PROGRAMAÇÃO DE SISTEMAS II

Professor Esp. Ronie Cesar Tokumoto

GRADUAÇÃO
Unicesumar

UNICESUMAR
Av. Guedner, 1610 - Jardim Aclimação
Cep 87050-900 - MARINGÁ - PARANÁ
unicesumar.edu.br
44 3027.6360

UNICESUMAR EDUCAÇÃO A DISTÂNCIA
NEAD - Núcleo de Educação a Distância
Bloco 4 - MARINGÁ - PARANÁ
unicesumar.edu.br
0800 600 6360

FICHA CATALOGRÁFICA

C397 **CENTRO UNIVERSITÁRIO DE MARINGÁ.** Núcleo de Educação a Distância; **TOKUMOTO**, Ronie Cesar.

Programação de Sistemas II. Ronie Cesar Tokumoto.
Maringá-Pr.: UniCesumar, 2018.
148 p.
"Graduação - EaD".

1. Programação. 2. Sistemas . EaD. I. Título.

ISBN: 978-85-459-0845-6 CDD - 22 ed. 005
CIP - NBR 12899 - AACR/2

Ficha catalográfica elaborada pelo bibliotecário
João Vivaldo de Souza - CRB-8 - 6828

Impresso por:

UniCesumar

Reitor
Wilson de Matos Silva

Vice-Reitor
Wilson de Matos Silva Filho

Pró-Reitor de Administração
Wilson de Matos Silva Filho

Pró-Reitor de EAD
Willian Victor Kendrick de Matos Silva

Presidente da Mantenedora
Cláudio Ferdinandi

NEAD - Núcleo de Educação a Distância

Direção Operacional de Ensino
Kátia Coelho

Direção de Planejamento de Ensino
Fabrício Lazilha

Direção de Operações
Chrystiano Mincoff

Direção de Mercado
Hilton Pereira

Direção de Polos Próprios
James Prestes

Direção de Desenvolvimento
Dayane Almeida

Direção de Relacionamento
Alessandra Baron

Head de Produção de Conteúdos
Rodolfo Encinas de Encarnação Pinelli

Gerência de Produção de Conteúdos
Gabriel Araújo

Supervisão do Núcleo de Produção de Materiais
Nádila de Almeida Toledo

Coordenador de Conteúdo
Fabiana de Lima

Design Educacional
Isabela Agulhon

Iconografia
Ana Carolina Martins Prado

Projeto Gráfico
Jaime de Marchi Junior
José Jhonny Coelho

Arte Capa
André Morais de Freitas

Editoração
Robson Yuiti Saito

Revisão Textual
Alisson Pepato
Cíncitia Prezoto Ferreira

Ilustração
Marcelo Goto

Professor
Wilson de Matos Silva
Reitor

Viver e trabalhar em uma sociedade global é um grande desafio para todos os cidadãos. A busca por tecnologia, informação, conhecimento de qualidade, novas habilidades para liderança e solução de problemas com eficiência tornou-se uma questão de sobrevivência no mundo do trabalho.

Cada um de nós tem uma grande responsabilidade: as escolhas que fizermos por nós e pelos nossos farão grande diferença no futuro.

Com essa visão, o Centro Universitário Cesumar assume o compromisso de democratizar o conhecimento por meio de alta tecnologia e contribuir para o futuro dos brasileiros.

No cumprimento de sua missão – "promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária" –, o Centro Universitário Cesumar busca a integração do ensino-pesquisa-extensão com as demandas institucionais e sociais; a realização de uma prática acadêmica que contribua para o desenvolvimento da consciência social e política e, por fim, a democratização do conhecimento acadêmico com a articulação e a integração com a sociedade.

Diante disso, o Centro Universitário Cesumar almeja ser reconhecido como uma instituição universitária de referência regional e nacional pela qualidade e compromisso do corpo docente; aquisição de competências institucionais para o desenvolvimento de linhas de pesquisa; consolidação da extensão universitária; qualidade da oferta dos ensinos presencial e a distância; bem-estar e satisfação da comunidade interna; qualidade da gestão acadêmica e administrativa; compromisso social de inclusão; processos de cooperação e parceria com o mundo do trabalho, como também pelo compromisso e relacionamento permanente com os egressos, incentivando a educação continuada.

Professor
Janes Fidélis Tomelin
Pró-Reitor de
Ensino de EAD

Professora
Kátia Solange Coelho
Diretoria de Graduação
e Pós-graduação

Seja bem-vindo(a), caro(a) acadêmico(a)! Você está iniciando um processo de transformação, pois quando investimos em nossa formação, seja ela pessoal ou profissional, nos transformamos e, consequentemente, transformamos também a sociedade na qual estamos inseridos. De que forma o fazemos? Criando oportunidades e/ou estabelecendo mudanças capazes de alcançar um nível de desenvolvimento compatível com os desafios que surgem no mundo contemporâneo.

O Centro Universitário Cesumar mediante o Núcleo de Educação a Distância, o(a) acompanhará durante todo este processo, pois conforme Freire (1996): "Os homens se educam juntos, na transformação do mundo".

Os materiais produzidos oferecem linguagem dialógica e encontram-se integrados à proposta pedagógica, contribuindo no processo educacional, complementando sua formação profissional, desenvolvendo competências e habilidades, e aplicando conceitos teóricos em situação de realidade, de maneira a inseri-lo no mercado de trabalho. Ou seja, estes materiais têm como principal objetivo "provocar uma aproximação entre você e o conteúdo", desta forma possibilita o desenvolvimento da autonomia em busca dos conhecimentos necessários para a sua formação pessoal e profissional.

Portanto, nossa distância nesse processo de crescimento e construção do conhecimento deve ser apenas geográfica. Utilize os diversos recursos pedagógicos que o Centro Universitário Cesumar lhe possibilita. Ou seja, acesse regularmente o AVA – Ambiente Virtual de Aprendizagem, interaja nos fóruns e enquetes, assista às aulas ao vivo e participe das discussões. Além disso, lembre-se que existe uma equipe de professores e tutores que se encontra disponível para sanar suas dúvidas e auxiliá-lo(a) em seu processo de aprendizagem, possibilitando-lhe trilhar com tranquilidade e segurança sua trajetória acadêmica.

AUTOR

Professor Esp. Ronie Cesar Tokumoto

Pós-Graduado em Docência no Ensino Superior pela Unicesumar, Pós-Graduação em Tutoria em Educação a Distância e Gestão Escolar Integrada e Práticas Pedagógicas pela Faculdade Eficaz. Possui graduação em Bacharelado em Informática pela Universidade Federal do Paraná (2001). Experiência na área de educação desde 1994 em escolas profissionalizantes e colégios estaduais. Experiência em Coordenação de Curso Técnico no ano de 2013 e Participação em Palestras e Eventos Organizados para o Curso durante o ano de 2013. Atualmente, atua como Tutor Mediador no EAD-Unicesumar e como Professor na rede Estadual de Ensino do Paraná em Sarandi-PR, em Cursos Técnicos Profissionalizantes.

Para informações mais detalhadas sobre sua atuação profissional, pesquisas e publicações, acesse o currículo, disponível em: <http://lattes.cnpq.br/2653232632701400>.

PROGRAMAÇÃO DE SISTEMAS II

SEJA BEM-VINDO(A)!

Seja bem-vindo(a), caro(a) aluno(a), nesta segunda parte da disciplina. Conceitos complementares aos da primeira parte serão abordados de forma a se fundir durante a apresentação das aulas ao vivo para, assim, moldar uma única linha de raciocínio, servindo de base para o estudo da programação em linguagem Java.

Essa linguagem, diferentemente da linguagem C tratada em outras disciplinas do curso, segue um outro paradigma de programação, chamado de orientação a objetos, que tem seus conceitos trabalhados graficamente em outras disciplinas do curso.

Esse paradigma ainda é aceito por versões mais recentes da linguagem C, como C++, C#, Visual C e Objective C, mas estas não são foco deste material, servindo apenas de exemplo para comparação.

A linguagem Java é puramente orientada a objetos, diferentemente da linguagem C++, que aceita tanto a programação orientada a objetos quanto a baseada no paradigma estruturado em que não se usam classes e métodos, mas sim registros e funções. Além disso, é uma linguagem muito popular na comunidade mundial de programadores, tendo, inclusive, inúmeras fontes de aquisição de conhecimentos, troca de experiências e aquisição de códigos prontos.

Nesta segunda parte da disciplina de Programação de sistemas, os conceitos do paradigma orientado a objetos são tratados por meio de seus conceitos mais importantes e exemplos de aplicação, que podem ser livremente adaptados para o aprendizado da linguagem Java e criação de aplicações.

A melhor forma de se aprender a programar em Java ou qualquer outra linguagem é por meio da prática de criação de códigos em quantidade com as mais diversas finalidades, permitindo que, a cada nova codificação, problemas recorrentes sejam solucionados, e a prática leve a uma experiência que permita a criação de aplicações em menor tempo e com melhor qualidade.

Outra prática fundamental da programação feita em quantidade é a percepção de códigos, que acabam tendo trechos repetitivos e que possam ser reaproveitados para aplicações futuras, gerando um recurso muito importante chamado reuso que, por sua vez, acaba fornecendo material para a criação de bibliotecas própria recurso muito importante no dia a dia de profissionais de desenvolvimento.

O mais importante no estudo da programação é compreender, primeiramente, os conceitos e, a medida em que ocorrem os avanços, desenvolver códigos baseados nos exemplos do livro, buscando fontes complementares para aperfeiçoar seus conhecimentos.

Bons estudos!

SUMÁRIO

UNIDADE I

ORIENTAÇÃO A OBJETOS - PARTE I

UNIDADE II

ORIENTAÇÃO A OBJETOS - PARTE II

UNIDADE V

TÓPICOS ADICIONAIS

Professor Esp. Ronie Cesar Tokumoto

UNIDADE I

ORIENTAÇÃO A OBJETOS - PARTE I

Objetivos de Aprendizagem

- Conhecer os conceitos de base do paradigma orientado a objetos em Java.
- Compreender os conceitos de atributos.
- Compreender os conceitos de visibilidade.
- Saber diferenciar classes, métodos e atributos.
- Compreender como se comportam objetos.
- Compreender como instanciar e usar objetos.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Programação orientada a objetos em Java
- Atributos
- Visibilidade
- Métodos
- Classes
- Objetos

UniCesumar

INTRODUÇÃO

Iniciando os estudos desta disciplina, temos, nesta primeira Unidade, a apresentação dos conceitos fundamentais do paradigma orientado a objetos voltado à programação em linguagem Java.

As diferenças de programação nas diferentes formas de programação são estudadas na disciplina de Paradigmas de Linguagens de Programação, na qual paradigmas como o Estruturado e o Orientado a Objetos são vistos.

Aqui, o objetivo é conhecer apenas os aspectos voltados ao paradigma orientado a objetos, em que termos, como classe e objeto são comuns e, juntamente com outros conceitos, formam um dos melhores paradigmas já criados.

Assim, esta unidade se inicia com uma ideia do paradigma e suas características básicas, para depois adentrar nos conceitos de atributo, método e classe tão fundamentais na programação orientada a objetos.

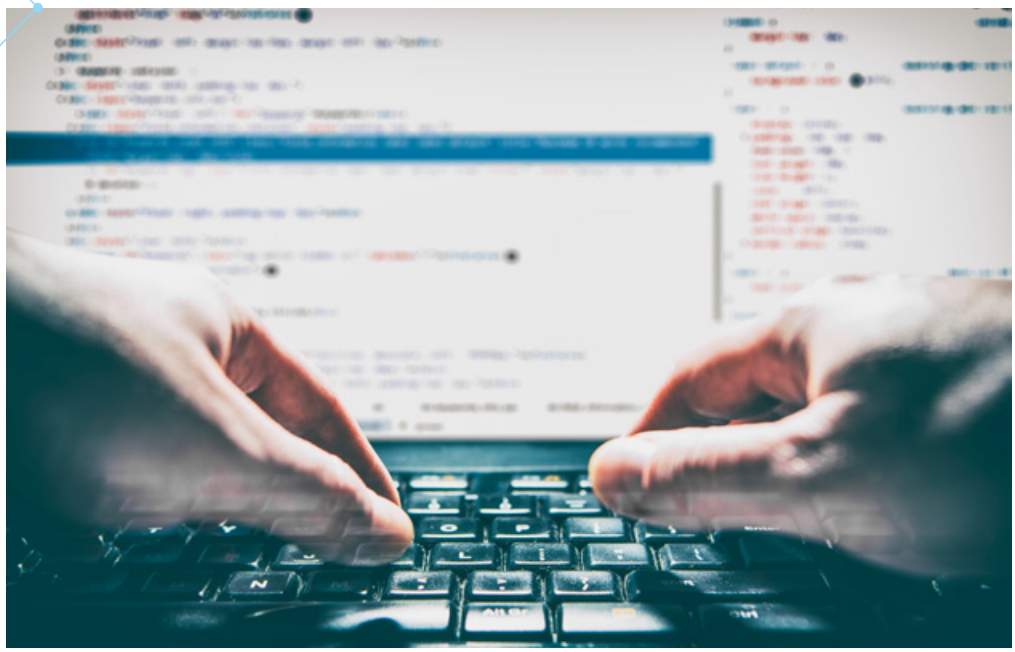
Esses três conceitos são elementares e devem ser estudados até que estejam claros na mente do estudante, pois, sem eles, a continuidade dos estudos fica comprometida e pode impedir a compreensão da ideia e dos demais conceitos do paradigma.

Juntamente com esses conceitos básicos, é abordada a chamada visibilidade que, por sua vez, representa outro pilar da orientação a objetos, criando mecanismos que permitem a escolha daquilo que cada parte de um código tem acesso durante a execução.

Finalizando a unidade, é conceituada a ideia de objeto, fechando a parte de conceitos fundamentais da orientação a objetos, mostrando como uma estrutura genérica, chamada classe, é utilizada como "molde" para a quantidade desejada de replicações desta para uso durante a execução da aplicação.

Na segunda Unidade, os conceitos serão aprofundados para complementar o conhecimento sobre o paradigma da orientação a objetos, permitindo a correta aprendizagem da linguagem Java e como embutir, de forma adequada, esses conceitos em qualquer projeto de aplicação.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA

Basicamente, a programação Java segue o paradigma orientado a objetos e distribui as funções que um programa deve executar, por meio das chamadas classes, contendo atributos e métodos que são componentes essenciais dentro desta forma de programação.

Não é possível programar seguindo o paradigma estruturado, por exemplo, pois a base dos programas em Java são as Classes com todas as suas características e, além disso, não são aceitos ponteiros ou funções utilizados em outras linguagens de programação, como na linguagem C (MANZANO, 2011).

Conhecer as diferenças entre paradigmas de programação existentes pode auxiliar no aprendizado e correta aplicação de uma linguagem em cada tipo de projeto a ser realizado.

Uma linguagem orientada a objetos pode representar uma evolução na reutilização de código em relação à programação estruturada, mas, ao mesmo tempo,

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

pode gerar conflitos na lógica e na forma como se pensa e projeta um software, devido às suas características próprias.

É preciso mudar a forma de pensar durante o desenvolvimento de códigos em linguagens baseadas neste tipo de paradigma. Além do mais, muitos programadores ou até empresas de desenvolvimento de software nem cogitam o uso desse paradigma, justamente pelo tempo de aprendizado em função dos benefícios imediatos possíveis na visão dessas empresas (MANZANO, 2011).

Aqueles que iniciam seu ingresso no universo da programação por meio de linguagens orientadas a objeto não sentem dificuldades, pois já aprendem os fundamentos da programação moldando sua lógica dentro desse paradigma, sem a necessidade de adaptação.

Algumas características da programação orientada a objetos podem ser complexas e causar dúvidas durante o aprendizado, mas nem todas as técnicas são obrigatórias, pois apenas o essencial pode ser suficiente para uma codificação capaz de criar a solução esperada.

Toda programação orientada a objetos na linguagem Java, basicamente, se constitui nos conceitos do uso de classes e seus membros (atributos e métodos), mantendo uma base sólida na linguagem C++, tornando-as muito semelhantes e facilitando a migração de estudos, caso se conheça alguma das duas.

Resumidamente, uma classe é um modelo chamado de abstração que, por sua vez, traz as principais características de um elemento da aplicação que possa ser tratado como tendo características próprias e que possa ser independente do restante do código, exceto por dados que precise receber para realizar suas funções por meio das chamadas mensagens que representam os parâmetros (DEITEL, 2010).

Toda classe pode possuir métodos que representam ações que podem manipular dados nessas classes, e atributos que representam os dados que ela pode conter.

A seguir, os principais conceitos vão sendo colocados para que se possa ter um conhecimento mais concreto dos fundamentos desse paradigma e como o mesmo se aplica na linguagem Java.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATRIBUTOS

Os atributos são essenciais na programação orientada a objetos, pois eles são responsáveis por armazenar os dados que devem ser processados em cada método, permitindo que cada classe possa conter um conjunto de dados que a represente e identifique dentro das funções de um programa.

Sem atributos, um método se torna restrito a poucas ações possíveis, limitando-se a utilizar dados recebidos por parâmetros. Ademais, uma classe sem atributos não possui características definidas para conter dados que possam ser processados pelos seus métodos.

Em geral, os atributos funcionam como variáveis comuns da programação do paradigma estruturado e sua declaração segue o modelo em que se indica um tipo de dado aceito pelo atributo, como **int** ou **float**, por exemplo, seguido

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

pelo nome dado a esse atributo (com todas as letras minúsculas por convenção) e, opcionalmente, um valor pode ser indicado para a sua inicialização. Usa-se o símbolo de ponto e vírgula para indicar o fim da declaração do atributo.

//código1

```
Sintaxe:
    tipo nomeAtributo <, outroAtributo> < = > < valor > ;
Exemplo:
    char letra;
    int idade = 0;
    double cpf, rg;
```

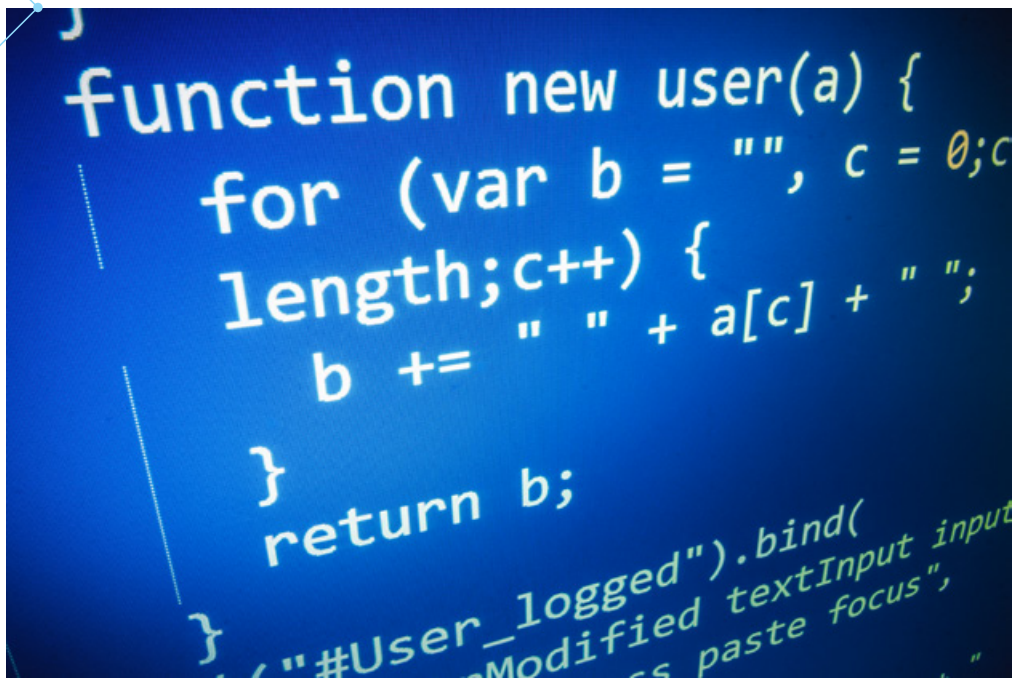
Nos exemplos do código1, temos a declaração de atributos que reservam espaços na memória, de acordo com a quantidade de bytes necessários para armazenar qualquer um dos valores admitidos dentro do intervalo aceito pelos tipos indicados (**char, int, double**).

No primeiro exemplo, temos a declaração de uma variável do tipo caractere, capaz de armazenar qualquer caractere aceito pela linguagem, lembrando que esse tipo de dado se refere à apenas um único caractere por variável. Caso se deseje armazenar uma quantidade maior, é preciso outro tipo de dado.

No segundo exemplo, além da declaração da variável idade, na mesma instrução é atribuído um valor inicial para essa variável, para que possa ser garantida a correta inicialização da mesma, reduzindo a chance de problemas em tempo de execução que, por sua vez, podem ser ocasionados por valores indesejados alocados anteriormente no espaço de memória ocupado pela variável.

Finalmente, na terceira declaração, são criadas, numa mesma instrução, duas variáveis com um mesmo tipo de dado, economizando linhas e código, simplificando sua leitura.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



VISIBILIDADE

Quando são desenvolvidos projetos dentro do paradigma orientado a objetos, uma das características fortes se chama encapsulamento, assunto que será tratado na unidade seguinte.

O encapsulamento se refere à possibilidade de acesso a dados resumindo a ideia, sendo necessário algum mecanismo para garantir esse recurso. Para associar o tipo de visibilidade a atributos, métodos e classes, utilizamos as palavras reservadas **public, private e protected**.

Cada termo representa um diferente tipo de visibilidade para elementos da orientação a objetos, seguindo a lógica de que cada nível oferece diferentes níveis de acesso de um elemento dentro do código da aplicação.

A visibilidade tem relação com a possibilidade de um elemento de um código Java como um método ter acesso aos dados de atributos de outro método, sendo

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

o método um elemento que pode conter vários atributos, como será visto logo a seguir na unidade.

Quando se usa a palavra reservada **public**, supõe-se que a vontade do programador é de permitir que todos os demais elementos do código acessem esse elemento (MANZANO, 2011).

Desse modo, uma classe declarada como public pode ser acessada por outras classes em um projeto, tornando sua visibilidade a mais ampla possível pela chamada visibilidade pública.

Já a opção de visibilidade **private** torna um elemento acessível apenas dentro do bloco ao qual esteja associado, impedindo que quaisquer outros elementos do código possam ter acesso direto ao mesmo (MANZANO, 2011). Essa opção é usada, normalmente, em atributos e é a declaração padrão de visibilidade para atributos em linguagem Java.

Por fim, a opção **protected** é intermediária às duas anteriores e representa um tipo especial de visibilidade associada à outra característica muito forte da orientação a objetos chamada herança, que também será tratada durante o avanço dos estudos (MANZANO, 2011).

Como nem todos os conceitos foram tratados ainda, alguns pontos podem estar confusos, mas aos poucos, com a evolução nos estudos, existe a tendência de se compreender melhor as ideias apresentadas e todos os assuntos começarão a se relacionar e a formar uma base de conhecimento (MANZANO, 2011).

Para ilustrar esses conceitos de visibilidade, os exemplos apresentados nos Códigos 2, 3 e 4 serão muito úteis, pois mostram situações um que todos os tipos de visibilidade são associados a elementos da programação Java. Dessa forma, será mais fácil compreender a forma como se declara visibilidade junto com o tipo de cada atributo e método, ou em classes, garantindo os conceitos de encapsulamento e herança.

Uma dica levada em consideração na programação Java por muitos desenvolvedores é a padronização de atributos como private, métodos como protected ou private, e classes como public.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

MÉTODOS

Tendo uma sintaxe semelhante à declaração de uma classe, no caso dos métodos, assim como nas classes, pode-se indicar a visibilidade do mesmo para o restante do código por meio de termos, como **public** ou **private**, e um nome para o método (com todas as letras em minúsculo por convenção).

Podemos, ainda, indicar um tipo de retorno para essa classe por meio de termos como **int** ou **void**, permitindo que esse método tenha a possibilidade de retornar algum valor para o trecho de código que o chamou, e seus parâmetros de entrada com seus tipos e nomes para os mesmos, definidos entre parênteses, sendo que não são obrigatórios.

//código2

```
Sintaxe:
  visibilidade tipo nomeMetodo (<tipo parametro, tipo parametro2>)
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
Exemplo:
    public static void main (String [] args) {
            protected int nomeVariavel;
                <instruções;>
    }
```

No exemplo do código2, **main** é declarado com visibilidade pública para todo o código, sem tipo de retorno, podendo receber parâmetros na chamada do software em quantidade variada, separados por espaços em branco. Essa característica é própria desse método em particular, pois sendo o primeiro método a ser chamado durante a execução do software, ele tem sua declaração de parâmetros com uma sintaxe diferenciada, se desejado.

Poderia ser feita uma declaração comum, como nos demais métodos, seguindo o exemplo do código3, que mostra a declaração de parâmetros derivado a mesma sintaxe da declaração e atributos.

//código3

```
public int exemploMetodo (int X, float Y) {
        protected int variavel;
          <instruções;>
}
```

No exemplo anterior, temos dois parâmetros, X e Y, declarados com seus respectivos tipos indicados, podendo ser manipulados livremente dentro do método por serem considerados como variáveis locais do método.

Os métodos são os responsáveis diretos pelo processamento em orientação a objetos; e as classes, normalmente, possuem pelo menos um método para realizar a manipulação dos processos referentes à classe.

Existe um conceito importante a respeito do uso de método, que envolve a forma como se podem utilizar seus parâmetros, chamado de sobrecarga.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

A sobrecarga se refere à possibilidade de um método, que possua parâmetros de entrada de dados, ter variações em seu funcionamento, devido às diferentes possibilidades geradas pelo uso de diferentes parâmetros que, por conseguinte, possam ser passados ao ser chamado durante a execução.

Isto permite que, em uma mesma classe, mais de um método possua o mesmo nome e que cada um deles seja chamado pelo interpretador automaticamente, devido a esses diferentes parâmetros passados pela aplicação.

Geralmente, esses diferentes métodos com nomes iguais são criados para propósitos iguais ou semelhantes, mas que possam processar dados recebidos que sejam de tipos diferentes, evitando erros de execução pelo fato de a linguagem Java ser fortemente tipada.

O código4 traz um exemplo de código contendo uma classe com dois métodos de mesmo nome, mas com variações de parâmetros, ilustrando como se poderia utilizar esta técnica, lembrando que os métodos funcionariam perfeitamente se tivessem nomes diferentes:

//código4

```
public class Exemplo
{
   public void exemploMetodo(int numero) {
          System.out.print ("Imóvel número "+ numero);
   }
   public void método(int numero, int complemento) {
          System.out.print ("Imóvel número "+ numero+" - "+
                              complemento);
   }
}
```

Neste exemplo, é visível que um endereço contém número e pode, ou não, conter complemento em um endereço ta como apartamento, mas no caso de casas

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

que dividem um mesmo terreno, podem ser utilizadas letras como complemento, ou bloco em condomínios de prédios.

Nesses casos, outras cópias do método poderiam ser geradas para contemplar essas variações, como é possível observar no exemplo do código5 e que poderia ser acrescentado ao código4 do exemplo.

//código5

```
public void método(int numero, int complemento, char casa) {
        System.out.print ("Imóvel número "+ numero+ " - "+
                          complemento+ " - casa "+ casa);
}
public void método(int numero, int complemento, char casa,
                   char bloco) {
      System.out.print ("Imóvel número "+ numero+ " - "+
      complemento+ " - ap. "+ apartamento"+ " - bloco "+  bloco);
}
```

Por fim, tipos de retorno para métodos não podem servir como único elemento de diferenciação entre métodos de mesmo nome para gerar sobrecarga, pois tipos de retorno são ignorados pelo compilador.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

REFLITA

Observando os conceitos fundamentais da orientação a objetos, é perceptível a semelhança com a programação estruturada, mas, aos poucos, é possível perceber as grandes diferenças.

CLASSES

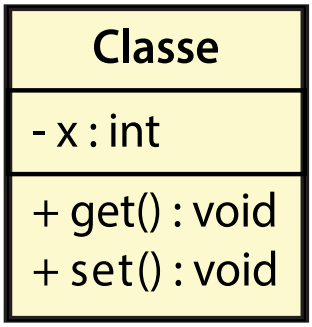


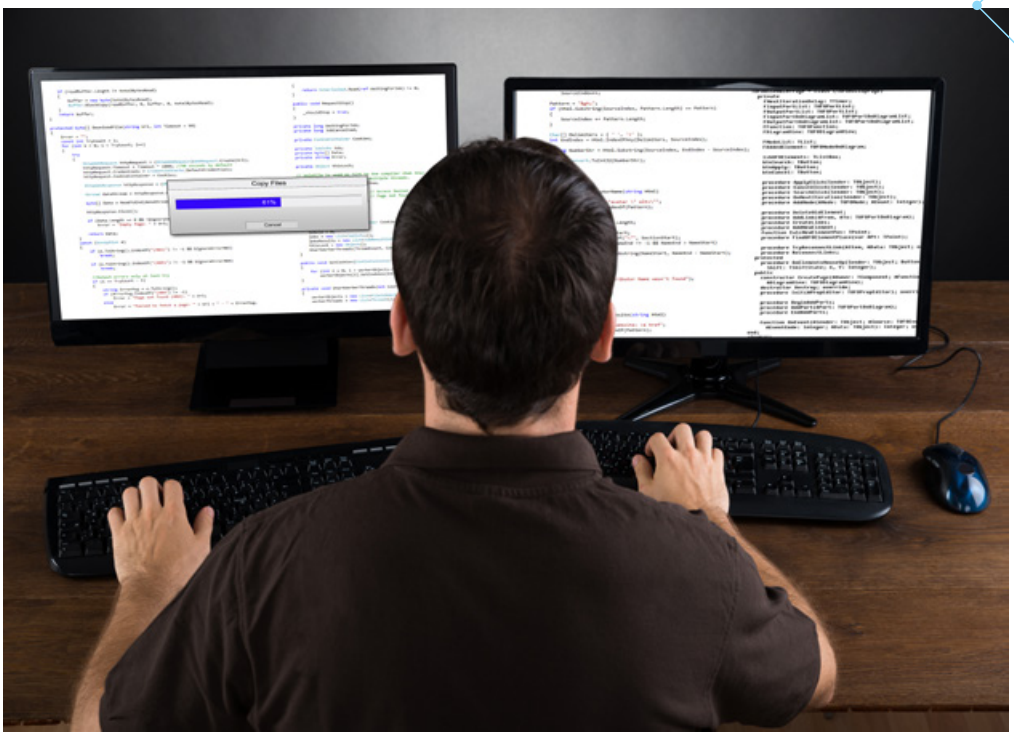
Figura 1 - Diagrama de classes para representar Classe
Fonte: o autor.

Classes são essenciais para a programação em Java devido ao seu paradigma de programação; por esse motivo, todo programa em Java possui, pelo menos, uma classe. A Figura 1 mostra um exemplo de classe representada por um dos elementos utilizados no chamado diagrama de classes da linguagem UML.

Nessa representação da Figura 1, temos indicados todos os elementos citados até o momento, como o nome da classe na parte superior; seu atributo **x** do tipo inteiro, indicado pela palavra reservada **int**; com visibilidade privada, indicada pelo sinal - na parte central; e seus métodos **get()** e **set()** sem tipos de retorno em função da palavra reservada **void**, com visibilidade pública pelos sinais de + na parte inferior da ilustração.

A declaração de uma classe em Java consiste na indicação de sua visibilidade por meio de **public,** como exemplo, a palavra reservada **class** e o nome da classe (geralmente iniciando por letra maiúscula por convenção).

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



//código6

Sintaxe:

```
visibilidade tipo nomeDaClasse
```

Exemplo:

```
public class ExemploClasse {
        public float exemploVariavel1;
        public int exemplo Metodo () {
                private int exemploVariavel2;
                <instruções>
        }
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

No exemplo do código6, a declaração mostra a criação de uma classe genérica de nome CLASSE, com visibilidade pública e com métodos também públicos pertencentes a ela.

É importante observar que existe a declaração de um atributo chamado ATRIBUTO2 de visibilidade privada, que pertence a MÉTODO, declarado dentro de CLASSE, representando uma das características da orientação a objeto chamada encapsulamento.

Outro atributo, chamado ATRIBUTO de visibilidade pública, pertence à CLASSE e, além de poder ser visualizado por todos os métodos de CLASSE, pode, também, ser acessado por outras classes e seus métodos.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

SAIBA MAIS

Classes, métodos e atributos são elementos da orientação a objetos que possuem nomenclatura para identificação e, para evitar confusões durante a programação, é interessante estabelecer ou seguir uma das regras existentes para nomenclatura desses nomes para uma associação mais rápida.

Uma forma seria iniciar nomes de classes com letra maiúscula e as demais minúsculas. No caso de métodos e atributos, pode-se escrever o nome de ambos com minúsculas, pois métodos sempre têm parênteses depois do nome, distinguindo os dois tipos. Assim, exemplos poderiam ser **Classe**, **método()** e **atributo**.

Complementando, em casos de nomes compostos, é interessante iniciar as partes posteriores do nome com letras maiúsculas para facilitar a leitura e interpretação do nome do elemento. Por exemplo, um atributo poderia ter um nome composto como **nomeCompleto** e uma classe poderia se chamar **ImovelNovo**.

Fonte: o autor.

OBJETOS

Uma classe representa uma estrutura criada para representar uma abstração de algo dentro do propósito da aplicação, servindo apenas como referência para que o código possa replicar a estrutura da classe em memória, para cada vez que seja necessário utilizar a estrutura da classe para receber um conjunto de dados ou realizar o processamento de algum método da classe.

Assim, é possível associar a ideia de classe e objeto com analogias do mundo real, como a ideia de imaginar uma classe referente a carro. Objetos seriam as ocorrências de carros diversos que poderiam acontecer em uma revenda, por exemplo.

Dessa forma, a classe representaria todas as características (atributos) e ações (métodos) possíveis de serem realizadas com esses carros, como atributos modelo, marca, cor, ano etc. Os métodos associados poderiam conter venda, compra, reparo etc.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Na chamada instanciação de um objeto, o processo é semelhante à declaração de uma variável, em que a classe funciona como tipo de dado mais complexo ao qual uma variável é associada, representando o objeto.

Na prática, para a classe carro, o cadastro de um Fiat Uno Branco 2010 seria incluído em um objeto instanciado da classe carro, da mesma forma que outros objetos seriam instanciados para armazenar dados de outros carros.

No código7, é possível compreender como funciona o processo de instanciação de uma classe, na qual a palavra reservada new é fundamental por ser a responsável por ativar o objeto para uso durante a execução da aplicação.

//código7

```
public class CLASSE {
        public float ATRIBUTO;
        public int METODO () {
                private int VARIAVEL;
                <instruções>
        }
}
        CLASSE objClasse = new CLASSE();
```

No exemplo, é possível perceber que uma classe de nome CLASSE, contendo atributo e método, é usada para a declaração de um objeto chamado objClasse, e esse objeto é instanciado na mesma instrução por meio da palavra reservada new.

Outra forma de se declarar e instanciar o mesmo objeto pode ser visto na variação do código7 no código8. Nele, é possível perceber que a definição e instância do objeto são feitos em linhas diferentes.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código8

```
public class CLASSE {
        public float ATRIBUTO;
        public int METODO(){
                private int VARIAVAEL;
                <instruções>
        }
}
        CLASSE objClasse;
        objClasse = new CLASSE();
```

Nesse exemplo do código8, é possível esclarecer o quanto é importante o processo de instanciar uma classe para criar um novo objeto.

A linha **CLASSE objClasse;** apenas declara um novo objeto de nome objClasse, mas ainda o deixa inativo, sendo impossível adicionar dados à sua estrutura enquanto a linha **objClasse = new CLASSE();** não for adicionada ao código para ativar o objeto.

Com isso em mente, é possível compreender a importância da palavra reservada new e sua função dentro do paradigma orientado a objetos, permitindo a adição de dados em objetos criados como se fosse parte de uma lista ou pilha de estruturas de dados.

Pode ocorrer de um objeto instanciado possuir métodos que recebam parâmetros de nome igual ao de atributos da classe ao qual pertencem. Isso não parece ser uma boa prática de programação, mas é totalmente aceitável pelo compilador.

Desta forma, teríamos uma situação semelhante à indicada no trecho de código do código9, em que tanto o atributo da classe quanto o parâmetro do método possuem o mesmo nome:

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
//código9
public class CLASSE{
    public float VALOR;
    public int METODO (float VALOR){
            private int VARIAVEL;
            VARIAVEL = VALOR;
            VARIAVEL = this.VALOR;
    }
}
```

Nesse exemplo do código9, a solução para a dúvida que pode ocorrer em relação a qual valor de VALOR (atributo ou parâmetro) está sendo acessado, a palavra reservada this representa uma espécie de coringa.

This é um tipo especial de objeto que significa que, dentro da execução do método **METODO** do exemplo, deve ser utilizado o atributo da classe e não o parâmetro do método, que por padrão teria prioridade por estar, de certa forma, no mesmo bloco de código.

Já na instrução sem a palavra this, é utilizado o VALOR relacionado ao parâmetro do método como padrão, em função da instrução estar dentro do próprio bloco de instruções do método e ter prioridade no uso.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

CONSIDERAÇÕES FINAIS

Nesta primeira Unidade do livro, foram tratados assuntos relativos aos fundamentos do paradigma da orientação a objetos, base para a programação de aplicações em linguagem Java.

Após a introdução de algumas ideias sobre o tema, foram vistos alguns dos conceitos fundamentais que o aluno deve estudar exaustivamente por meio deste livro, com o auxílio de outras fontes, como as sugeridas ao longo das unidades.

A unidade teve foco no aprendizado dos conceitos de classe, método e atributo, além de outros pontos relacionados e fundamentais ao aprendizado da programação orientada a objetos.

Classes representam modelos de estruturas que servirão para atender demandas do projeto por meio de chamadas abstrações, as quais serão tratadas durante a disciplina. Contém elementos para estruturar componentes da aplicação chamados de métodos, podendo armazenar dados nos chamados atributos.

Esses métodos são similares às funções da programação estruturada, mas possuem características diferentes, como a forma com que podem ser acessados e sua sintaxe e semântica.

Já os atributos representam estruturas de dados simples, com tipos definidos em função da característica de linguagem fortemente tipada, funcionando como variáveis, respeitando as características da orientação a objetos, como visibilidade, por exemplo.

Sobre a visibilidade, outro ponto desta unidade, foi complementado os três conceitos anteriores com a ideia de um destes elementos poderem, ou não, ser acessados por outros dentro da aplicação durante a execução.

Finalizando os conceitos da unidade, a ideia de objeto foi colocada para mostrar como são aplicadas as classes na prática, por meio de instâncias chamadas objetos, responsáveis por realizar o processamento dos métodos e armazenamento de dados em atributos.

Na próxima unidade, é dada continuidade no estudo do paradigma com conceitos que permitem completar o aprendizado da orientação a objetos.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATIVIDADES

1. A orientação a objetos permite o desenvolvimento de projetos baseados em um paradigma de programação bastante complexo, mas com várias vantagens sobre o paradigma tradicional estruturado. **Quais os elementos básicos desse paradigma?**

2. Dentro da programação em linguagem Java, existem parâmetros importantes. **Quais os tipos de visibilidade existentes em linguagem Java?**

3. Em métodos que utilizam parâmetros para entrada de dados na sua chamada, é possível criar mecanismos para controlar o funcionamento do método de acordo com a quantidade de parâmetros que forem passados ao método durante a execução. **Como se chama essa funcionalidade?**

4. Uma das dúvidas comuns do paradigma orientado a objetos é a diferença entre classe e objeto no momento de se compreender a linguagem e a técnica da orientação a objetos. **Comente qual seria essa diferença fundamental.**

5. Seguindo a ideia da questão anterior, existe uma palavra reservada própria da linguagem Java utilizada para instanciar objetos a partir de classes. **Qual é essa palavra?**

LEITURA COMPLEMENTAR

A linguagem de representação UML é capaz de ilustrar, por meio de diagramas diversos, a estrutura e os processos de um sistema de qualquer complexidade, permitindo a pessoas leigas uma maior compreensão da extensão de um sistema.

Diagramas como os diagramas de Casos de Uso são ótimos para mostrar a clientes e usuários quem são os chamados atores que interagem com o sistema e quais são as ações atribuíveis de acordo com seus papéis no funcionamento do mesmo.

Diagramas de Classe são mais complexos e indicados para uso interno na desenvolvedora para servir de base para o trabalho de análise e projeto do sistema, além de servir como documentação do sistema após a conclusão do projeto.

Essa documentação é composta por diversos elementos e todos os diagramas UML podem ajudar a compor essa documentação, servindo tanto para compreensão da estrutura do sistema e atendimento ao cliente quanto para manutenções e evoluções que possam ser necessárias durante o ciclo de vida do sistema.

Mesmo conceitos mais complexos da orientação a objetos, como os presentes na Unidade II, são contemplados por esses diagramas UML, como o caso da herança e encapsulamento.

Existem diversos outros diagramas com funções complementares da linguagem UML, como o diagrama de sequência, que verifica a comunicação entre objetos, detalhando como os métodos são utilizados dentro das classes.

Outro diagrama importante é o diagrama de estados, que permite mapear os possíveis estados que um objeto pode ter, no caso de ser limitada a quantidade desses estados.

O chamado diagrama de comunicação também é interessante, pois complementa o diagrama de sequência, ordenando os estados sem se basear no tempo, mas sim na sequência em que ocorrem.

Sendo chamada de linguagem, a UML muitas vezes é confundida com linguagem de programação capaz de gerar aplicações, mas, na realidade, é uma forma de ilustrar todos os aspectos de um sistema na etapa de análise, facilitando a passagem para as etapas seguintes de um projeto.

Fonte: o autor.

MATERIAL COMPLEMENTAR

LIVRO

Java - A Referência Completa
Autor: Herbert Schildt
Editora: Alta Books
Sinopse: atualizado para a versão sete da linguagem Java, o livro traz, de forma bem organizada, todos os princípios fundamentais da linguagem, além de conceitos adicionais importantes.

REFERÊNCIAS

DEITEL, P. J. **Java:** como programar. 8. Ed. São Paulo: Pearson Prentice Hall, 2010.

MANZANO, J. A. N. G. **Java 7:** programação de computadores: guia prático de introdução, orientação e desenvolvimento. São Paulo: Érica, 2011.

GABARITO

1) Classe, Método e Atributo.

2) Público, Protegido e Privado.

3) Sobrecarga de métodos.

4) Uma classe representa um modelo baseado na abstração de um componente do projeto e a partir desta classe contendo métodos e atributos próprios. Além disso, são instanciados os chamados objetos que seriam ocorrências destas classes para uso na aplicação. Com isto, temos que a classe é única e capaz de gerar a quantidade de objetos que desejarmos para o processamento de dados dentro da aplicação.

5) New.

Professor Esp. Ronie Cesar Tokumoto

ORIENTAÇÃO A OBJETOS - PARTE II

UNIDADE II

Objetivos de Aprendizagem

- Conhecer os fundamentos complementares do paradigma orientado a objetos.
- Compreender os conceitos de herança.
- Compreender os conceitos de encapsulamento.
- Conhecer métodos estáticos e o funcionamento do polimorfismo.
- Saber como estruturar classes para abstrair problemas do mundo real.
- Conhecer técnicas para a elaboração de classes e métodos abstratos.
- Conhecer algumas dicas de como estruturar classes.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Herança
- Encapsulamento
- Métodos estáticos
- Polimorfismo
- Classes abstratas
- Métodos abstratos
- Dicas para elaboração de classes

INTRODUÇÃO

Nesta segunda Unidade, segue a contextualização dos conceitos do paradigma orientado a objetos iniciados na primeira Unidade, na qual foram introduzidos os conceitos de classe, método e atributo.

Tendo sido tratados conceitos como declaração de classes, métodos e atributos, também foram tratados conceitos de visibilidade e sobrecarga, complementando a unidade.

Nesta segunda Unidade, veremos conceitos mais aprofundados do paradigma relacionados ao comportamento desses elementos, aumentando a eficiência do modelo e suas aplicações. Um desses conceitos é a herança, que permite que classes tenham associações entre si em forma de dependência, criando uma relação que aceita que classes possam acessar atributos e métodos umas das outras.

Outro conceito muito importante se chama encapsulamento, que oferece um mecanismo de proteção a dados e processos baseado no conceito da visibilidade, visto na unidade anterior.

Um terceiro conceito que compõe a base desta unidade é o polimorfismo que, por sua vez, oferece a possibilidade de um método ser reescrito para adaptar-se a necessidades variadas dentro de uma aplicação.

Complementando os assuntos da unidade, serão abordados os conceitos de abstração e do uso de métodos estáticos, ampliando a gama de funcionalidades relacionadas a classes e métodos em orientação a objetos.

Na próxima unidade, estudaremos os conceitos relacionados à programação em Java, para compreensão dos chamados controles de execução do software, como controles de fluxo e laços de repetição, assuntos fundamentais para a compreensão da linguagem.

Esses conceitos complementam outros relacionados à programação na disciplina de Programação de Sistemas I, em relação à programação em linguagem Java, fazendo parte do conjunto básico que deve ser estudado para se programar na linguagem.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

HERANÇA

Um dos pilares da orientação a objetos é chamado de herança, na qual existe uma forte ligação entre classes que podem compartilhar elementos, mantendo uma relação de dependência.

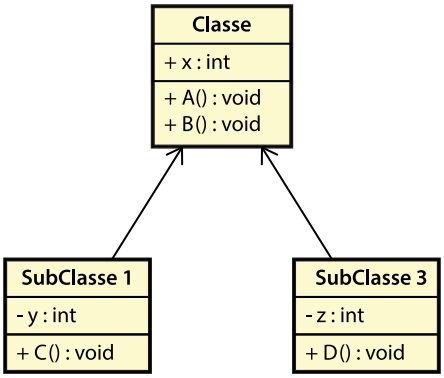


Figura 1 - Diagrama de classes para representar Herança
Fonte: o autor.

A possibilidade de termos uma classe genérica é muito importante na reutilização de um código, mas, em muitos casos, é necessário que uma classe assim tenha especializações, ou seja, possa ter classes que mantenham as características da classe genérica e que possam agregar mais detalhamento.

Continuando com o exemplo de uma classe de tratamento de dados pessoais, poderia haver uma classe genérica para os dados, mas poderiam ser criadas outras classes que, a partir de dados da classe geral, poderiam realizar ações complementares, como no caso dos dados de um cadastro se referir à pessoa física ou jurídica sofrer um diferente tratamento em seus dados.

Desse modo, surge a ideia de classes e superclasses, em que uma superclasse pode estar servindo de base para outras classes que usufruem de seus métodos e atributos, da mesma forma que uma classe também pode usufruir de características de mais de uma superclasse.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

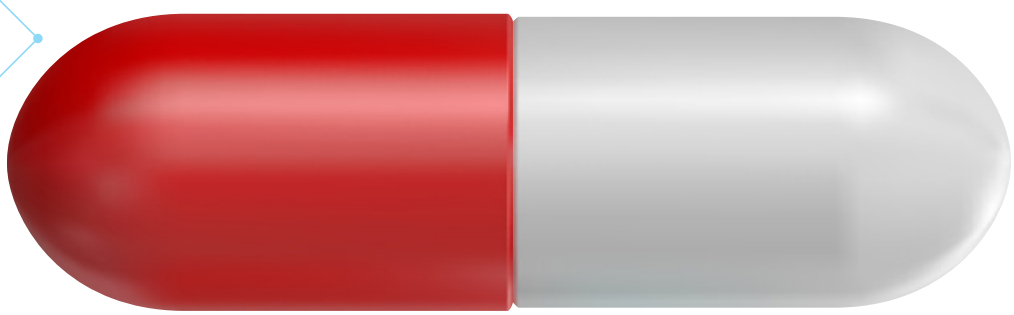
Dessa forma, podemos imaginar uma situação em que uma classe de pessoa física se vale do atributo nome de uma superclasse pessoa, assim como outra classe pessoa jurídica também podem se utilizar do mesmo atributo, mas dentro de seus processos de forma diferente da classe pessoa física.

Outro caso seria de uma classe pessoa jurídica estar ligada à superclasse pessoa para usufruir daquilo que lhe seja conveniente, assim como de outra superclasse compras, que pode, também, permitir a distinção entre pessoa jurídica e física, possuindo tratamento diferenciado de dados em cada caso.

//código1

```
public class Classe{
        public tipo atributo <, atributo>;
        protected tipo metodo (<tipo atributo, tipo atributo>){
                visibilidade tipo variavel <, variavel>;
                <instruções>
        }
}
public class Classe2 extends Classe{
        public tipo atributo <, atributo>;
        private tipo metodo2 (<tipo atributo, tipo atributo>){
                visibilidade tipo variavel <, variavel>;
                <instruções>
                Classe.atributo = 0;
                }
        }
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



ENCAPSULAMENTO

É uma forte característica da programação orientada a objeto, permitindo que o acesso a um método e tudo que envolva seu funcionamento seja isolado do restante do código. Esse isolamento inclui seus atributos e a lógica envolvendo os processos executados pelo método (MANZANO, 2011).

Essa forma de isolamento permite que outros métodos ou classes possam acessar esse método encapsulado por meio de chamadas, ao mesmo tempo em que podem permitir, ou não, a comunicação por meio de parâmetros de entrada ou saída, caracterizando a troca de mensagens característica desse paradigma de programação.

Além do isolamento, o encapsulamento é um dos principais fatores que contribuem para facilitar o reaproveitamento de código, em que é possível que um método ou toda uma classe com seus componentes sejam reaproveitados em outros programas, por meio da passagem de mensagens e retorno de valores.

Encapsular, na programação orientada a objeto, tem a ver com a visibilidade que componentes de uma classe possuem em relação a outras classes do código, ou seja, podemos usar atributos de visibilidade na declaração desses componentes para torná-los encapsulados ou não (DEITEL, 2010).

Podemos citar três tipos de visibilidade para indicar a existência ou não do encapsulamento em um código, podendo ser do tipo público, privado ou protegido.

Por meio dessas opções, podemos implementar, ou não, a ideia do encapsulamento em classes e seus componentes, de acordo com as necessidades do projeto do software em desenvolvimento.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Por meio do diagrama de classes da figura, é possível visualizar se o encapsulamento é utilizado em uma classe, servindo de base para a criação do código que contemple este projeto.

Quando a visibilidade é indicada como pública, qualquer componente é acessível por qualquer parte do código, e seus valores podem ser manipulados, durante a execução, a qualquer momento.

É a opção mais indicada para classes, pois necessitam ser visíveis por todo o código e, também, para alguns atributos ou métodos, dependendo das necessidades do software a ser implementado.

Quando a visibilidade utilizada é privada, o componente se torna restrito a uso local no momento em que este é apenas executado, para depois poder ter seu valor perdido.

No modo privado, podemos, por exemplo, ter atributos privados dentro de um método e estes serem criados junto com o método e destruídos ao final da execução do mesmo, tendo seus valores utilizáveis apenas neste intervalo de tempo de execução.

Também se pode encapsular um método e ser visível para métodos da mesma classe, tendo seu acesso restringido fora dessa classe, independentemente da visibilidade de seus atributos ou atributos e métodos da mesma classe.

Observe o exemplo de classe do código2, em que estão destacadas as palavras reservadas utilizadas para indicar a visibilidade de cada elemento e formalizar os aspectos do encapsulamento, indicando o nível de acesso para cada elemento (classe, método e atributo).

//código2

```
public class Classe {
        public tipo atributo <, atributo>;
        protected tipo método (<tipo atributo, tipo atributo>){
                  private tipo variavel <, variavel>;
                  <instruções>
       }
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

MÉTODOS ESTÁTICOS

Pode-se usar a palavra reservada **static** para tratar situações, em que um dos conceitos fundamentais da orientação a objetos não seja desejável, a instanciação de objetos (DEITEL, 2010). Além do mais, pode-se usar essa palavra em alguns casos específicos, quando um método único precisa ser utilizado de forma direta, sem que toda a classe tenha que ser instanciada sem necessidade.

Dessa forma, um método estático pode ser utilizado de forma independente, como no uso de funções no paradigma imperativo, ou seja, torna-se algo mais prático e sem muitas regras para acessar sua JAVAfuncionalidade.

O código3 traz um exemplo de declaração de método estático, em que a palavra static indica que a classe que possui o método não precisa ser instanciada para que o método seja utilizado.

//código3

```
public static void main(int valor){
      <instruções>
}
```

O método principal de uma aplicação em Java, chamado **main()**, sempre é definido como **static**, para que não haja problemas de execução quando a JVM (máquina virtual Java) tentar interpretar e executar a aplicação Java, além do fato de não haver, ainda, nenhum objeto instanciado que impeça o método **main()** de ser chamado.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Isso vale para quaisquer outros métodos que não necessitem estar vinculados à instanciação de objetos para serem utilizados dentro de uma aplicação Java.

POLIMORFISMO

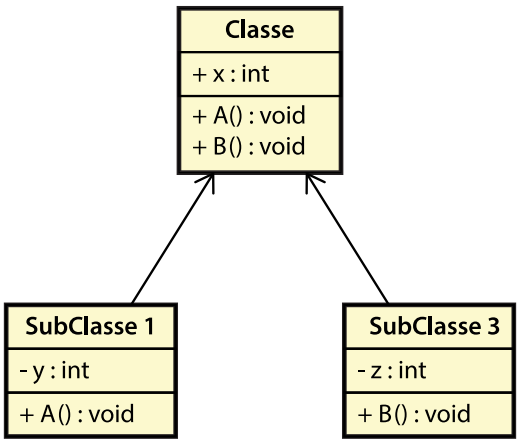


Figura 2 - Diagrama de classes para representar Polimorfismo
Fonte: o autor.

Essa é outra técnica importante dentro do paradigma de programação orientada a objeto, pois aceita que classes ou métodos genéricos possam ser adaptados para usos mais específicos, dependendo das necessidades de um programa (BARNES, 2009).

Um exemplo seria uma classe de tratamento de dados pessoais, em que um método poderia ser responsável pela validação do número de RG de um cadastro. Nesse caso, a regra de validação varia de estado para estado, diferentemente do que ocorre com o CPF que, por ser um documento federal, tem uma única regra de validação.

Então, dentro de uma classe de tratamento de dados, um método de validação de RG poderia ser ajustado de acordo com um parâmetro que informa o estado de origem do documento:

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código4

```
public class Classe {
        public tipo atributo <, atributo>;
        protected tipo metodo (<tipo atributo, tipo atributo>) {
                protected tipo variavel <, variavel>;
                    <instruções>
         }
 }
public class Classe2 extends Classe {
        public tipo atributo <, atributo>;
        private tipo metodo (<tipo atributo, tipo atributo>) {
                private tipo variavel <, variavel>;
                     <novas instruções>
        }
}
```

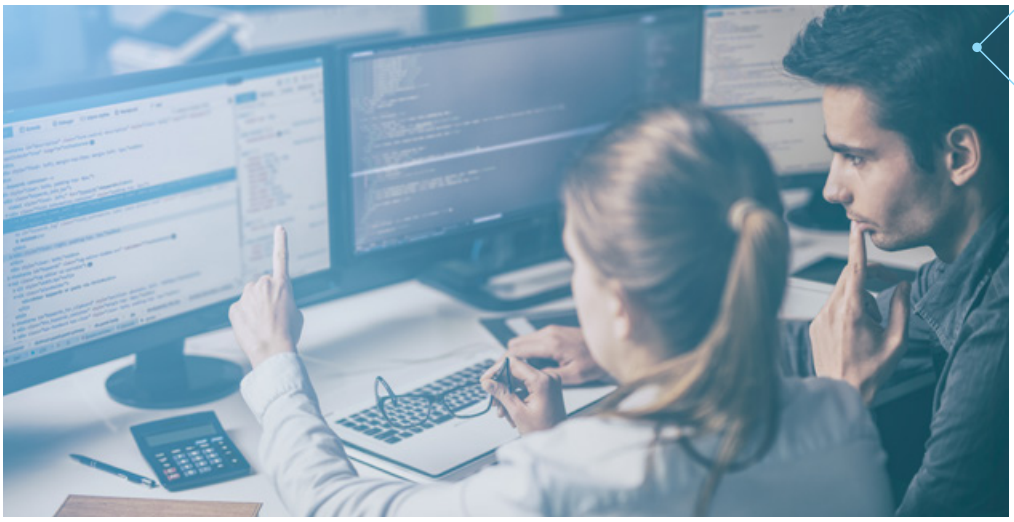
CLASSES ABSTRATAS

As classes são a base do paradigma orientado a objetos. Temos, como fundamento da classe, a sua instanciação para a criação e objetos que, por sua vez, criam cópias em tempo de execução das classes para processamento de dados por meio de métodos, por exemplo.

Toda classe é criada com a intenção de servir de molde para objetos que, consequentemente, serão usados pela aplicação e a criação de uma classe. A ideia de ser instanciada pode não fazer sentido inicialmente, porém, existe um tipo especial de classe chamada abstrata.

Uma classe abstrata é criada para não ser instanciada e utiliza, em sua declaração, a palavra chave abstract para informar ao compilador que a classe não pode servir de base para a instanciação de objetos (BARNES, 2009).

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



Dessa forma, pode parecer sem fundamento algo desse tipo, mas esse tipo de classe é adequado para o uso de recursos especiais, como a herança e o polimorfismo, que serão vistos mais adiante.

//código5

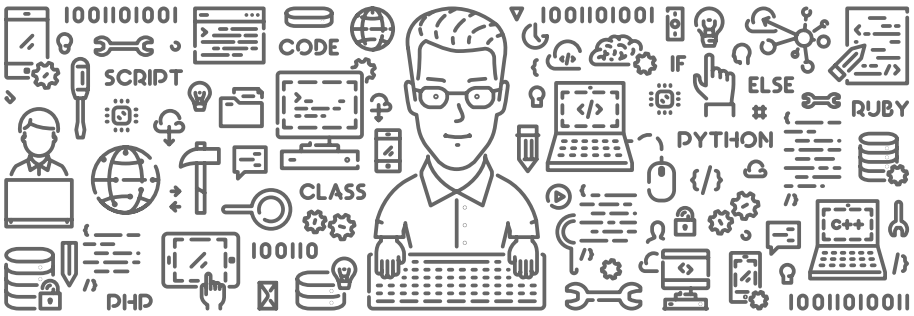
```
abstract class Classe {
        protected double x;
        protected int y;
        public double getDados () {
               return this.x;
        }
}
class SubClasse extends Classe {
      public double getDados() {
             return this.x * this.y;
}
```

REFLITA

Conhecer os conceitos da orientação a objetos é fundamental na programação Java, mas é essencial a prática na criação de código orientado a objetos para saber como aplicar cada conceito.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

PROGRAMMING



MÉTODOS ABSTRATOS

Um método abstrato, assim como a classe abstrata, tem o propósito de servir aos fundamentos de herança e polimorfismo, permitindo que sejam reescritos quando necessários (BARNES, 2009).

Dessa forma, um método abstrato é apenas declarado dentro da classe abstrata, de forma a não conter instruções de processamento, podendo ser redefinido por qualquer subclasse da classe com o método abstrato.

Assim, é importante observar que, pelo fato de uma classe possuir um método abstrato e este não possuir instruções, é obrigatório que exista pelo menos uma subclasse associada a ela para reescrever a função do método abstrato, evitando que o programa não possa ser compilado.

//código6

```
abstract class Classe {
        abstract double getDados();
}
class SubClasse extends Classe {
      public double getDados() {
                        return this.x * this.y;
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

DICAS PARA ELABORAÇÃO DE CLASSES

Dados de classes devem ser mantidos ocultos do restante do código, por motivo de segurança e para garantir o encapsulamento. Assim, é recomendável que atributos de classes e métodos de uma classe sejam definidos como privados.

Ao instanciar uma classe, é importante garantir que toda a estrutura de dados a ser tratada pela classe tenha recursos para reduzir erros em tempo de execução, como, por exemplo os valores indevidos em atributos (DEITEL, 2010).

Para isso, uma técnica muito simples é a inicialização de atributos com valores padrão básicos que não gerem erros na execução do código, aumentando o nível de segurança da aplicação em termos de funcionalidade.

Pode-se usar recursos como sobrecarga em métodos chamados construtores, atendendo diferentes demandas na inicialização de atributos e parâmetros, de acordo com as possíveis funcionalidades de um método, por exemplo. Esse tipo de método é tratado ao longo do livro e representa uma funcionalidade muito importante da linguagem Java.

Classes com muitos atributos simples, que sejam componentes que integram um mesmo dado, podem ser separados em classes menores para melhorar a legibilidade do código e facilitar seu manuseio, como no caso de um endereço que possua tipo de logradouro, nome do logradouro, número e complemento como tipos simples associados à essa estrutura de dados.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Alguns atributos podem não necessitar de métodos para tratar seus dados, pois estes podem ser definidos no momento em que esse atributo é inicializado em um construtor.

Outra regra útil para facilitar a legibilidade e manutenção de código é definir atributos com escopos padrão que, além da padronização, ajuda a reduzir a possibilidade de desenvolver código com erros de lógica ou falhas em encapsulamento, por exemplo (DEITEL, 2010).

Assim, classes podem ser definidas como públicas, subclasses podem ser públicas ou protegidas pela herança (uma das fortes características da orientação a objetos), atributos e métodos privados para garantir encapsulamento.

Classes complexas podem ser divididas em classes que mantenham contato entre si por meio do envio de mensagens durante a execução, ou a redução de classes em subclasses, para que a visibilidade seja reduzida dos seus elementos e tudo que for mais específico fique restrito ao uso pela superclasse por meio da herança (DEITEL, 2010).

Finalmente, nomes de elementos como classes, métodos e atributos devem ser lógicos e fazer sentido dentro do objetivo da aplicação, facilitando a legibilidade do código e sua manutenção.

SAIBA MAIS

Um conceito muito importante dentro da utilização de métodos se refere ao uso de recursos de hardware, mais especificamente da memória por parte dos objetos, quando instanciados na execução da aplicação.

Um construtor é um recurso que permite atribuir valores padrão para atributos e parâmetros, de forma a evitar que valores indesejáveis contidos na memória no momento da alocação da mesma sejam transferidos para o objeto.

Pode-se definir um construtor para inicializar o atributo valor de uma classe, com o propósito de melhorar a funcionalidade da aplicação ou facilitar o uso pelo usuário, com valores que, na maioria das vezes, se repetem, como o nome do estado em um cadastro de endereços.

Fonte: adaptado de Barnes. (2009)

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

CONSIDERAÇÕES FINAIS

Nestas duas primeiras Unidades do livro, os conceitos de orientação a objetos fundamentais foram trabalhados e servirão de base para todo o estudo da programação em linguagem Java.

Mais especificamente na segunda Unidade, os assuntos estavam relacionados a características do paradigma, mostrando como a orientação a objetos é capaz de oferecer boas formas de se estruturar uma aplicação.

Aspectos como a herança, permite que classes associaem-se a uma superclasse com nível hierárquico maior que as demais, em que métodos e atributos dessa superclasse podem ser compartilhados com as subclasses ligadas a ela.

O encapsulamento é outro aspecto trabalhado, no qual elementos de um código podem ser acessíveis ou não a outras partes desse código, por meio da chamada visibilidade, no qual podemos definir o nível de acessibilidade de elementos do código como públicos, protegidos ou privados.

Essa visibilidade, se bem aplicada, permite a criação de aplicações mais seguras e com funcionalidade transparente, devido a maior independência de métodos e classes entre si.

Dessa maneira, o reaproveitamento de código é muito maior e feito de forma mais fácil, pois uma classe encapsulada recebe apenas alguns parâmetros e devolve algum resultado por meio da passagem de mensagens entre objetos instanciados por meio dela.

O polimorfismo é outra importante técnica que oferece a possibilidade de alterar um método de acordo com as diversas demandas durante a execução da aplicação que, por sua vez necessitam de diferentes funções.

Foram abordados, também, outros conceitos, como o da abstração, que cria classes com métodos também abstratos, que não são instanciadas e servem como base para o polimorfismo, em que estes podem ser reescritos em outros pontos do código.

Na próxima unidade, a programação em si da linguagem Java é o foco, por meio de conceitos de controle de desvio de fluxo e o uso de laços de repetição.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATIVIDADES

1. Dentre os paradigmas existentes em programação, temos o paradigma orientado a objetos que se fundamenta em três componentes essenciais. **Cite quais são esses componentes.**

2. Classes podem conter atributos e métodos próprios. Associado a esses elementos, existe uma característica muito importante relacionado a como eles podem ser acessados e manipulados dentro da aplicação. **Como se chama essa característica?**

3. O encapsulamento é uma característica muito importante do paradigma orientado a objetos e permite que uma aplicação se torne mais segura por meio do ocultamento e bloqueio de acesso a classes, atributos e métodos, se desejado. **Quais as palavras reservadas utilizadas para definir o tipo de encapsulamento desses elementos?**

4. Herança é uma característica da orientação a objetos, que permite certo relacionamento entre classes. **Como isso ocorre? Descreva os detalhes que achar suficientes para comprovar essa relação.**

5. Existe uma característica muito útil que permite que uma subclasse possa alterar métodos definidos em sua classe-pai, desde que os métodos em ambas as classes sejam declaradas com o mesmo nome. **Como se chama essa característica?**

O desenvolvimento de aplicações em programação usando ambientes integrados de desenvolvimento (IDE) é muito comum entre profissionais que gostam dos benefícios que cada diferente IDE pode proporcionar, como compilação e execução no próprio ambiente, o que torna mais fácil o teste de um software.

Recursos adicionais, como cores e auxílio avançado à codificação também são recursos muito úteis que agilizam o processo de desenvolvimento, sendo muito indicadas aos profissionais de empresas de desenvolvimento.

Esses ambientes também controlam mais facilmente a criação de pacotes de aplicações, por meio de uma organização mais fácil dos componentes de um projeto, com a criação de toda a estrutura de arquivos e pastas para composição de uma aplicação.

Como padrão, ao criarmos um novo projeto, além de toda a estrutura básica de pastas, temos auxílio na implementação de código por meio de janelas que, por conseguinte, pedem as informações básicas para que o editor já traga a base do código, como nomes de pacote e classes, dentro dos padrões popularizados de nomenclatura, alternando maiúsculas e minúsculas nos nomes de cada elemento do projeto.

O código7 mostra um exemplo em que o pacote que representa o todo da aplicação tem seu nome "app1" todo em letras minúsculas, tal qual se chamada a aplicação, e o nome "APP1" todo em maiúsculas, para identificar o nome da classe criada que dará nome ao arquivo do tipo JAVA.

//código7

```
package app1;
public class APP1 {
  public static void main(String[] args) {
  }
}
```

Desse modo, temos a indicação de como fica a estrutura desse projeto "app1", utilizado para ilustrar a organização de elementos do projeto criado na IDE Netbeans, sabendo que pode haver variações dependendo da estrutura do projeto e de variações nas versões da IDE.

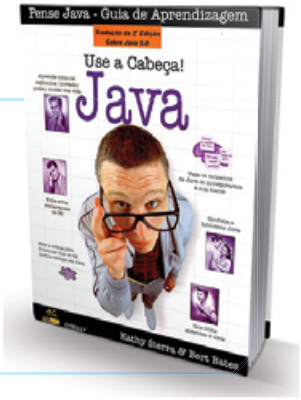
Da mesma forma que todos os demais conteúdos tratados neste material, esta parte pode ser uma ferramenta muito útil para o aprendizado da programação em Java, sendo que, com o passar do tempo e a constante prática, pesquisa de novas ferramentas e acompanhamento das evoluções da linguagem, o profissional tende a se tornar um ótimo programador Java, assim como de qualquer linguagem.

Fonte: o autor.

MATERIAL COMPLEMENTAR

LIVRO

Use a Cabeça! Java – 2ª Edição
Kathy Sierra, Bert Bates
Editora: Alta Books
Sinopse: este livro tem uma dinâmica diferente que atrai muitos estudantes, usando uma forma descontraída e que mostra importantes conceitos da linguagem Java com um linguajar leve, com muitas imagens e atividades.



REFERÊNCIAS

DEITEL, P. J. **Java**: como programar. 8. Ed. São Paulo: Pearson Prentice Hall, 2010.

MANZANO, J. A. N. G. **Java 7**: programação de computadores: guia prático de introdução, orientação e desenvolvimento. São Paulo: Érica, 2011.

BARNES, D. J.; KÖLLING, M. **Programação orientada a objetos com Java**. 4. Ed. São Paulo: Pearson Prentice Hall, 2009.

GABARITO

1) Desvio de fluxo é controlar a escolha de blocos de instruções que serão executadas de acordo com decisões tomadas pelo software durante a execução, e laços de repetição são blocos de instruções executadas por um determinado número de vezes pré-definidas ou não.

2) if (idade >= 18)

3) while e do...while

4) for (i=0; i<10;i++)

   {}

5) i = 0;

   while (i<10)

   {}

Professor Esp. Ronie Cesar Tokumoto

ESTRUTURAS DE CONTROLE DE EXECUÇÃO

UNIDADE III

Objetivos de Aprendizagem

- Conhecer as alternativas de controle de execução de blocos de instruções.
- Saber diferenciar quais estruturas usar em situações diversas, como as de controle de fluxo e laços de repetição, contados ou condicionais.
- Compreender os princípios de desvio de fluxo e laço de repetição.
- Saber aplicar as diferentes variações de laços de repetição.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Controle da execução das aplicações
- Instruções de controle de fluxo
- Laços de repetição contados
- Laços de repetição condicionais
- Variações de laços

UniCesumar

INTRODUÇÃO

Na unidade anterior, foram trabalhados conceitos complementares do paradigma orientado a objetos como herança, polimorfismo e encapsulamento, que compõem alguns dos conceitos mais importantes.

Esses conceitos são fundamentais para a compreensão de tudo que se refere à linguagem Java, pois ela é totalmente orientada a objetos, e uma aplicação bem elaborada pode conter todos esses recursos.

Nesta unidade, são tratados outros aspectos fundamentais da linguagem Java e de praticamente todas as demais linguagens de programação: o controle da execução dos processos.

Um dos aspectos contemplados é o controle de fluxo na execução da aplicação, que trata de como se podem escolher diferentes processamentos a serem realizados, por meio de condições baseadas em operadores lógicos.

Instruções de desvio permitem que um programa possa contemplar diversas alternativas de execução com um mesmo código, sem a necessidade de versões distintas para cada situação.

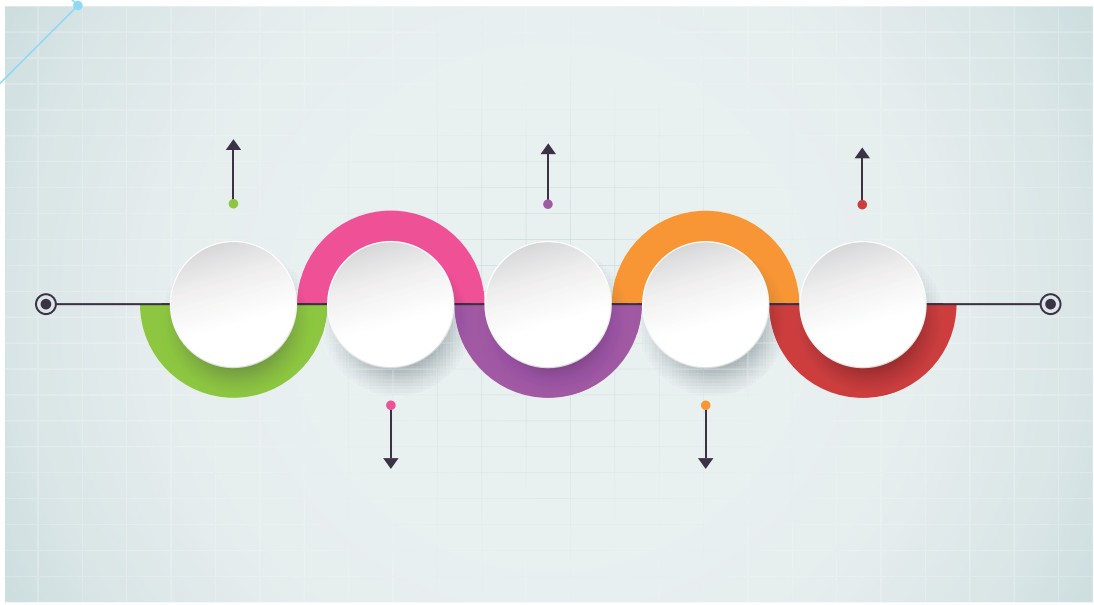
Além dos desvios, pode surgir a necessidade de executar diversas vezes um mesmo processamento, sem que ele tenha que estar replicado no código. Para isso, temos os chamados laços de repetição, que permitem que essa ação seja realizada de forma simples.

Esses laços podem ser com uma quantidade pré-definida de passos ou baseados em condições que sofram interferências externas ou de processamento das instruções contidas no próprio laço.

Laços e desvios representam estruturas fundamentais de programação, e a linguagem Java contém recursos muito bons para trabalhar com esses tipos de instruções.

O uso desses tipos de instruções permite uma variação muito grande de implementações de código. Esta unidade trabalha alguns exemplos criados por um excelente desenvolvedor, mostrando como diferentes códigos podem atender aos mesmos objetivos.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



CONTROLE DA EXECUÇÃO DAS APLICAÇÕES

O desenvolvimento de aplicações em Java dispõe de diversos mecanismos para oferecer um controle completo dos processos que ocorrem dentro da execução dessas aplicações.

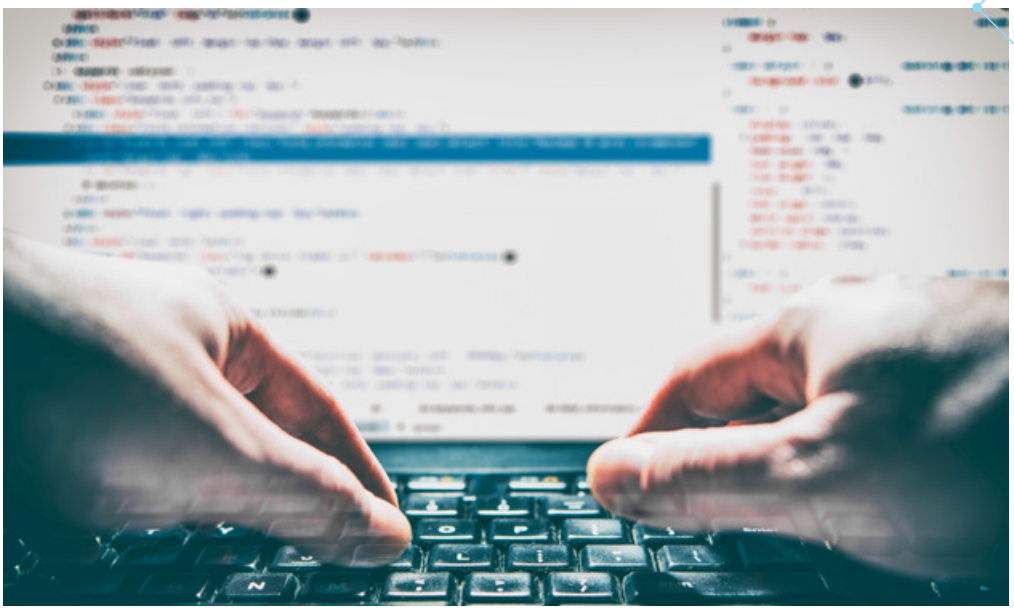
O chamado fluxo de execução se refere à forma com que uma aplicação executa seus processos, alternando ações a serem realizadas, gerando diferentes processamentos normalmente a cada execução.

Uma aplicação que não tenha mecanismos de controle do fluxo de execução funciona de forma linear, sem que haja mudanças na forma como é executada em todas as situações em que for acionada.

Apenas aplicações muito simples funcionam dessa forma, mas a grande maioria contém diversos mecanismos para controlar ações, permitindo escolhas referentes a quais processos devem ocorrer e o momento em que devem ocorrer.

Assim, uma aplicação pode possuir processos controlados pelos chamados desvios e laços, que representam a base para que seja possível o controle do fluxo de execução em uma aplicação.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.



INSTRUÇÕES DE CONTROLE DE FLUXO

O controle do fluxo de execução de uma aplicação é muito importante, pois, por meio destes desvios, a aplicação se torna mais completa e capaz de lidar com situações esperadas ou não, podendo depender da interação de um usuário ou não (MANZANO, 2011).

Tal controle de fluxo cria alternativas em tempo de execução, tornando a aplicação capaz de lidar com situações previstas como escolhas de valores para um atributo ou parâmetro de um método, ou imprevistas como o tratamento de erros.

Existe o chamado aninhamento de instruções de controle de fluxo, que consiste na inserção de uma instrução desse tipo dentro de outro, aumentando a quantidade de situações verificadas pelo controle, sendo que essas instruções aninhadas podem realizar suas verificações fundamentadas em condições baseadas ou não em um mesmo dado.

O código1 mostra a sintaxe básica dos comandos de controle de fluxo utilizados nesta unidade, para servir de base para a construção dos exemplos que serão utilizados em seguida (DEITEL, 2010).

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código1

```
if {...}
if {...} else {...}
if {...} else if {...} ... else {...}
switch (<expressão>) { case <valor>: <instruções> break; ...
                       default: <instruções> }
```

No código2, é apresentado um exemplo de código utilizando uma instrução de controle de fluxo, a fim de descobrir se um número armazenado em uma variável "x" é positivo ou negativo.

//código2

```
if (x >= 0)
   System.out.println ("Número Positivo");
else
   System.out.println ("Número Negativo");
```

Outra implementação possível para a mesma situação, mas contemplando a possibilidade do número digitado ser zero e não negativo e positivo é mostrado no código3.

//código3

```
if (x < 0)
   System.out.println ("Número Negativo");
else {
      if (x > 0)
          System.out.println ("Número Positivo");
      else
           System.out.println ("Número Zero");
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Em seguida, o código4 traz outra variação de código para analisar números, mas que identifica o valor armazenado na variável “x” para informar esse mesmo valor por extenso ao usuário.

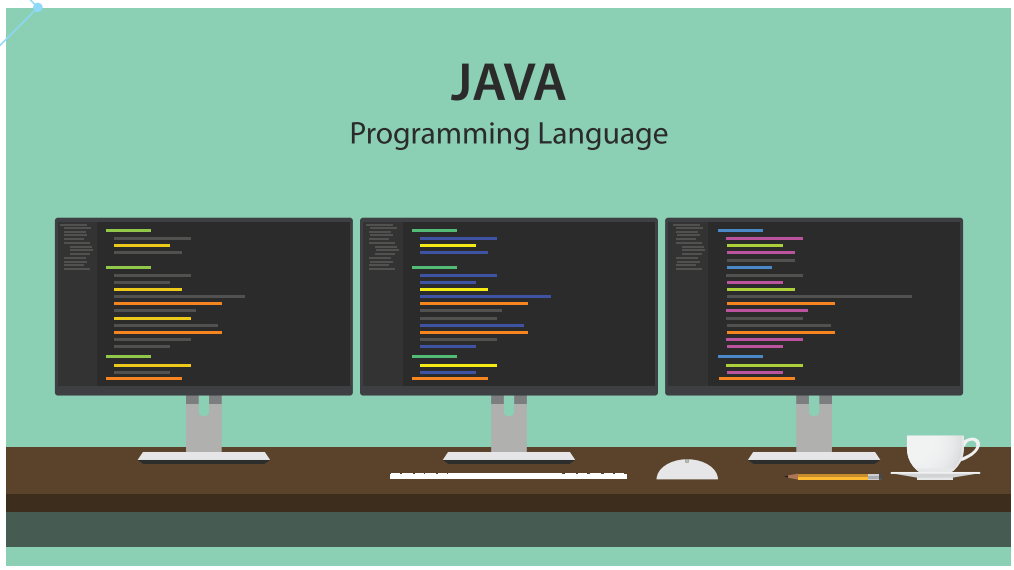
//código4

```
switch (x) {
        case 0:
System.out.println (“Número Zero”);
break;
      case 1:
             System.out.println (“Número Um”);
             break;
      case 2:
             System.out.println (“Número Dois”);
             break;
      default:
             System.out.println (“Outro Número”);
             break;
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

REFLITA

Programar não é uma receita de bolo, em que sempre usamos as mesmas medidas dos mesmos ingredientes. É por esse motivo que a programação é tão interessante.



LAÇOS DE REPETIÇÃO CONTADOS

Os laços de repetição servem para gerar repetições controladas, em que, a princípio, é controlada a quantidade de repetições a serem realizadas previamente à execução do bloco de instruções que, por sua vez, está associado a esse laço.

Desse modo, por meio de um contador que pode ser incrementado ou decrementado, o laço pode manter o controle de suas repetições, evitando o chamado loop infinito.

Equívocos na elaboração da lógica do laço e repetição, ou da própria sintaxe da instrução, podem ocasionar repetições não programadas do bloco de instruções associadas, podendo gerar até o loop infinito.

Basicamente, uma instrução for é composta de uma variável de controle de iterações que recebe um valor inicial, seguida da indicação do valor final para iterações e a condição para incremento dessa variável.

Dentro de uma instrução for, é possível que seja colocada uma outra instrução a ser executada, tantas vezes quantas forem as iterações do laço ou até um bloco de instruções delimitadas por chaves.

O codigo mostra um exemplo de instrução de laço de repetição **for**, para ilustrar uma forma de declaração que pode ser alterada livremente, de acordo com a necessidade e quantidade de iterações que devam ocorrer.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

No exemplo a seguir, no código5, a instrução é declarada por completo com todos os seus elementos e, dentro do laço, é usado o comando println para exibir em qual iteração se encontra a execução do laço.

//código5

```
for (i = 0; i<10; i++) {
        System.out.println ("Iteração "+i);
}
```

Em seguida, no código6, temos um exemplo de uso de uma instrução incompleta, que exibe as iterações que ocorrem durante a execução do laço de repetição, por meio do mesmo método println aplicado à variável i.

Um ponto importante é que o controle de incremento ou decremento do laço é executado dentro do próprio laço, por meio do operador incremento aplicado à variável i.

//código6

```
for (i = 0; i < 10) {
        System.out.println ("Iteração "+i);
        I++;
}
```

Já o exemplo apresentado no código7 mostra o mesmo resultado, mas elaborado de forma distinta do exemplo anterior, em que é perceptível a versatilidade do comando **for** em relação aos seus parâmetros de execução, aceitando a omissão destes sem prejudicar a execução, desde que sejam devidamente controladas as iterações por entre de outros meios.

No caso do código7, a inicialização da variável de controle é feita antes da instrução **for** iniciar, por meio da atribuição do valor inicial à variável de controle, e a forma como ocorre o incremento ou decremento das iterações é realizado dentro do próprio laço, como no código6.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código7

```
i = 0;
for (; i < 10;) {
      System.out.println ("Iteração "+i);
      i++;
}
```

No código8, como todos os parâmetros são omitidos, além de não haver inicialização nem condição de fim para o laço, não há incremento ou decremento, pois nem mesmo variável de controle é indicada.

Assim, o laço ocorre infinitamente, sendo indicado para aplicações que nunca devem parar de ser executadas, mas é preciso controle sobre o uso de laços desse tipo, pois, quando usado em pontos indevidos do código, pode criar situações indesejadas, nas quais a aplicação precise ser encerrada de forma forçada.

//código8

```
for (;;) {
}       // laço infinito
```

No código9, o exemplo mostra uma versão mais elaborada da instrução de laço, em que o controle de parada do laço de repetição ocorre em função de certa interação do usuário durante a execução do laço.

Neste exemplo, a aplicação aguarda o pressionamento da tecla **s**, para que a execução do laço seja interrompida, sendo que outras teclas inseridas aparecerão, mas não irão interromper o laço.

//código9

```
for (i = 0; (char) System.in.read() != 's'; i++) {
      System.out.println ("Iteração "+i);
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Já no código10, você pode observar um exemplo em que a própria instrução de laço é utilizada para modificar o valor de uma variável, incrementando seu valor de acordo com as iterações do laço por meio do operador +=.

//código10

```
for (i = 0; i < 10; x += i++) ; // comando sem bloco interno
                                    efetua soma
```

O laço do código11 é um laço vazio, que pode servir para ocupar o processamento por algum tempo, de forma a servir como pausa durante a execução de uma aplicação. Pode ser um recurso útil em casos em que um hardware muito potente pode afetar a usabilidade da aplicação, tornando-a rápida demais.

//código11

```
for (int i = 0; i < 10; i++) {
}
```

//código12

```
import java.util.Scanner;
public class Tabuada {
   public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     int n1, i;
     System.out.print("Entre com o valor para tabuada: ");
     n1 = sc.nextInt();
     for (i=1; i<=10; i++) {
        System.out.println(n1 + " X " + i + " = " + n1*i);
     }
   }
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

No código12, é proposta uma aplicação completa capaz de exibir a tabuada de um valor informado de forma simples e prática, em que alterações no código podem gerar uma aplicação mais agradável ao uso com recursos adicionais, como permitir que mais de uma tabuada fosse exibida, por exemplo.

Para isso, bastaria incluir as instruções em negrito como um bloco dentro de um laço, controlado pela interação do usuário, como exibido no código9.

LAÇOS DE REPETIÇÃO CONDICIONAIS

Complementando os laços contados, temos os laços condicionais, que incluem, em sua definição, regras para finalizar a execução das repetições, tendo maior facilidade na ocorrência de erros na definição do laço.

Temos como base a verificação da regra de validação do laço logo em sua definição, permitindo, inclusive, a verificação da ocorrência ou não do bloco de instruções contidas no laço uma única, várias ou nenhuma vez.

Um variação permite que a verificação seja realizada ao final do laço, garantindo que, pelo menos uma vez, a execução do bloco de instruções do laço ocorram.

No exemplo apresentado no código13, é exibido um laço controlado usando o comando while que, por meio de uma condição, verifica se a mesma já não foi atendida antes da execução do laço e, caso contrário, executa uma iteração do laço para, então, realizar nova verificação.

Nesse exemplo, o código13 não apresenta instruções dentro do laço, mas nada impede que, se o valor da variável de controle x continha qualquer valor superior a zero, esse laço ocorra infinitamente, como ocorre no exemplo no código8.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código13

```
while (x > 0) {
}
```

Já no código14, o exemplo traz a exibição do valor da iteração contado pela variável **i** na linha, contendo o método **println**, mas, nesse caso, a verificação da condição da condição ocorre apenas no final da iteração do laço e, assim, temos a garantia de que este seja executado pelo menos uma vez.

//código14

```
do {
   System.out.println ("Iteração "+i);
   i++;
} while (i<10);
```

Um ponto importante a ser observado é que esse tipo de laço condicional pode realizar o que o laço contado for faz, por exemplo, mas é capaz de criar laços mais complexos, que dependem de processamento das instruções internas dele ou de interação externa de usuários ou dados.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Já no código15, o laço, a princípio, tem como padrão executar diversas iterações do laço até que a condição de a variável de controle seja menor que 10, mas a instrução de controle de fluxo **if** pode interromper a execução do laço.

Caso a variável de controle **i** tenha o valor 7 em alguma das iterações do laço, o comando **break** tem a função de interromper instruções em execução, retomando o ponto seguinte na execução do código.

//código15

```
do {
      System.out.println ("Iteração "+i);
      I++;
      if (i == 7)
                  break;   // Força saída antecipada do laço
} while (i<10);
```

O código16 traz um exemplo de laço em que a instrução de desvio de fluxo if, quando satisfeita, desvia a execução para o ponto indicado no comando break.

Esse ponto se localiza dentro do próprio laço no exemplo, mas pode estar em qualquer parte do código, de acordo com a lógica e o que seja necessário para a funcionalidade da aplicação.

//código16

```
do
   ponto: {
          System.out.println ("Iteração "+i);
          I++;
          if (i == 7)
                    break ponto;  // Força reinício do laço
   } while (i<10);
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

No código17, temos um exemplo de código completo de uma aplicação simples de geração de tabuada, como no exemplo do código12, citado anteriormente.

A diferença nessa versão é o uso do laço condicional para executar o cálculo e exibição da tabuada. Também é possível incrementar essa implementação com funções adicionais, sem maiores problemas.

//código17

```
import java.util.Scanner;
public class Tabuada2 {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1, i;

        System.out.print("Entre com o valor para tabuada: ");
        n1 = sc.nextInt();
        i=1;
        while (i<=10) {
              System.out.println(n1 + " X " + i + " = " + n1*i);
              I++;
}
}
}
```

No código18, a seguir, temos a mesma aplicação, porém controlada de forma diferente, com a verificação da condição de parada ao final do laço, obrigando que seja executada pelo menos uma iteração no laço.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código18

```
Import java.util.Scanner;
public class Tabuada3 {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1, i;

        System.out.print("Entre com o valor para tabuada: ");
        n1 = sc.nextInt();
        i=1;
        do {
            System.out.println(n1 + " X " + i + " = " + n1*i);
            I++;
        } while (i<=10);
}
}
```

VARIAÇÕES DE LAÇOS

Podem-se criar laços contados utilizando instruções de laços condicionais, criando variáveis de controle que são tratadas dentro dos laços, de forma a agir como nos laços contados, sendo incrementadas ou decrementadas por meio de expressões adequadas, tomando o devido cuidado com a lógica envolvida na verificação da variável e controle do laço.

Outra forma de controle de laços de repetição e de desvio de fluxo se dá por meio de instruções que, por sua vez, podem interromper e retomar à execução da aplicação de forma controlada.

É possível o uso de instruções **while** e **do...while** para laços contados, assim como o uso dos comandos **break** e **continue** para controlar desvios, oferecendo novas formas de uso para instruções conhecidas.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

O código19 traz um exemplo de implementação da tabuada, em que um comando for teria que executar 100 iterações de acordo com sua definição, mas como a tabuada utiliza apenas 10 iterações, foi criada uma instrução condicional, que interrompe a execução do laço assim que o valor da variável de controle ultrapassar o limite definido.

//código19

```
import java.util.Scanner;
public class Tabuada4 {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1, i;
        System.out.print("Entre com o valor para tabuada: ");
        n1 = sc.nextInt();
        for (i=1; i<=100; i++) {
             if (i <= 10) {
                 System.out.println(n1 + " X " + i + " = " + n1*i);
             }
             else {
                   break;
             }
        }
}
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

A criação de programas envolve uma série de conceitos que, juntos, têm a função de oferecer mecanismos para a solução de problemas diversos para as mais diversas atividades, em que cada problema necessita de programas diferentes para se chegar ao resultado esperado, assim como se pode, também, ter diferentes interpretações do problema e soluções implementadas para cada interpretação.

A diversidade de soluções possíveis para cada problema é que torna a programação tão interessante e ao mesmo tempo trabalhosa, pois permite que a atividade de programar não seja rotineira, exigindo muito da capacidade mental de cada profissional.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Desse modo, temos que observar que, para cada problema proposto neste material, existe mais de uma solução possível em praticamente todos os casos, e, por isso, é preciso considerar que os exemplos disponíveis são apenas alternativas que podem ser modificadas e melhoradas de acordo com o estilo e conhecimento de cada programador.

Empresas também podem adotar padrões de codificação, para que os projetos desenvolvidos por seus colaboradores sejam os mais próximos possível desses padrões, facilitando manutenções preventivas e corretivas, além de atualizações e otimizações do código.

Estruturas de decisão permitem que o programador possa oferecer alternativas diversas em tempo de execução ao programa ou ao usuário, dependendo de haver, ou não, a necessidade de interação do mesmo com o aplicativo em cada processo, permitindo que ações ocorram em função dessas decisões tomadas , ou não, automaticamente.

Para resolver o problema a seguir, foram implementados diferentes maneiras para demonstrar que um código que se propõe a atender uma necessidade específica pode ser pensado de formas diferentes por programadores, sendo que um pode ter desempenho e lógica diferente das demais soluções propostas. Essas soluções foram implementadas originalmente por Jacques Sauvé, que conseguiu, em seus exemplos, mostrar como existem soluções distintas para um mesmo problema.

Imagine um código para verificar qual entre três valores informados é o menor e qual o maior, sendo que a ordem dos números informados possa ser aleatória, sem prejudicar a correta avaliação e comparação entre esses valores.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código20

```
package p2.exemplos;
import java.util.Scanner;

/*
* "Ler 3 números inteiros da entrada, imprimir o menor e o maior"
*
* Autor: Jacques Sauvé
*/

public class MinMax1 {
public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int n1, n2, n3;

       System.out.print("Entre com o primeiro inteiro: ");
       n1 = sc.nextInt();
       System.out.print("Entre com o segundo inteiro: ");
       n2 = sc.nextInt();
       System.out.print("Entre com o terceiro inteiro: ");
       n3 = sc.nextInt();
       if (n1 > n2) {
           if (n1 > n3) {
               if (n2 < n3) {
                   System.out.println("O menor numero eh: " + n2);
               } else {
                   System.out.println("O menor numero eh: " + n3);
               }
                   System.out.println("O maior numero eh: " + n1);
               } else {
               if (n1 < n2) {
                  System.out.println("O menor numero eh: " + n1);
               } else {
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
                System.out.println("O menor numero eh: " + n2);
                }
                System.out.println("O maior numero eh: " + n3);
                }
            } else {
            if (n2 > n3) {
                if (n1 < n3) {
                    System.out.println("O menor numero eh: " + n1);
                } else {
                    System.out.println("O menor numero eh: " + n3);
             }
             System.out.println("O maior numero eh: " + n2);
             } else {
               if (n1 < n2) {
                  System.out.println("O menor numero eh: " + n1);
               } else {
                   System.out.println("O menor numero eh: " + n2);
               }
               System.out.println("O maior numero eh: " + n3);
             }
}
}
}
```

Fonte: Sauvé ([2017], on-line)[1].

Nesse primeiro exemplo, a lógica é a mais comum, em que um dos valores é comparado com os demais. Nesse caso, uma comparação é feita entre o primeiro e segundo valor para definir depois se o primeiro valor é, também, maior ou menor que o terceiro valor. Desse modo, por meio dessas comparações, é possível avaliar qual é o menor entre eles para depois indicar o maior, baseando-se no primeiro valor. Caso contrário, se o primeiro valor não for maior que o segundo, este é comparado ao terceiro, seguindo uma lógica semelhante de comparações para definir o menor e depois o maior valor, deixando, neste momento, o primeiro valor em segundo plano.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código21

```
package p2.exemplos;
Import java.util.Scanner;

/*
* "Ler 3 números inteiros da entrada, imprimir o menor e o maior"
*
* Autor: Jacques Sauvé
*/

public class MinMax2 {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1, n2, n3;

        System.out.print("Entre com o primeiro inteiro: ");
        n1 = sc.nextInt();
        System.out.print("Entre com o segundo inteiro: ");
        n2 = sc.nextInt();
        System.out.print("Entre com o terceiro inteiro: ");
        n3 = sc.nextInt();

        int mínimo;
        int máximo;
        if (n1 > n2) {
            if (n1 > n3) {
                if (n2 < n3) {
                    mínimo = n2;
                } else {
                  mínimo = n3;
                }
                máximo = n1;
                } else {
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
                    if (n1 < n2) {
                        mínimo = n1;
                    } else {
                        mínimo = n2;
                    }
                    máximo = n3;
                }
            } else {
              if (n2 > n3) {
                    if (n1 < n3) {
                        mínimo = n1;
                    } else {
                        mínimo = n3;
                    }
                    máximo = n2;
              } else {
                if (n1 < n2) {
                    mínimo = n1;
                } else {
                    mínimo = n3;
                }
                máximo = n3;
              }
            }
            System.out.println("O menor numero eh: " + mínimo);
            System.out.println("O maior numero eh: " + máximo);
        }
}
```

Fonte: Sauvé ([2017], on-line)[1].

Nessa outra implementação, ocorre uma mudança na forma como são tratados os resultados, pois, baseado numa mesma lógica, o que muda é a forma como a informação de qual o menor e maior valor são passados como resultado.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Nessa segunda versão, utilizam-se variáveis auxiliares para armazenar o menor e o maior valor que, ao final do processamento dos valores, são utilizadas para exibir o menor e, em seguida, o maior valor dentre os três informados.

É clara a necessidade de uso de recursos adicionais nessa versão, tornando-a mais custosa para o processamento e memória, sem trazer benefícios perceptíveis em termos de quantidade de linhas de código ou desempenho durante a execução.

//código22

```
package p2.exemplos;

import java.util.Scanner;

/*
* "Ler 3 números inteiros da entrada, imprimir o menor e o maior"
*
* Autor: Jacques Sauvé
*/

public class MinMax3 {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;
        int mínimo = Integer.MAX_VALUE;
        int máximo = Integer.MIN_VALUE;

        System.out.print("Entre com o primeiro inteiro: ");
        num = sc.nextInt();
        if (num < mínimo) {
                mínimo = num;
        }
        if (num > máximo) {
                máximo = num;
        }
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
        System.out.print("Entre com o segundo inteiro: ");
        num = sc.nextInt();
        if (num < mínimo) {
               mínimo = num;
        }
        if (num > máximo) {
               máximo = num;
        }

        System.out.print("Entre com o terceiro inteiro: ");
        num = sc.nextInt();
        if (num < mínimo) {
               mínimo = num;
        }
        if (num > máximo) {
               máximo = num;
        }

        System.out.println("O menor numero eh: " + mínimo);
        System.out.println("O maior numero eh: " + máximo);
}
}
```

Fonte: Sauvé ([2017], on-line)[1].

Nessa terceira versão, as variáveis auxiliares são inicializadas por meio de constante padrão da linguagem, de forma que seja garantida a funcionalidade do código, pois, sem a correta inicialização, pode ocorrer de um endereço de memória atribuído a uma variável conter algum valor indesejado, interferindo na análise lógica feita pelas estruturas condicionais.

Essa é uma técnica bastante comum em programação, que, por sua vez, tem a função de garantir valores desejados para variáveis, a partir do momento em que possam ser utilizadas na execução do programa.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Assim, atribuindo a variável **mínimo** ao menor valor aceito pelo tipo de dado associado à variável, e a variável **máximo** ao maior valor, qualquer valor digitado como entrada poderia ser corretamente avaliado como menor ou maior entre os valores que vão sendo inseridos pelo usuário.

Outro ponto diferencial nessa implementação é que, a cada valor inserido, este já é analisado com as variáveis auxiliares, para que, em caso de condições que vão sendo atendidas, realizem a troca de valores dessas variáveis auxiliares, criando uma situação em que: como cada valor inserido já seja prontamente avaliado, uma mesma variável possa ser utilizada para armazenar todos os valores inseridos, sabendo que ,a cada novo valor pedido, o anterior possa ser descartado sem problemas.

Dessa forma, houve uma otimização de recursos de memória entre a segunda versão e essa terceira, fato que não influencia muito o desempenho nesta implementação específica, mas que, em larga escala, pode representar uma perceptível melhoria no uso de recursos e desempenho em tempo de execução.

//código23

```
package p2.exemplos;
Import java.util.Scanner;

/*
* "Ler 3 números inteiros da entrada, imprimir o menor e o maior"
*
* Autor: Jacques Sauvé
*/

public class MinMax4 {
public static void main(String[] args) {
       final int NÚMEROS_A_LER = 3;
       Scanner sc = new Scanner(System.in);
       int mínimo = Integer.MAX_VALUE;
       int máximo = Integer.MIN_VALUE;
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9610 de 19 de fevereiro de 1998.

```
        for (int i = 0; i < NÚMEROS_A_LER; i++) {
              System.out.print("Entre com o proximo inteiro: ");
              int num = sc.nextInt();
              if (num < mínimo) {
                    mínimo = num;
              }
              if (num > máximo) {
                    máximo = num;
              }
        }
        System.out.println("O menor numero eh: " + mínimo);
        System.out.println("O maior numero eh: " + máximo);
  }
  }
```

Fonte: Sauvé ([2017], on-line)[1].

Agora, temos uma implementação que combina estrutura de decisão com laço de repetição, de forma a reduzir as linhas de código em função da existência de uma estrutura de dados em forma de matriz unidimensional para armazenar os três valores.

Nesse caso, o uso do laço de repetição precisa ser cuidadosamente pensado, para que seu bloco de instruções interno seja capaz de analisar os valores de forma adequada a cada iteração do laço e inclusão de valores na matriz de valores.

Para avaliar a validade dessa regra de comparação repetitiva dentro do laço de repetição, podemos utilizar o código23 para perceber que, a cada valor pedido, a lógica de comparação não se altera nas três situações, provando que o uso de uma estrutura de repetição seria válido nesse caso. Aqui, então, houve uma clara redução de linhas de código, mantendo um uso semelhante de recursos de memória, porém, de forma distinta, com uma matriz de três elementos em vez de três variáveis diferentes. Uma vantagem clara dessa implementação, usando laço de repetição, ocorre à medida que a quantidade de valores e comparações iguais é realizada.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Se tivéssemos que usar a mesma lógica, mas para quantidades maiores de valores, o código tenderia a aumentar na mesma proporção da quantidade de valores, ao passo que, usando laços de repetição, não haveria aumento da quantidade de linhas de código, porém apenas a alteração da quantidade de iterações a serem realizadas, indicadas na instrução de repetição ou condição de parada.

Um fato interessante e de grande valor para programação é testar as diferentes alternativas, com o objetivo de observar seus desempenhos, verificando se o uso de códigos mais elaborados logicamente é algo vantajoso sobre programas de lógica mais simples em tempo de codificação, simplicidade de manutenção e esforço mental para desenvolvimento.

SAIBA **MAIS**

Estruturas de decisão oferecem funcionalidades muito importantes dentro da programação, pois permitem que o fluxo de execução do programa seja alterado de acordo com a análise de condições em tempo de execução.

Assim, sabemos que uma estrutura de decisão oferece meios de um programa possuir diferentes funções que podem ser programadas para desviar seu fluxo de execução de acordo com parâmetros recebidos, processamentos efetuados ou, até mesmo, de entrada de dados realizada por usuários.

Além das estruturas de decisão, os laços de repetição oferecem outras funcionalidades excelentes para, junto das estruturas de decisão, tornarem as aplicações muito mais completas, por meio da otimização gerada de código e melhoria na lógica de programação.

Fonte: o autor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

CONSIDERAÇÕES FINAIS

Nesta unidade, foram vistos conceitos de programação essenciais em qualquer linguagem de programação. Sem esses conceitos, programadores iniciantes podem não compreender a base lógica da elaboração de códigos, e os mais experientes sentiriam falta de mecanismos para resolver problemas comuns no desenvolvimento de software.

Iniciando com o chamado desvio de fluxo, comandos como **if** e **else** são capazes de oferecer formas de uma aplicação tomar decisões automáticas ou por meio da interação com usuário, por exemplo.

O uso de instruções a partir do comando **switch** oferece uma segunda alternativa para o uso de desvios em programas, de uma maneira mais fácil, em termos de codificação e compreensão, mas com limitações.

Já o conteúdo relacionado aos laços de repetição traz instruções baseadas na repetição de instruções por uma quantidade de vezes de maneira controlada e predefinida, dependendo de interações com o usuário ou do próprio software para encerrar o laço e seguir com a execução da aplicação.

Comandos como **while** servem para definir blocos de instruções que dependem de condições para serem executados e essa quantidade de iterações depende dessa condição e de como ela é trabalhada durante a execução do laço.

Há variações como o comando while, verificado apenas no final do bloco de instruções, fato que obriga o bloco ser executado pelo menos uma vez e sendo, também, controlado internamente por interações ou pelo processamento das instruções.

Os laços representam recursos muito úteis em programação, pois permitem a criação de laços que podem reduzir a quantidade de linhas de código, além de criar situações, como laços infinitos e combinações entre laços e desvios para problemas mais complexos.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATIVIDADES

1. Dentro da programação em linguagem Java, existem instruções com funcionalidades bem definidas. **Qual a diferença entre desvio de fluxo e laço de repetição?**

2. Instruções de desvio de fluxo são muito usadas em programação. **Cite um exemplo de desvio que se baseia na análise do valor da idade para a tomada de decisões.**

3. Existe um tipo de laço de repetição controlado no início do laço e outro que são controlados no fim, ambos baseados em condições. **Quais são as palavras-chave utilizadas nesses casos?**

4. Imaginando a confecção de um pequeno código em linguagem Java, dê um exemplo de **laço de repetição que realize 10 iterações.**

5. Baseado na ideia da atividade 4, **crie uma forma de realizar a mesma quantidade de iterações com um comando de laço condicional que analise no início a condição.**

Conexão com Bancos de Dados

Existe uma biblioteca voltada às conexões com bancos de dados chamadas JDBC (Java Database Connextivity) que utiliza comandos da linguagem SQL própria para gerenciamento de bancos de dados.

Para criar uma conexão, um método chamado DriverManager.getConnection() utiliza alguns parâmetros para realizar essa tarefa, indicando a aplicação e o endereço (URL - Uniform Resource Locator) para conexão com o banco de dados, junto com usuário e senha de acesso.

A classe DriverManager oferece outros métodos como getConnection para capturar a URL JDBC de conexão, que possui um formato específico dividido em três partes seguindo o modelo **jdbc:subprotocolo:subnome**, sendo a primeira parte apenas a indicação de da URL ser padrão JDBC, e as demais representam o driver e a fonte de dados com a qual será realizada a conexão.

//código23

```
import                                  java.sql.Connection;
import                                java.sql.DriverManager;
import                                 java.sql.SQLException;
import                               javax.swing.JOptionPane;
import                                 javax.swing.JTextArea;


 public class ExemploConnection {
        private Connection con = null;
        private String usuario = null;
        private String senha = null;
        private String urlConexao = null;
 public ExemploConnection() {
        urlConexao = "jdbc:derby://localhost:1527/ead-bd";
        usuario = "ead";
        senha = "ead";
        this.conectarBanco();
        this.desconectaBanco();
 }
```

```
public static void main(String[] args) {
        new ExemploConnection();
}
private void conectarBanco() {
         String saida = "";
         try {
         con = DriverManager.getConnection(urlConexao, usuário,
                                           senha);
         saida += "Informações de Conexão";
         saida += "\nFabricante SGDB: " +  con.getMetaData().
                   DatabaseProductName();
         saida += "\nVersão SGDB: " + con.getMetaData().
                   getDatabaseProductVersion();
         saida += "\nDriver SGDB: "+ con.getMetaData().
                   getDriverName();
         saida += "\nVersão Driver SGDB: " + con.getMeta
                   Data().getDriverVersion();
         saida += "\nUrl de Conexão: " + con.getMetaData().
                   getURL();
         saida += "\nUsuário: " 49 + con.getMetaData().
                   getUserName();
         JTextArea saidaArea = new JTextArea();
         saidaArea.setText(saida);
        JOptionPane.showMessageDialog(null, saidaArea, "Conexão
        realizada com Sucesso", JOptionPane.INFORMATION_MESSAGE);
}
```

```
catch ( SQLException ex) {

  JOptionPane.showMessageDialog(null, ex.getMessage(), "Erro ao
              conectar no banco", JOptionPane.ERROR_MESSAGE);
  }
}

private void desconectaBanco() {
        try {
             if (!con.isClosed()) {
                 con.close();
            }
         } catch (SQLException ex) {
           JOptionPane.showMessageDialog(null, ex.getMessage(),
          "Erro ao desconectar", JOptionPane.ERROR_MESSAGE);
         }
 }
}
```

É importante perceber como se pode utilizar bancos de dados em aplicações Java, pois esse estudo futuro pode oferecer ótimas oportunidades de trabalho.

A disciplina dupla de banco de Dados irá complementar bem esses conceitos tratados nesta leitura.

Fonte: adaptado de Junior e Pereira (2016).

MATERIAL COMPLEMENTAR

LIVRO

Programação Java para Web
Décio Heinzelmann Luckow, Alexandre Altair de Melo
Editora: Novatec
Sinopse: livro baseado no desenvolvimento de aplicações web com banco de dados, incluindo conceitos de javaserver etc.

REFERÊNCIAS

DEITEL, P. J. **Java:** como programar. 8. Ed. São Paulo: Pearson Prentice Hall, 2010.

MANZANO, J. A. N. G. **Java 7**: programação de computadores: guia prático de introdução, orientação e desenvolvimento. São Paulo: Érica, 2011.

JUNIOR, E. A. O.; PEREIRA, R. L. **Programação II**. Maringá: UniCesumar, 2016.

REFERÊNCIAS ON-LINE

[1] Em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/p2/html/intro/intro.htm>. Acesso em: 1 jun. 2017.

GABARITO

1. Desvio de fluxo é controlar a escolha de blocos de instruções que serão executadas de acordo com decisões tomadas pelo software durante a execução, e laços de repetição são blocos de instruções executadas por um determinado número de vezes pré-definidas ou não.

2. if (idade >= 18)

3. while e do...while

   for (i=0; i<10;i++)

   {}

4. i = 0;

   while (i<10)

   {}

5. i = 0;

   while (i<10)

   {}

Professor Esp. Ronie Cesar Tokumoto

UNIDADE IV

BIBLIOTECAS JAVA

Objetivos de Aprendizagem

- Conhecer algumas das bibliotecas existentes em Java.
- Compreender os conceitos da criação de aplicações gráficas.
- Saber os conceitos de pacote e interface.
- Compreender os conceitos básicos de fluxo de dados.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Bibliotecas
- Swing (interface gráfica de usuário)
- Pacotes
- Interface

UniCesumar

INTRODUÇÃO

Na unidade anterior, tratamos alguns dos conceitos básicos mais importantes da programação, envolvendo desvios de execução e laços de repetição, em aplicações, que representam mecanismos para aumentar a complexidade lógica e aumentar a versatilidade do que é possível realizar com a linguagem Java.

Nesta unidade, outros conceitos importantes serão tratados para complementar os tópicos abordados, auxiliando ainda mais na elaboração de códigos para resolver os mais diversos problemas.

As chamadas bibliotecas representam um excelente recurso para auxiliar programadores no desenvolvimento de aplicações, oferecendo recursos para todo tipo de situação.

Existem bibliotecas dedicadas a tarefas, como gerenciar entrada e saída de dados para uma aplicação, permitir a criação de gráficos diversos e interfaces gráficas para uso em softwares para web -ou não -, em dispositivos móveis ou computadores que podem estar ligados, ou não, em rede.

As chamadas bibliotecas podem oferecer recursos poderosos no desenvolvimento de software, por meio de códigos já implementados e que acompanham o pacote de desenvolvimento, ou criados por terceiros que deixam estas disponíveis para uso livre.

Também é possível criar bibliotecas próprias para ampliar a gama de facilidades no desenvolvimento de aplicações, por meio da redução de retrabalho pelo reuso desses códigos.

Uma biblioteca que receberá atenção é a biblioteca gráfica chamada Swing, capaz de criar interfaces gráficas com certa facilidade e com ótima qualidade, oferecendo diversas alternativas de criação dessas interfaces.

Complementando os conceitos tratados na unidade, veremos os conceitos de pacotes e interfaces que permitem recursos extras para melhorar o desenvolvimento de software em Java.

Com o andamento dos estudos nas duas disciplinas de Programação de Sistemas, é possível obter, ao menos, uma base para o desenvolvimento de aplicações em Java.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

BIBLIOTECAS

O uso de bibliotecas é importante em qualquer linguagem de programação, pois permite que novos programas criados tenham menor número de linhas de código e possam aproveitar códigos já implementados anteriormente pelo próprio programador ou por outros desenvolvedores em formato de arquivo, que agrupam, dentro de si diversas classes com seus respectivos métodos e atributos devidamente encapsulados, garantindo que sejam úteis ao programa que utilize a biblioteca em questão (MANZANO, 2011).

Tal encapsulamento é essencial para garantir a independência da classe importada em relação ao código criado, normalmente sem necessidade de alterações. Se necessário, é possível realizar mudanças por meio do polimorfismo, normalmente presente no paradigma orientado a objetos.

//código1

```
import java.util.Scanner;
import java.util.Random;
import java.util.List;
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
import java.util.*;
import javax.swing.*.
import java.awt.List;
import java.awt.*;
```

No exemplo do código1, temos uma série de instruções usando a palavra-chave **import**, para agregar bibliotecas ao código da aplicação durante o processo de geração dessa aplicação para execução futura.

Como a quantidade de bibliotecas padrão de Java já representa uma lista considerável de opções, e as criadas pela comunidade de programadores representam outra lista maior ainda, apenas algumas serão citadas neste material, mas outras podem ser pesquisadas em diversas fontes bibliográficas indicadas, ou na própria Internet.

Uma observação importante é que a importação pode ser mais específica, pela indicação direta das classes necessárias da biblioteca, ou mais genérica, usando o símbolo * para trazer todas as classes que pertencem à biblioteca (MANZANO, 2011).

É fácil perceber que especificar as classes a serem utilizadas na importação de certa biblioteca favorece a programação, pois além de facilitar a identificação do porquê de ter sido importada, não sobrecarrega a aplicação com classes não utilizadas em excesso.

Outra observação é com relação às linhas em negrito, que mostram a importação de classes com mesmo nome para dentro da mesma aplicação, podendo gerar erros na atribuição de chamadas a essas classes, caso não forem bem especificadas no código.

O ideal é a completa indicação do nome da biblioteca junto ao nome da classe, no momento da utilização de algum método relativo a estas, para que o compilador não busque métodos em classes erradas.

O código2 mostra uma forma de resolver esse problema, por meio da especificação completa da classe na chamada de algum método desta no código. No exemplo a seguir, a classe **List** utilizada se refere à biblioteca **AWT** (ORACLE, on-line)[1].

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código2

```
import java.awt.List;
import java.util.List;

public class TesteImports {
        public static void main(String[] args) {
                java.awt.List awtLista = new java.awt.List();
                List utilLista = new ArrayList();
        }
}
```

A seguir, algumas bibliotecas conhecidas são descritas para exemplificar suas funções e sintaxe de uso de alguns de seus métodos, para auxiliar os desenvolvedores iniciantes a compreender como utilizar qualquer uma delas.

JAVA 2D

Biblioteca para criação de conteúdo 2D como gráficos, texto e manipulação de imagens, como uma extensão para a biblioteca AWT, podendo criar imagens vetoriais e editar fotos com efeitos típicos como num editor de imagens comum (ORACLE, on-line)[1].

//código3

```
setBackground( Color.black );
```

J3D (JAVA 3D)

J3D é uma biblioteca para criação de gráficos em 3D e plug-ins do tipo applet para navegadores, sendo esta uma tecnologia que está sendo descontinuada pelos navegadores.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código4

```
Background bg = new Background(0.1f, 0.1f, 0.7f);
```

JAVAFX

Biblioteca gráfica para Java, com o intuito de permitir a criação de animações, interfaces, efeitos especiais, funções arrastar e soltar (drag and drop), áudio e vídeo.

Na versão 8 da linguagem, será incluída como API padrão no pacote JDK, em função da sua popularidade e uso extensivo pela comunidade de desenvolvedores (ORACLE, on-line)[1].

//código5

```
Caixa.setFont (Font.font ("", FontWeight.BOLD, 50));
```

REFLITA

As bibliotecas representam uma das maiores ferramentas de trabalho para programadores, pois trazem inúmeras funcionalidades prontas, que permitem maior agilidade à padronização de trabalho.

AWT (ABSTRACT WINDOWING TOOLKIT)

Biblioteca muito popular para a criação de interfaces gráficas, contendo janelas e botões, por exemplo, sendo muito utilizada pelos desenvolvedores, além de ser uma das bibliotecas mais populares.

Ela é uma biblioteca muito eficaz, pois, estando mais próxima do sistema operacional, deixa por conta deste a realização da criação de cada componente, podendo, com isso, oferecer maior desempenho.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Um ponto negativo é que, por depender dos recursos que os diferentes sistemas operacionais podem oferecer, acaba ficando mais limitada e correndo o risco de uma mesma aplicação ter diferenças visuais, dependendo da plataforma que esteja sendo executada (DEITEL, 2010).

//código6

```
JOptionPane.showMessageDialog (null,"Fechando Aplicativo");
```

SWING (INTERFACE GRÁFICA DE USUÁRIO)

Essa outra biblioteca é compatível com a biblioteca AWT, mas cria por conta própria todos os componentes que sejam implementados pelo usuário, sendo que, na biblioteca AWT, essa tarefa é delegada à capacidade gráfica do sistema operacional (DEITEL, 2010).

Além do mais, acaba sendo uma biblioteca de mais alto nível, com maior consumo de recursos de hardware e menor desempenho, porém capaz de recursos mais avançados e com a capacidade de manter uma maior compatibilidade visual entre as aplicações executadas em diferentes plataformas, justamente por não depender das capacidades do sistema operacional.

//código7

```
JButton botao = new JButton ("OK");
```

Complementando os estudos das bibliotecas, um pequeno aprofundamento da biblioteca Swing pode mostrar como a linguagem Java possui recursos muito bons de programação, facilmente utilizados.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

O exemplo apresentado no código7 traz uma pequena aplicação que soma dois valores inteiros, utilizando janelas individuais para cada etapa da execução, sem criar uma janela fixa para isso.

A biblioteca Swing foi parcialmente inserida no código, bastando a parte relacionada à classe **JOptionPane** que, por sua vez, é capaz de criar janelas de interação com o usuário (DEITEL, 2010).

Uma curiosidade da aplicação é que as caixas de diálogo recebem valores em formato String, ou seja, texto que não pode ser somado aritmeticamente, mas que, caso tenham o formato correto, podem ser convertidos.

A conversão fica por conta do método **parseDouble**, pertencente à classe **Double**, que contém métodos para manipulação de valores do tipo double para conversões necessárias em aplicações (DEITEL, 2010).

Buscando maiores detalhes na documentação Java, é possível encontrar uma referência a todos os parâmetros que são aceitos por métodos das classes da biblioteca Swing para, assim, compreender como podem ser adicionados recursos extras em caixas de diálogo, por exemplo.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

O método **JOptionPane.showMessageDialog** é um exemplo de método com vários parâmetros, em que o valor **null** do primeiro parâmetro se refere ao posicionamento da janela de diálogo, que por conseguinte, não tendo coordenadas definidas, centraliza a caixa na tela em função deste valor **null**.

Outro parâmetro contém um texto entre aspas, seguido de uma concatenação (+ de texto com outro elemento) com o valor da soma, que deverá ser exibido como mensagem da caixa de diálogo, representando o resultado do processamento na aplicação.

Finalmente, a mensagem **"Soma de Valores"** deve aparecer na barra de títulos da janela de exibição do resultado, em uma janela sinalizada como sendo de passagem de mensagens, devido ao método **PLAIN_MESSAGE** utilizado como parâmetro final da instrução.

//código8

```
import javax.swing.JOptionPane;
public class Soma
{
  public static void main (String args[]) {
        String valor1 = JOptionPane.showInputDialog ("Primeiro
                                    valor: ");
        String valor2 = JOptionPane.showInputDialog ("Segundo
                                    valor: ");
                double n1 = Double.parseDouble (valor1);
                double n2 = Double.parseDouble (valor2);
                double soma =  n1 + n2;
                JOptionPane.showMessageDialog (null, "O
resultado da soma é: " + soma, "Soma de Valores", JOptionPane.
PLAIN_MESSAGE);
        }
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Os exemplos código1, código2, código3 e código4 mostram as etapas da execução da aplicação desde a compilação e execução dentro do **Prompt de Comando** até a execução, com um teste de funcionalidade em interface gráfica.

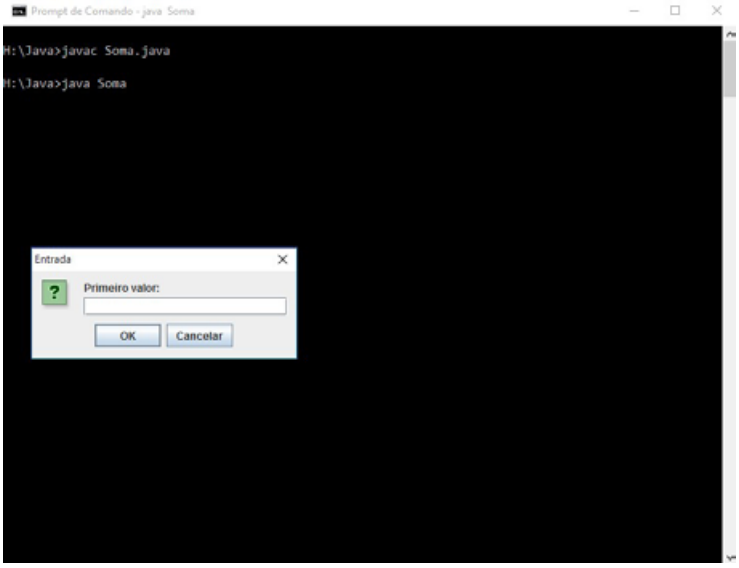


Figura 1 - Processo de compilação e execução da aplicação
Fonte: o autor.

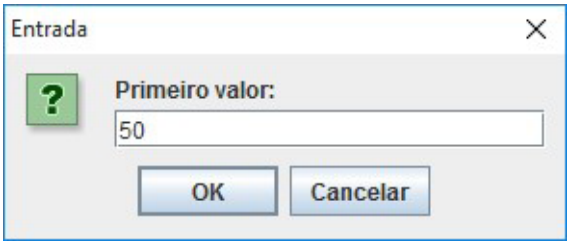


Figura 2 - Primeira caixa de diálogo da aplicação
Fonte: o autor.

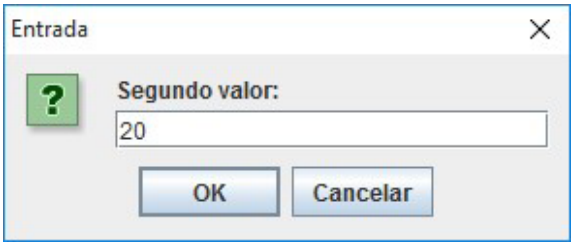


Figura 3 - Segunda caixa de diálogo da aplicação
Fonte: o autor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

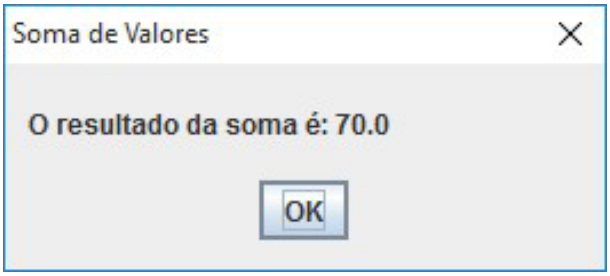


Figura 4 - Terceira caixa de diálogo da aplicação
Fonte: o autor.

Para um teste mais completo, foram inseridos valores e formatos distintos para teste de aceitação de valores, em que um valor inteiro foi inserido na primeira janela, e outro, fracionado, foi inserido na segunda janela.

Sem nenhum problema, a aplicação conseguiu realizar a conversão de **String** em valores do tipo **double**, observando o uso do ponto para separar a parte inteira da fracionária em Java.

SAIBA MAIS

Um importante recurso da linguagem Java, que para muitos programadores iniciantes pode não ser familiar, é o tipo **String**.

Pelo nome iniciado por letra maiúscula, automaticamente já ocorre a associação com classes, mas, pelo fato de não ser necessária sua instanciação como objeto, fica claro que se trata de outro tipo de elemento.

String funciona como um tipo especial de dado, que permite a manipulação de cadeias de caracteres que, em outras linguagens como C, é feita por vetores de caracteres ou pela adição do tipo por meio de bibliotecas.

Fonte: o autor.

Em linguagem Java, dois pacotes ou bibliotecas muito utilizados para programação em modo gráfico são AWT e Swing, capazes de oferecer classes e métodos para criação de janelas e demais elementos gráficos para criação de aplicativos e jogos.

O pacote AWT (Abstract Window Toolkit) é o pacote criado para manipulação de janelas e todos os componentes associados ao uso deste tipo de interface gráfica.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Um trecho de código utilizando o pacote AWT, que é acionado em um navegador por meio de uma chamada dentro do código HTML, é mostrado no Quadro 1:

Quadro 1 - Trecho de código Java utilizando pacote AWT

`import Java.AWT.Graphics;`
`public class Nome extends Java.Applet.Applet {`
`public void paint (graphics g) {`
`g.DrawString ("TEXTO");`
`}`
`}`
	Arquivo.java para ser compilado e gerar bytecode

`<HTML>`
`<HEAD></HEAD>`
`<BODY>`
`<APPLET CODE="Nome.class"`
`WIDTH="100%" WEIGHT="100%"></APPLET>`
`</BODY></HTML>`
	Arquivo HTML com chamada ao bytecode

Fonte: Manzano (2011).

O pacote Swing, por sua vez, é mais utilizado e, além de agregar praticamente todas as principais funções do pacote AWT, possui outras classes e métodos adicionais que complementam e potencializam o desenvolvimento de aplicações de modo gráfico, tornando-se um pacote mais popular em termos de uso pelos profissionais do desenvolvimento de software em Java.

No Quadro 2, temos o trecho de código Java contendo chamada ao pacote Swing para uso na criação de caixa de diálogo, para exibição de uma mensagem qualquer ao usuário:

Quadro 2 - Trecho de código Java utilizando pacote Swing

```
import Javax.Swing.JOptionPane;
class Nome {
      public static void main (String Args[ ]) {
            JOptionpane.ShowMessageDialog (Null, "TEXTO");
            System.Exit (0);
      }
}
```

Fonte: Manzano (2011).

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

PACOTES

Uma forma de se desenvolver projetos contendo diversos arquivos em linguagem Java é por meio da criação de pacotes que interligam os arquivos, mediante uma linha de instrução indicada no início de cada arquivo Java.

Desse modo, um pacote pode servir a dois propósitos, sendo, um deles, o de unir diversas classes que representem partes de um projeto maior, em que o resultado final é visto pelo usuário como uma aplicação única, e não um agrupamento de programas gerados em arquivos distintos.

Outro ponto positivo é que o empacotamento de classes aumenta a segurança da aplicação, pois torna as classes internas ao pacote encapsuladas, tornando-as privadas e inacessíveis de fora do pacote, a princípio.

Com essa prática, é possível, além de unir códigos diferentes em um mesmo projeto, quebrar códigos que poderiam se tornar muito extensos em partes menores que, por conseguinte, podem receber manutenções mais facilmente.

Essa forma de programação também melhora o reuso de código, pois cada arquivo Java pode conter apenas classes e métodos responsáveis por determinados tipos de tarefas dentro de objetivos mais específicos.

Além do mais, essa forma também influencia no encapsulamento tão desejado na programação orientada a objetos, em que o fato de partes do código estarem

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

em arquivos separados aumenta ainda mais a proteção a métodos e classes.

Para criar um pacote basta inserir o comando **package** no nome do mesmo, logo no início de cada arquivo Java do conjunto de arquivos que compõem o pacote.

Um detalhe importante é a configuração da variável de ambiente CLASSPATH, como informado anteriormente neste livro, para que todos os pacotes e arquivos que compõem estes pacotes sejam encontrados.

//código9

```
package pacoteExemplo;
class Um {
        <instruções>
}
Class Dois {
      <instruções>
}
```

Esse código deve ser gravado com o nome de **Um.java** em uma pasta chamada **pacoteExemplo**, assim como o nome do pacote indicado pelo comando **package**.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

INTERFACE

Interface é como classes abstratas com métodos sem corpo, ou seja, uma interface não especifica como fazer algo, mas o que deve ser feito. Além do mais, tem como diferença fundamental em relação a uma classe o fato de seus métodos não possuírem corpo com instruções, sem implementação.

A classe é responsável por codificar uma interface; em duas classes podem implementar uma mesma interface de formas distintas, mas contendo instruções para os mesmos métodos.

Por meio do uso de interfaces, trabalha-se fortemente com a ideia do polimorfismo em Java, permitindo que uma classe possa ter diversas implementações de seus métodos.

A seguir, o código10 mostra um código de interface em que o uso do tipo de visibilidade **public** indica que quaisquer classes podem acessar métodos dessa interface, ou a omissão da visibilidade, restringindo uso dos métodos às classes da interface.

É possível ver, no exemplo a seguir, que os métodos são indicados dentro da interface e o uso do símbolo de ponto e vírgula indica que apenas o cabeçalho do método será codificado na interface.

//código10

```
<public> interface nomeInterface {
         int Metodo (int parametro);
          int variavel;
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Um exemplo completo de uso é mostrado a seguir, ilustrando como é codificada uma interface e de que forma pode ser utilizada dentro de um programa Java.

//código11

```
class Teste implementsTesteInterface {
      int x;

      Teste() {
            x = 0;
      }
      public int getX() {
            x = x+1;
            return x;
      }
      public void zera() {
             x = 0;
}
}

classTesteMain {
     public static void main (String args[]) {
            Teste objeto = new Teste();
            while (x <10)
                  x = objeto.getX();
                  System.out.println ("Fim após"+ x+ "
                                        iterações");

     }
}
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

O uso de interfaces tem muita ligação com o paradigma da orientação a objetos e, usando a palavra-chave **extends**, podemos implementar a ideia de herança como em classes Java comuns, tais como as trabalhadas até o momento.

//código12

```
interface Teste1 {
      void x();
      void y();
}
interface Teste2 extends Teste1 { // Teste2 herda Teste1 void z(); }

class Exemplo implements Teste2 {
       public void x() {
           System.out.println ("Método X");
       }
       public void y() {
           System.out.println ("Método Y");
       }
       public void z() {
           System.out.println ("Método Z");
       }

       classExtensao {
              public static void main (String args []) {
                     Extensao objeto = new Extensao();
                      objeto.x();
                      objeto.y();
                      objeto.z();
              }
        }
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Finalizando esta unidade, foi possível compreender que, além de conhecer a sintaxe básica e comandos essenciais de uma linguagem, saber utilizar bibliotecas adicionais facilita o trabalho do desenvolvedor, como foi mostrado por meio do uso da biblioteca gráfica em exemplos dados.

SAIBA MAIS

Entrada e saída por meio de fluxo de dados: na linguagem Java, os processos de entrada e saída são realizados por fluxo de dados, em que o fluxo representa a ação de entrada de dados para processamento e, também, a produção de dados para retorno de algum processamento.

Entendendo como entrada e saída o uso de conjunto do software da aplicação com o hardware, entendemos que esse fluxo pode ser composto por dispositivos, como monitores, impressoras e discos de armazenamento permanente.

Fonte: o autor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

CONSIDERAÇÕES FINAIS

Com o término desta unidade, é possível ter ideia de como o uso de bibliotecas ajuda no processo de desenvolvimento de projetos de software, pois, com o seu uso, a tarefa de implementação se torna menos exaustiva e é concluída em um tempo menor.

Alguns exemplos de bibliotecas citados dentro da unidade permitem que o aluno tenha uma noção de como utiliza-las e quais tipos de recursos obterão.

As bibliotecas têm as mais diversas finalidades, sendo, em geral, bem específicas, facilitando sua identificação por parte do programador que precisa conhecer uma gama de bibliotecas que mais utilize, sabendo as finalidades e funcionalidades embutidas nas mesmas.

Bibliotecas gráficas, por exemplo, oferecem recursos interessantes no desenvolvimento de aplicações com interfaces gráficas modernas, capazes de estar adequadas aos padrões atuais visuais da Internet e dos softwares comerciais desenvolvidos no mercado.

Os mecanismos de entrada e saída de dados, tratadas na primeira disciplina de Programação de Sistemas, se unem aos mostrados nesta unidade, auxiliando o processo de desenvolvimento e reduzindo a quantidade de linhas de código, em função da necessidade de importar as bibliotecas no início do código e usar livremente seus métodos e classes ao longo do programa criado.

Se não houvesse essas bibliotecas, seriam necessárias muito mais horas de trabalho e retrabalho ao se desenvolver novas aplicações, devido à necessidade de reescrita constante de código.

Outro ponto é que, sem o uso de bibliotecas, o desenvolvedor necessita de conhecimentos muito avançados da linguagem de programação em si, além de diversos outros recursos, como hardware, rede, Internet, computação gráfica, entre outros.

Na próxima unidade, serão tratados os conceitos de encerramento do material, contemplando alguns tópicos interessantes que podem acabar sendo úteis dependendo de como seja a atuação de cada um, no futuro, dentro da área de Tecnologia da Informação.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATIVIDADES

1. Bibliotecas são arquivos que contém trechos de código que podem ser utilizados diversas vezes por vários programas. **Sendo assim, cite o nome das bibliotecas gráficas citadas no livro.**
2. Dentro da ideia de bibliotecas, algumas têm finalidades bem específicas. **Existe alguma diferença relevante entre as bibliotecas AWT e Swing?**
3. Um dos benefícios de otimizar a programação é a possibilidade de obter o chamado "reuso de código". **Em que isso impacta na programação de aplicações?**
4. Dentro da biblioteca Swing, temos diversos elementos disponíveis para uso. **Sendo assim, cite um exemplo de método da biblioteca Swing.**
5. Dentro das possibilidades de programação em Java, **qual a finalidade do uso de pacotes em Java?**

Programação Funcional com a Biblioteca Swing

O uso de interfaces gráficas nos dias atuais está tão comum que quase não se criam aplicações em modo texto, a não ser em casos em que se utilize hardwares específicos e com pouco poder de processamento, ou em casos em que seja necessário o maior desempenho possível em detrimento da aparência.

A biblioteca Swing traz recursos muito bons para trabalho com o desenvolvimento de aplicações web e de computação gráfica, por meio das bibliotecas AWT e Swing.

No código13, apresentado a seguir, veremos um exemplo de código que instancia um objeto botao1 que, ao ser clicado, ativa o evento que está ligado ao método **actionPerformed** para exibir a mensagem de saída.

//código13

```
JButton btn1 = new JButton ("OK");
btn1.addActionListener (new ActionListener() {
public void action Performed (ActionEvent e) {
        System.out.println ("Botão Acionado");
}
});
```

No exemplo do código13, temos uma ideia semelhante, porém utilizando uma forma mais moderna de programação usando o paradigma funcional, em que, além da sintaxe, o código em si acaba tendo uma forma diferente, mais compacta e complexa em termos de interpretação.

//código13

```
JButton btn2 = new JButton ("OK");
btn2.addActionListener ( (event) -> System.out.println ("Botão
                                    Acionado") );
```

A API Swing é uma biblioteca importante para o desenvolvimento de aplicações em Java que necessitem de interface gráfica, assim como a API AWT, JavaFX, Apache Wicket ou GWT, por exemplo.

Além disso, oferece poderosos recursos para criação de interfaces gráficas de boa qualidade e capazes de suprir uma boa parcela das necessidades dos profissionais de desenvolvimento.

Eventos podem ser definidos em classes sem nome e, muitas vezes, quando é preciso apenas atribuir um comportamento de um elemento (botão, campo etc.), é possível passar um objeto instanciado da classe como parâmetro a um evento ActionListener sem nome, por exemplo.

Fonte: o autor.

MATERIAL COMPLEMENTAR

LIVRO

Projetos de POO em Java
Martins, F. Mário
Editora: Fca Editora
Sinopse: livro com bom embasamento teórico para o desenvolvimento de projetos baseados no paradigma da orientação a objetos.

REFERÊNCIAS

DEITEL, P. J. **Java**: como programar. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

MANZANO, J. A. N. G. **Java 7**: programação de computadores: guia prático de introdução, orientação e desenvolvimento. São Paulo: Érica, 2011.

REFERÊNCIAS ON-LINE

[1] Em: <http://education.oracle.com/pls/web_prod-plq-dad/ou_product_category.getPage?p_cat_id=267>. Acesso em: 2 jun. 2017.

GABARITO

1) Java 2D, Java FX, J3D, AWT, Swing.

2) A biblioteca AWT é menos completa que a biblioteca Swing, a qual contém praticamente tudo que AWT possui e mais diversas funcionalidades extras.

3) O reuso é o aproveitamento de código, a qual pode ser importante quando são desenvolvidas diversas aplicações que, por sua vez, podem compartilhar recursos e trechos de código. As bibliotecas foram criadas, também, em função desse reuso de código.

4) ShowMessageDialog().

5) Um pacote pode servir a dois propósitos, sendo estes: unir diversas classes que representem partes de um projeto maior, em que o resultado final é visto pelo usuário como uma aplicação única, e não um agrupamento de programas gerados em arquivos distintos.

Professor Esp. Ronie Cesar Tokumoto

UNIDADE V

TÓPICOS ADICIONAIS

Objetivos de Aprendizagem

- Conhecer algumas das mais importantes inovações da programação na versão 8 da linguagem.
- Saber que existem formas diferentes de se programar na linguagem Java, além do paradigma orientado a objetos.
- Conhecer alguns recursos adicionais disponíveis para o desenvolvimento de aplicações em Java, como Javabeans.
- Ser capaz de gerenciar projetos por meio de ambientes integrados de desenvolvimento.
- Conhecer os conceitos básicos de persistência na programação em Java.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- A versão 8 da linguagem Java
- Programação funcional em Java
- Javabeans
- Gerenciando projetos com Netbeans
- Persistência em Java usando JPA

UniCesumar

INTRODUÇÃO

Finalmente, chegando à última unidade da segunda parte da disciplina de Projeto de Sistemas utilizando como base a linguagem Java, já tratamos a grande maioria dos tópicos a serem trabalhados nesta disciplina.

Estudamos os temas ligados ao uso de bibliotecas para permitir o reuso de código e uma redução no tempo de produção de software, acarretando em redução de custos, mão de obra e garantia de prazos de entrega de produtos.

As bibliotecas permitem que o programador possa replanejar seu trabalho, visando dedicar uma fatia de seu tempo para conhecê-las, para estar sempre atualizado e descobrindo elementos que possam contribuir com a melhoria constante necessária na produção de software.

São vistos temas que possuem relação com o desenvolvimento de software em Java, além de permitir que o aluno possa conhecer pontos adicionais do processo de desenvolvimento em Java.

A unidade inicia mostrando características da versão 8 da linguagem Java, trazendo, de forma resumida, alguns conceitos novos inseridos na linguagem.

Um desses conceitos é a adaptação da sintaxe e semântica da linguagem para a aceitação do paradigma funcional que, por sua vez, está revolucionando a programação com códigos mais elaborados e menores, gerando maior desempenho e novos recursos.

Na programação funcional, é possível perceber como alguns conceitos podem ser modificados basicamente em função das chamadas funções lambda, que representam um desses pilares da versão 8 da linguagem.

Javabeans também são citados, agregando novas ideias ao reuso de código e permitindo o desenvolvimento de aplicações com funcionalidades distintas.

Finalizando a unidade, é comentado o uso de um ambiente integrado de desenvolvimento de software, para mostrar que o uso de uma ferramenta mais elaborada que o editor de textos e o prompt de comando do sistema operacional também pode trazer vantagens ao desenvolvedor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

A VERSÃO 8 DA LINGUAGEM JAVA

Desde 2004, quando a Sun Microsystems lançou a quinta versão da linguagem Java, não se criava tanta expectativa com o lançamento de uma nova versão da linguagem Java (ORACLE, on-line)[1].

Mudanças gerais na linguagem em si e de seu compilador, bibliotecas e na própria Máquina Virtual para execução dos arquivos bytecode trouxeram um agregado interessante de inovações.

Um dos pontos mais importantes dessa oitava versão é a implementação de conceitos de programação funcional, que permitem mudar a forma como são criadas e manipuladas as funções, inclusive em tempo de execução.

Essa mudança não altera o paradigma orientado a objetos da linguagem, mas acrescenta uma nova gama de potencialidades para a programação em Java, trazendo seu uso profissional para um novo nível.

As chamadas funções Lambda seguem a linha de programação funcional de linguagens como **Lisp** e, além do mais, não precisam ser declaradas com nome e

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

tipo de retorno, além de permitirem que a quantidade de linhas de código possa ser reduzida em certos casos.

Exemplos de funções Lambda válidas em linguagem Java podem ser observadas no código1, em que cada função executa uma tarefa simples, porém clara:

//código1

```
(float x, float y) -> { return x * y; }
() ->System a.out.println ("Texto Exemplo");
() ->{ return 100 };
```

Na primeira, temos dois parâmetros de entrada que, por sua vez, resultam em um retorno com a multiplicação desses dois valores passados como parâmetros. Na segunda função, temos uma função sem parâmetros, que basicamente exibe um texto na tela, podendo ser útil para mensagens de alerta, por exemplo. Finalmente, a terceira função apenas retorna um valor inteiro, que pode se referir a alguma constante utilizada para cálculos em um programa.

O uso das chaves para delimitar os comandos da função é necessário apenas em casos em que a função detenha um bloco de vários comandos e não apenas um. Nesse caso, é dispensável o uso desses símbolos delimitadores.

Nos exemplos da Figura 2, vemos, na primeira parte, um exemplo comum de código Java sem o uso das funções Lambda, em que é instanciada a Thread para execução em paralelo.

Já na segunda implementação, temos, além da redução de código, o uso da sintaxe funcional, que associa direto o comando à função "t", definida em "Teste", sendo, depois, instanciada em paralelo como no primeiro código.

Finalmente, na terceira implementação, temos que uma função Lambda pode ser diretamente passada como parâmetro para um método, tornando essa forma de programação ainda mais interessante para a linguagem.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código2

```
Teste = new Teste() {
      public void teste() {
      System.out.println ("Thread");
      }
};
new Thread (t).start ();
```

//código3

```
Teste t = () -> System.out.println ("Thread");
      new Thread (t).start ();
```

//código4

```
new Thread (
    () -> System.out.println ("Thread")
).start ();
```

Além da adição da programação funcional, uma mudança significativa ocorreu com a biblioteca (API) de datas e horas, que foi simplificada em seu uso, tornando-a mais intuitiva, trazendo melhorias no mecanismo de controle de execução de métodos, identificando com precisão os seus chamadores, aumentando a segurança na execução de aplicações.

PROGRAMAÇÃO FUNCIONAL EM JAVA

Ações como percorrer listas para encontrar um elemento é algo bastante comum e pode ser implementado, como visto no código5, com uma sintaxe um pouco diferente da convencional em Java.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Neste código5, até a própria sintaxe do comando de laço controlado for é diferente, aumentando sua versatilidade, mas dificultando a sua interpretação (ORACLE, on-line)[1].

//código5

```
System.out.println ("Lista de números pares");
List<Integer> lista = Arrays.asList (2, 4, 6, 8, 10);
for (Integer i: lista) {
      System.out.println (i);
}
```

Utilizando funções lambda com classes de coleções Java, ações específicas, tais como percorrer listas em busca de elementos, são algo simples de implementar, como é visto no código6.

//código6

```
System.out.println ("Imprime todos os elementos da lista!");
List<Integer>list = Arrays.asList (1, 2, 3, 4, 5, 6, 7);
list.forEach (n ->System.out.println (n));
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Usando o método **forEach** na lista do exemplo apresentado no código6, este método já espera por uma função do tipo lambda, como argumento que rodará um laço de repetição, o qual executa a função lambda para percorrer a lista de elementos a cada iteração e retornando a impressão do elemento seguinte.

O código6 pode ser incrementado com à adição de um controle simples, para assim analisar o resultado da função lambda antes de exibir seu resultado, adicionando um comando de desvio de fluxo **IF**, que filtra os elementos encontrados, utilizando apenas os pares por meio da operação de cálculo do resto da divisão do elemento da lista por 2.

Caso seja 0 o resto da divisão, esse elemento é um número par, sendo, então, exibido, e, caso contrário, a condição do desvio falha e o elemento não é aproveitado na busca.

//código7

```
System.out.println ("Imprime todos os elementos pares da lista!");
List<Integer>list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
list.forEach(n -> {
if (n % 2 == 0) {
    System.out.println(n);
}
});
```

Podemos, também, alterar no código8, para que, antes de retornar um valor, a instrução mesma realize ações, como cálculos e conversões. No código8, é possível observar que o valor encontrado é multiplicado antes de ser retornado ao usuário pelo método **System.out.printl**.

//código8

```
System.out.println ("Duplica valores dos elementos da lista");
List<Integer> lista = Arrays.asList (1, 2, 3, 4, 5);
lista.forEach (i ->System.out.println (i * 2));
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Uma interface chamada “Comparator” é capaz de auxiliar o trabalho com listas de dados, permitindo a comparação para o processo de ordenação da lista, sendo que, para cada parâmetro de comparação, é criada uma interface.

Para implementar a classe Texto do codigo 9, seriam necessários apenas os atributos nome, o construtor da classe e os métodos **getter** e **setter** padrões para a classe. Observe o código que utilizaria a classe Texto da figura 9, para criar um processo de ordenação de dados para uma lista de nomes (ORACLE, on-line)[1].

Neste exemplo do código9, a instrução em negrito cria os objetos para armazenar 3 instâncias da classe **Texto,** em que são atribuídos nomes quaisquer, para depois serem comparados entre si.

//código9

```
System.out.println (“Ordenação de Nomes”);
List<Texto> lista = Arrays.asList (new Texto (“Nome1”, 10),
        new Texto (“Nome2”, 20), new Texto (“Nome3”, 30));
Collections.sort (lista, new Comparator<Texto>() {
@Override
publicintcomparacao (Texto nome1, Texto nome2){
        return nome1.getNome().comparacao (nome2.getNome());
}
});
lista.forEach (p ->System.out.println (p.getNome()));
```

Novos paradigmas são criados com a evolução da TI, mas isso obriga a constante atualização do profissional. Até que ponto esse constante aprendizado é válido?

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

JAVABEANS

Componentes reutilizáveis de software criados em Java são chamados de JavaBeans, sendo que o chamado Bean se refere a uma classe Java, tendo seus atributos e métodos expostos, seguindo convenções de nomes como no caso dos métodos setter e getter (BARNES, 2009).

Como características, temos que um Bean representa um objeto que agrega vários outros objetos, facilitando sua manipulação e passagem como argumento único e com construtores nulos.

Algumas vantagens do uso de JavaBeans podem ser colocadas como a possibilidade de controle das propriedades de um Bean por meio de aplicações externas; permitir que um Bean receba eventos de outros objetos ou envie eventos para outros objetos com os quais possuem contato; podem ser ajustados externamente por meio de softwares.

Existem, também, desvantagens claras em relação ao uso e segurança, pois, não tendo construtores para seus métodos, podem receber entradas de dados inválidas por usuários ou de frameworks automatizados e passarem despercebidos.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Também há a necessidade de criação de métodos getter e setter para todas as propriedades de um JavaBean.

A seguir, o código9 mostra um exemplo de classe que cria uma interface capaz de gerar dados serializáveis, ou seja, que podem ser gravados em arquivo texto.

Dentro dessa classe, são criados os métodos **getter** e **setter**, a fim de realizar as ações básicas para o atributo da classe chamado **nome**. Existe, também, um método construtor para inicializar um valor vazio, reduzindo a chance de erros na execução por valores indevidos no atributo, quando ocorrer a alocação de memória para esse atributo.

//código9

```
public class ListaNomes implements java.io.Serializable {
        private String nome;

        public String getNome() {
                return nome;
        }
        public void setNome(String nome) {
                this.nome = nome;
        }

        publicListaNomes() {
                this.nome="";
        }
}
```

Já no código10, temos um exemplo de código HTML que ativa o bean, criado pelo arquivo Java, para utilizar seus recursos em um formulário comum que, por meio da tag<form>, indica que os dados serão postados em local indicado após preenchimento e clique no botão "Enviar Dados".

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

O código da figura 10 está simplificado apenas para fins didáticos. Programadores web podem adaptar esse código HTML de acordo com as características que achar conveniente para cada uso.

O código da Figura 9 deve ser gravado em arquivo próprio de nome **ListaNomes.java**, em função do nome da classe, e o arquivo da figura 10 pode ser gravado com qualquer nome com a extensão correta, como, por exemplo, **cadastro.jsp**.

//código10

```
<!DOCTYPE html>
<html>
<head>
     <meta http-equiv="Content-Type" content="text/html;
                                     charset=ISO-8859-1">
     <title>Uso de JavaBean</title>
</head>
<body>
     <jsp:useBean id="lista" class="pacote.ListaNomes"></
                                                  jsp:useBean>
     <jsp:setPropertyproperty="*" name="lista"/>
     <form action="listanome.jsp" method="post">
          Nome: <input type="text" name="nome"><br>
          <input type="submit" value="Enviar Dados">
     </form>
</body>
</html>
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

GERENCIANDO PROJETOS COM NETBEANS

Projetos em **Netbeans** são gerenciados por um módulo integrado ao ambiente de desenvolvimento chamado **Ant**, responsável por executar scripts de compilação, execução e depuração dos programas implementados (BARNES, 2009).

Para a criação desses projetos, existem os chamados **Templates**, que seriam modelos que facilitam o desenvolvimento por terem sua base codificada, servindo como ponto de partida para o desenvolvimento de novas aplicações, sem a necessidade de iniciar do zero novos programas Java ou de qualquer linguagem que já esteja vinculada à IDE.

Duas opções fundamentais estão disponíveis para uso, sendo a opção Standard Templates, que contém os modelos mais comuns, todos gerenciados pela ferramenta Ant integrada ao IDE Netbeans, e a outra chamada de **Free-formTemplates**, responsável pelo gerenciamento de scripts Ant criados pelo próprio desenvolvedor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

Ant é uma ferramenta de gerenciamento da construção de aplicações Java, utilizando script previamente implementado para compilação, depuração e organização de projetos.

Realizados como tarefas (**tasks**), todos os procedimentos efetuados pela ferramenta Ant são armazenados como arquivos do tipo build-impl.xml, mas mantém no arquivo build.xml os scripts contendo as tarefas de construção de uma aplicação (BARNES, 2009).

Dentro desse arquivo build-impl.xml, existe uma divisão por seções, que tem como objetivo melhorar a legibilidade das aplicações, além de conter divisões, tais como **Initialization,** para indicar argumentos de entrada, propriedades do projeto e usuários cadastrados.

Compilation serve para organizar o processo de compilação, de forma a organizar em pastas todas as classes e códigos fontes compilado.

A seção **Jar** define arquivos compactados do tipo JAR, assim como a seção **Javadoc** se refere aos scripts para documentação dentro do padrão do utilitário Javadoc.

Execution e **Debugging** tratam, respectivamente, das seções para definições das tarefas de execução e controle de depuração da execução de programas criados.

Após avançadas algumas etapas na escolha do novo projeto, é possível utilizar um gerenciador de layouts para facilitar a elaboração visual de aplicações, trazendo modelos de posicionamento de componentes de um novo projeto.

Opções como **FlowLayout**, **BorderLayout**, **GridLayout**, **BoxLayout** e **CardLayout** apresentam possibilidades de alinhamentos para o posicionamento dos elementos a serem implementados no código Java, utilizando padrões de layouts já conhecidos e popularizados.

O Netbeans divide os tipos de projeto Java de acordo com categorias indicadas na janela de criação de novos projetos, tendo como passo inicial a escolha entre as seguintes opções:

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

General, que trata de aplicações baseadas na plataforma **J2SE**, sub dividindo-se em:

- **Java Application**
- **Java Class Library**
- **Java Project withExistingSouces**
- **Java Project withExistingAnt Script**.

Essas opções servem para iniciar projetos de novas aplicações Java ou bibliotecas de classes. Também permite que novas aplicações sejam criadas, partindo de outras classes anteriormente criadas e que possam ser reaproveitadas, assim como projetos com scripts Ant pessoais já codificados.

A categoria Web possui os templates **Web Application**, **Web Project withExistingSources** e **Web Project withExistingAnt Script,** que representam, respectivamente, projetos novos para web, reaproveitando ou não códigos já escritos ou utilizando scripts Ant já criados.

A categoria Enterprise se refere a aplicações corporativas em que os templates **Enterprise Application** e **Enterprise ApplicationwithExistingSources** iniciam projetos usando, ou não, códigos já criados.

Os templates **EJB Module** e **EJB Module withExistingSources** para criação de componentes EJB com ou sem o uso de códigos já criados, além do template **EJB Module withExistingAnt Script** para projetos existentes com scripts previamente codificados.

Finalmente, a categoria **Samples**, que contém exemplos prontos de projetos Java que podem ser modificados livremente para auxílio ao estudo ou desenvolvimento profissional, sendo todos de código aberto.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

PERSISTÊNCIA EM JAVA USANDO JPA

JPA é um framework utilizado para persistência em bancos de dados relacionais. Servindo tanto para aplicações desktop ou web, a biblioteca possui bons recursos para lidar com o conceito (DEITEL, 2010).

Persistência se refere ao processo de manipulação de objetos Java comuns, para que sejam utilizados em bancos de dados **SQL**, reduzindo o trabalho com programação de scripts SQL.

Pela JPA, as chamadas entidades relativas ao conceito de banco de dados recebem o nome de Classes de Entidades, com atributos responsáveis pelas relações com o banco de dados.

Uma Classe de Entidade deve possuir chave primária, conter o mesmo nome da tabela ao qual se refere no banco de dados (ou a classe ser anotada com @),

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

possuir construtores sem parâmetros, entre outras características mais específicas, como implementar interface do tipo Serializable.

Uma entidade equivale a um objeto e, pela persistência, é possível a gravação de dados no banco por meio do chamado Mapeamento objeto-relacional, sendo esses objetos chamados de **POJOs** (Plain Old Java Objects) (DEITEL, 2010).

POJOs são objetos Java simples. O JavaBean se enquadra nessa definição, tendo definições e estrutura simples e correta, usando método construtor sem parâmetros e com métodos getters e setters para os atributos.

Por meio do Mapeamento objeto-relacional, as tabelas do banco de dados são representadas por meio de classes, e os registos em cada tabela são representados como instâncias de classes.

Assim, ao invés de se utilizar comandos SQL, uma interface é utilizada, não tendo correspondência direta entre tabelas armazenadas no banco de dados e classes em linguagem Java.

São necessários metadados para o uso da persistência. Metadados podem ser criados por meio de arquivos **XML** ou **Java annotations**. Escolhendo annotations, é possível verificar se uma classe é persistente ou se os métodos são colunas de uma determinada tabela.

O código11 traz um exemplo de criação de classe usando persistência, em que uma pequena agenda de contatos ilustra o uso da técnica pela criação de uma entidade **AGENDA** implementada como serializável, permitindo, além da manipulação de dados, a gravação em arquivo texto simples de forma organizada para compor o banco de dados.

No exemplo a seguir temos a declaração inicial da classe como serializável e a indicação do nome para a entidade, para, depois, iniciar a codificação dos métodos **getter** e **setter** para manipulação dos atributos da classe:

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

//código11

```
import javax.persistence.*;
import java.io.Serializable;
@Entity(name = "AGENDA") // Entidade
public class Agenda implements Serializable
{
        private long id;
        private String nome;
        private String endereço;
        private long telefone;

        // Métodos GETTER e SETTER para a classe
        public String getId() {
               return id;
        }
         public void setId(long id) {
                this.id = id;
        }
         public String getNome() {
                return nome;
         }
         public void setNome(String nome) {
                this.nome = nome;
         }
         public String getEndereco() {
                return endereco;
         }
         public void setEndereco(String endereco) {
                this.endereco = endereco;
         }
         public String getTelefone() {
                return telefone;
         }
```

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

```
        public void setTelefone(String telefone) {
                this.telefone = telefone;
        }
}
```

Algumas considerações finais são cabíveis para complementar o estudo da persistência. Uma dessas considerações se refere ao fato de a JPA permitir que uma classe persistente herde métodos e atributos de classes não-persistentes, da mesma forma que classes persistentes podem herdar de outras persistentes, e não-persistentes herdem de persistentes.

Não é possível a herança entre classes persistentes criadas e classes padrões do pacote JDK, como **java.net.Socket** e **java.lang.Thread**, além de que atributos de uma superclasse não persistentes herdados por uma classe persistente não possam ser também persistentes.

Maiores informações podem ser consultadas na documentação oficial fornecida pela Oracle em seu site oficial, nas áreas dedicadas à linguagem Java e, mais especificamente, ao textos relativos à persistência.

SAIBA MAIS

Dentro do paradigma de programação orientado a objetos, temos que o grande foco fica na correta estruturação do problema, por meio da sua abstração, para a elaboração de classes aptas de atender os requisitos mínimos capazes de satisfazer clientes e usuários de uma aplicação.

Assim, desde pequenos a grandes problemas, as classes que representam as entidades do problema devem ser bem pensadas, de forma que contenham os atributos corretos com suas visibilidades ajustadas para garantir o encapsulamento adequado.

Além disso, métodos devem ser bem elaborados, para que o processamento de dados sejam eficiente e com o mínimo de erros possível, pois uma solução mal pensada pode gerar uma aplicação fraca e incapaz de atender às necessidades do usuário.

Outras regras como a herança, em que classes se comunicam entre si, por meio da passagem de mensagens entre os objetos e correta definição de tudo embutido em uma classe, já pode representar uma carga enorme de trabalho.

Fonte: o autor.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

CONSIDERAÇÕES FINAIS

Nesta unidade, foram vistos complementos para as demais unidades que podem contribuir para uma melhor preparação do profissional e mostrar alternativas para o desenvolvedor.

Primeiramente, foram citados conceitos relevantes que foram introduzidos na oitava versão da linguagem Java, que ampliará sua capacidade de produzir código otimizado e com funcionalidades adicionais.

Estas versão manteve a base todas as funcionalidades essenciais das versões anteriores sem que estas fossem alteradas significativamente, mas sendo melhoradas quando possível ou conveniente.

Uma das principais inovações desta versão da linguagem foi a introdução do uso de alguns fundamentos da programação, dentro do paradigma funcional que trabalha toda sua codificação baseada em funções.

Assim, unindo os dois paradigmas, é possível obter códigos extremamente eficientes, ideias para aplicações de alto desempenho e que muitas vezes ficam como software embarcado em diversos tipos de hardware.

Os chamados Javabeans também representam formas diferenciadas de desenvolvimento de componentes reutilizáveis, que conseguem unir objetos de forma a reduzir a quantidade de parâmetros a serem passados pela aplicação. Isso melhora a comunicação entre aplicação e agentes externos, mas com a ausência de construtores, a segurança do software é reduzida.

Em seguida, temos informações a respeito do uso de uma IDE bem popular, para mostrar algumas funcionalidades existentes no uso de um ambiente integrado para o desenvolvimento, como assistentes de codificação e a não necessidade de interação com o Prompt de Comandos do sistema operacional.

Finalizando esta unidade, são introduzidos os conceitos de persistência para aplicações Java, visando a conexão com bancos de dados de forma segura e consistente.

Reprodução proibida. Art. 184 do Código Penal e Lei 9.610 de 19 de fevereiro de 1998.

ATIVIDADES

1. Com as inovações da linguagem Java, chegamos a oitava versão. **Qual o nome genérico dado às funções trazidas do paradigma funcional para a oitava versão da linguagem Java?**

2. Complementando os exemplos da unidade, **dê um exemplo de função do paradigma funcional em Java 8 como as utilizadas na unidade.**

3. Os Javabeans são recursos interessantes da programação Java. **Cite uma característica negativa de um componente Javabean.**

4. Mantendo o foco no uso de Javabeans na linguagem Java, **como se pode utilizar um componente Javabean criado?**

5. Finalizando as atividades baseadas em Javabean, **cite benefícios do uso de IDEs como o Netbeans.**

Uma versão atualizada do código exemplo da classe é mostrado no código12, contendo comentários posteriores para melhor compreensão da persistência:

//código12

```
import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
@Entity(name = "AGENDA")        // Entidade
public class Agenda implements Serializable
{
  @Id
  @Column (name="ID", nullable = false)
  @GenerateValue (strategy = GenerationType.AUTO)
  private long id;
  @Column (name="NOME", nullable = false, length = 50);
  private String nome;
  @Column (name="ENDERECO", nullable = false, length = 100);
  private String endereco;
  @Column (name="TELEFONE", nullable = false, length = 20);
  private long telefone;
  @Versao
  @Column(name = "ATUALIZADO")
  private Date atualizado;

  // Métodos GETTER e SETTER para a classe
  public String getId() {
         return id;
  }
  public void setId(long id) {
         this.id = id;
  }
```

```
    public String getNome() {
              return nome;
    }
    public void setNome(String nome) {
              this.nome = nome;
    }
    public String getEndereco() {
              return endereco;
    }
    public void setEndereco(String endereco) {
              this.endereco = endereco;
    }
    public String getTelefone() {
              return telefone;
    }
    public void setTelefone(String telefone) {
              this.telefone = telefone;
    }
}
```

Todas as anotações usadas são definidas na API **javax.persistence**, possibilitando a definição da entidade **AGENDA**, já indicada no código anterior de número 11. Uma observação importante é a existência de uma anotação chamada @Table, que é usada no caso do uso de nomes diferentes entre a classe e a entidade.

Nesse exemplo do código12, cada anotação inclusa tem uma função específica, como o caso de **@ID**, que indica que o atributo a seguir representa a chave primária da tabela, lembrando que uma chave não pode conter dados repetidos, garantidos pela anotação **@GenerateValue**, que contém o parâmetro **strategy = GenerationType.AUTO**, o qual gera valores sequenciais automaticamente para evitar repetições.

Outro argumento importante utilizado em outros pontos do código é **nullable = false**, o qual impede que atributos sejam deixados em branco, aumentando a confiabilidade das informações do banco de dados.

Em seguida, **@Column** nomeia uma coluna da tabela pelo atributo name, caso os nomes do atributo e da coluna sejam diferentes por desejo dos envolvidos na elaboração do banco de dados e da aplicação.

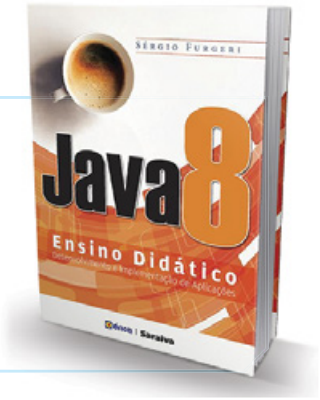
A anotação @Version armazena um valor indicativo de versão dos dados, para efeito de controle de alterações realizadas nos valores da tabela, permitindo principalmente que a aplicação possa detectar acessos múltiplos não permitidos a uma mesma versão de um registro de dados da tabela, indicando algum tipo de exceção ocorrida ao usuário e que pode ser tratada pela aplicação internamente.

Fonte: adaptada de Deitel (2010).

MATERIAL COMPLEMENTAR

LIVRO

Java 8. Ensino Didático. Desenvolvimento e Implementação de Aplicações
Autor: Sérgio Furgeri
Editora: Érica
Sinopse: livro especificamente voltado para a versão 8 da linguagem Java, trazendo suas principais inovações aliadas aos conceitos básicos da linguagem.



BARNES, D. J.; KÖLLING, M. **Programação orientada a objetos com Java**. 4. Ed. São Paulo: Pearson Prentice Hall, 2009.

DEITEL, P. J. **Java**: como programar. 8. Ed. São Paulo: Pearson Prentice Hall, 2010.

REFERÊNCIAS ON-LINE

[1] Em: <http://education.oracle.com/pls/web_prod-plq-dad/ou_product_category.getPage?p_cat_id=267>. Acesso em: 2 jun. 2017.

GABARITO

1) Lambda.

2) () ->System a.out.println (“Texto Exemplo”);

3) Falta de métodos construtores.

4) Por meio de código HTML.

5) Facilidades na codificação e auxílio na compilação e execução da aplicação.

CONCLUSÃO

A linguagem Java se encontra entre as três linguagens mais utilizadas do mercado, se mantendo forte dentro da comunidade de desenvolvedores de software, com constantes adaptações, inclusive eu nicho de atuação.

Algumas atividades antes comuns no desenvolvimento em Java, como plug-ins para navegadores, já não representa mais o foco do desenvolvimento.

Vale a pena se manter atualizado e buscar novas formas de desenvolvimento de software, em diferentes frentes que estejam em alta no mercado, para não perder espaço que pode ser difícil recuperar dentro do mercado.

O desenvolvimento de aplicações web ou aplicativos para dispositivos móveis ainda representa uma boa opção de trabalho, mas surgem outras formas de desenvolvimento de software em Java que têm conseguido muitos profissionais para trabalhar. Um exemplo disso é o desenvolvimento orientado a serviços e componentes.

Manter-se focado em novas tecnologias e técnicas de trabalho com o desenvolvimento de software é muito importante para aqueles que desejam se tornar desenvolvedores, além de engenheiros de software.

Por meio das unidades, foram vistos importantes conceitos como os relacionados ao paradigma orientado a objetos, contemplando classes, métodos e atributos, assim como encapsulamento, herança e polimorfismo.

Também foram vistas estruturas de desvio de fluxo da execução e a elaboração de laços de repetição, contendo diferentes tipos de instruções com variações de uso possíveis.

Outro aspecto fundamental do material é o uso de bibliotecas diversas, facilitando o trabalho de desenvolvimento de aplicações em Java para as mais diversas finalidades.

Desse modo, entendemos que a programação em linguagem Java abre um leque adicional a tudo o que foi estudado em disciplinas anteriores, permitindo a ligação com disciplinas que tratam da modelagem de software, além da comparação da programação no paradigma mais tradicional estruturado, estudado com a linguagem de programação C.