momento, a fim de que cessem as iterações, saindo da execução repetida do bloco da estrutura, sem que ocorra o looping infinito.

Exemplo:

```
int x = 1;
do {
   System.out.println(x);
   x++;
} while (x <= 10);
// assim como no exemplo dado para a estrutura "while",
// o exemplo acima imprime na tela da console
// ...os números inteiros de 1 a 10
```

A seguir, um exemplo das estruturas de repetição while e do – while, com resultados diferentes:

```
int x = 400;
while (x <= 10) {
   System.out.println(x);
   x++;
}
// neste caso nada será impresso na console,
// já que a condição inicial é falsa
do {
   System.out.println(x);
   x++;
} while (x <= 10);
// neste caso será impresso pelo menos o valor 400,
// de forma que como a condição é falsa, não haverá
// mais iterações, saindo da estrutura de repetição.
```

2.5 Packages (pacotes)

Num projeto Java, criamos packages (ou pacotes) para separarmos as classes em categorias (o equivalente a pastas, quando queremos organizar nossos arquivos no computador), e com isso podemos organizar as classes de nosso projeto.

Fisicamente (por exemplo, se procurarmos no Explorer do Windows), um pacote é identificado como um ou mais diretórios que contêm internamente uma ou mais classes Java.

Por exemplo,

```
package model;
```

é o diretório model que fica na raiz do projeto Java.

Por exemplo,

```
package com.mysql.jdbc;
```

é o diretório com/mysql/jdbc (três diretórios, um dentro do outro) que fica na raiz do projeto Java.

Quando uma classe está localizada em um pacote, internamente, na classe, essa situação precisa estar definida (em código), identificando o pacote em que ela se localiza, por meio da palavra reservada package.

```
package nome.do.pacote;
//esta Classe pertence ao pacote "nome.do.pacote"
```

Quando vamos utilizar (ou referenciar/importar) ao longo da classe uma outra classe que está em outro pacote ou em outra biblioteca de classes, somos obrigados a importar aquela classe, utilizando a palavra reservada import, e indicando o caminho até ela:

```
import nome.do.pacote.Classe;
```

Observação

Uma biblioteca de classes é um conjunto de classes que ficam agrupadas geralmente em um arquivo com extensão .jar, o qual foi criado com o intuito de conter diversas classes que podem ser utilizadas por vários projetos ao mesmo tempo. Nesse arquivo, todas as classes estão dispostas em packages diversos. Esse é um arquivo compactado específico que pode ser preparado pelo Eclipse, exportando-se um projeto como arquivo jar (não executável).

Saiba mais

O link a seguir contém um texto que faz parte de um tutorial oficial do Java, do site da Oracle, e que explica como e para que geramos arquivos .jar.

ORACLE. *The Java™ tutorials*: using JAR files – the basics. [s.d.]c. Disponível em: https://cutt.ly/YTfIwV2. Acesso em: 21 out. 2021.

Caso se queira importar **todas** as classes do pacote, no lugar do nome da classe, colocamos o caractere * (asterisco).

```
import nome.do.pacote.*;
```

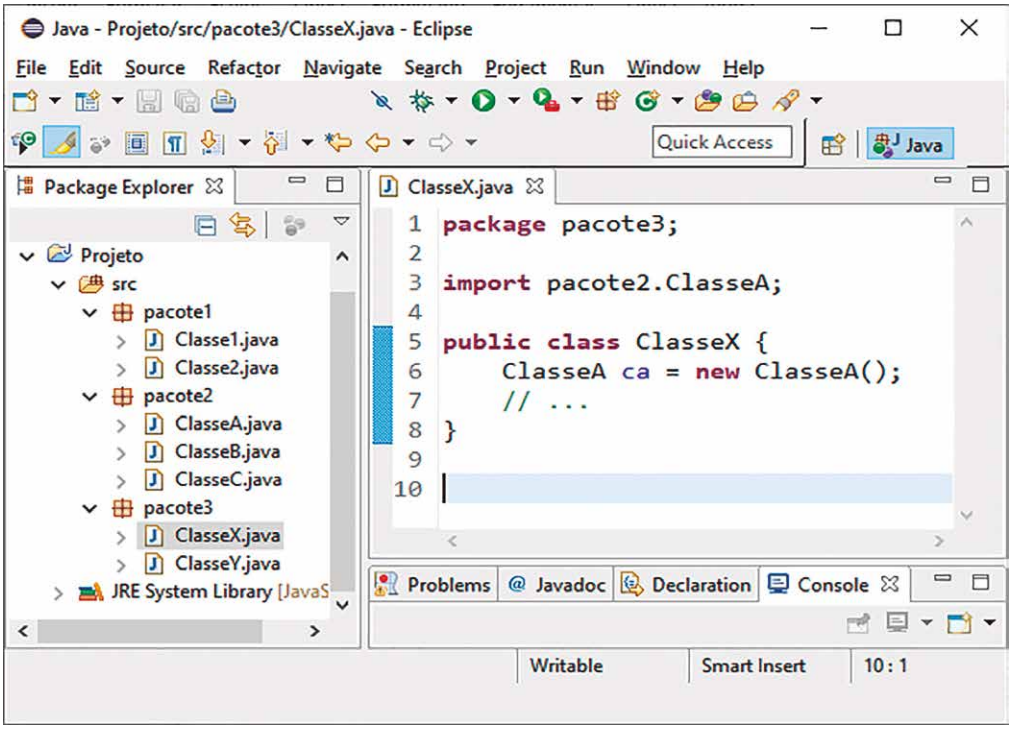

Figura 16 – Projeto organizado em packages

Na imagem, podemos observar um projeto Java organizado em três pacotes, sendo que o pacote pacote1 contém duas classes, o pacote pacote2 contém três classes e o pacote pacote3 contém duas classes. Percebe-se que, na área central, vê-se o código da classe ClasseX, que pertence ao pacote pacote3 e por isso contém definida a seguinte linha de código:

```
package pacote3;
```

Também se observa que um dos atributos da ClasseX é um objeto da ClasseA que está em outro pacote, e por isso precisou ser importado com o seguinte comando:

```
import pacote2.ClasseA;
```

Para finalizar esta unidade, observe os exemplos de aplicação a seguir. As resoluções são apresentadas na sequência.

Exemplos de aplicação

1 – Criar uma classe (ClasseA) com o método main que deverá ter duas variáveis do tipo inteiro (v1 e v2) contendo quaisquer valores numéricos. Criar uma terceira variável do tipo inteiro (v3) que deverá receber o cálculo da soma das duas variáveis iniciais, e no final o programa deverá mostrar na tela da console o valor dessa soma com o texto:

"Soma = <valor>"

2 – Criar uma classe (ClasseB) com dois atributos (a1 e a2) do tipo double e o método main. Criar um método chamado mostrarValores que exibe na tela da console em linhas diferentes os valores dos atributos com o texto ("Valor do atributo a1: <valor>" e na linha seguinte o texto equivalente ao outro aributo). No método main deve-se inserir valores nos atributos, sendo que para isso deve-se antes instanciar a classe; a partir dela, inserir os valores nos atributos; e, ao final, mostrar os valores dos atributos chamando o método mostrarValores.

3 – Criar uma classe com o nome Calculadora (sem o método main). Essa classe deverá possuir três atributos públicos do tipo double (n1, n2 e res) e quatro métodos públicos sem retorno (somar, subtrair, multiplicar e dividir), de modo que cada método deverá realizar a respectiva conta com os dois atributos (n1 e n2) e, sem retornar valor algum, inserir o valor do resultado no atributo res.

Criar também a classe TesteCalc com o método main, que deverá realizar (a partir da classe Calculadora) e mostrar o resultado das seguintes contas:

a) 5 + 3,7

b) 18,4 – 23,2

c) 7 * 1,5 + 4,2

d) 13,8 / 4,2 + 1,8

4 – Criar uma classe (de nome Verificadora), sem o método main, que possui dois métodos que recebem um valor numérico inteiro como parâmetro, de modo que o método verificaParImpar**( )** deverá retornar a string "PAR", "IMPAR" ou "NULO" (este último, para o caso de receber o valor zero) de acordo com o valor do módulo (valor absoluto ou valor equivalente positivo) do número recebido por parâmetro, e o método verificaNegPos() deverá retornar a string "POSITIVO", "NEGATIVO" ou "ZERO", dependendo do valor recebido por parâmetro.

Essa classe Verificadora deverá possuir também um método de nome verificarValor(), que recebe um número como parâmetro e imprime o resultado segundo os dois outros métodos da classe, de modo a mostrar na tela da console um texto equivalente ao seguinte (imaginando que este último método tenha recebido o valor -2):

"O número -2 é NEGATIVO e tem módulo PAR."

Criar uma classe TesteVerificadora que deverá instanciar a classe Verificadora e em seguida verificar (a partir do método verificarValor()) os seguintes números: 101, 24, -17 e -8.

5 – Criar uma classe (ClasseC) com o método intervaloComFor(...) e o método intervaloComWhile(...) de forma que ambos recebam como parâmetro dois números inteiros, um representando o início de um intervalo e o outro representando o final do intervalo. Eles deverão mostrar na tela da console

os números inteiros do intervalo (inclusive eles mesmos) de acordo com as especificações a seguir (observação: o primeiro método deverá utilizar laço de repetição for e o outro, laço de repetição while):

- Criar um terceiro método com o nome chamarMetodos(...) que recebe dois números inteiros como parâmetro e simplesmente aciona os dois métodos anteriores na sequência com os mesmos números.

- O programa deverá dar como resultado na tela da console um texto que deverá estar no seguinte formato de amostragem (imaginando que o método chamarMetodos(...) tenha recebido os valores 3 e 9):

  "...
  Resultado com for:
  3, 4, 5, 6, 7, 8, 9
  Resultado com while:
  3, 4,
  5, 6,
  7, 8,
  9
  ..."

Criar uma classe TesteC que instancia a ClasseC deste enunciado e roda o método chamarMetodos(...) com os valores:

a) 3 e 9 (o mesmo do exemplo mostrado no enunciado).

b) -7 e 10.

Resolução

1 –

```
public class ClasseA {
   public static void main(String[] args) {
     int v1 = 7;
     int v2 = 20;
     int v3 = v1 + v2;
     System.out.println("Soma = " + v3);
   }
}
```

2 –

```
public class ClasseB {
    public double a1;
    public double a2;
    public static void main(String[] args) {
      ClasseB cb = new ClasseB();
      cb.a1 = 73.42;
      cb.a2 = -105.77;
      cb.mostrarValores();
    }
    public void mostrarValores() {
      System.out.println("Valor do atributo a1: " + a1);
      System.out.println("Valor do atributo a2: " + a2);
    }
}
```

3 –

```
public class Calculadora {
    public double n1;
    public double n2;
    public double res;
    public void somar() {
      this.res = n1 + n2;
    }
    public void subtrair() {
      this.res = n1 - n2;
    }
    public void multiplicar() {
      this.res = n1 * n2;
    }
    public void dividir() {
      this.res = n1 / n2;
    }
}
public class TesteCalc {
    public static void main(String[] args) {
      // Instanciando a Classe Calculadora
      Calculadora calc = new Calculadora();
      // Realizando a conta: 5 + 3,7
      // ..lembrando que os números decimais devem ser
      // ..descritos com ponto ao invés de vírgula
      calc.n1 = 5;
```

```
        calc.n2 = 3.7;
        calc.somar();
        System.out.println("Resultado: 5 + 3,7 = " + calc.res);
        // Realizando a conta: 18,4 – 23,2
        calc.n1 = 18.4;
        calc.n2 = 23.2;
        calc.subtrair();
        System.out.println("Resultado: 18,4 – 23,2 = "
                                                        + calc.res);
        // Realizando a conta: 7 * 1,5 + 4,2
        //..obs.: para realizar 2 contas devemos nos utilizar
        //..de um recurso que é inserir o resultado num dos
        //..atributos.
        calc.n1 = 7;
        calc.n2 = 1.5;
        calc.multiplicar();
        calc.n1 = calc.res;
        calc.n2 = 4.2;
        calc.somar();
        System.out.println("Resultado: 7 * 1,5 + 4,2 = "
                                                        + calc.res);
        // Realizando a conta: 13,8 / 4,2 + 1,8
        calc.n1 = 13.8;
        calc.n2 = 4.2;
        calc.dividir();
        calc.n1 = calc.res;
        calc.n2 = 1.8;
        calc.somar();
        System.out.println("Resultado: 13,8 / 4,2 + 1,8 = "
                                                        + calc.res);
    }
}
```

4 –

```
public class Verificadora {
    public String verificaParImpar(int valor) {
        // tornando o valor recebido em seu valor absoluto
        if (valor < 0) valor *= -1;
        String txt = "NULO";
        if (valor != 0) {
                txt = "PAR";
                int res = valor % 2;
                if (res != 0) txt = "IMPAR";
```

```
    }
    return txt;
  }
  public String verificaNegPos(int valor) {
    String txt = "ZERO";
    if (valor != 0) {
          txt = "POSITIVO";
          if (valor < 0) txt = "NEGATIVO";
    }
    return txt;
  }
  public void verificarValor(int valor) {
    String txt = "O número " + valor;
    txt += " é " + verificaNegPos(valor);
    txt += " e tem módulo " + verificaParImpar(valor) + ".";
    System.out.println(txt);
  }
}
public class TesteVerificadora {
  public static void main(String[] args) {
    // Instanciando a Classe Verificadora
    Verificadora ver = new Verificadora();
    // Realizando as verificações solicitadas
    ver.verificarValor(101);
    ver.verificarValor(24);
    ver.verificarValor(-17);
    ver.verificarValor(-8);
  }
}
```

5 –

```
public class ClasseC {
  public void intervaloComFor(int ini, int fin) {
    String txt = "";
    for (int x = ini; x <= fin; x++) {
          txt += x;
          if (x != fin) txt += ", ";
    }
    System.out.println(txt);
  }
  public void intervaloComWhile(int ini, int fin) {
    String txt = "";
    int x = ini;
```

```
        int cont = 0;
        while (x <= fin) {
                txt += x;
                if (x != fin) txt += ", ";
                if (cont == 1) {
                        txt += "\n";
                        cont = 0;
                } else {
                        cont++;
                }
                x++;
        }
        System.out.println(txt);
    }
    public void chamarMetodos(int ini, int fin) {
        System.out.println("Resultado com for:");
        intervaloComFor(ini, fin);
        System.out.println("Resultado com while:");
        intervaloComWhile(ini, fin);
    }
}
public class TesteC {
    public static void main(String[] args) {
        ClasseC cc = new ClasseC();
        cc.chamarMetodos(3, 9) {
        cc.chamarMetodos(-7, 10) {
    }
}
```

Resumo

Nesta unidade, vimos alguns conceitos básicos de programação e como eles são aplicados na linguagem Java.

Quando programamos nessa linguagem, trabalhamos basicamente apenas com três elementos: as **classes**, que são os modelos que utilizamos para construir os objetos (as instâncias daquela classe) em memória; os **métodos**, que são as ações definidas nas classes e que somente serão acionados se forem explicitamente "chamados" no programa; e os **atributos**, que representam as informações que caracterizam as classes.

Utilizamos como IDE (ambiente integrado de desenvolvimento) o Eclipse, que é um programa gratuito criado pela IBM e que facilita a criação de projetos em Java.

A linguagem Java possui algumas convenções de programação que regulamentam desde a forma com que se programa até os nomes dos elementos que trabalhamos. Vimos que, ao programarmos, devemos colocar todos os atributos no início da classe e que os métodos devem estar depois dos atributos. O nome de uma classe deve iniciar com letra maiúscula, enquanto o nome dos métodos e dos atributos devem iniciar com letra minúscula.

Os métodos devem ser descritos com a seguinte sintaxe:

```
modificadores tipoDeRetorno nomeDoMetodo(parâmetros) {...}
```

A sintaxe dos parâmetros do método (quando houver) deve ser:

(tipo1 parametro1, tipo2 parametro2, ..., tipoN parametroN)

O modificador é um termo que pode definir como o elemento poderá ser acessado (modificadores de acesso), ou como ele se comportará (modificadores de comportamento) no sistema.

Os atributos devem ser descritos com a seguinte sintaxe:

```
modificadores tipo nomeDoAtributo;
```

Ou, ainda:

```
modificadores tipo nomeDoAtributo = valorInicial;
```

Num programa, os comentários (que podem ser inseridos como comentário de linha utilizando `//`, ou comentário multilinha utilizando `/*...*/`), são colocados no sistema para que este possa ser explicado a todo desenvolvedor ou analista que for dar manutenção ou implementar o sistema. Esses comentários não interferem na execução do código, de forma que são ignorados quando o sistema gera o arquivo compilado do programa.

Quando geramos um programa (uma classe), o fazemos em um arquivo com extensão .java, que é um "arquivo fonte" com o código criado pelo desenvolvedor. Quando vamos executar esse programa, o Eclipse gera automaticamente o arquivo compilado desse programa, que é um arquivo com extensão .class utilizado pela JVM para rodar as aplicações.

A linguagem Java é fortemente tipada, o que significa que para que possamos utilizar variáveis locais ou globais, elas devem ser previamente declaradas no programa. A sintaxe dessa declaração é:

```
tipoDaVariavel nomeVariavel;
```

Ou, ainda:

```
tipoDaVariavel nomeVariavel = valor;
```

Seu tipo define o tipo da informação que nós podemos guardar nela. Os tipos mais utilizados são: int (para valores inteiros), double (para valores reais), string (que é considerada uma classe e é utilizada para guardar frases e palavras) e boolean (para valores lógicos).

As estruturas condicionais são aquelas que definem se um bloco de linhas de código, definido pelo espaço entre a abertura e o fechamento de chaves, será ou não executado, dependendo do resultado de determinadas condições (comparações). Na linguagem Java existem algumas estruturas condicionais, como a estrutura simples (if), a estrutura composta (if – else), a estrutura encadeada (if – else if – else) e a estrutura case (switch – case), sendo que nesta última a única comparação possível é a de igualdade. Nas estruturas condicionais, apenas um bloco de código é executado: aquele cuja primeira condição, ou comparação, tiver resultado verdadeiro. Suas sintaxes são:

Da estrutura if (geral):

```
if (condicao_01) {
        <bloco da condição 01>
} else if (condicao_02) {
        <bloco da condição 02>
...
} else if (condicao_N) {
        <bloco da condição N>
} else {
        <bloco do else>
}
```

Da estrutura switch – case:

```
switch (variavel) {
        case valor_01:
                <bloco do valor 01>
                break;
        case valor_02:
                <bloco do valor 02>
                break;
        ...
        case valor_N:
                <bloco do valor N>
                break;
        default:
                <bloco do default>
                break;
}
```

As estruturas de repetição são aquelas que permitem que um determinado conjunto de códigos seja executado repetidas vezes. No Java temos três estruturas de repetição: a estrutura for, em que a repetição é controlada por uma variável de controle (em geral uma variável contadora); a estrutura while, em que a repetição se dá enquanto uma condição for verdadeira, sendo que essa condição é verificada antes de cada iteração; e a estrutura do – while, em que a repetição também se dá enquanto uma condição for verdadeira, sendo que neste caso essa condição é verificada ao final de cada iteração. Suas sintaxes são:

Da estrutura for:

```
for (<início>; <condição>; <variação>) {
                <bloco de código a ser executado repetidas vezas>
}
```

Da estrutura while:

```
while (<condição de repetição>) {
        <bloco de código a ser executado repetidas vezas>
}
```

Da estrutura do – while:

```
do {
        <bloco de código a ser executado repetidas vezas>
} while (<condição de repetição>);
```

Exercícios

Questão 1. (Sugep – UFRPE/2018. Adaptada) Leia o texto a seguir, a respeito do paradigma de orientação a objetos:

O paradigma de orientação a objetos surge como o advento da reutilização de código e a facilidade na manutenção. Seu princípio é a construção de código implementando as entidades do mundo real, por meio do conceito de classes que possuem relação entre si. Pode-se considerar que a programação orientada a objetos (POO) está embasada nos seguintes pilares.

- **Abstração**: como estamos lidando com objetos do mundo real, por exemplo, carro, casa, pessoa etc., precisamos imaginar como esses objetos vão se integrar dentro do nosso sistema e modelar seu comportamento, abstraindo o comportamento e características específicas de cada um.

- **Encapsulamento**: não importa, para um código que invoca um método, saber como outro vai ser executado. O encapsulamento evita a utilização inapropriada dos atributos de uma classe, de modo a possibilitar o controle de seus valores por meio dos seus métodos. Trata-se de uma característica que traz principalmente segurança ao código.

- **Herança**: assim como no mundo real, a herança em programação orientada a objetos seria a capacidade de uma classe herdar de outra métodos e atributos, sendo, portanto, uma característica relacionada ao reúso de código.

- **Polimorfismo**: existem animais capazes de se adaptar a algumas necessidades do mundo real e se comportar de forma diferenciada, em alguns casos. Essa particularidade também é possível no paradigma de orientação a objetos. Mesmo herdando o comportamento de outra classe, a classe herdeira pode modificar o seu comportamento em determinadas situações.

Adaptado de: SILVA, F. M. da. *Paradigmas de programação*. Porto Alegre: Sagah, 2019. p. 21.

Com base na leitura e nos seus conhecimentos sobre programação orientada a objetos (POO), avalie as afirmativas.

I – POO é um paradigma de programação, sugerindo um padrão de desenvolvimento que independe da linguagem utilizada.

II – O conceito de herança está fortemente atrelado ao reúso de código, o que pode aumentar a produtividade da aplicação.

III – Apesar de Java ter sido projetada para orientação a objetos, os códigos nessa linguagem podem ser escritos utilizando-se programação estruturada.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa E.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: POO não é uma linguagem de programação, e sim um paradigma de programação, ou seja, é uma modelagem de escrita de código. Para que uma linguagem de programação seja considerada orientada a objetos, deve possibilitar a implementação dos conceitos de herança, polimorfismo e encapsulamento. Alguns exemplos dessas linguagens são C++, C#, Python e, é claro, Java.

II – Afirmativa correta.

Justificativa: a herança diz respeito à capacidade de uma classe herdar métodos e atributos de outra classe. Está, portanto, atrelada ao reúso de código e ao aumento da produtividade da aplicação.

III – Afirmativa correta.

Justificativa: o paradigma de programação estruturada é aquele que, geralmente, aprendemos no nosso primeiro contato com programação. Linguagens como Cobol, C e Pascal implementam esse paradigma. Ele é baseado em três estruturas básicas:

- **sequência** (os passos são executados linearmente, um após o outro);
- **decisão** (utiliza comandos como if-else e switch, que "decidem" se determinado trecho de código será executado ou não);
- **iteração** (execução repetitiva de determinado trecho de código, que pode ser feita por comandos como for e while).

Por mais que seja considerada uma linguagem orientada a objetos, é possível, sim, desenvolvermos códigos em Java utilizando programação estruturada. Esses códigos, muitas vezes, resultam do "vício" do programador em seguir o paradigma estruturado.

Questão 2. (IF-MT/2019) Considere o trecho de código, escrito na linguagem Java, apresentado a seguir.

```
 1 - public class MinhaClasse
 2 - {
 3 -    public static void main(String[] args)
 4 -    {
 5-         int numero1 = 13;
 6 -        int numero2 = 31;
 7 -        if (numero1 = numero2)
 8 -            System.out.print("Os numeros sao iguais");
 9 -        else
10 -          System.out.print("Os numeros sao diferentes");
11 -    }
12 - }
```

É correto afirmar que:

A) A sua execução apresentará a mensagem: Os numeros sao iguais.

B) A sua execução apresentará a mensagem: Os numeros sao diferentes.

C) A sua compilação apresentará uma mensagem de erro na linha 1.

D) A sua compilação apresentará uma mensagem de erro na linha 3.

E) A sua compilação apresentará uma mensagem de erro na linha 7.

Resposta correta: alternativa E.

Análise da questão

É fácil fazermos a análise do objetivo do código, que consiste apenas na comparação entre os valores das variáveis numero1 e numero2, seguida da exibição de uma mensagem. Porém, justamente na comparação, na linha 7, ocorre um deslize que fará com que seja apresentada uma mensagem de erro durante a compilação.

Em Java, o operador = é um operador de atribuição. Observe o exemplo a seguir.

int x = 23;

Nessa linha, foi declarada a variável x, do tipo inteiro, à qual foi atribuído o valor 23. Para compararmos valores entre si, devemos utilizar o operador relacional ==, que verifica se dois operandos são iguais entre si. Portanto, para que o código seja corrigido, a linha 7 deve ser modificada para

```
if (numero1 == numero2)
```

É importante notar que, em Java, blocos de comandos que têm apenas uma instrução não precisam, obrigatoriamente, estar envolvidos por chaves. Esse é o caso do bloco do if e do bloco do else da questão. Logo, a ausência de chaves não geraria uma mensagem de erro durante a compilação. No entanto, é considerada uma boa prática sempre envolver blocos de declaração por um par de chaves, de forma a evitar problemas em futuras edições do código.