Programação Java

com Ênfase em Orientação a Objetos

Douglas Rocha Mendes

Novatec

CAPÍTULO 1

Introdução à linguagem Java

O capítulo 1 inclui o histórico da linguagem Java, detalhes que formam a linguagem, a plataforma Java, uma breve descrição sobre orientação a objetos, operadores matemáticos, tipos de dados primitivos e, por fim, as estruturas de controle e repetição. No final do capítulo está disponível uma lista de exercícios e o laboratório 1, que exercita a teoria apresentada no capítulo. Esse laboratório será progressivo e se estenderá até o capítulo 9.

1.1 Histórico da linguagem

A tecnologia Java foi criada como uma ferramenta de programação de um projeto da Sun Microsystems, chamado The Green Project, iniciado por Patrick Naughton, Mike Sheridan e James Gosling, em 1991. Esse projeto tinha como principal objetivo criar uma nova plataforma para a computação interativa, ou seja, a linguagem de programação não era o principal objetivo do projeto. No verão de 1992 foi gerada a primeira demonstração do projeto, que representou um sistema executando em um handheld com capacidade de controle remoto que ainda oferecia uma interface sensível ao toque (touchscreen) interativa. Esse handheld foi chamado de *7 (star seven), por esta ser a forma de atender (ou puxar) chamadas telefônicas entre os telefones dos integrantes da equipe. A figura 1.1 apresenta um exemplo da aparência do *7.

O *7 foi capaz de controlar uma grande variedade de dispositivos de uso doméstico, enquanto apresentava uma interface com animação. O sistema criado para o handheld foi executado em um novo processador independente de linguagem de programação. A linguagem utilizada nesse sistema foi chamada de Oak ("carvalho", em inglês), com base na paisagem que James Gosling tinha de sua janela.

Figura 1.1 – Handheld *7(Star seven).

Em 23 de maio de 1995, John Gage, diretor da Sun Microsystems, e Marc Andreessen, um executivo da Netscape, anunciaram o lançamento da plataforma Java, composta pela Java Virtual Machine (JVM) e pela API (Application Programming Interface) Java. Tal plataforma foi, então, inserida no Netscape Navigator, o principal browser de acesso à Internet usado na época.

1.2 Características da linguagem de programação Java

A linguagem de programação Java representa uma linguagem simples, orientada a objetos, multithread, interpretada, neutra de arquitetura, portável, robusta, segura e que oferece alto desempenho. É importante observar que a tecnologia Java é composta de uma linguagem de programação e de uma plataforma (API e a máquina virtual). A seguir, cada uma das características citadas é descrita.

1.2.1 Simples

A linguagem Java é considerada simples porque permite o desenvolvimento de sistemas em diferentes sistemas operacionais e arquiteturas de hardware, sem que o programador tenha que se preocupar com detalhes de infra-estrutura. Dessa forma, o programador consegue desempenhar seu trabalho de uma forma mais produtiva e eficiente.

Pensando em tornar a linguagem o mais simples possível, os projetistas da linguagem Java optaram por não implementar o uso do conceito de herança múltipla, de sobrecarga de operadores, ponteiros nem a operação aritmética com esse tipo de dado. Essas características podem ser encontradas em outras linguagens, como C ou C++.

1.2.2 Orientada a objetos

A linguagem Java foi criada seguindo o paradigma da orientação a objetos e, por isso, traz de forma nativa a possibilidade de o programador usar os conceitos de herança, polimorfismo e encapsulamento. O paradigma da orientação a objetos existe desde a década de 70, mas somente após o sucesso da linguagem Java é que o paradigma ganhou credibilidade. O paradigma de orientação a objetos traz um enfoque diferente da programação estruturada, no sentido de adotar formas mais próximas do mecanismo humano para gerenciar a complexidade de um sistema. Nesse paradigma, o mundo real é visto como sendo constituído de objetos autônomos, concorrentes, que interagem entre si, e cada objeto tem seu próprio estado (atributos) e comportamento (métodos), semelhante a seu correspondente no mundo real.

Quando desenvolvemos programas orientados a objetos e estruturados temos dois paradigmas totalmente diferentes. A forma de pensar e escrever o código são diferentes. É importante observar que muitos programadores usam a linguagem Java, mas continuam pensando no formato estruturado. Essa má prática de programação é muito freqüente em estudantes e profissionais que utilizam Java diariamente. Este livro objetiva minimizar esta má prática do uso da linguagem Java. A seguir é apresentada uma comparação entre o desenvolvimento de um sistema bancário usando o paradigma da programação estruturada e o paradigma da programação orientada a objetos.

1.2.2.1 Comparação entre o paradigma da programação estruturada e paradigma da orientação a objetos

Considere o exemplo de um sistema bancário com os seguintes requisitos funcionais:

- Manter o cliente (envolvendo inserir cliente, remover cliente, selecionar cliente e atualizar cliente).
- Manter conta (envolvendo abrir conta, fechar conta, alterar conta, pesquisar conta e remover conta).
- Movimentar caixa (envolvendo sacar, depositar e transferir).
- Registrar movimento (envolvendo gerar histórico).
- Emitir relatórios contábeis.

No paradigma utilizado pelo modelo estruturado seria necessário criar um programa para manter clientes, um programa para manter conta, um programa para movimentar caixa e um programa para emitir os relatórios. Cada programa deverá

ser executado por um programa principal, ou seja, um programa com o método `main()`. Cada programa e também o programa principal têm um conjunto de variáveis específicas para suas necessidades. Cada variável que precisar ser compartilhada com outros programas deve ser criada como global e, se estivermos usando a linguagem C, esta deveria ser externalizada (definir a variável como `extern` em outro programa) para outros programas. Um ambiente estruturado nos limita apenas a identificar os programas que serão envolvidos na implementação do sistema.

Neste modelo de programação, a reutilização é pequena, e a redundância é grande. No paradigma procedural, quando existe alguma modificação, muitas vezes pelo alto acoplamento entre os programas, é necessário alterar vários programas e novamente testar todo o sistema. No modelo estruturado ficamos limitados a identificar os programas necessários e, no máximo, a criar funções que possam ser reutilizadas em outros programas. No paradigma da orientação a objetos temos outras possibilidades, tais como o uso de encapsulamento para oferecer segurança à classe, herança que permite a reutilização de código, polimorfismo e padrões de projeto, todos explorados de forma detalhada neste livro.

No caso do paradigma da orientação a objetos, inicialmente precisamos identificar nossas classes com seus atributos e métodos. Como técnica para identificar classes temos a análise dos substantivos e locuções substantivas (dois substantivos juntos), listados pelos requisitos funcionais (Use Cases). Assim, para o exemplo apresentado, teríamos as classes Conta, Pessoa, Cliente, Gerente de Relacionamento, Contabilidade, Movimentação e Histórico, entre outras. Percebam que nesta abordagem identificamos os substantivos e locuções substantivas descritos pelos requisitos funcionais.

É importante observar que no paradigma da programação orientada a objetos não nos preocupamos em definir quantos programas seriam necessários e sim quais seriam os substantivos utilizados pelo sistema. Cada substantivo representa um objeto no mundo real e uma classe em nosso sistema. Cada classe seria representada por um novo programa Java. Outra vantagem do paradigma da orientação a objetos é a possibilidade de usar uma ferramenta de modelagem para a geração de código.

1.2.3 Multithread

A plataforma Java permite a criação de programas que implementam o conceito multithread, incluindo sofisticados mecanismos de sincronização entre processos. O multithreading é uma técnica de programação concorrente, que permite projetar e implementar aplicações paralelas de forma eficiente.

1.2.4 Interpretada

A linguagem Java é interpretada, ou seja, após a compilação é gerado um arquivo intermediário (nem texto nem executável) no formato bytecode, que poderá ser executado em qualquer arquitetura (Windows, Linux, Mac e Unix) que tenha uma máquina virtual Java instalada. A linkedição do programa no formato bytecode é realizada no momento de sua execução de forma simples e totalmente gerenciada pela JVM (Java Virtual Machine).

1.2.5 Independência de arquitetura

A linguagem Java está projetada para dar suporte a sistemas que serão implementados em plataformas heterogêneas (hardware e software), como ambiente Unix, Linux e Mainframe. Nesses ambientes, o sistema deve ser capaz de ser executado em diferentes hardwares, como servidor Unix da HP ou servidor Unix da IBM. Para acomodar essa situação de interoperabilidade, o compilador Java gera os programas em um formato conhecido por bytecode (um formato intermediário de código projetado para permitir que múltiplos hardwares e softwares executem o mesmo código), permitindo que um programa Java seja executado em qualquer arquitetura.

1.2.6 Portabilidade

O que garante a portabilidade dos programas desenvolvidos em Java é a Máquina Virtual Java (Java Virtual Machine – JVM). Trata-se de uma especificação na qual o compilador Java de cada plataforma irá se basear para gerar o código em bytecode.

Na linguagem de programação Java, todo código-fonte deve ser primeiramente escrito em um arquivo no formato de texto e ser gravado em um arquivo com a extensão `.java`. Após a compilação desse arquivo pelo compilador (`javac.exe`), um novo arquivo será automaticamente criado com o mesmo nome do arquivo-fonte, entretanto, com a extensão `.class`. O arquivo com extensão `.class` gerado representa um arquivo no formato bytecode e poderá ser executado em qualquer plataforma que tiver uma JVM instalada. Uma vez gerado o arquivo no formato bytecode, será possível executar o programa com o arquivo `java.exe`. O programa `java.exe` executa um programa Java como uma instância da JVM. A figura 1.2 apresenta as etapas para compilação e execução de um programa Java.

Na figura 1.2 há um programa Java sendo submetido ao compilador que irá gerar um arquivo bytecode com a extensão `.class` independente da arquitetura do sistema operacional utilizado. Esse `.class` poderá ser executado por qualquer JVM em qualquer ambiente operacional.

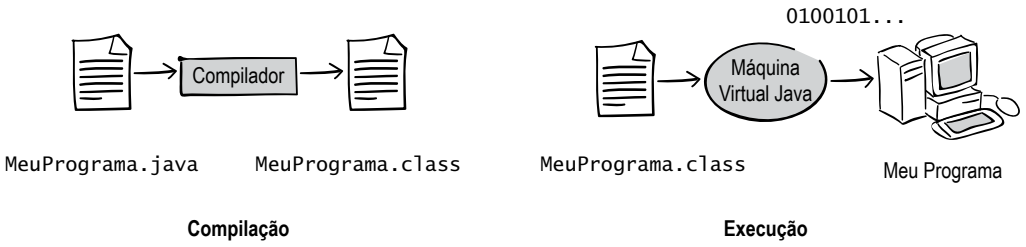


Figura 1.2 – Etapas para compilação e execução de um programa Java.

Devido à JVM estar disponível em inúmeras plataformas, o mesmo arquivo com extensão `.class` poderá ser executado em diferentes sistemas operacionais, como Windows, Solaris, Linux ou Mac OS. A figura 1.3 apresenta um ambiente onde o mesmo programa `.class` poderia estar em execução em múltiplos sistema operacionais.

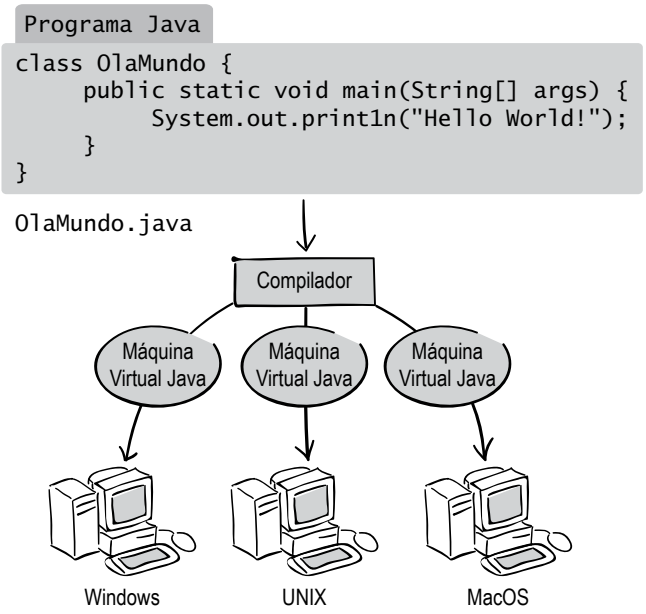


Figura 1.3 – Etapas para compilação e execução de um programa Java.

1.2.7 Alto desempenho

O desempenho sempre foi um fator de comparação entre a linguagem Java, que é interpretada, e as linguagens compiladas. A plataforma Java oferece um bom desempenho, pois executa um código que foi previamente analisado e convertido para um formato intermediário. Outro elemento que auxilia no bom desempenho é o recurso de *garbage collector* (coletor de lixo), que é executado em segundo plano

como uma thread com baixa prioridade, procurando liberar memória que não está sendo mais utilizada. Isso faz com que a memória liberada seja reutilizada por outra parte do sistema, gerando um bom desempenho do sistema.

A linguagem Java também permite que um programa seja compilado em uma plataforma (Windows, Linux ou Unix) específica. Entretanto, nesse caso o programa não poderá ser portado em outra plataforma automaticamente, exigindo a necessidade de uma nova compilação.

1.2.8 Robusta

A linguagem Java foi projetada para gerar sistemas confiáveis, pois fornece já em tempo de compilação, por exemplo, uma checagem para identificar código não-alcançável. Entenda-se como código não-alcançável uma linha de código que por algum motivo na lógica de programação nunca será executada. Como exemplo podemos ter um comando `return` e logo abaixo a impressão de uma string. Nesse caso, a string nunca seria impressa, por isso o compilador java gera um erro. A linguagem java também oferece uma checagem para identificar variáveis que foram definidas, porém não foram inicializadas. O modelo de gerenciamento de memória é extremamente simples, sendo que após a alocação de memória por meio do operador `new` não é necessário que o programador libere esse espaço alocado, pois o garbage collector realiza essa atividade.

Outros fatores que contribuem para o rótulo de linguagem robusta são:

- Sistema de tipo rígido que, em tempo de compilação, verifica erros comuns, como `if (cpf = 1)`. Nesse caso estamos atribuindo um valor, e não realizando a comparação. O correto seria apresentar o comando `if (cpf == 1)`.
- Inicialização das variáveis e atributos inteiros com o valor 0 automaticamente ou com vazio no caso de variáveis ou atributos do tipo string.
- Manipulação de exceções é parte integrante da linguagem.

1.2.9 Segura

A linguagem Java foi criada para operar em ambientes distribuídos, o que significa que segurança é de extrema importância. Com as características projetadas na linguagem Java, e principalmente na JVM, podemos garantir que em um ambiente de rede nenhum programa Java permitirá que outro programa escrito em qualquer outra linguagem possa se esconder em um código Java a fim de se instalar automaticamente.

1.3 Plataforma Java

O termo genérico plataforma representa o hardware e o software onde um programa Java pode ser executado. Como exemplo de plataformas temos: o Windows, HPUX – Unix da HP, AIX – Unix da IBM, Linux, Solaris e Mac OS. Muitas plataformas podem ser descritas como uma combinação do sistema operacional e do hardware que oferece o suporte. No caso do Java o termo plataforma refere-se somente ao software onde são executados os programas Java. A plataforma Java é composta de Java Virtual Machine (JVM) e Java Application Programming Interface (API).

A JVM representa a base da plataforma Java e pode ser instalada na maioria dos sistemas operacionais disponíveis no mercado. A API Java representa uma grande coleção de classes prontas que fornecem uma grande quantidade de facilidades ao programador. Esse conjunto de classes deve ser instalado no computador por meio do download do J2SDK na versão desejada. Tais classes são agrupadas em bibliotecas conhecidas como pacotes. Conforme apresentado na figura 1.4 vemos que a plataforma Java corresponde a uma API (conjunto de classes) e uma máquina virtual, que é responsável por executar um programa Java, ficando disponível ao usuário após a instalação no computador do J2SDK na versão escolhida.

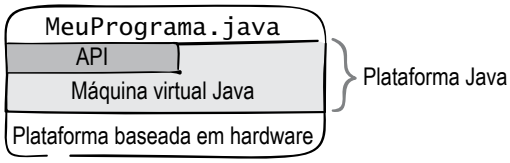


Figura 1.4 – Plataforma Java.

1.4 Por que usar Java?

A plataforma Java oferece aos programadores e analistas de sistemas um conjunto completo de classes para o desenvolvimento de sistemas web (Servlet, JSP, JavaServer Faces (JSF)), desktop (Swing, SWT) e batch (método `main()`). Com essas classes, o tempo para o desenvolvimento é reduzido e a qualidade do sistema fica muito melhor.

Aprender a programar em Java pode ser uma tarefa fácil para quem já conhece as linguagens C ou C++ e domina o paradigma da orientação a objetos. Programadores experientes nessas linguagens percebem que em Java a quantidade de código que deve ser construído é menor, bem como a legibilidade e qualidade são superiores. O conceito de *write once* (escreva uma vez), *run anywhere* (execute em qualquer lugar) pode realmente ser alcançado com programas escritos Java. É importante observar

que, para obter as vantagens da linguagem Java, precisamos conhecer muito bem o paradigma da orientação a objetos, caso contrário o programador continuará desenvolvendo programas com a linguagem Java em formato estruturado.

1.5 Meu primeiro programa Java

Antes de começar a escrever um programa Java é necessário garantir que esteja instalado no sistema operacional o Java SE Development Kit (J2SDK) na versão desejada. Caso não esteja, podemos realizar o download do site da Sun e realizar a instalação. Neste livro, utilizamos o J2SDK na versão 1.5 para a compilação e execução de nossos exemplos. Como segundo ponto importante, é preciso definir se esse programa será desenvolvido em uma IDE (Integrated Development Environment) ou iremos usar um bloco de notas oferecido pelo sistema operacional. Essa segunda opção torna o desenvolvimento um pouco mais trabalhoso. Neste livro, usamos a IDE do Eclipse, por tratar-se de uma IDE gratuita e estar presente nas principais empresas de desenvolvimento de sistemas e instituições de ensino.

1.5.1 Dicas para o uso da IDE Eclipse

É importante observar que antes da execução do Eclipse precisamos garantir que a versão 1.5 do J2SE ou superior já tenha sido instalada no computador. Alguns métodos e classes usados nos exemplos deste livro fazem referência a essa versão. Caso o leitor use uma versão anterior ao J2SE 1.5, teremos alguns exemplos apresentando erro de compilação.

Quando executamos o ícone do Eclipse é necessário informar onde fica nossa área de trabalho (workspace). Essa área é um diretório no sistema operacional, que contém todos os nossos programas. Depois de definido onde ficará a workspace, é necessário criar um projeto na IDE, por meio do menu de opções **File > New > Project > Java Project**. Podemos dar o nome de `livroJava` ou qualquer outro nome para nosso projeto. Lembre-se de que esse nome de projeto também será representado no sistema operacional como um diretório.

A partir desse momento devemos clicar em cima do nome do projeto e criar um pacote que será também representado no sistema operacional como um diretório. Em seguida será possível criar uma classe que será representada no sistema operacional por um arquivo com a extensão `.java`. O nome da classe escolhida deverá ser igual ao usado no programa 01.01. O nome da classe `HelloWorld` deve ser igual ao nome do arquivo `.java` criado dentro do diretório que representa o pacote, respeitando letras maiúsculas e minúsculas.

É importante observar que para executar o programa 01.01 sem erros precisamos passar dois parâmetros no momento da execução. Temos duas opções para a execução de um programa java, sendo a primeira a utilização do comando

```
java -cp . modulo01.HelloWorld 10 20
```

ou por meio da IDE Eclipse em **Run > Run > Arguments > Program Arguments.** Visualize o espaço em branco e coloque os dois parâmetros no espaço vazio separados por espaço. A seguir apresentamos detalhadamente o que cada linha desse programa representa e como devemos proceder para sua execução.

Programa 01.01

```
package modulo01;
1   // Imprime na tela os argumentos recebidos e a frase Hello World
2   public class HelloWorld {
3       public static void main(String args[]) {
4       System.out.println("Parâmetro 01: " + args [0]);
5       System.out.println("Parâmetro 02: " + args [1]);
6       System.out.println("Hello World");
    }
}
```

A seguir, descreveremos o que cada parte do programa representa. A linha 1 representa um comentário em Java. Os comentários são ignorados pelo compilador, mas podem ajudar outros programadores a entender o propósito de seu programa. Há dois tipos de comentários em Java. O primeiro é representado por `/*` e `*/`. Esse tipo de comentário oferece a possibilidade de comentar um bloco de informações, que pode se estender por várias linhas. O segundo formato disponível é representado por `//` e tem como objetivo comentar apenas uma linha do programa.

A linha 2 representa a definição da classe, que deve ser sempre seguida por `{`. Em seguida, podemos criar métodos e atributos. Para cada método devemos ter um `{` e, no final de cada método, devemos ter um `}` que formaliza o final do método. Também no final da classe precisamos de um `},` para formalizar que a classe se encerra nesse ponto. Todo o código da classe deve ficar entre o `{` e o `}`. Qualquer código colocado fora desses limites apresentará erro na compilação. Existem algumas exceções que representam códigos que devem ser definidos fora desse limite. São citados a seguir:

- Código de inicialização estático ou não-estático.
- Comando `import`.
- Comando `package`.

O capítulo 2 apresenta os comandos `import` e `package`. O capítulo 3 descreve de forma detalhada o código de inicialização.

A linha 3 apresenta a definição do método `main()`. Todo programa desenvolvido para desktop ou batch (programas Java executados sem interatividade com o usuário) deve ter o método `main()` definido.

A JVM inicia a execução do programa a partir desse método. A palavra reservada `public` define que o método pode ser chamado por qualquer outro objeto e, nesse caso, quem o faz é a JVM. A palavra reservada `static` define que o método poderá ser executado sem a necessidade de criação de um objeto da classe. Um objeto representa o modelo da classe em tempo de execução, e este livro realiza uma abordagem completa desse conceito. A palavra reservada `void` informa o tipo de retorno do método que, no caso do método `main()`, deve ser sempre `void`. A JVM não está preparada para receber valores de retorno do programa executado. Em seguida, há o nome do método e o parâmetro que recebe quando executado.

O método `main()` aceita um parâmetro do tipo vetor de strings. Cada string desse array pode representar um parâmetro passado na execução do programa. Dessa forma, é possível mudar o fluxo de execução do programa considerando os parâmetros recebidos por meio desse parâmetro. Para usar esses parâmetros o programa Java deve ser executado da seguinte forma, em uma linha de comando:

```
D:\workspace\livro>java -cp . modulo01.HelloWorld 10 20
Parâmetro 01: 10
Parâmetro 02: 20
Hello World
```

É importante observar que antes de executar o comando

```
java -cp . modulo01.HelloWorld 10 20
```

devemos nos posicionar no diretório do projeto e, em seguida, executar o comando apresentado. Neste exemplo temos o diretório `D:\workspace\livro>`. Pode ser que esse diretório seja diferente, dependendo do ambiente do leitor. Outro ponto importante é o diretório que representa o nome do pacote. No diretório do pacote (neste exemplo, `modulo01`) utilizado no comando deve estar gravado o arquivo com o nome da classe seguido da extensão `.class` (`HelloWorld.class`). Caso contrário, um erro será apresentado, e o programa não será executado.

O parâmetro `10` estará disponível no programa para ser acessado por meio da posição 0 do vetor `args (args[0])`, enquanto o parâmetro `20` estará disponível para ser acessado por meio da posição 1 do vetor `args (args[1])`, conforme as linhas 4 e 5 do programa 01.01.

Nas linhas 4, 5 e 6 estamos realizando a impressão dos valores dos parâmetros recebidos e também da frase "`Hello World`", por meio do comando `System.out.println`. Em que `System` representa o nome da classe, `out` representa um atributo estático do tipo da classe `PrintStream` e `println()` representa um método do objeto `out`. O capítulo 8 apresenta detalhadamente a classe `PrintStream`.

O centro de qualquer aplicação Java é seu método `main()`, pois será a partir dele que todos os outros métodos requeridos para executar uma aplicação serão executados. Se o interpretador por meio da JVM não encontrar o método `main()`, o programa não será executado.

1.6 Introdução à orientação a objetos

Muitos alunos que começam a estudar a linguagem Java têm grande dificuldade para entender a diferença entre uma classe e um objeto, bem como qual o real significado de um atributo e de um método. Para esclarecer tais dúvidas vamos comparar os elementos contidos em uma classe com os elementos contidos no ambiente de um banco de dados. Não que uma classe e uma tabela sejam iguais, apenas usaremos essa comparação como um meio para que o leitor entenda o significado desses dois importantes conceitos.

Podemos dizer que toda tabela de banco de dados tem um nome, assim como uma classe também. No caso da classe, ela deve iniciar com letra maiúscula e ser um substantivo ou locução substantiva (dois substantivos juntos). A tabela 1.1 apresenta um exemplo de nomes para uma tabela e uma classe.

Tabela 1.1 – Relação entre nome de tabela e classe

CONTA_CORRENTE	ContaCorrente
APOLICE_SEGURO	ApoliceSeguro
VEICULO	Veiculo

Uma tabela de banco de dados tem colunas, que representam a estrutura da tabela. Uma classe tem o mesmo conceito, mas o chamamos de atributos, logo, uma coluna de uma tabela representa um atributo de uma classe. É importante observar que o atributo de uma classe deve começar com letra minúscula e, se for composto, as próximas letras devem ser maiúsculas (recomendação que faz parte das práticas recomendadas de programação). A tabela 1.2 apresenta exemplos de nomes usados para colunas e atributos.

Tabela 1.2 – Exemplo de nomes de colunas e atributos

Tabela	Classe
nome_cliente	nomeCliente
saldo_conta	saldoConta
numero_conta	numeroConta
numero_agencia	numeroAgencia

Uma tabela de banco de dados pode conter operações que venham a manipular suas colunas a fim de gerar resultados que possam ser atualizados em outra tabela ou gravados em um arquivo texto no formato de um relatório. Essas operações que manipulam as colunas de uma tabela são conhecidas por *stored procedures*, e uma tabela de banco de dados pode conter nenhuma ou várias dessas operações. Os métodos em uma classe representam as operações que a classe contém. Tais operações usam os atributos da classe, seja para alteração ou apenas para leitura, e processam e geram resultados que podem ser usados para a geração de um relatório.

Como boa prática de programação Java, podemos definir o nome de um método Java começando com letra minúscula e sendo um verbo no infinitivo (terminado em *ar*, *er*, *ir*) seguido de um substantivo. Como exemplo de nomes de métodos podemos ter: `calcularPedido()`, `arquivarDocumento()`, `imprimirFichaCadastral()`, `efetuarSaque()`, `efetuarDeposito()`, entre outros. A tabela 1.3 apresenta exemplos de nomes usados para stored procedures e métodos.

Tabela 1.3 – Exemplo de nomes de stored procedures e métodos

Tabela	Classe
sp_recuperar_cliente_por_cpf(int cpf)	recuperarClienteporCPF (int cpf)
sp_recuperar_cliente_por_nome (varchar nome)	recuperarClienteporNome (String nome)
sp_atualizar_saldo (decimal valor)	atualizarSaldo (double valor)
sp_remover_tansacoes (int cpf)	removerTansacoes (int cpf)

Não podemos esquecer que uma tabela contém linhas, e estas representam os registros gravados na tabela. Podemos ter milhares de linhas gravadas em uma tabela sendo que o limite seria a capacidade do disco rígido do servidor de banco de dados.

Em um ambiente orientado a objetos essas linhas das tabelas representarão nossos objetos quando o programa estiver em execução, ou seja, quando iniciar sua execução, o programador poderá selecionar uma ou mais linhas da tabela e, para tratá-las em memória, o programa Java atribuirá a elas um objeto do tipo da classe correspondente. Esse objeto representa o modelo da classe no qual ele está

baseado. Na tabela 1.4 cada linha da tabela corresponde a um registro da tabela e, quando tratado pelo programa Java no ambiente de produção, será processado como um objeto.

Tabela 1.4 – Exemplo de registros e objetos

Tabela				Classe			
nome_cliente	saldo_conta	numero_conta	numero_agencia	nomeCliente	saldoConta	numeroConta	numeroAgencia
Mauricio Oliveira	1000	9876543	1996	Mauricio Oliveira	1000	9876543	1996
Carlos Chiarello	9000	1234531	0003	Carlos Chiarello	9000	1234531	0003
Daniela Freire	850	9087670	0057	Daniela Freire	850	9087670	0057
Domingos Lucio	8569	9878652	0015	Domingos Lucio	8569	9878652	0015
Hideson Alves	1234	7654321	0090	Hideson Alves	1234	7654321	0090

1.6.1 Classes em Java

Uma classe define o modelo de um objeto, ou seja, todas as características que o objeto contém foram definidas pela classe. É importante considerar que uma classe não representa nenhum objeto em particular, pois é só um modelo. O programa 01.02 apresenta como criar uma classe em Java. Não podemos esquecer que o nome da classe e o nome do arquivo `.java` devem ser exatamente iguais. Uma classe pode definir entre o abre e fecha chaves (`{` e `}`) somente atributos, métodos e códigos de inicialização.

Programa 01.02

```
package modulo01;
// Representação de uma classe com atributos e métodos
public class ContaCorrente {
    int conta;
    int agencia;
    double saldo;
    static double cpmf; // atributo estático
    String nome;
    public String getNome() {
        return this.nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
```

```
    void efetuarSaque(double valor) {
        this.saldo = this.saldo - valor;
    }
    void efetuarDeposito(double valor) {
        this.saldo = this.saldo + valor;
    }
    void imprimirSaldo() {
        System.out.println(this.saldo);
    }
    public int getAgencia() {
        return this.agencia;
    }
    public void setAgencia(int agencia) {
        this.agencia = agencia;
    }
    public int getConta() {
        return this.conta;
    }
    public void setConta(int conta) {
        this.conta = conta;
    }
    public double getCpmf() {
        return cpmf;
    }
    public void setCpmf(double cpmf) {
        ContaCorrente.cpmf = cpmf;
    }
    public double getSaldo() {
        return this.saldo;
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
}   // Nenhuma linha de código poderá ser definida abaixo deste símbolo de fecha chaves
```

A classe do programa 01.02 representa uma conta corrente com seus atributos, métodos de acesso (`getXXX()`) e métodos modificadores (`setXXX()`). Os métodos de acesso e os modificadores de uma classe devem ser usados para que o programador acesse ou altere seus atributos, sem a necessidade de fazê-lo de forma direta. Apesar de ser possível realizar o acesso ou alteração de forma direta, segundo as boas práticas de programação e o conceito de encapsulamento, um atributo deve ser sempre acessado por meio de seus métodos de acesso ou métodos modificadores. Outro elemento que também fortalece esse modo de programação é o uso de frameworks como o Spring, que exige que um atributo tenha seus métodos de acesso e seus métodos modificadores.

A classe `ContaCorrente` apresentada no programa 01.02 representa exatamente uma tabela do banco de dados, onde definimos suas colunas (atributos) para receberem registros. Em uma classe Java, diferentemente do ambiente de banco de dados, que

usa comandos SQL (`select`, `update`, `insert` etc.) e stored precedures para acessar suas colunas, para acessar ou alterar os atributos de uma classe, podemos usar os métodos de acesso (`getXXX()`) ou os métodos modificadores (`setXXX()`). Em um método modificador podemos definir regras de validação para os atributos, evitando que um valor inválido seja atribuído a um atributo.

Como exemplo de validação de um atributo, podemos especificar que a agência não pode ser inicializada com um parâmetro (`int agencia`) com valor menor ou igual a 0. Esse teste poderia ocorrer dentro do método `setAgencia (int agencia)`. Todos os programas que usarem a classe `ContaCorrente` automaticamente estarão se beneficiando dessa regra de validação. Caso o teste seja realizado em cada uma das classes Java que use este atributo, com certeza haverá redundância de código.

1.6.2 Objetos em Java

O termo objeto e instância são sinônimos, entretanto, este livro enfatiza o termo objeto. A criação de um objeto na linguagem Java envolve o operador `new` e pode ser feita por meio do comando

```
Classe variavel = new Classe();
```

O programa 01.03 apresenta a criação de objetos.

Programa 01.03

```
package modulo01;
// Criação de objetos em Java
public class Principal {
    public static void main(String args[]) {
        ContaCorrente objeto1 = new ContaCorrente();   // criando o objeto 1
        ContaCorrente objeto2 = new ContaCorrente();   // criando o objeto 2
    }
}
```

Esse programa está representando apenas como devemos criar um objeto. Se for executado não será exibido nenhum resultado na tela, pois seu objetivo foi simplesmente apresentar a sintaxe usada na linguagem Java para realizar a criação de objetos.

1.6.3 Mensagens em Java

Para usar um objeto após sua criação, devemos enviar mensagens. Uma mensagem é a forma de comunicação entre objetos. Para isso é preciso criar um objeto, identificar o nome do método a ser executado e, caso necessário, identificar também os

parâmetros que o método recebe ou retorna. Depois de ter esses dados em mãos podemos enviar mensagens usando o seguinte comando:

```
objeto1.efetuarSaque(100,00);
```

Neste exemplo estamos enviando uma mensagem para que o objeto `objeto1` efetue um saque de R$ 100,00 reais.

O programa 01.04 apresenta a classe `ExemploMensagem` realizando dois saques, sendo o primeiro por meio do `objeto1` e, o segundo, por meio do `objeto2`.

Programa 01.04

```
package modulo01;
// Envio de mensagens para o objeto
public class ExemploMensagem {
    public static void main(String args[]) {
        ContaCorrente objeto1 = new ContaCorrente();
        objeto1.efetuarSaque(41.80);
        ContaCorrente objeto2 = new ContaCorrente();
        objeto2.efetuarSaque(131.10);
    }
}
```

Esse programa apresenta a sintaxe para que o programador execute métodos em Java. Primeiramente precisamos criar um objeto com o operador `new` e, em seguida, podemos executar os métodos do objeto. A execução de um método em Java é semelhante à execução de uma sub-rotina COBOL ou uma função C.

1.6.4 Métodos em Java

Toda classe Java contém pelo menos um método chamado de método construtor (método com o mesmo nome da classe e sem parâmetros de entrada). Podem existir vários outros construtores e métodos, mas o método construtor sem parâmetros, se não for explicitamente criado pelo programador, será implicitamente criado pelo compilador Java sempre que nenhum outro tiver sido criado. Caso o programador crie um construtor diferente do construtor sem parâmetros, o compilador Java não mais criará esse construtor de forma automática. Neste caso o programador também deverá criar o construtor sem parâmetros explicitamente. O capítulo 3 deste livro apresenta o uso do conceito de construtor de forma detalhada.

É importante observar que dentro dos métodos podemos criar variáveis, mas estas têm validade (escopo) somente para uso dentro do método que as criou. Não devemos confundir variáveis de um método com os atributos que são criados logo abaixo da definição de uma classe. Percebam que os atributos não pertencem a

nenhum método específico, logo, podem ser acessados por qualquer um deles. O programa 01.05 apresenta uma classe Java com dois métodos, além da diferença entre variáveis e atributos.

Programa 01.05

```
package modulo01;
// Exemplo de métodos, variáveis e atributos
import java.util.Date;
public class ExemploMetodo {
    private int  meuAtributo = 0;      // pode ser usado por qualquer método
    public static void main(String[] args) {
        Date today = new Date();       // today representa uma variável
        System.out.println(today);
        imprimir();     // executando o método imprimir
    }
    public static void imprimir() {
        // esta variável poderá ser usada somente no método imprimir()
        int  minhaVariavel = 0;
        System.out.println("método imprimir");
    }
}
```

Esse programa cria um atributo chamado `meuAtributo (private int meuAtributo = 0)`, além dos métodos `main()` e `imprimir()`. O método `imprimir()` cria uma variável chamada `minhaVariavel`. É importante observar que o método `main()` está executando o método `imprimir()`. Como já informado, uma variável poderá ser utilizada somente dentro de um método.

1.6.5 Assinatura de um método

Todo método definido tem uma assinatura que garante que não haja dois métodos iguais no programa. Quando enviamos mensagens para um método, a JVM (Java Virtual Machine) procura executar o método buscando identificá-lo pela sua assinatura. Em Java é perfeitamente possível codificar em uma classe dois ou mais métodos com o mesmo nome, mas com assinaturas diferentes.

Os elementos do método que fazem parte da sua assinatura são:

- nome do método;
- quantidade de parâmetros existentes;
- tipo de cada parâmetro;
- ordem desses parâmetros;

Obs.: o tipo de retorno de um método não faz parte de sua assinatura.

O programa 01.06 apresenta um problema na definição de métodos com assinaturas iguais.

Programa 01.06

```
package modulo01;
public class ContaCorrenteSobrecarga {
    int conta;
    int agencia;
    double saldo;
    void efetuarSaque(double valor) {
        this.saldo = this.saldo - valor;
    }
    void efetuarDeposito(double valor) {
        this.saldo = this.saldo + valor;
    }
    void imprimirSaldo() {
        System.out.println(this.saldo);
    }
    void imprimirAtributos() {
        System.out.println("Método imprimirAtributos()");
    }
    void imprimirAtributos(int a) {
        System.out.println("Método imprimirAtributos (int a)");
    }
    void imprimirAtributos(char a) {
        System.out.println("Método imprimirAtributos (char a)");
    }
    void imprimirAtributos(int a, char b) {
        System.out.println("Método imprimirAtributos (int a, char b)");
    }
    void imprimirAtributos(char b, int a) {
        System.out.println("Método imprimirAtributos (char b, int a)");
    }
    // int imprimirAtributos(char a) {}
    /*  a linha anterior irá gerar um erro por estar duplicando a definição
        do método: void imprimirAtributos(char a). */
}
```

Esse programa apresenta o método `imprimirAtributos()` diversas vezes, mas todas com assinaturas diferentes. Também apresenta um exemplo em que apenas alteramos o tipo do retorno do método (`// int imprimirAtributos (char a) {}`), mantendo os outros elementos intactos. É importante observar que o tipo de retorno do método não faz parte da sua assinatura e, por isso, a linha `int imprimirAtributos (char a) {}` precisou ser comentada. A definição desse método gera conflito com a definição do método presente na linha `void imprimirAtributos(char a)`. Nesse caso, para o compilador Java os métodos são iguais, e um erro de compilação será indicado.

1.7 Elementos básicos da linguagem Java

1.7.1 Variáveis em Java

Uma variável representa um endereço de memória no qual um valor está armazenado. Quando definimos variáveis em Java devemos considerar seu nome, tipo e um valor de inicialização. Lembre-se de que uma variável será definida dentro de um método, enquanto um atributo é definido dentro da classe. O nome da variável representa o rótulo usado para identificá-la no código. O tipo da variável determina o conjunto de valores que ela pode assumir, e o valor de inicialização deve ser atribuído à variável no momento de sua criação. É importante observar que uma variável definida internamente a um método só será visível dentro do método, enquanto um atributo será visível por todos os métodos da classe.

1.7.2 Atributos em Java

Conforme já informado, um atributo deve ser criado após a definição da classe e fora de qualquer método, sendo que o atributo pode ser ainda classificado como não-estático ou estático.

Um atributo não-estático existe enquanto o objeto estiver ativo, ou seja, seu valor será liberado depois que o objeto não for mais usado. Um atributo estático existe enquanto a classe na qual ele foi definido estiver carregada na JVM. O programa 01.07 mostra um exemplo do uso de atributo estático e não-estático.

Programa 01.07

```
package modulo01;
// Representação de uma classe com atributos e métodos
public class ContaCorrente {
    int conta;
    int agencia;
    double saldo;
    static double cpmf; // atributo estático
    String nome;
    public String getNome() {
        return this.nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    void efetuarSaque(double valor) {
        this.saldo = this.saldo - valor;
    }
```

```
    void efetuarDeposito(double valor) {
        this.saldo = this.saldo + valor;
    }
    void imprimirSaldo() {
        System.out.println(this.saldo);
    }
    public int getAgencia() {
        return this.agencia;
    }
    public void setAgencia(int agencia) {
        this.agencia = agencia;
    }
    public int getConta() {
        return this.conta;
    }
    public void setConta(int conta) {
        this.conta = conta;
    }
    public double getCpmf() {
        return cpmf;
    }
    public void setCpmf(double cpmf) {
        ContaCorrente.cpmf = cpmf;
    }
    public double getSaldo() {
        return this.saldo;
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
}   // Nenhuma linha de código poderá ser definida abaixo deste símbolo de fecha chaves
```

Na classe `ContaCorrente` definimos o atributo `cpmf` como estático, assim, no método `setCpmf(double cpmf)` fizemos a atribuição ao atributo estático por meio do nome da classe (`ContaCorrente.cpmf = cpmf`), ou seja, um atributo estático deve sempre ser acessado pelo nome da classe. Apesar de ser possível acessar um atributo estático usando um objeto, devemos optar pelo nome da classe. O programa 01.08 mostra um exemplo de uso do atributo estático apresentado no programa 01.07.

Programa 01.08

```
package modulo01;
// Exemplo do uso de atributo estático
public class PrincipalEstatico {
    public static void main(String args[]) {
        ContaCorrente objeto1 = new ContaCorrente();
        // acessando o atributo estático por meio do nome da classe
        ContaCorrente.cpmf = 0.0038;
        objeto1.saldo = 200;
        ContaCorrente objeto2 = new ContaCorrente();
```

```
        objeto2.saldo = 400;
        System.out.println("Manipulando objetos em Java\u2122");
        System.out.println("\u00A9 Sun Microsystems, Inc.");
        // atributo estático sendo acessado por um objeto
        System.out.println("objeto1 atributo estático: " + objeto1.cpmf);
        System.out.println("objeto1 atributo não estático: " + objeto1.saldo);
        // acessando o atributo estático por meio de um objeto
        System.out.println("objeto2 atributo estático: " + objeto2.cpmf);
        System.out.println("objeto2 atributo não estático: " + objeto2.saldo);
        objeto2.cpmf = 0.0010;
        System.out.println("objeto1 atributo estático: " + objeto1.cpmf);
        System.out.println("objeto1 atributo não estático: " + objeto1.saldo);
        // acessando o atributo estático por meio do nome da classe
        System.out.println("objeto1 atributo estático. Acessado pelo nome da classe: " + ContaCorrente.cpmf);
        System.out.println("objeto1 atributo não estático: " + objeto1.saldo);
    }
}
```

O resultado da execução do programa é:

```
Manipulando objetos em Java™
© Sun Microsystems, Inc.
objeto1 atributo estático: 0.0038
objeto1 atributo não estático: 200.0
objeto2 atributo estático: 0.0038
objeto2 atributo não estático: 400.0
objeto1 atributo estático: 0.0010
objeto1 atributo não estático: 200.0
objeto1 atributo estático. Acessado pelo nome da classe: 0.0010
objeto1 atributo não estático: 200.0
```

No programa 01.08 estamos acessando o atributo estático por meio de um objeto (`objeto2.cpmf = 0.0010`) e do nome da classe (`ContaCorrente.cpmf = 0.0038`). O acesso realizado por meio do nome do objeto foi usado apenas para demonstrar que, quando alteramos o valor de um objeto, os outros objetos passam a visualizar o valor alterado. Porém, de acordo com as boas práticas, devemos sempre acessar um atributo estático por meio do nome da classe.

1.7.3 Tipos primitivos

Os tipos primitivos da linguagem Java são muito parecidos com os usados na linguagem C, assim, se você tiver experiência com a linguagem C, não terá dificuldades para entender os tipos primitivos da linguagem Java. A tabela 1.5 mostra os tipos primitivos da linguagem Java.

Tabela 1.5 – Tabela dos tipos primitivos da linguagem Java

Tipo	Tamanho/Formato	Valores válidos
boolean	8 bits	true ou false
byte	8 bits	-128 a 127
short	16 bits	-32.768 a +32.767
int	32 bits	- 2.147.483.648 a +2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807
float	32 bits IEEE 754 floating point	- 3,4028234663852886E+38 a 3,4028234663852886E+38
double	64 bits IEEE 754 floating point	- 4,94065645841246544E-324 a 4,94065645841246544E-324
char	16 bits Caractere Unicode	'\u0000' a '\uFFFF' – 0 a 65.535

1.7.4 Tipos de dados Referência

Toda classe criada pelo programador ou pela Sun representa um tipo de dado diferente dos tipos primitivos (`int`, `double`, `float` etc.) já apresentados. Um exemplo é a classe `ContaCorrente`. Ela representa um novo tipo de dado.

Quando criamos objetos do tipo da classe `ContaCorrente` estamos criando um atributo ou variável do tipo referência. Uma variável do tipo referência, quando passada como parâmetro, terá um comportamento diferente de um tipo de dado primitivo, ou seja, caso o parâmetro passado seja alterado no método chamado, este terá seu valor alterado também quando retornar a execução ao método ativador. Se passarmos um parâmetro do tipo primitivo, as alterações realizadas neste não serão refletidas no método ativador.

1.7.5 Operadores aritméticos

Os operadores aritméticos representam as operações básicas da matemática. A tabela 1.6 apresenta os operadores, seu uso e uma breve descrição.

Tabela 1.6 – Operadores aritméticos

Operador	Uso	Descrição
+	v1 + v2	Adição
-	v1 - v2	Subtração
*	v1 * v2	Multiplicação
/	v1 / v2	Divisão. A divisão entre dois números inteiros resulta em outro número inteiro. Ex.: 11 / 3 = 3
%	v1 % v2	Módulo. O operador % (módulo) retorna o resto da divisão. Ex.: 39 % 5 = 4

O programa 01.09 mostra um exemplo do uso dos operadores aritméticos.

Programa 01.09

```
package modulo01;
// Exemplo do uso dos operadores aritméticos
public class ExemploOperadorAritmetico {
    public static void main(String args[]) {
        int var1 = 5;   // variável var1
        int var2 = 2;   // variável var2
        System.out.println("var1 = " + var1);
        System.out.println("var2 = " + var2);
        System.out.println("-var2 = " + (-var2));      // imprime como valor negativo
        System.out.println("var1+ var2 = " + (var1 + var2));
        System.out.println("var1- var2 = " + (var1 - var2));
        System.out.println("var1* var2 = " + (var1 * var2));
        System.out.println("var1/ var2 = " + (var1 / var2));
        System.out.println("(float) var1/ var2 = " + ((float) var1 / var2));
        System.out.println("var1% var2 = " + (var1 % var2));
        System.out.println("var2 = " + var2);
    }
}
```

O resultado da execução do programa é:

```
var1= 5
var2 = 2
-var2 = -2
var1+ var2 = 7
var1- var2 = 3
var1* var2 = 10
var1/ var2 = 2
(float) var1/ var2 = 2.5
var1% var2 = 1
var2 = 2
```

1.7.6 Operadores unários

Os operadores unários realizam as operações básicas levando em consideração apenas uma variável. A tabela 1.7 apresenta os operadores unários.

Tabela 1.7 – Operadores unários

Operador	Uso	Descrição
++	v1++	Incremento de uma unidade. Quando se coloca o operador após a variável, o incremento será feito após o uso do valor da mesma na expressão (usa-se o valor e depois ele é incrementado).
	++v1	Quando se usa o operador antes da variável, o incremento será feito antes do uso do valor da mesma na expressão (incrementa-se primeiro e depois se usa o valor).
--	v1--	Decremento de uma unidade.
	--v1	Quando se usa o operador antes da variável, o decremento será feito antes do uso do valor da mesma na expressão (decrementa-se primeiro e depois se usa o valor).

O programa 01.10 mostra um exemplo do uso dos operadores unários.

Programa 01.10

```
package modulo01;
// Exemplo do uso dos operadores unários
public class ExemploOperadorUnario {
    public static void main(String args[]) {
        int var1 = 10;
        int var2 = 20;
        int res = 0;
        res = var1 + var2;
        System.out.println("res: " + res);     // imprime 30
        // após a execução do operador = será executado o operador ++ da variável var1
        res = var1++ + var2;    // var1 vai valer 11 após a execução do operador =
        System.out.println("res: " + res);     // imprime 30
        res = var1 + var2;
        System.out.println("res: " + res);     // imprime 31
        res = var1 + --var2;
        System.out.println("res: " + res);     // imprime 30
    }
}
```

O resultado da execução do programa é:

```
res: 30
res: 30
res: 31
res: 30
```

1.7.7 Operadores relacionais

Os operadores relacionais realizam as operações de igualdade e comparação. A tabela 1.8 apresenta os operadores relacionais.

Tabela 1.8 – Operadores relacionais

Operador	Uso	Descrição
==	v1 == v2	Igualdade
!=	v1 != v2	Desigualdade
<	v1 < v2	Menor
<=	v1 <= v2	Menor ou igual
>	v1 > v2	Maior
>=	v1 >= v2	Maior ou igual

O programa 01.11 mostra um exemplo do uso dos operadores relacionais.

Programa 01.11

```
package modulo01;
// Exemplo do uso dos operadores relacionais
public class ExemploOperadorRelacional {
    public static void main(String args[]) {
        int var1 = 27;
        int var2 = 74;
        System.out.println("var1 = " + var1);
        System.out.println("var2 = " + var2);
        System.out.println("var1 == var2 -> " + (var1 == var2));
        System.out.println("var1 != var2 -> " + (var1 != var2));
        System.out.println("var1 < var2 ->  " + (var1 < var2));
        System.out.println("var1 > var2 ->  " + (var1 > var2));
        System.out.println("var1 <= var2 -> " + (var1 <= var2));
        System.out.println("var1 >= var2 -> " + (var1 >= var2));
    }
}
```

O resultado da execução do programa é:

```
var1 = 27
var2 = 74
var1 == var2 -> false
var1 != var2 -> true
var1 < var2 -> true
var1 > var2 -> false
var1 <= var2 -> true
var1 >= var2 -> false
```

1.7.8 Operadores lógicos

Os operadores lógicos são muito usados em estruturas de controle e loop. A tabela 1.9 apresenta os operadores lógicos.

Tabela 1.9 – Operadores lógicos

Operador	Uso	Descrição
&&	v1 && v2	"E" lógico
||	v1 || v2	"OU" lógico
!	!(v1 < v2)	Negação lógica
Os operadores apresentados têm como resultado um valor do tipo boolean.		

1.7.9 Operadores de atribuição

Os operadores de atribuição são usados para atribuir um novo valor a uma variável. A tabela 1.10 mostra os operadores de atribuição.

Tabela 1.10 – Operadores de atribuição

Operador	Uso	Equivale a:
=	v1 = v2;	Atribuição
+=	v1 += v2;	v1 = v1 + v2;
-=	v1 -= v2;	v1 = v1 - v2;
*=	v1 *= v2;	v1 = v1 * v2;
/=	v1 /= v2;	v1 = v1 / v2;
%=	v1 %= v2;	v1 = v1 % v2;

1.7.10 Estruturas de controle

A linguagem Java disponibiliza os comandos `if` e `else` que, de forma seletiva, definem qual bloco de comandos deverá ser executado. Se a condição do comando `if` for avaliada como verdadeira será executado o bloco de comandos dentro do `if`. Caso contrário, o bloco de comandos dentro do `else` será executado. O programa 01.12 mostra um exemplo do comando `if`.

Programa 01.12

```
package modulo01;
// Exemplo do comando if
public class ExemploIf {
    public static void main(String args[]) {
        int var1 = 20;
        int var2 = 10;
        // modo de uso: if (condicao)
        if (var1 > var2) {
            // bloco de comandos do if
            System.out.println("var1 é maior que var2");
        }
    }
}
```

A condição (`var1 > var2`) sempre retornará um valor lógico, ou seja, verdadeiro ou falso. Caso verdadeiro, o bloco de comandos abaixo da condição será executado, caso contrário, não haverá execução. Caso o comando `if` retorne falso, podemos encadear a execução de um segundo bloco de comandos do comando `else`. O programa 01.13 mostra um exemplo envolvendo o comando `if` e o `else`.

Programa 01.13

```
package modulo01;
// Exemplo comando if e do comando else
public class ExemploIfElse {
    public static void main(String args[]) {
        int var1 = 10;
        int var2 = 20;
        // modo de uso: if (condicao)
        if (var1 > var2) {
            // bloco de comandos do if
            System.out.println("var1 é maior que var2");
        } else { // condição avaliada como falso
            // bloco de comandos do else
            System.out.println("var1 é menor que var2");
        }
    }
}
```

Além dos comandos `if` e `else` há o comando `switch`, que também realiza a execução de um bloco de comandos de acordo com uma decisão. Muitos programadores se perguntam quando devem usar o `switch` e quando devem utilizar o comando `if`. Qual dos dois comandos será mais útil?

O uso de cada um dos comandos depende da composição da estrutura de decisão. Devemos optar por usar o comando `switch` quando, no teste realizado, usarmos uma mesma variável, igualando-a com vários valores diferentes. O programa 01.14 mostra um exemplo do uso do comando `if`, enquanto o programa 01.15 mostra um exemplo do mesmo programa com o comando `switch`.

Quando usamos o comando `switch` devemos também usar em conjunto o comando `case`, que com base no valor da variável do comando `switch` define qual opção será executada. Para que somente um entre os vários comandos `case` seja executado, devemos executar o comando `break`, logo após a execução dos comandos contidos no bloco do comando `case` selecionado.

Quando quisermos que um mesmo bloco de comandos seja executado, para dois ou mais diferentes valores do comando `switch`, devemos encadear a execução dos comandos `case` sem a execução do comando `break` entre eles. Somente executamos um comando `break` ao final dos comandos executados. É importante observar que o comando `break` interrompe a execução do `case`.

Há também o comando `default`, que representa uma exceção a todas as opções listadas nos comandos `case`. É importante observar que no comando `switch` só são aceitas variáveis do tipo `int` ou `char`. O uso de outro tipo primitivo gera um erro de compilação.

Programa 01.14

```
package modulo01;
// Exemplo do comando if
import java.io.IOException;
public class ExemploIf02 {
    public static void main(String[] args) throws IOException {
        System.out.println("Digite uma das letras da palavra Java: ");
        int numero = System.in.read(); // lê do teclado apenas um caractere
        if (((char) numero == 'J')  || ((char) numero == 'j')
            || ((char) numero == 'A') || ((char) numero == 'a')
            || ((char) numero == 'V') || ((char) numero == 'v')) {
            System.out.println("Letra digitada está correta.");
        } else if ((char) numero == (char) 13) {
            System.out.println("Foi digitado apenas um <enter>.");
        } else {
            System.out.println("Letra digitada está incorreta.");
        }
    }
}
```

Nesse programa leremos no teclado um caractere, e caso seja igual às letras da palavra Java, exibirá a mensagem `"Letra digitada está correta."`. Caso contrário, veremos `Letra digitada está incorreta`. A linha `else if ((char) numero == (char) 13)` do programa 01.14 está verificando se o usuário digitou a tecla <enter>. O valor dessa tecla na tabela ASCII é igual ao valor 13 no formato caractere. Ou seja, se o usuário não digitar nenhuma letra e apenas teclar <enter> no teclado, será emitida uma mensagem de alerta. Neste exemplo usamos o comando `import java.io.IOException`. O comando considera dois elementos: o comando `import` e a classe de exceção `IOException`. Este livro aborda o comando `import` e apresenta de forma detalhada, no capítulo 7, todos os elementos necessários para o tratamento de exceções em Java.

O programa 01.15 representa outra forma de obter o mesmo resultado que tivemos com o programa 01.14, usando o comando `switch` em vez do `if`.

Programa 01.15

```
package modulo01;
// Exemplo do comando switch. Uma alternativa ao uso do comando if
import java.io.IOException;
public class ExemploSwitch {
    public static void main(String[] args) throws IOException {
        System.out.println("Digite uma das letras da palavra Java: ");
        int numero = System.in.read(); // lê do teclado apenas um caractere
        switch ((char) numero) {
        case 'J':
        case 'j':
        case 'A':
        case 'a':
```

```
        case 'V':
        case 'v':
            System.out.println("Letra digitada está correta.");
            break;
        case (char) 13: // utilizado para tratar o enter
            System.out.println("Foi digitado apenas um <enter>.");
            break;
        default:
            System.out.println("Letra digitada está incorreta.");
        }
    }
}
```

1.7.11.1 Processo de avaliação com os operadores &&, &, || e |

A linguagem Java realiza a avaliação curta quando usamos os operadores lógicos `&&` e `||`. Ou seja, dependendo da avaliação parcial do comando `if`, por exemplo, não será necessário a avaliação da outra expressão. Quando usamos o operador `&&`, caso a primeira condição avaliada em um comando `if` seja falsa, automaticamente a segunda expressão não será avaliada. Quando usamos o operador `||`, caso a primeira condição avaliada em um comando `if` seja verdadeira, automaticamente a segunda expressão não será avaliada. Caso seja necessário que toda a expressão seja avaliada independentemente das validações anteriores, podemos usar os operadores `&` e `|`.

O programa 01.16 exemplifica o uso desses operadores. Neste exemplo usamos a classe `Console` para enfatizar o comportamento dos operadores apresentados. A classe `Console` poderá ser obtida na internet ou na página deste livro no site da Novatec. É importante observar que até a versão 1.4 do J2SE os mecanismos de leitura de dados a partir do teclado eram de uso complexo. Para contornar essa complexidade foi criada a classe `Console`, que tem como objetivo facilitar a leitura dos dados em programas Java.

É importante observar que a classe `Console` usada neste livro não corresponde à classe `Console` presente no pacote `java.io.Console`. A classe `java.io.Console` foi criada na versão 1.6 do J2SE e deve ser usada quando um programa java é executado a partir de um console, ou seja, não atende às necessidade dos exemplos presentes neste livro. A classe `Console` usada neste livro deverá ser obtida da internet ou do site da editora Novatec, conforme comentado.

Para executar os exemplos deste livro será necessário que o leitor tenha a classe `Console` em seu ambiente de desenvolvimento. É importante observar também que em outros exemplos será necessário realizar o download de arquivos com a extensão `.jar`, com o objetivo de evitar os erros de compilação causados pela falta deles. Quando houver algum exemplo que use algum recurso especial, descreveremos no

próprio programa essa necessidade. Todos os recursos necessários para a execução dos exemplos deste livro poderão ser obtidos na página da editora Novatec.

A partir da versão 1.5 do J2SE foi lançada a classe `Scanner`, que pode ser usada como uma alternativa à classe `Console`. A vantagem de usá-la é que já está presente na instalação do J2SE versão 1.5 ou superior, enquanto a classe `Console` deve ser obtida em sites externos. O capítulo 8 apresenta de forma detalhada a classe `Scanner`. Contudo, faremos um uso intensivo dessa classe para realizar a leitura de dados por meio do teclado em diversos exemplos deste livro.

No exemplo apresentado no programa 01.16 usamos a classe `Console` para realizar a leitura dos dados a partir do teclado. Como podemos observar, essa classe está gravada no pacote `modulo01.estudodecaso.util`, ou seja, para que o exemplo funcione, é necessário colocar a classe `Console` na mesma estrutura hierárquica de pacotes usada neste exemplo, além de executar o comando `import` para evitar um erro de compilação.

Para não gerar erro de compilação ao digitar o programa é preciso, a partir do nome do projeto, criar o pacote `modulo01.estudodecaso.util` e copiar a classe `Console` obtida do site para dentro do pacote. A partir desse momento, o exemplo funcionará corretamente. É importante observar que esse mesmo procedimento será usado por outros exemplos deste livro. O capítulo 2 apresenta de forma detalhada o conceito de pacotes (`package`, em inglês) e o comando `import`.

Programa 01.16

```
package modulo01;
// Exemplo do uso do comando if com os operadores &&, ||, &, e |
// A classe Console precisa estar disponível no pacote modulo01.estudodecaso.util
import modulo01.estudodecaso.util.Console;
public class ExemploOperadorLogico {
    public static void main(String args[]) {
        if (Console.readInt("\nNúmero 1: ") > 10 && Console.readInt("Número 2: ") > 10) {
            System.out.println("Os dois números são maiores que 10");
        } else {
            System.out.println("O primeiro ou o segundo número não é maior que 10");
        }
        if (Console.readInt("\nNúmero 3: ") > 10 || Console.readInt("Número 4: ") > 10) {
            System.out.println("Um dos números é maior que 10");
        } else {
            System.out.println("O terceiro ou o quarto número não é maior que 10");
        }
        if (Console.readInt("\nNúmero 5: ") > 10 & Console.readInt("Número 6: ") > 10) {
            System.out.println("Os dois números são maiores que 10");
        } else {
            System.out.println("O quinto ou o sexto número não é maior que 10");
        }
```

```
        if (Console.readInt("\nNúmero 7: ") > 10 | Console.readInt("Número 8: ") > 10) {
            System.out.println("Um dos números é maior que 10");
        } else {
            System.out.println("O sétimo ou o oitavo número não é maior que 10");
        }
    }
}
```

1.7.11 Estruturas de repetição

O primeiro comando que abordaremos é o `for`, que executa um bloco de comandos determinado várias vezes. A figura 1.5 apresenta a sintaxe do comando `for`, e a representação gráfica do comando quando executado de forma procedural.

Código do comando for	Representação gráfica

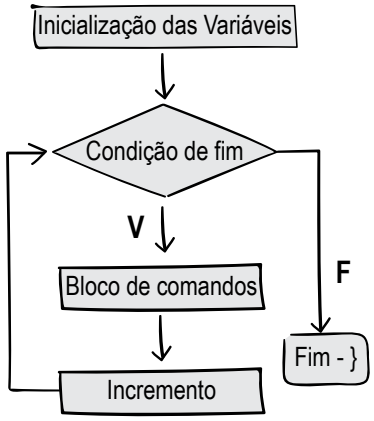
`for (inicialização;`
`  condição de fim;`
`  incremento)`
`{`
`  bloco de comandos`
`}`
	

Figura 1.5 – Comando for.

O programa 01.17 exemplifica o comando `for`.

Programa 01.17

```
package modulo01;
// Exemplo do comando for
public class ExemploFor {
    public static void main(String[] args) {
        for (int var1 = 0; var1 < 27; var1++) {
            System.out.println(var1);
        }
    }
}
```

O programa exibirá os números entre 0 e 26 na tela do computador.

Conforme comentado, devemos usar o comando `for` quando sabemos de antemão quantas vezes o loop deverá ser executado. Quando o fim de um loop depender da entrada do usuário devemos optar pelo uso do comando `while`, que executa um bloco de comandos enquanto sua condição for verdadeira. A figura 1.6 mostra um exemplo da sintaxe do comando `while`, e a representação gráfica do comando.

Código do comando while	Representação gráfica

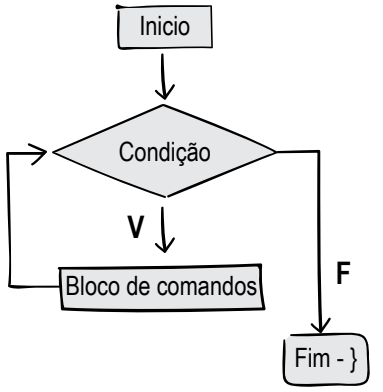
`while (condição)`
`{`
`  bloco de comandos`
`}`
	

Figura 1.6 – Comando while.

O programa 01.18 mostra um exemplo do uso do comando `while`.

Programa 01.18

```
package modulo01;
// Exemplo do comando while
import java.io.IOException;
public class ExemploWhile {
    public static void main(String[] args) throws IOException {
        System.out.println("Digite f para terminar: ");
        int letra = ' ';
        while ((char)letra != 'f') {
            // lê do teclado apenas um caractere
            letra = System.in.read();
        }
    }
}
```

Há ainda uma terceira opção de comando de loop conhecido por `do while`. Difere do `while` no sentido de permitir que pelo menos uma execução do bloco de comandos seja executada antes de testar a condição. O bloco de comandos será executado enquanto a condição for verdadeira. A figura 1.7 mostra um exemplo da sintaxe do comando `do while`, e a representação gráfica do comando.

O programa 01.19 mostra um exemplo do comando `do while`.

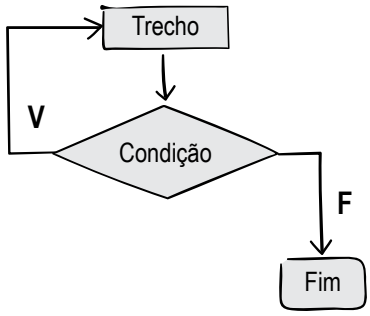
Código do comando do-while	Representação gráfica
do { bloco de comandos } while (condição);	

Figura 1.7 – Comando do while.

Programa 01.19

```
package modulo01;
// Exemplo do comando do while
import java.io.IOException;
public class ExemploDoWhile {
    public static void main(String[] args) throws IOException {
        System.out.println("Digite f para terminar: ");
        int letra = ' ';
        do {
            // lê do teclado apenas um caractere
            letra = System.in.read();
        } while ((char)letra != 'f');
    }
}
```

O comando `do while` será usado nos laboratórios deste livro para a validação da entrada dos dados via teclado. Teremos 9 laboratórios práticos que, de forma progressiva, praticarão os comandos e conceitos apresentados no livro.

1.7.12 Comando break

O comando `break` tem a função de interromper a execução de um loop. É possível também, com o comando `break`, interromper um loop presente em um nível superior ao loop onde o `break` será executado. Para isso usamos labels. O programa 01.20 exemplifica o comando `break`, com e sem uso de labels.

Programa 01.20

```
package modulo01;
// Exemplo do comando break
import java.io.IOException;
import java.util.Scanner;
```

```
public class ExemploBreak {
    public static void main(String[] args) throws IOException {
        System.out.println("Digite f para terminar: ");
        int letra = ' ';
        while (true) {
            // lê do teclado apenas um caractere
            letra = System.in.read();
            if ((char) letra == 'f') {
                break;  // quebra o loop do while (true)
            }
        }
        System.out.println("O loop foi quebrado");
        System.out.println("Usando o comando break com label");
        breakLabel();
    }
    private static void breakLabel() {
        int[] tLista = { 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100
    };
        int i;
        int num = 0;
        Scanner sc = new Scanner(System.in);
        labelbreak: while (true) { // primeiro nível do loop e a definição do label
            do { // segundo nível do loop
                System.out.println("Digite um número. (0 - fim)");
                num = sc.nextInt();
                if (num == 0) {
                    break labelbreak;   // quebra o primeiro nível do loop
                }
                // valida o número lido
                if (num < 0 || num > 100) {
                    System.out.println("Digitar números entre 1 e 100");
                }
            } while (num < 0 || num > 100);
            for (i = 0; i < tLista.length; i++) {
                if (tLista[i] == num)
                    break;  // quebra o loop do for
            }
            if (i < tLista.length)
                System.out.println("O número foi encontrado no vetor e está na posição " + i);
            else
                System.out.println("O número não foi encontrado no vetor");
        }
    }
}
```

1.7.13 Comando continue

O comando `continue` tem a função de fazer com que a condição do comando de loop seja novamente testada, mesmo antes de alcançar o fim do comando. O programa 01.21 mostra um exemplo do comando `continue`.

Programa 01.21

```
package modulo01;
// Exemplo do comando continue
public class ExemploContinue {
    public static void main(String args[]) {
        for (int i = 0; i <= 30; i++) {
            if ((i > 10) && (i < 20)) {
                continue;
            }
            // apresenta na tela quando o i não estiver entre 10 e 20
            System.out.println("i = " + i);
        }
    }
}
```

1.7.14 Comandos de entrada

Os comandos de entrada disponíveis nas primeiras versões da linguagem Java sempre representaram uma grande dificuldade para o desenvolvedor. Somente na versão do J2SE 5.0 é que o problema foi resolvido, com a inclusão da classe `Scanner`, disponível no pacote `java.util`. Ou seja, para usar essa classe devemos usar o comando `import java.util.Scanner` antes da criação da classe. O programa 01.22 traz um exemplo do uso da classe `Scanner` realizando a leitura dos dados pelo teclado. O programa 01.23 mostra um segundo exemplo do uso da classe `Scanner`. Neste exemplo, realizaremos a leitura dos dados usando como fonte de dados um arquivo. Para executar esse programa e ter acesso ao arquivo, este deverá ser criado dentro do pacote onde o código-fonte do programa Java está gravado.

Programa 01.22

```
package modulo01;
// Exemplo da classe Scanner lendo dados do teclado
import java.util.Scanner;
public class ExemploScanner {
    public static void main(String args[]) {
        System.out.print("Digite um número inteiro: ");
        Scanner var = new Scanner(System.in);
        int numero = var.nextInt(); // declara e inicia a variável
        System.out.println("Valor digitado = " + numero);
        System.out.print("Digite uma string composta: ");
        /*  definindo que o delimitador de leitura do objeto sc é o enter. Para formalizar
            que o delimitador é o enter usamos os caracteres especiais \r\n.
            O caracter padrão do comando Scanner é o espaço em branco. */
        Scanner sc = new Scanner(System.in).useDelimiter("\r\n");
        /*  como o delimitador do objeto sc é igual a enter podemos usar o método
            next() para ler strings compostas. */
```

```
        String nome = sc.next();
        System.out.println("String digitada = " + nome);
        System.out.print("Digite uma string composta: ");
        // criando um novo objeto Scanner, sem alteração do delimitador padrão
        Scanner sc1 = new Scanner(System.in);
        // para o objeto sc1 ler uma string composta precisamos do método nextLine
        String nome1 = sc1.nextLine(); // usando o método nextLine
        System.out.println("String digitada = " + nome1);
    }
}
```

Programa 01.23

```
package modulo01;
// Exemplo da classe Scanner lendo dados de um arquivo
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class ExemploScannerArquivo {
    public static void main(String args[]) throws FileNotFoundException {
        System.out.println("Números contidos no arquivo: ");
        /*  o arquivo numeros.txt deve ficar gravado dentro do pacote modulo01 e possuir
            o seguinte conteúdo: 1 3 4 6 34 23. */
        Scanner sc = new Scanner(new File(".\\modulo01\\numeros.txt"));
        while (sc.hasNextLong()) {
            long aLong = sc.nextLong();
            System.out.println ("Número: " + aLong);
        }
    }
}
```

A classe `Scanner` permite a leitura de tipos de dados primitivos e strings. No primeiro exemplo do programa 01.22 a fonte usada para leitura foi o teclado (`System.in`), e o tipo primitivo lido, o `int`. No segundo exemplo, a fonte de leitura foi um arquivo, e o tipo de dado primitivo, o `long`.

1.7.15 Comandos de saída

Nos exemplos já apresentados no livro usamos em diferentes lugares o comando `System.out.println ("texto qualquer")` para apresentar um determinado resultado na tela. Esse comando é o mais usado para a apresentação de dados quando o desenvolvimento do programa Java é batch. Este exemplo usa o atributo estático `out`, definido na classe `System`, que por sua vez executa o método `println()`. O atributo `out` representa um `stream` de saída padrão e é um atributo do tipo da classe `PrintStream`, que será detalhada no capítulo 8.

Na versão do J2SE 1.5 foi disponibilizado na classe `PrintStream` o método `printf()`, que oferece os mesmos recursos já conhecidos na linguagem C. Logo, podemos usar um formato de saída padrão muito parecido com o disponível na linguagem C. O programa 01.24 exemplifica o atributo `out` usando o método `printf()`. É importante observar que o método `printf()` está presente apenas a partir da versão 1.5 do J2SE. Caso seu ambiente ainda não esteja configurado para essa versão, um erro de compilação será exibido.

Programa 01.24

```
package modulo01;
// Exemplo do atributo out e do método printf
public class ExemploPrintf {
    public static void main(String args[]) {
        System.out.printf("|%8d|\n",   820);
        System.out.printf("|%2.6f|\n", 1223.4432);
        System.out.printf("|%2.2f|\n", 1223.4432);
        System.out.printf("|%10.2f|\n", 1223.4432);
        System.out.printf("|%010.2f|\n", 1223.4432);
        System.out.printf("|%20f|\n", 1223.4432);
        System.out.printf("|%.2f|\n", 1223.4432);
        System.out.printf("|%10s|\n", "abcdefghijklmnopqrstuvxywz");
        System.out.printf("|%10.8s|\n", "abcdefghijklmnopqrstuvxywz");
        System.out.printf("|%10s|\n", "abcde");
        // alinhado a esquerda -
        System.out.printf("|%-10s|\n", "abcde");
        // arredondamento na sexta casa
        System.out.printf("%f\n", 10.123456589);
        System.out.printf("%f\n", 10.123456389);
    }
}
```

O resultado desse programa será:

```
|     820|
|1223,443200|
|1223,44|
|   1223,44|
|0001223,44|
|         1223,443200|
|1223,44|
|abcdefghijklmnopqrstuvxywz|
|  abcdefgh|
|     abcde|
|abcde     |
10,123457
10,123456
```

1.8 Exercícios sobre classes e objetos

1.8.1 Exercício 1

Desenvolver uma classe Java chamada `Apolice` com os seguintes atributos: `nomeSegurado`, `idade` e `valorPremio`. A classe `Apolice` deverá conter os seguintes métodos:

Método	Descrição
`imprimir()`	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe `Apolice`. Para imprimir em Java usa-se o comando `System.out.println(NOME_ATRIBUTO)`.
`calcularPremioApolice()`	Este método não retorna valor e deverá calcular o valor do prêmio seguindo as seguintes regras: caso a idade seja superior a 18 e menor ou igual a 25 anos, use a fórmula: `valorPremio += (valorPremio * 20)/100`. Quando a idade for superior a 25 e menor ou igual a 36 anos, use a fórmula `valorPremio += (valorPremio *15)/100`. Quando a idade for superior a 36, use a fórmula `valorPremio += (valorPremio * 10)/100`.
`oferecerDesconto()`	Este método não retorna valor, mas recebe o parâmetro `cidade`, que irá conter o nome da cidade para o cálculo do desconto. Caso a cidade seja Curitiba, dê um desconto no valor do prêmio de 20%. Caso seja Rio de Janeiro, dê um desconto no valor do prêmio de 15%. Caso seja São Paulo, dê um desconto no valor do prêmio de 10%. Caso a cidade seja Belo Horizonte, dar um desconto no valor do prêmio de 5%.

O programa 01.25 apresenta o resultado do exercício 1. Essa classe pode ser comparada a uma tabela de banco de dados com colunas e stored procedures. Como a classe não deverá conter o método `main()`, não poderá ser executada. O exercício 2 usará essa classe e implementará o método `main()`.

É importante observar que usamos a palavra reservada `this` com o objetivo de formalizar que a variável é um atributo. O capítulo 3 apresenta maiores detalhes sobre essa palavra reservada.

Programa 01.25

```
package modulo01.exemplosbasicos;
// Resposta do exercício 1
public class Apolice {
    String nome;
    int idade;
    double valorPremio;
    public void imprimir() {
        System.out.println("Nome:" + this.nome);
```

```
        System.out.println("Idade:" + this.idade);
        System.out.println("Valor Prêmio:" + this.valorPremio);
    }
    public void calcularPremioApolice() {
        if ((this.idade >= 18) && (this.idade <= 25)) {
            this.valorPremio += (this.valorPremio * 20) / 100;
        }
        if ((this.idade > 25) && (this.idade <= 36)) {
            this.valorPremio += (this.valorPremio * 15) / 100;
        }
        if (this.idade > 36) {
            this.valorPremio += (this.valorPremio * 10) / 100;
        }
    }
    public void oferecerDesconto(String nomeCidade) {
        if (nomeCidade.equals("Curitiba")) {
            this.valorPremio -= (this.valorPremio * 20) / 100;
        }
        if (nomeCidade.equals("Rio de Janeiro")) {
            this.valorPremio -= (this.valorPremio * 15) / 100;
        }
        if (nomeCidade.equals("São Paulo")) {
            this.valorPremio -= (this.valorPremio * 10) / 100;
        }
        if (nomeCidade.equals("Belo Horizonte")) {
            this.valorPremio -= (this.valorPremio * 5) / 100;
        }
    }
}
```

1.8.2 Exercício 2

Desenvolver uma segunda classe Java chamada `PrincipalApolice` com a seguinte estrutura:

- Criar o método `main()` conforme o padrão da linguagem Java. Nesse método, criar um objeto da classe `Apolice` usando o comando: `Apolice novoObj = new Apolice()`. Para cada atributo da classe atribuir um valor coerente.
- Executar o método `imprimir()` e analisar o que será impresso na tela.
- Em seguida, executar o método `calcularPremioApolice()`.
- Executar o método `imprimir()` novamente e analisar o que será exibido na tela.
- Executar o método `oferecerDesconto()` passando como parâmetro a cidade de Curitiba.
- Executar o método `imprimir()` novamente e analisar o que será exibido na tela.

O programa 01.26 mostra o resultado do exercício 2.

Programa 01.26

```
package modulo01.exemplosbasicos;
// Resposta do exercício 2
public class PrincipalApolice {
    public static void main(String[] args) {
        Apolice novoObj = new Apolice();
        novoObj.idade = 25;
        novoObj.nome = "Gustavo Baravieira Costa";
        novoObj.valorPremio = 500;
        System.out.println();
        System.out.println("Imprimindo os dados inicializados");
        novoObj.imprimir();
        novoObj.calcularPremioApolice();
        System.out.println();
        System.out.println("Imprimindo os dados após a execução do método calcularPremioApolice");
        novoObj.imprimir();
        novoObj.oferecerDesconto("Curitiba");
        System.out.println();
        System.out.println("Imprimindo os dados após a execução do método oferecerDesconto");
        novoObj.imprimir();
    }
}
```

O resultado da execução do programa será:

```
Imprimindo os dados inicializados
Nome:Gustavo Baravieira Costa
Idade:25
Valor Premio:500.0

Imprimindo os dados após a execução do método calcularPremioApolice
Nome:Gustavo Baravieira Costa
Idade:25
Valor Premio:600.0

Imprimindo os dados após a execução do método oferecerDesconto
Nome:Gustavo Baravieira Costa
Idade:25
Valor Premio:480.0
```

1.8.3 Exercício 3

Desenvolver uma classe chamada `Acampamento` com os seguintes atributos: `nome`, `equipe`, `idade`. Em seguida, implementar os seguintes métodos:

Método	Descrição
`imprimir()`	Este método não retorna valor e deve exibir os atributos na tela.
`separarGrupo()`	Este método não retorna valor e deverá verificar as seguintes condições: se a idade estiver entre 6 e 10 anos, atribuir A ao atributo `equipe`; se a idade estiver entre 11 e 20, atribuir B ao atributo `equipe`; se a idade for superior a 21 anos, atribuir C ao atributo `equipe`.

O programa 01.27 apresenta o resultado do exercício 3. Essa classe pode ser comparada a uma tabela de banco de dados com suas colunas e suas `stored procedures`. Como a classe não deverá conter o método `main()`, não poderá ser executada. O exercício 4 usará essa classe.

Programa 01.27

```
 package modulo01.exemplosbasicos;
// Resposta do exercício 3
public class Acampamento {
    // atributos da classe
    String nome;
    String equipe;
    int idade;
    public void imprimir() {
        System.out.println("Nome:   " + this.nome);
        System.out.println("Equipe: " + this.equipe);
        System.out.println("Idade:  " + this.idade);
    }
    public void separarGrupo() {
        if ((this.idade >= 6) && (this.idade <= 10)) {
            this.equipe = "A";
        }
        if ((this.idade >= 11) && (this.idade <= 20)) {
            this.equipe = "B";
        }
        if (this.idade >= 21) {
            this.equipe = "C";
        }
    }
}
```

1.8.4 Exercício 4

Desenvolver uma segunda classe Java chamada `PrincipalAcampamento` com a seguinte estrutura:

- Criar o método `main()` conforme o padrão da linguagem Java.
- Criar um objeto da classe `Acampamento` e atribuir valores a seus atributos.

- Executar o método `imprimir()` e analisar o que será exibido na tela.
- Executar o método `separarGrupo()`.
- Executar o método `imprimir()` novamente e analisar o que será exibido na tela.

O programa 01.28 mostra o resultado do exercício 4.

Programa 01.28

```
package modulo01.exemplosbasicos;
// Resposta do exercício 4
public class PrincipalAcampamento {
    public static void main(String[] args) {
        Acampamento novoObj = new Acampamento();
        novoObj.idade = 22;
        novoObj.nome = "Rafael Zanetti";
        // não precisamos atribuir valor para equipe, pois ela será definida por meio da idade
        System.out.println("Imprimindo os dados inicializados");
        novoObj.imprimir();
        novoObj.separarGrupo();
        System.out.println();
        System.out.println("Imprimindo os dados após a execução do método separarGrupo");
        novoObj.imprimir();
    }
}
```

O resultado da execução do programa será:

```
Imprimindo os dados inicializados
Nome:   Rafael Zanetti
Equipe: null
idade:  22

Imprimindo os dados após a execução do método separarGrupo
Nome:   Rafael Zanetti
Equipe: C
idade:  22
```

1.8.5 Exercício 5

Desenvolver uma classe chamada `Computador` com os seguintes atributos: `marca`, `cor`, `modelo`, `numeroSerie`, `preco`. Implementar os seguintes métodos:

Método	Descrição
`imprimir()`	Este método não retorna valor e deve exibir os atributos na tela.
`calcularValor()`	Não retorna valor e deverá verificar as seguintes condições: caso a marca seja HP, acrescentar 30% ao preço; caso seja IBM, acrescentar 50% ao preço; caso seja qualquer outra, manter o preço original.
`alterarValor()`	Este método recebe um valor como parâmetro. Atribuir este valor ao atributo `preço`, caso o valor do parâmetro recebido seja maior que 0. Caso seja maior que 0, o método `alterarValor()` deverá, além de atribuir o valor ao atributo `preco`, retornar 1. Caso contrário, não atribuir o valor ao atributo `preco` e retornar 0.

O programa 01.29 mostra o resultado do exercício 5.

Programa 01.29

```
package modulo01.exemplosbasicos;
// Resposta do exercício 5
public class Computador {
    String marca, cor, modelo;
    int nSerie;
    double preco;
    public void imprimir() {
        System.out.println("Marca:             " + this.marca);
        System.out.println("Cor:               " + this.cor);
        System.out.println("Modelo:            " + this.modelo);
        System.out.println("Número de série:   " + this.nSerie);
        System.out.println("Preço:             " + this.preco);
    }
    public void calcularValor() {
        if (this.marca.equals("HP")) {
            this.preco = this.preco * 1.30;
        }
        if (this.marca.equals("IBM")) {
            this.preco = this.preco * 1.50;
        }
    }
    public int alterarValor (double novoValor) {
        if (novoValor > 0) {
            this.preco = novoValor;
            return 1;
        }
        return 0;
    }
}
```

1.8.6 Exercício 6

Desenvolver uma segunda classe Java chamada `PrincipalComputador` com a seguinte estrutura:

- Criar o método `main()` conforme o padrão da linguagem Java.
- Criar um objeto da classe `Computador` e atribuir valores a seus atributos. Atribuir HP ao atributo marca.
- Executar o método `imprimir()` e analisar o que será exibido na tela.
- Executar o método `calcularValor()`.
- Executar o método `imprimir()` e analisar o que será exibido na tela.
- Criar um segundo objeto e atribuir valores a seus atributos. Atribuir IBM ao atributo `marca` do novo objeto.
- Executar o método `calcularValor()` do novo objeto.
- Executar o método `imprimir()` do novo objeto e analisar o que será exibido na tela.
- Executar para o novo objeto o método `alterarValor()` com um valor positivo.
- Verificar no método `main()` o retorno do método `alterarValor()` e mostrar a mensagem de "Alterado" caso este retorne 1, e valor "Não Alterado" caso retorne 0.
- Executar para o novo objeto o método `alterarValor()` com um valor negativo.
- Verificar no método `main()` o retorno do método `alterarValor()` e mostrar a mensagem de "Valor Alterado" caso este retorne 1, e "valor Não Alterado" caso retorne 0.
- Executar para o novo objeto o método `imprimir()` e analisar o que será exibido na tela.

O programa 01.30 mostra o resultado do exercício 6.

Programa 01.30

```
package modulo01.exemplosbasicos;
// Resposta do exercício 6
public class PrincipalComputador {
    public static void main(String[] args) {
        Computador novoObj = new Computador();
        novoObj.marca = "HP";
        novoObj.cor = "Preto";
        novoObj.modelo = "DV6383";
        novoObj.nSerie = 987654312;
        novoObj.preco  = 3000;
        System.out.println("Imprimindo os dados inicializados");
        novoObj.imprimir();
        novoObj.calcularValor();
        System.out.println();
        System.out.println("Imprimindo os dados após a execução do método calcularValor");
        novoObj.imprimir();
        Computador novoObj01 = new Computador();
        novoObj01.marca     = "IBM";
        novoObj01.cor       = "Branco";
        novoObj01.modelo    = "IBM583";
        novoObj01.nSerie    = 9873312;
        novoObj01.preco     = 4000;
        novoObj01.calcularValor();
        System.out.println();
        System.out.println("Imprimindo dados após a execução do método calcularValor");
        novoObj01.imprimir();
        int ret = novoObj01.alterarValor(2000);
        if (ret > 0) {
            System.out.println("Valor alterado");
        } else {
            System.out.println("Valor NÃO alterado");
        }
        System.out.println();
        System.out.println("Imprimindo dados após a execução do método alterarValor");
        novoObj01.imprimir();
        System.out.println();
        System.out.println("Executando o método alterarValor com valor negativo");
        ret = novoObj01.alterarValor(-1300);
        if (ret > 0) {
            System.out.println("Valor alterado");
        } else {
            System.out.println("Valor NÃO alterado");
        }
        System.out.println();
        System.out.println("Imprimindo dados após a execução do método alterarValor");
        novoObj01.imprimir();
    }
}
```

O resultado da execução do programa será:

```
Imprimindo os dados inicializados
Marca:              HP
Cor:                Preto
Modelo:             DV6383
Número de Série:    987654312
Preço:              3000.0

Imprimindo os dados após a execução do método calcularValor
Marca:              HP
Cor:                Preto
Modelo:             DV6383
Número de Série:    987654312
Preço:              3900.0

Imprimindo dados após a execução do método calcularValor
Marca:              IBM
Cor:                Branco
Modelo:             IBM583
Número de Série:    9873312
Preço:              6000.0
Valor alterado

Imprimindo dados após a execução do método alterarValor
Marca:              IBM
Cor:                Branco
Modelo:             IBM583
Número de Série:    9873312
Preço:              2000.0

Executando o método alterarValor com valor negativo
Valor NÃO alterado

Imprimindo dados após a execução do método alterarValor
Marca:              IBM
Cor:                Branco
Modelo:             IBM583
Número de Série:    9873312
Preço:              2000.0
```

1.8.7 Exercício 7

Desenvolver uma classe Java chamada `ContaCorrente` com a seguinte estrutura:

Atributos: `conta`, `agencia`, `saldo` e `nomeCliente`

Métodos:

Método	Descrição
`sacar()`	Retorna valor 1 caso o saque seja realizado ou 0 se não houver saldo suficiente na conta. Deverá receber como parâmetro o valor a ser sacado.
`depositar()`	Realizar o depósito do valor recebido como parâmetro. Não deve retornar valor.
`imprimir()`	Exibir na tela os atributos da classe. Esse método não retorna nada.

O programa 01.31 mostra o resultado do exercício 7.

Programa 01.31

```
package modulo01.exemplosbasicos;
// Resposta do exercício 7
public class ContaCorrente {
    int conta, agencia;
    double saldo;
    String nomeCliente;
    public int sacar(double valor) {
        if (this.saldo >= valor) {
            this.saldo = this.saldo - valor;
            return 1;
        }
        return 0;
    }
    public void depositar(double valor) {
        this.saldo = this.saldo + valor;
    }
    public void imprimir() {
        System.out.println("Número da conta: " + this.conta);
        System.out.println("Número da agência: " + this.agencia);
        System.out.println("Saldo da conta corrente: " + this.saldo);
        System.out.println("Nome do cliente: " + this.nomeCliente);
    }
}
```

1.8.8 Exercício 8

Desenvolver uma segunda classe Java chamada `PrincipalContaCorrente` com a seguinte estrutura:

Criar um atributo da classe `ContaCorrente` para ser usado pelos métodos da classe para realizar saques e depósitos. Não se esquecer de executar o operador `new` para o atributo criado.

Obs.: atenção, pois ao executar o programa só poderemos fazer um saque se já tivermos realizado um depósito.

Métodos:

Método	Descrição
`main()`	Implementá-lo conforme o padrão da linguagem Java. O método `main()` deverá criar um loop para o usuário escolher entre as opções cadastrar, sacar, depositar ou consultar. Se for selecionada a opção sacar, executar o método `execSaque()`. Se for selecionado depositar, executar o método `execDeposito()`. Para a opção consultar, executar o método `execConsulta()`. Para a opção cadastrar, executar o método `execCadastro()`.
`execSaque()`	Solicitar ao usuário que digite um valor e executar o método `sacar()` da classe `ContaCorrente` usando o atributo criado. Testar o retorno do método `sacar()`. Se for retornado 1, exibir "saque realizado", caso contrário, exibir "saque não realizado".
`execDeposito()`	Solicitar ao usuário que digite um valor e executar o método `depositar()` da classe `ContaCorrente` usando o objeto criado anteriormente.
`execConsulta()`	Apresentar os atributos na tela executando o método `imprimir()` da classe `ContaCorrente`.
`execCadastrar()`	Solicitar que o usuário realize a leitura dos dados via teclado e em seguida realize a atribuição dos valores lidos do teclado aos atributos do objeto da classe `ContaCorrente`, criado como atributo dessa classe.

O programa 01.32 mostra o resultado do exercício 8.

Programa 01.32

```
package modulo01.exemplosbasicos;
// Resposta do exercicio 8
import java.util.Scanner;
public class PrincipalContaCorrente {
    ContaCorrente cc = new ContaCorrente();
    public static void main(String[] args) {
        PrincipalContaCorrente obj = new PrincipalContaCorrente();
        int op = 0;
        while (op != 9) {
            Scanner sc = new Scanner(System.in);
            System.out.println("1 - Cadastrar");
            System.out.println("2 - Saque");
            System.out.println("3 - Depósito");
            System.out.println("4 - Consultar saldo");
            System.out.println("9 - Sair");
            System.out.println("Entre com uma opção: ");
            op = sc.nextInt();
            switch (op) {
            case 1:
                obj.execCadastrar();
                break;
```

```
            case 2:
                obj.execSaque();
                break;
            case 3:
                obj.execDeposito();
                break;
            case 4:
                obj.execConsulta();
                break;
            }
        }
    }
    public void execDeposito() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Entre com o valor para o depósito: ");
        double valor = sc.nextDouble();
        this.cc.depositar(valor);
        System.out.println("Depósito realizado");
    }
    public void execSaque() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Entre com o valor para o saque: ");
        double valor = sc.nextDouble();
        int ret = this.cc.sacar(valor);
        if (ret == 1) {
            System.out.println("Saque realizado");
        } else {
            System.out.println("Saque NÃO realizado");
        }
    }
    public void execConsulta() {
        this.cc.imprimir();
    }
    public void execCadastrar() {
        // o conteúdo \r\n define que o separador entre strings é o enter. O padrão é o espaço
        Scanner sc = new Scanner(System.in).useDelimiter("\r\n");
        System.out.println("Entre com o nome do cliente:       ");
        this.cc.nomeCliente = sc.nextLine();
        System.out.println("Entre com o número da agência:     ");
        this.cc.agencia = sc.nextInt();
        System.out.println("Entre com o número da conta:       ");
        this.cc.conta = sc.nextInt();
        System.out.println("Entre com o saldo do cliente:      ");
        this.cc.saldo = sc.nextDouble();
    }
}
```

1.8.9 Exercício 9

Desenvolver uma classe Java chamada `Eleitoral` com a seguinte estrutura:

Atributos: `nome` e `idade`

Métodos:

Método	Descrição
`imprimir()`	Seguir a mesma especificação dos demais métodos. O método `imprimir()` deverá executar o método `verificar()` como último comando.
`verificar()`	O método `verificar()` não retorna valor nem recebe parâmetro. Deve exibir na tela mensagens de acordo com as seguintes condições: caso a idade seja inferior a 16 anos, exibir na tela "Eleitor não pode votar"; para idade superior ou igual a 16 anos e inferior ou igual a 65, exibir na tela "Eleitor deve votar". Para idade superior a 65 anos, exibir a tela "Voto facultativo".

O programa 01.33 mostra o resultado do exercício 9.

Programa 01.33

```
package modulo01.exemplosbasicos;
// Resposta do exercício 9
public class Eleitoral {
    String nome;
    int idade;
    public void imprimir() {
        System.out.println("Nome do eleitor:   " + this.nome);
        System.out.println("Idade do eleitor:  " + this.idade);
        // executar o método verificar para que este imprima seu resultado na tela
        verificar();
    }
    public void verificar() {
        if (this.idade < 16) {
            System.out.println("Eleitor não pode votar");
        }
        if ((this.idade >= 16) && (this.idade <= 65)) {
            System.out.println("Eleitor deve votar");
        }
        if (this.idade > 65) {
            System.out.println("Voto facultativo");
        }
    }
}
```

1.8.10 Exercício 10

Desenvolver a classe `PrincipalEleitoral` com a seguinte estrutura:

Atributo: `valor`

Método:

Método	Descrição
`main()`	Implementá-lo conforme o padrão da linguagem Java. Criar um objeto da classe `Eleitoral` e atribuir valores aos parâmetros. Executar o método `imprimir()` e analisar os valores exibidos na tela.

O programa 01.34 mostra o resultado do exercício 10.

Programa 01.34

```
package modulo01.exemplosbasicos;
// Resposta do exercício 10
public class PrincipalEleitoral {
    public static void main(String[] args) {
        Eleitoral el = new Eleitoral();
        el.idade = 20;
        el.nome = "Thiago F. P. David ";
        el.imprimir();
    }
}
```

1.8.11 Exercício 11

Desenvolver uma classe Java chamada `Estoque` com a seguinte estrutura:

Atributos: `nomeProduto`, `valor` e `quantidade`.

Métodos:

Método	Descrição
`imprimir()`	Seguir a mesma especificação dos demais métodos.
`verificarDisponibilidade()`	Deve retornar um valor inteiro e receber um parâmetro inteiro. O método `verificarDisponibilidade()` deverá retornar 1 caso existam produtos disponíveis ou 0 em caso contrário. A existência de produtos disponíveis significa que o atributo tem quantidade maior que 0 e maior ou igual ao parâmetro recebido.
`removerProdutos()`	O método `removerProdutos()` retorna um inteiro e deverá receber como parâmetro a quantidade de elementos que serão removidos. Antes da remoção deve-se verificar se há disponibilidade do produto solicitado. Para isso executar, o método `verificarDisponibilidade()` e, caso este retorne 1, o método remover estoque poderá diminuir o valor recebido como parâmetro do total do atributo quantidade. O método `removerProdutos()` deverá retornar 1, caso tenha sucesso na remoção dos produtos. Caso contrário, retornar 0 informando que não foi possível remover a quantidade solicitada.

O programa 01.35 mostra o resultado do exercício 11.

Programa 01.35

```
package modulo01.exemplosbasicos;
// Resposta do exercício 11
public class Estoque {
    String nomeProduto;
    int quantidade;
    double valor;
    public void imprimir() {
        System.out.println("Nome do produto:       " + this.nomeProduto);
        System.out.println("Quantidade do produto: " + this.quantidade);
        System.out.println("Valor do produto:      " + this.valor);
    }
    public int verificarDisponibilidade (int quant) {
        if ((this.quantidade > 0) && (this.quantidade >= quant)) {
            return 1;
        }
        return 0;
    }
    public int removerProdutos (int quant) {
        int ret = verificarDisponibilidade(quant);
        if (ret == 1) {
            this.quantidade = this.quantidade - quant;
            return 1;
        }
        return 0;
    }
}
```

1.8.12 Exercício 12

Desenvolver a classe `PrincipalEstoque` com a seguinte estrutura:

Método:

Método	Descrição
`main()`	Implementá-lo conforme o padrão da linguagem Java. Criar um objeto da classe `Estoque` e atribuir valores aos parâmetros.

Criar três objetos da classe `Estoque` e atribuir valores para os atributos. Exercitar a chamada dos métodos para que seja possível analisar todas as possibilidades que os métodos criados retornam.

O programa 01.36 mostra o resultado do exercício 12.

Programa 01.36

```
package modulo01.exemplosbasicos;
// Resposta do exercício 12
public class PrincipalEstoque {
    public static void main(String[] args) {
        // criando o objeto es1 do tipo Estoque
        Estoque es1 = new Estoque();
        es1.nomeProduto = "Mochilas";
        es1.quantidade = 100;
        es1.valor = 10;
        es1.imprimir();
        int ret = es1.verificarDisponibilidade(100);
        if (ret == 1) {
            System.out.println("Produto na quantidade informada disponível");
        }
        else{
            System.out.println("Produto na quantidade informada NÃO disponível");
        }
        ret = es1.verificarDisponibilidade(500);
        if (ret == 1) {
            System.out.println("Produto na quantidade informada disponível");
        }
        else{
            System.out.println("Produto na quantidade informada NÃO disponível");
        }
        // criando o objeto es2 do tipo Estoque
        Estoque es2 = new Estoque();
        es2.nomeProduto = "Pastas";
        es2.quantidade = 50;
        es2.valor = 250;
        // criando o objeto es3 do tipo Estoque
        Estoque es3 = new Estoque();
        es3.nomeProduto = "Telefones";
        es3.quantidade = 150;
        es3.valor = 59;
        ret = es3.verificarDisponibilidade(120);
        if (ret == 1) {
            System.out.println("Produto na quantidade informada disponível");
        }
        else {
            System.out.println("Produto na quantidade informada NÃO disponível");
        }
    }
}
```

1.9 Laboratório 1

A figura 1.8 representa o diagrama de classes usado pelo laboratório 1.

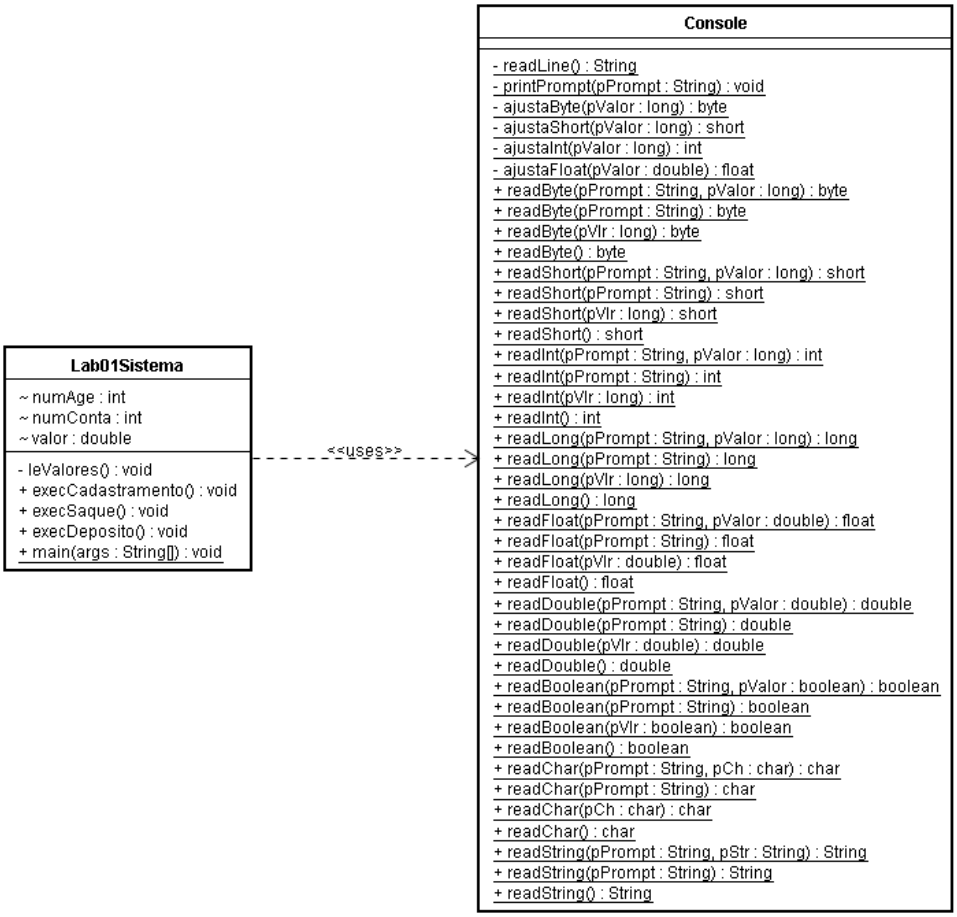


Figura 1.8 – Diagrama de classes do laboratório 1.

1.9.1 Objetivo

Esse laboratório tem como objetivo aplicar os conhecimentos referentes a construção de classes Java, comandos de entrada e saída, classe `String`, estruturas de controle, estruturas de repetição, criação de classe, objetos e métodos.

Introdução:

Antes de iniciarmos o laboratório é necessário:

- Criar o projeto `NomeProjeto` (sugestão: usar nome de uma equipe).
  - Criar o pacote `modulo01.estudodecaso.view;`
  - Criar o pacote `modulo01.estudodecaso.model`
  - Criar o pacote `modulo01.estudodecaso.util`

- Realizar o `import` da classe `Console` para o pacote `modulo01.estudodecaso.util`. É importante observar que também podemos usar a classe `Scanner` presente a partir da versão 1.5 do J2SE para realizar a leitura dos dados pelo teclado.

1.9.2 Definição

Fazer uma classe Java chamada `Lab01Sistema` no pacote `modulo01.estudodecaso.view` que fique em loop mostrando o menu a seguir, até que se entre com a opção de fim.

Layout:

1 - Cadastramento

2 - Saque

3 - Depósito

9 - Fim

Opção :

Para a opção "1 - Cadastramento" deve-se solicitar que o usuário leia os dados referentes a uma conta corrente. Para isso, desenvolver o método chamado `execCadastramento()` que mostre os textos e realize a leitura dos dados conforme o layout a seguir:

Layout método cadastramento

Número da agência: 1555

Número da conta: 1112223

Nome do cliente: Webster Freire

Saldo: 9999.99

Confirma cadastramento (S/N): s

Cadastramento realizado com sucesso.

Para a opção "2 - Saque" deve-se ativar o método `execSaque()`, que não recebe nenhum parâmetro. Inicialmente, o método `execSaque()` deve solicitar o número de agência e conta e o valor do saque. Posteriormente será implementado um código adicional a esse método.

Layout método saque

Número da agência: 1996

Número da conta: 8728232

Valor do saque: 120.99

Confirma saque (S/N): s

Saque efetuado com sucesso.

De maneira similar desenvolver o método `execDeposito()` para a opção "3 - Depósito". Nada será passado como parâmetro para o método, e ele deve solicitar que seja digitado com o teclado o valor para agência, conta e valor do depósito.

Layout método depósito

Número da agência: 1996

Número da conta: 1288273

Valor do depósito: 10000.99

Confirma depósito (S/N): s

Depósito efetuado com sucesso.

Ao final da ativação de cada método deve-se solicitar uma confirmação. Se for digitado 'S' ou 's', o processamento será executado. Se digitado 'N' ou 'n', deve-se sair do método indicando que a operação não foi realizada. Para determinar se essa parte do código está funcionando, colocar nos respectivos métodos as mensagens "<Operação> realizada" ou "<Operação> cancelada" para confirmação e cancelamento, respectivamente. Nas mensagens anteriores "<Operação>", substituir por "Cadastramento", "Saque" ou "Depósito" nos respectivos métodos.

1.9.2.1 Sugestões

Fazer um loop infinito e, caso se entre com a opção "9", terminar com `break` ou usar o comando `System.exit (0)`. Esse comando encerra definitivamente o programa.

A leitura das variáveis poderá ser feita por meio da classe `Console` disponível no pacote `modulo01.estudodecaso.util`. Também é possível realizar a leitura com a classe `Scanner`.

Exemplo do uso da classe `Console`:

```
int numAge = Console.readInt ("Digite o número da agência: ");
```

Essa forma de uso da classe é possível devido aos métodos serem estáticos.

Exemplo do uso da classe `Scanner`:

```
Scanner sc = new Scanner (System.in);
System.out.println ("Digite o número da agência: ");
int numAge = sc.nextInt();
```

O método `main()` deve ser criado na classe `Lab01Sistema`.

Lembre-se de importar a classe `Console` para o pacote `modulo01.estudodecaso.util` do projeto criado para que seja possível usar a classe `Console`. Após a importação, deve-se realizar na classe do laboratório o `import` da classe `Console` por meio do comando `import modulo01.estudodecaso.util.*` antes de iniciar o desenvolvimento da classe. Esse procedimento deverá ser aplicado caso seja usada a classe `Console` para a leitura dos dados. Caso a opção seja pelo uso da classe `Scanner`, deve-se então importar o pacote `util` da linguagem Java por meio do comando `import java.util.Scanner;`. É importante observar que o pacote `modulo01.estudodecaso.util` criado e o pacote `java.util` onde está localizada a classe `Scanner` não se relacionam.

1.9.3 Solução do laboratório 1

O programa 01.37 apresenta a solução do laboratório 1. A classe `Console` refere-se a uma classe que realiza a leitura de dados via teclado. Essa classe oferece uma alternativa aos complexos mecanismos de leitura de dados até a versão 1.4 do Java. A partir da versão 1.5 do J2SE foi desenvolvida a classe `Scanner`, que atua como uma classe substituta da classe `Console`. A classe `Console` deve ser obtida de um site da Internet, enquanto a classe `Scanner` já é instalada no J2SE 1.5. O capítulo 8 aborda os detalhes da classe `Scanner`. A classe `Console` poderá ser obtida na página deste livro no site da Novatec.

É importante observar que o uso da variável `this` nessa solução se refere a uma formalização de que a variável usada é um atributo. O capítulo 3 aborda de forma detalhada a palavra reservada `this`. Outra questão muito importante sobre essa solução é que se não estiver instalada a versão 1.5 ou superior do J2SE na máquina não será possível usar a classe `Scanner`. Para verificar a versão instalada, deve-se executar o comando `java -version` no console do sistema operacional.

Programa 01.37

```
package modulo01.estudodecaso.view;
// Resposta do Laboratório 1
import java.util.Scanner;
// A classe Console precisa estar disponível no pacote modulo01.estudodecaso.util.
import modulo01.estudodecaso.util.Console;
public class Lab01Sistema {
    int numAge;
    int numConta;
    double valor;
    private void leValores() {
        do {
            this.numAge = Console.readInt("Número da agência:  ");
        } while (this.numAge <= 0);
        do {
            this.numConta = Console.readInt("Número da conta:  ");
        } while (this.numConta <= 0);
        do {
            this.valor = Console.readDouble("Valor:            ");
        } while (this.valor <= 0.0);
    }
    public void execCadastramento() {
        String nome = null;
        // usando a classe Scanner como uma opção de leitura
        Scanner sc = new Scanner(System.in);
        char opcao;
        leValores();
        do {
            System.out.println("Nome do cliente:                  ");
            nome = sc.nextLine();
        } while (nome.equals(""));
        opcao = Console.readChar("Confirma cadastramento (S/N):    ");
        if ((opcao == 'S') || ((opcao == 's'))) {
            System.out.println("Cadastramento realizado com sucesso.");
        } else {
            System.out.println("Cadastramento não realizado.");
        }
    }
    public void execSaque() {
        char opcao;
        leValores();
        opcao = Console.readChar("Confirma saque (S/N): ");
        if ((opcao == 'S') || ((opcao == 's'))) {
            System.out.println("Saque efetuado com sucesso.");
        } else {
            System.out.println("Saque não realizado.");
        }
    }
```

```
    public void execDeposito() {
        char opcao;
        leValores();
        opcao = Console.readChar("Confirma depósito (S/N): ");
        if ((opcao == 'S') || ((opcao == 's'))) {
            System.out.println("Depósito efetuado com sucesso.");
        } else {
            System.out.println("Depósito não realizado.");
        }
    }
    public static void main(String[] args) {
        char opcao;
        Lab01Sistema obj = new Lab01Sistema();
        while (true) {
            System.out.println("Entre com a opção desejada");
            System.out.println("1 - Cadastramento");
            System.out.println("2 - Saque");
            System.out.println("3 - Depósito");
            System.out.println("9 - Fim");
            opcao = Console.readChar("Opção: ");
            if (opcao == '9')
                break;
            switch (opcao) {
            case '1':
                obj.execCadastramento();
                break;
            case '2':
                obj.execSaque();
                break;
            case '3':
                obj.execDeposito();
                break;
            default:
                System.out.println("Opção inválida. Reentre.");
            }
        }
    }
}
```