UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
Departamento de Ciência da Computação

Introdução à Ciência da Computação com Java e Orientação a Objetos

Fabio Kon
Alfredo Goldman
Paulo J. S. Silva

Editado e Revisado por:
Raphael Y. de Camargo

São Paulo, 23 de fevereiro de 2006

A nossas esposas e filhos,

fonte de força e alegria em nossas vidas.

Agradecimentos

Este livro não seria possível sem a colaboração de inúmeros alunos e professores do IME/USP.

Leo Kazuhiro Ueda e Nelson Posse Lago atuaram como assistentes de ensino na primeira vez em que esta disciplina foi ministrada e foram responsáveis por inúmeras contribuições. O apêndice sobre o Dr. Java foi preparado pelo Leo. Fabiano Mitsuo Sato, George Henrique Silva e Igor Ribeiro Sucupira foram monitores da disciplina também em 2003 e colaboraram com alguns exercícios.

Raphael Y. de Camargo realizou um excelente trabalho na edição e revisão do livro além de colaborar com alguns exercícios. O Prof. João Eduardo Ferreira, nosso colega no ensino da disciplina de introdução, o Prof. Valdemar Setzer, nosso experiente e sábio colega de departamento, e o Prof. Marcos Chaim, da USPLeste, nos deram inúmeras sugestões úteis que, sempre que possível, foram incorporadas ao texto final.

Agradecemos ao Prof. Walter Savitch da Universidade da California em San Diego por ter autorizado o uso de sua classe para entrada de dados.

Finalmente, agradecemos aos alunos e professores que não foram citados mas que deram sugestões e nos incentivaram a escrever este livro.

Sumário

Lista de Figuras

Prefácio

Caros leitores, sejam bem-vindos ao maravilhoso mundo da Ciência da Computação. Com este livro, vocês terão a oportunidade de aprender os elementos básicos da programação de computadores e terão contato com alguns dos conceitos fundamentais da Ciência da Computação.

Este livro é resultado da experiência dos autores na disciplina de Introdução à Ciência da Computação ministrada no IME/USP utilizando orientação a objetos e a linguagem Java de 2003 a 2006. Desde 2005, esta abordagem também é aplicada na USPLeste com os 180 alunos do curso de Sistemas de Informação. Influenciada pelo método proposto pela ACM (*Association for Computing Machinery*) e pelo IEEE (*Institute of Electrical and Electronics Engineers*) em seu currículo *Objects-first*, nossa abordagem evita o problema de se ensinar a programar sem o uso de objetos para depois exigir do estudante uma mudança de paradigma com a introdução de objetos. Elimina-se assim a necessidade de dizer "esqueça a forma que programamos até agora; agora vamos aprender o jeito correto".

Apresentando os conceitos de orientação a objetos desde o início do ensino de programação, a abordagem adotada aqui permite que o aluno inicie sua formação em programação de forma equilibrada. Aprendendo conceitos tanto algorítmicos quanto estruturais ao mesmo tempo em que toma contato com práticas fundamentais de programação como testes automatizados e o uso de nomes claros para os elementos do código, o leitor obtém uma visão básica porém ampla do que é mais importante para o desenvolvimento de software.

Esperamos que este livro seja útil tanto para alunos de cursos superiores de Informática quanto para profissionais do mercado e outros curiosos que queiram adquirir conhecimentos na área de desenvolvimento de software. Nosso objetivo é que este livro os ajude na fascinante empreitada de aprender a programar de forma elegante e eficaz e que ele permita um bom início rumo a uma formação sólida em programação que é uma condição fundamental que os desafios da informática do século XXI nos impõem.

Abraços e boa jornada!

São Paulo, março de 2006.

Alfredo Goldman, Fabio Kon e Paulo J. S. Silva

Capítulo 1

Teatro de Objetos

Vamos iniciar a nossa jornada ao fascinante mundo da orientação a objetos de forma dramática: com um Teatro de Objetos. Se você está usando este livro em um curso, reserve juntamente com seu professor uma parte da primeira aula para realizar a encenação descrita neste capítulo. Se você trabalha em uma empresa, reuna-se com colegas de trabalho ou com amigos interessados em programação para exercitar suas qualidades dramáticas. Liberte o artista que existe dentro de você!'

O objetivo do Teatro de Objetos é fazer com que os alunos vivenciem um jogo interativo do qual participam vários "objetos" realizando diferentes formas de ações e comunicações. Os conceitos de orientação a objetos empregados no teatro não são explicitamente explicados já no primeiro capítulo mas serão abordados ao longo do livro. Em um curso de Introdução á Ciência da Computação, normalmente a primeira metade da primeira aula é dedicada a uma conversa informal com os alunos explicando quais são os objetivos da disciplina (e do curso inteiro, se for o caso). É sempre interessante também conversar sobre contatos prévios que os alunos tiveram com informática e com programação. É bom deixar claro que este livro pode ser acompanhado por uma pessoa que nunca viu um computador na frente em sua vida, mas que aprender a programar não é uma tarefa fácil, é preciso se empenhar. Na segunda metade da aula, exercitamos as habilidades dramáticas: para ilustrar o funcionamento de um programa de computador complexo, vamos fazer de conta que somos partes de um programa de computador trabalhando em conjunto para atingir um certo objetivo. Se você estiver organizando a encenação com seus colegas, você pode fazer um certo suspense sobre qual é o objetivo (simular uma disputa de pênaltis) e sobre como ele será alcançado.

1.1 Disputa de pênaltis

A "peça" que encenaremos representará uma disputa de pênaltis entre dois times e contará com a participação de cerca de 26 atores desempenhando 8 papéis. Se você não tiver 26 atores disponíveis, você pode elaborar a sua própria adaptação da peça. Os papéis são:

- Técnico (2 atores)
- Juiz (1 ator)
- Bandeirinha (2 atores)
- Gandula (1 ator)

- Jogador. Os jogadores são divididos em dois tipos:
  - Goleiro (2 atores desempenham este papel)
  - Batedor de pênalti (10 atores)
- Torcedor. Os torcedores são divididos em dois tipos:
  - Torcedor educado (4 atores)
  - Torcedor mal-educado (4 atores)

O professor (ou o diretor da peça, que pode ser você) será responsável por escolher as pessoas que desempenharão cada papel. Se houver limitação de espaço, é conveniente que os 8 torcedores fiquem concentrados em uma área separada (por exemplo, em uma suas próprias carteiras, se o local for uma sala de aula) para não tumultuar muito o ambiente. Obviamente, o diretor pode também aumentar ou diminuir o número de torcedores e de batedores de pênalti. Para desempenhar o papel de torcedores, uma boa dica é o diretor escolher pessoas que pareçam bem barulhentas e faladoras (por exemplo, a turma do fundão numa sala de aula :-). Ao escolher os atores, o diretor deverá entregar um cartão preso com um barbante que ficará pendurado no pescoço do ator e conterá informações sobre o papel desempenhado pelo ator. As informações são:

1. Nome do papel
2. Mensagens que o personagem é capaz de entender
3. Atributos do personagem

Os três tipos de informação acima já devem vir pré-escritos à caneta no cartão mas os valores dos atributos do personagem devem ser escritos na hora a lápis pelo diretor. Alguns papéis, como o de juiz, não possuem nenhum atributo. Outros papéis podem possuir um ou mais atributos, o jogador, por exemplo, pode possuir como atributos o nome do time ao qual pertence e o número da sua camisa. No caso de o jogador ser um goleiro, o atributo "número da camisa" pode vir escrito a caneta como valendo 1.

Além do cartão que fica pendurado no pescoço do ator, cada ator recebe um script descrevendo o seu comportamento: para cada mensagem recebida pelo ator, o script descreve quais ações devem ser tomadas pelo ator.

O diretor não deve esquecer de trazer uma bola para esta atividade e deve tomar cuidado para que nenhuma janela seja quebrada durante a realização da atividade. O tempo total estimado para a realização da atividade é de 50 minutos. A maior parte do tempo é gasto explicando-se os procedimentos. A encenação em si, demora entre 5 e 10 minutos dependendo da qualidade dos batedores de pênalti e dos goleiros.

Eis a descrição detalhada dos dados que deverão aparecer nos cartões descritivos e no script (comportamento) de cada um dos 26 atores participantes da encenação. Para obter versões PDF dos cartões prontas para impressão, visite `www.ime.usp.br/~kon/livros/Java/TeatroDeObjetos.pdf`

1. Goleiro

   - Cartão de Identificação
     - Nome do Papel: Goleiro
     - Mensagens que entende: `SuaVez`, `Cobrança Autorizada`, `VenceuOTimeX`
     - Atributos: `Time:` , `Camisa número:` 1
   - Comportamento (*Script*)
     - mensagem: `SuaVez` ⇒ ação: posiciona-se na frente do gol e fica esperando pela cobrança do pênalti.
     - mensagem: `CobrançaAutorizada` ⇒ ação: concentra-se na bola que será chutada pelo adversário e faz de tudo para não deixar que a bola entre no gol. O goleiro não pode se adiantar antes do chute do adversário. Após a cobrança sair do gol para dar lugar ao goleiro adversário.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` é igual ao atributo `Time` no seu cartão de identificação, comemore; caso contrário, xingue o juiz (polidamente! :-).

2. Batedor de Pênalti

   - Cartão de Identificação
     - Nome do Papel: Batedor de Pênalti
     - Mensagens que entende: `SuaVez`, `CobrançaAutorizada`, `VenceuOTimeX`
     - Atributos: `Time:` , `Camisa número:`
   - Comportamento
     - mensagem: `SuaVez` ⇒ ação: posiciona-se na frente da bola e fica esperando pela autorização do juiz.
     - mensagem: `CobrançaAutorizada` ⇒ ação: chuta a bola tentando marcar um gol. Após a cobrança voltar para junto do seu técnico para dar lugar à próxima cobrança.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` é igual ao atributo `Time` no seu cartão de identificação, comemore; caso contrário, xingue o juiz (polidamente! :-).

3. Torcedor Educado

   - Cartão de Identificação
     - Nome do Papel: Torcedor Educado
     - Mensagens que entende: `Ação`, `VenceuOTimeX`
     - Atributos: `Time:` , `Camisa número:` 12
   - Comportamento
     - mensagem: `Ação` ⇒ ação: assista ao jogo emitindo opiniões inteligentes sobre o andamento da peleja e manifestando o seu apreço e desapreço pelo desenrolar da disputa.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` é igual ao atributo `Time` no seu cartão de identificação, comemore e faça um comentário elogioso sobre o seu time; caso contrário, elogie o adversário e parabenize o seu time pelo empenho.

4. Torcedor Mal-Educado

   - Cartão de Identificação
     - Nome do Papel: Torcedor Mal-Educado
     - Mensagens que entende: `Ação`, `VenceuOTimeX`
     - Atributos: `Time:` , `Camisa número: 12`
   - Comportamento
     - mensagem: `Ação` ⇒ ação: assista ao jogo emitindo opiniões duvidosas sobre o andamento da peleja e manifestando a sua raiva ou alegria pelo desenrolar do jogo.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` é igual ao atributo `Time` no seu cartão de identificação, xingue o adversário. Caso contrário, xingue o adversário desesperadamente (mas, por favor, não se esqueça que estamos em uma universidade).

5. Juiz

   - Cartão de Identificação
     - Nome do Papel: Juiz
     - Mensagens que entende: `Ação`, `Irregularidade`
   - Comportamento
     - mensagem: `Ação` ⇒ ação: coordene o andamento da disputa de pênaltis enviando mensagens `SuaVez` para o técnico do time batedor e para o goleiro defensor a cada nova batida. Quando os personagens estiverem a postos, emita a mensagem `CobrançaAutorizada`. Faça a contagem de gols e quando houver um vencedor, emita a mensagem `VenceuOTimeX` onde `TimeX` é o nome do time vencedor. A disputa de pênaltis é feita alternadamente, 5 cobranças para cada time. Se não houver um ganhador após as 5 cobranças, são feitas novas cobranças alternadamente até que haja um vencedor.
     - mensagem: `Irregularidade` ⇒ ação: se a mensagem foi enviada por um Bandeirinha, ignore a cobrança recém-efetuada e ordene que ela seja realizada novamente enviando a mensagem `RepitaCobrança` ao técnico apropriado.

6. Gandula

   - Cartão de Identificação
     - Nome do Papel: Gandula
     - Mensagens que entende: `CobrançaAutorizada`
   - Comportamento
     - mensagem: `CobrançaAutorizada` ⇒ ação: preste atenção à cobrança do pênalti. Após a conclusão da cobrança, pegue a bola e leve-a de volta à marca de pênalti.

7. Técnico

   - Cartão de Identificação
     - Nome do Papel: Técnico
     - Mensagens que entende: `SuaVez`, `RepitaCobrança`, `VenceuOTimeX`
     - Atributos: `Time:`
   - Comportamento
     - mensagem: `SuaVez` ⇒ ação: escolha um dos seus jogadores para efetuar a cobrança e envie a mensagem `SuaVez`. Não repita jogadores nas 5 cobranças iniciais.
     - mensagem: `RepitaCobrança` ⇒ ação: envie a mensagem `SuaVez` para o jogador que acabou de efetuar a cobrança.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` é igual ao atributo `Time` no seu cartão de identificação, comemore; caso contrário, diga que o seu time foi prejudicado pela arbitragem e que futebol é uma caixinha de surpresas.

8. Bandeirinha

   - Cartão de Identificação
     - Nome do Papel: Bandeirinha
     - Mensagens que entende: `Cobrança Autorizada`, `VenceuOTimeX`
   - Comportamento
     - mensagem: `CobrançaAutorizada` ⇒ ação: verifique se o goleiro realmente não avança antes de o batedor chutar a bola. Caso ele avance, envie uma mensagem `Irregularidade` para o Juiz.
     - mensagem: `VenceuOTimeX` ⇒ ação: se `TimeX` não é o nome do time da casa, distancie-se da torcida pois você acaba de se tornar um alvo em potencial.

Capítulo 2

História da Computação

Quais novidades veremos neste capítulo?

- a história da computação;
- evolução da arquitetura do computador;
- evolução das linguagens de programação.

A Computação tem sido considerada uma ciência independente desde meados do século XX. No entanto, desde a antiguidade, a humanidade busca formas de automatizar os seus cômputos e de criar máquinas e métodos para facilitar a realização de cálculos. Neste capítulo, apresentamos inicialmente uma linha do tempo que indica alguns dos principais eventos ocorridos na história da computação e na evolução da arquitetura dos computadores automáticos. Em seguida, apresentamos uma linha do tempo da evolução das linguagens de programação.

2.1 História da Computação e Arquitetura do Computador

- Ábaco (Soroban em japonês) (criado ~2000 anos atrás)
- Blaise Pascal, 1642 (*pai da calculadora*)
  - o primeiro computador digital
  - hoje, diríamos apenas que era uma calculadora super simplificada
  - capaz de somar
  - entrada através de discos giratórios
  - ajudou seu pai, coletor de impostos
- Leibniz
  - computador capaz de somar e multiplicar (inventou em 1671 e construiu em 1694)

  - criou o mecanismo de engrenagens do "vai-um" usado até hoje

- Joseph Marie Jacquard (1752 - 1834)
  - 1790: criou um sistema de tear semi-automático onde os desenhos – de flores, folhas e figuras geométricas – eram codificados em cartões perfurados
  - 1812: havia cerca de 11 mil teares de Jacquard na França
  - a máquina despertou muitos protestos de artesãos que temiam o desemprego que a máquina poderia causar
  - a máquina era capaz de desenhar padrões de alta complexidade como, por exemplo, um auto-retrato de Jacquard feito com 10 mil cartões perfurados

- Charles Babbage (professor de Matemática em Cambridge, Inglaterra)
  - 1812: notou que muito do que se fazia em Matemática poderia ser automatizado
  - iniciou projeto do "Difference Engine" (Máquina/Engenho/Engenhoca de Diferenças)
  - 1822: terminou um protótipo da máquina e obteve financiamento do governo para construí-la
  - 1823: iniciou a construção (usaria motor a vapor, seria totalmente automático, imprimiria o resultado e teria um programa fixo)
  - 1833: depois de 10 anos teve uma idéia melhor e abandonou tudo
  - Nova idéia: máquina **programável**, de propósito geral: "Analytical Engine" (Máquina Analítica)
    * manipularia números de 50 dígitos
    * memória de 1000 dígitos
    * *estações de leitura* leriam cartões perfurados similares ao de tear de Jacquard (nesta época, o auto-retrato de Jacquard pertencia a Babbage)
  - mas ele não conseguiu construí-lo
    * tecnologia mecânica da época era insuficiente
    * pouca gente via a necessidade para tal máquina
  - Ada Lovelace (*mãe da programação*) escreveu programas para o engenho analítico; inventou a palavra algoritmo em homenagem ao matemático Al-Khawarizmi (820 d.C.)
  - **Algoritmo**: seqüência de operações ou comandos que, aplicada a um conjunto de dados, permite solucionar classes de problemas semelhantes. Exemplos: algoritmo da divisão, da raiz cúbica, para resolução de equações do segundo grau, etc.
  - a máquina foi finalmente construída pelo governo inglês nos anos 1990 (e funciona!)

- Herman Hollerith, 1890
  - criou cartões perfurados para uso no censo americano
  - tecnologia levou à criação da International Business Machines (IBM)
  - até hoje dizemos que no final do mês os empregados recebem o “holerite” como sinônimo de contracheque

- Avanços nas calculadoras de mesa ⇒ Em 1890, as máquinas permitiam:
  - acumular resultados parciais
  - armazenamento e reentrada automática de resultados passados (memória)
  - imprimir resultados em papel
- MARK 1, criada em 1937 por Howard Aiken, professor de Matemática Aplicada de Harvard
  - calculadora eletromecânica com motor elétrico
  - pesava 5 toneladas, usava toneladas de gelo para refrigeração
  - multiplicava dois números de 23 dígitos em 3 segundos
- John Vincent Atanasoff, Universidade Estadual de Iowa
  - construiu o que é considerado o primeiro computador digital entre 1937 e 1942
  - calculadora com válvulas a vácuo (240 válvulas)
  - resolvia equações lineares, diferenciais e de balística
  - manipulava números binários
- Rumo à programabilidade
- Alan Turing,
  - Trabalhou para o exército inglês ajudando a quebrar o código criptográfico da máquina Enigma criada pelos alemães
  - Realizou importantes contribuições práticas e teóricas à Ciência da Computação
  - 1912: nasce em Londres
  - 1935: Ganha bolsa para realizar pesquisas no King's College, Cambridge
  - 1936: Elabora "Máquina de Turing", pesquisas em computabilidade
  - 1936-38: Princeton University. Ph.D. Lógica, Álgebra, Teoria dos Números
  - 1938-39: Cambridge. É apresentado à máquina Enigma dos alemães
  - 1939-40: "The Bombe", máquina para decodificação do Enigma criada em Bletchley Park
  - 1939-42: "quebra" Enigma do U-boat, aliados vencem batalha do Atlântico
  - 1943-45: Consultor-chefe anglo-americano para criptologia
  - 1947-48: Programação, redes neurais e inteligência artificial
  - 1948: Manchester University
  - 1949: Pesquisas sobre usos do computador em cálculos matemáticos avançados
  - 1950: Propõe "Teste de Turing" para inteligência de máquinas
  - 1952: Preso por homossexualidade, perde privilégios militares
  - 1953-54: Trabalho não finalizado em Biologia e Física; tem sua reputação e vida destruídas pelos militares ingleses

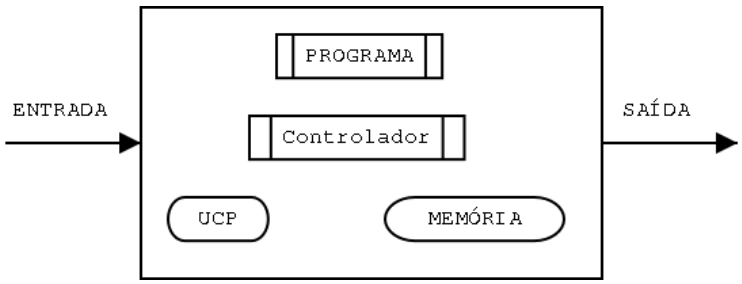

Figura 2.1: Arquitetura do ENIAC

  - 1954: Suicida-se em Wilmslow, Cheshire
  - Livro interessante sobre sua vida e obra: *Alan Turing: the Enigma* de Andrew Hodges, 2000
  - Sítio sobre a vida de Turing mantido pelo autor deste livro: `http://www.turing.org.uk/turing`

- ENIAC (Electronic Numerical Integrator and Computer), 1945

  - por alguns, considerado o primeiro computador eletrônico
  - números de 10 dígitos decimais
  - 300 multiplicações ou 5000 somas por segundo
  - 17486 válvulas, a queima de válvulas era quase que diária
  - 6000 comutadores manuais e centenas de cabos usados na programação
  - programação era muito difícil

- Arquitetura do ENIAC (ver Figura 2.1)

  - programa especificado manualmente em "hardware" com conexões semelhantes àquelas que as velhas telefonistas utilizavam
  - memória de dados separada do controle e separada do programa
  - o controle é formado por circuitos eletroeletrônicos

- John Von Neumann, matemático, 1945

  - estudo abstrato de modelos de computação levou à arquitetura do computador moderno
  - o programa deve ser guardado no mesmo lugar que os dados: na memória
  - Arquitetura de Von Neumann (ver Figura 2.2)
  - hoje em dia, há vários tipos de memória ROM, RAM, flash RAM, etc.)
  - o controlador em memória, levou à idéia de sistema operacional que temos hoje

  - Nunca diga nunca...

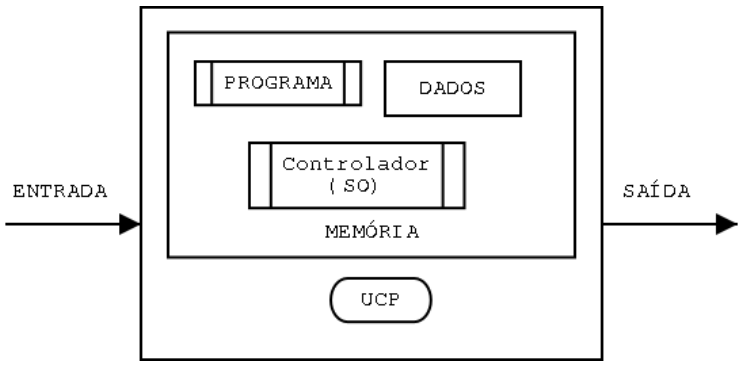

Figura 2.2: Arquitetura de Von Neumann

  * Aiken declarou em 1947 que nunca haveria necessidade de mais de um ou dois computadores programáveis e que os projetos neste sentido deveriam ser abandonados
  * Bill Gates declarou na década de 1980 que a Microsoft nunca desenvolveria um sistema operacional de 32 bits

- Anos 50
  - 1953: IBM vende 15 máquinas baseadas no modelo de Neumann
  - transistores
  - memória magnética ("magnetic core memory")
- Anos 60
  - circuitos impressos / circuitos integrados (*chips*)
  - crescimento segundo lei de Moore, que diz que o número de transistores em circuitos integrados duplica a cada 18 meses. Esta lei continua valendo até hoje
  - computação limitada a poucos computadores de grande porte
- Anos 70
  - indo contra o modelo centralizador da IBM, geração sexo, drogas e rock-and-roll da California exige a democratização da informática
  - revista esquerdista da Universidade da Califórnia em Berkeley *People's Computer Company* defende a criação de computadores pessoais e de cooperativas de informação
  - Steve Jobs cria a Apple em garagem em ~1975 e investe lucros do Apple II em shows de Rock (82)
  - nasce a Microsoft
  - governo da Califórnia apóia microinformática
- Anos 80

  - IBM lança PC (1981)
  - Apple lança MacIntosh (1984)
  - Xerox inventa e Apple comercializa interface baseada em janelas ("Windows")
  - Microsoft cresce comercializando o sistema operacional simplista MS-DOS para IBM-PCs (o DOS era uma versão simplificada do CPM que, por sua vez, era uma versão simplificada do UNIX)
  - algumas empresas começam a esconder código-fonte do software (antes era sempre aberto)
  - começa o movimento do software livre e software aberto

- Anos 90

  - Microsoft pega carona na explosão de vendas de PCs, utiliza técnicas de marketing agressivas (consideradas por alguns como moralmente questionáveis) para controlar o mercado de software, estabelecendo quase um monopólio em certas áreas
  - popularização da Internet e criação da Web
  - intensifica-se o movimento do software livre
  - nasce o Linux e uma nova forma de desenvolvimento de software baseada em comunidades distribuídas através da Internet
  - no final da década, o governo americano percebe o perigo que a Microsoft representa e inicia batalha judicial contra a empresa (em 2002, após a vitória dos conservadores na eleição nos EUA, o governo termina um processo judicial que havia iniciado; as sanções à empresa são mínimas)

- Século XXI

  - computadores de mão, telefones celulares, iPod
  - sistemas embutidos
  - computação ubíqua e pervasiva
  - grande crescimento da empresa Google através de serviços inovadores na Web

2.2 Evolução das Linguagens de Programação

Paralelamente à evolução do hardware dos computadores eletrônicos, ocorreu também a evolução do software e das linguagens de programação utilizadas para desenvolvê-lo. Inicialmente, as linguagens estavam bem próximas do funcionamento dos circuitos do hardware; paulatinamente, as linguagens foram se aproximando da linguagem natural utilizada pelos humanos em seu dia-a-dia.

- A máquina de Babbage só poderia ser programada com a troca física de engrenagens
- 1945, no ENIAC, a programação era feita mudando chaves e trocando a posição de cabos
- 1949-50, primeira linguagem binária, a programação era feita mudando os “comandos” de zero a um, e vice-versa

- 1951, Grace Hooper cria o primeiro compilador, A0, programa que transforma comandos em 0s e 1s
- 1957, primeira linguagem de programação de alto nível: FORTRAN (*FORmula TRANslating*) (John Backus da IBM)
- 1958, criação de um padrão universal de linguagem: ALGOL 58 (*ALGOritmic Language*) (origem da maioria das linguagens modernas). Primeira linguagem estruturada
- 1958, John McCarthy do MIT cria o LISP (*LISt Processing*), inicialmente projetada para uso em inteligência artificial. Nela tudo se baseia em listas. Ainda é usada hoje em dia
- 1959, FORTRAN era eficaz para manipulação de números, mas não para entrada e saída: foi criada COBOL (*COmmon Bussines Oriented Language*)
- 1964, criação do BASIC (*Beginners All-purpose Symbolic Instruction Code*)
- 1965, criação de uma linguagem específica para simulação (SIMULA I) por Ole-Johan Dahl and Kristen Nygaard da Universidade de Oslo. É considerada a base das linguagens orientadas a objetos
- 1966, criação da linguagem Logo para desenhos gráficos (a linguagem da tartaruga)
- 1967, Simula 67, uma linguagem de uso geral incluindo todos os conceitos fundamentais de orientação a objetos
- 1968, criação da linguagem PASCAL por Niklaus Wirth. Principal interesse: linguagem para o ensino. Combinou as melhores características de Cobol, Fortran e Algol, foi uma linguagem bem utilizada
- 1970, PROLOG, linguagem para programação lógica
- 1972, criação da linguagem C (Denis Ritchie). Supriu as deficiências da linguagem Pascal e teve sucesso quase imediato
- 1972-1980, linguagem Smalltalk, desenvolvida por Alan Kay, da Xerox, utilizando as melhores características de LISP, Simula 67, Logo e ambientes gráficos como Sketchpad; Orientação a Objetos ganha força
- 1983, criadas extensões de C incluindo suporte para OO: C++ e Objective-C
- 1987, linguagens baseadas em scripts, como Perl, desenvolvida por Larry Wall. Ferramentas de UNIX como sed e awk não eram suficientes
- 1994, Java é lançada pela Sun como a linguagem para a Internet
- 2000, lançamento do C# pela Microsoft, combinando idéias das linguagens Java e C++
- 2005, lançamento da versão 1.5 de Java

Neste livro, apresentamos, de forma muito sucinta, algumas das principais linguagens de programação. Para referências mais completas visite `www.princeton.edu/~ferguson/adw/programming_languages.shtml` e `en.wikipedia.org/wiki/Programming_language`.

Capítulo 3

Conversor de Temperaturas

Quais novidades veremos neste capítulo?

- primeiro programa em Java.

3.1 Analogia entre dramatização da disputa de pênaltis e Programação Orientada a Objetos

Terminologia Teatral	Terminologia de Programação Orientada a Objetos
personagem (papel)	classe (tipo)
ator	objeto
envio de mensagem	envio de mensagem, chamada de método ou chamada de função

Na dramatização, podíamos enviar uma mensagem (dizer alguma coisa) para um ator. Na programação orientada a objetos, podemos enviar uma mensagem para (ou chamar um método de) um objeto.

Os cartões de identificação definem os papéis dos atores e os scripts especificam o comportamento dos atores no decorrer da peça. A linguagem Java permite que especifiquemos a mesma coisa. Um cartão de identificação tem 3 partes, essas mesmas 3 partes aparecem na definição de uma classe em Java. Por exemplo, o bandeirinha em Java seria mais ou menos assim:

```
class Bandeirinha
{
  CobrançaAutorizada
  {
    // Verifica se o goleiro realmente não avança antes de o batedor...
  }
  VenceuOTime (NomeDoTime)
  {
    // Se NomeDoTime não é o nome do time da casa, distancie-se da torcida...
  }
}
```

3.2 Um exemplo real em Java: um conversor de Celsius para Fahrenheit

Sempre o primeiro passo antes de programar é analisar o problema.

$$\frac{F-32}{9} = \frac{C}{5} \Rightarrow F-32 = \frac{9}{5}C \Rightarrow F = \frac{9}{5}C+32$$

Traduzindo esta fórmula para Java temos `F=9*C/5+32`. A seguir iremos criar diversas classes para realizar a conversão entre Celsius e Fahrenheit.

1. **Primeira tentativa:** programa em Java para converter 40 graus Celsius para Fahrenheit.

```
class Conversor
{
  int celsiusParaFahrenheit()
  {
    return 9 * 40 / 5 + 32;
  }
}
```

   - para executar este conversor dentro do DrJava[1], podemos digitar o seguinte na janela do interpretador (chamada de *interactions*):

```
Conversor c1 = new Conversor();
c1.celsiusParaFahrenheit()
```

     o DrJava imprimirá o valor devolvido pelo método `celsiusParaFahrenheit` do objeto `c1`.

   - limitação: sempre converte a mesma temperatura.

2. **Segunda tentativa:** conversor genérico de temperaturas Celsius -> Fahrenheit. É capaz de converter qualquer temperatura de Fahrenheit para Celsius.

```
class Conversor2
{
  int celsiusParaFahrenheit(int c)
  {
    return 9 * c / 5 + 32;
  }
}
```

   - para executar este conversor, podemos digitar o seguinte na janela do interpretador:

```
Conversor2 c2 = new Conversor2();
c2.celsiusParaFahrenheit(100)
```

     o DrJava imprimirá o valor devolvido pelo método `celsiusParaFahrenheit` do objeto `c2`.

[1] Para mais informações sobre o DrJava consulte o Apêndice A.

- limitação: essa classe manipula apenas números inteiros. Mas, em geral, temperaturas são números reais, fracionários, então números inteiros não são suficientes. Quando o computador opera com números inteiros, os números são truncados, ou seja, 30.3 se torna 30 e 30.9 também se torna 30. Devido a esta limitação, se tivéssemos escrito a fórmula como $\frac{9}{5} * C + 32$, o cálculo seria errado, uma vez que $\frac{9}{5} = 1$ se considerarmos apenas a parte inteira da divisão. Assim, o programa calcularia apenas $1 * C + 32$.

Quando precisamos trabalhar com números reais, usamos números de ponto flutuante (*floating point numbers*). Esse nome se deriva do fato de que internamente os números são representados de forma parecida a potências de 10 onde, variando o expoente da potência, movemos (flutuamos) o ponto decimal para a esquerda ou para a direita. De *floating point* vem o tipo `float` da linguagem Java. Números do tipo `float` são armazenados em 4 bytes e tem uma precisão de 23 bits (o que equivale a aproximadamente 7 casas decimais). No entanto, quase sempre são utilizados números de ponto flutuante com precisão dupla que são chamados de `double`. Em Java, um `double` ocupa 8 bytes e tem precisão de 52 bits, o que equivale a aproximadamente 16 casas decimais. Daqui para frente sempre que precisarmos de números fracionários, vamos utilizar o tipo `double`. Mas lembre-se de que as variáveis do tipo `double` são apenas uma aproximação do número real, como podemos ver no exemplo abaixo:

```
> double x = 1.0
1.0
> x = x / 59049
1.6935087808430286E-5
> x = x * 59049
0.9999999999999999
```

Após dividir o número 1.0 pelo número $3^{10} = 59049$ e depois multiplicá-lo por este mesmo número, obtivemos um valor diferente do inicial. Por este motivo, devemos ter muito cuidado com os erros de arredondamento presentes quando utilizamos números de ponto flutuante.

3. **Terceira tentativa:** conversor genérico usando `double`

```
class Conversor3
{
  double celsiusParaFahrenheit(double c)
  {
    return 9.0 * c / 5.0 + 32.0;
  }
}
```

- para executar este conversor, podemos digitar o seguinte na janela do interpretador:

```
Conversor3 c3 = new Conversor3();
c3.celsiusParaFahrenheit(37.8);
```

- limitação: só faz conversão em um dos sentidos.

4. **Quarta e última versão:** conversão de mão dupla

```
class Conversor4
{
  double celsiusParaFahrenheit(double c)
  {
    return 9.0 * c / 5.0 + 32.0;
  }
  double fahrenheitParaCelsius(double f)
  {
    return 5.0 * (f - 32.0) / 9.0;
  }
}
```

- para executar este conversor, podemos digitar o seguinte na janela do interpretador:

```
Conversor4 c4 = new Conversor4();
c4.celsiusParaFahrenheit(37.8);
c4.FahrenheitParaCelsius(-20.3);
```

Note que, para que o seu programa fique bem legível e elegante, é muito importante o alinhamento dos abre-chaves `{` com os fecha-chaves `}` correspondentes. Em programas mais complexos, esse correto alinhamento (indentação) ajuda muito a tornar o programa mais claro para seres humanos.

Exercícios

1. Crie uma classe `Conversor5` que inclua também a escala Kelvin (K). Esta classe deve conter conversores entre as três escalas de temperatura (Celsius, Fahrenheit e Kelvin), totalizando seis funções. A relação entre as três escalas é dada por:

$$\frac{F-32}{9} = \frac{C}{5} = \frac{K-273}{5}$$

2. Iremos agora construir uma classe que calcula o valor de um número ao quadrado e ao cubo. Para tal, crie uma classe que contenha dois métodos. O primeiro método deve receber um número e devolver o seu quadrado e o segundo método deve receber um número e devolver o seu valor ao cubo. Escolha nomes para a classe e métodos que facilitem a compreensão de seu funcionamento.

3. Neste exercício iremos simular um jogo de tênis. A partida de tênis é composta por diversos papéis: jogador, juiz de cadeira, juiz de linha, treinador, gandula, torcedor.

   Para cada um destes papéis, descreva as mensagens que os atores de cada papel devem receber, e seu comportamento para cada uma delas. Utilize como modelo a descrição do papel `Bandeirinha` apresentada no início deste capítulo.

Capítulo 4

Testes Automatizados

Quais novidades veremos neste capítulo?

- comando `if` e `else`;
- comparações == (igualdade) e != (diferença);
- definição de variáveis inteiras e de ponto flutuante;
- impressão de texto;
- comentários.

Desde o início, a computação sempre esteve sujeita erros. O termo *bug*, para denotar erro, tem uma origem muito anterior (vem de um inseto que causava problemas de leitura no fonográfo de Thomas Edison em 1889). Várias outras histórias reais, ou nem tanto, também apareceram no início da informática. Infelizmente, é muito difícil garantir que não existam erros em programas. Uma das formas de se garantir que certos erros não vão ocorrer é testando algumas situações.

Apesar da célebre afirmação de Edsger Dijkstra [1] de que testes podem apenas mostrar a presença de erros e não a sua ausência, eles podem ser os nossos grandes aliados no desenvolvimento de programas corretos. Intuitivamente, quanto mais testes fizermos em um programa e quanto mais abrangentes eles forem, mais confiantes podemos ficar com relação ao seu funcionamento. Por outro lado, podemos usar o próprio computador para nos ajudar, isto é, podemos criar testes automatizados. Em vez de fazermos os testes "na mão", faremos com que o computador seja capaz de verificar o funcionamento de uma seqüência de testes. Veremos neste capítulo como desenvolver testes automatizados, passo a passo, para os conversores de temperatura vistos anteriormente.

No início da computação não havia uma preocupação muito grande com os testes, que eram feitos de forma manual pelos próprios programadores. Os grandes testadores eram os usuários finais. É interessante notar que isto acontece com alguns produtos ainda hoje. Com o aparecimento da Engenharia de Software ficou clara a necessidade de se efetuarem testes, tanto que em várias empresas de desenvolvimento de software existe a figura do testador, responsável por tentar encontrar erros em sistemas. Hoje existe uma tendência para se

[1] Um dos mais influenets membros da geração dos criadores da Ciência da Computação. A página `http://www.cs.utexas.edu/users/EWD` contém cópias de vários de seus manuscritos.

considerar que testes automatizados são muito importantes, devendo ser escritos mesmo antes de se escrever o código propriamente dito, técnica esta chamada de *testes a priori*.

Veremos como testar os diversos conversores de temperatura. Como testar o nosso primeiro programa em Java (Conversor).

```
Conversor c1 = new Conversor()

// a resposta esperada é o equivalente a 40C em F
if (c1.celsiusParaFahrenheit() == 104)
  System.out.println("Funciona");
else
  System.out.println("Não funciona");
```

Note que para fazer o teste utilizamos o comando condicional `if else`. O formato genérico deste comando é o seguinte.

```
if (CONDIÇÃO)
  COMANDO-1;
else
  COMANDO-2;
```

Se a `CONDIÇÃO` é verdadeira, o `COMANDO-1` é executado, caso contrário, o `COMANDO-2` é executado.

A classe `Conversor2` possui um método que aceita um parâmetro, veja o teste abaixo:

```
Conversor2 c2 = new Conversor2();

// cria duas variáveis inteiras
int entrada = 40;
int resposta = 104;

// a resposta esperada é o equivalente à entrada C em F
if (c2.celsiusParaFahrenheit(entrada) == resposta)
  System.out.println("Funciona");
else
  System.out.println("Não funciona");
```

Note que, para realizar o teste acima, definimos duas variáveis inteiras chamadas de `entrada` e `resposta`. A linha

```
int entrada = 40;
```

faz na verdade duas coisas. Primeiro, ela declara a criação de uma nova variável (`int entrada;`) e, depois, atribui um valor inicial a esta variável (`entrada = 40;`). Na linguagem Java, se o valor inicial de uma variável não é atribuído, o sistema atribui o valor 0 à variável automaticamente. Note que isso não é necessariamente verdade em outras linguagens como, por exemplo, C e C++.

Podemos também testar o `Conversor2` para outros valores. Por exemplo, para as entradas (e respostas): 20 (68) e 100 (212).

```
entrada = 20; // como as variáveis já foram declaradas acima, basta usá-las
resposta = 68;
if (c2.celsiusParaFahrenheit(entrada) == resposta)
  System.out.println("Funciona");
else
  System.out.println("Não funciona");
```

```
entrada = 100;
resposta = 212;
if (c2.celsiusParaFahrenheit(entrada) == resposta)
  System.out.println("Funciona");
else
  System.out.println("Não funciona");
```

No programa acima o texto *Funciona* será impresso na tela a cada sucesso, o que poderá causar uma poluição visual caso tenhamos dezenas ou centenas de testes. O ideal para um testador é que ele fique silencioso caso os testes dêem certo e chame a atenção caso ocorra algum erro. Podemos então mudar o programa para:

```
Conversor2 c2 = new Conversor2();

int entrada = 40;
int resposta = 104;
if (c2.celsiusParaFahrenheit(entrada) != resposta)
  System.out.println("Não funciona para 40");

entrada = 20;
resposta = 68;
if (c2.celsiusParaFahrenheit(entrada) != resposta)
  System.out.println("Não funciona para 20");

entrada = 100;
resposta = 212;
if (c2.celsiusParaFahrenheit(entrada) != resposta)
  System.out.println("Não funciona para 100");

System.out.println("Fim dos testes");
```

Note que o comando `if` acima foi utilizado sem a parte do `else`, o que é perfeitamente possível. Adicionamos também uma linha final para informar o término dos testes. Ela é importante no caso em que todos os testes dão certo para que o usuário saiba que a execução dos testes foi encerrada.

Uma forma de simplificar os comandos de impressão é usar a própria entrada como parâmetro, o que pode ser feito da seguinte forma:

```
System.out.println("Não funciona para " + entrada);
```

Criaremos agora os testes para o `Conversor4`. Mas, agora, devem ser testados os seus dois métodos. Introduziremos um testador automático criando uma classe com apenas um método que faz o que vimos.

```
class TestaConversor4
{
  int testaTudo()
  {
    Conversor4 c4 = new Conversor4();
    double celsius = 10.0;
    double fahrenheit = 50.0;

    if (c4.celsiusParaFahrenheit(celsius) != fahrenheit)
      System.out.println("C-> F não funciona para " + celsius);
    if (c4.fahrParaCelsius(fahrenheit) != celsius)
      System.out.println("F-> C não funciona para " + fahrenheit);
    celsius = 20.0;
    fahrenheit = 68.0;
```

```
    if (c4.celsiusParaFahrenheit(celsius) != fahrenheit)
      System.out.println("C-> F não funciona para " + celsius);
    if (c4.fahrParaCelsius(fahrenheit) != celsius)
      System.out.println("F-> C não funciona para " + fahrenheit);
    celsius = 101.0;
    fahrenheit = 213.8;
    if (c4.celsiusParaFahrenheit(celsius) != fahrenheit)
      System.out.println("C-> F não funciona para " + celsius);
    if (c4.fahrParaCelsius(fahrenheit) != celsius)
      System.out.println("F-> C não funciona para " + fahrenheit);
    System.out.println("Final dos testes");
    return 0;
  }
}
```

Comentários: Você deve ter notado que, no meio de alguns exemplos de código Java deste capítulo, nós introduzimos uns comentários em português. Comentários deste tipo são possíveis em praticamente todas as linguagens de programação e servem para ajudar os leitores do seu código a compreender mais facilmente o que você escreveu.
Comentários em Java podem ser inseridos de duas formas. O símbolo `//` faz com que tudo o que aparecer após este símbolo até o final da linha seja ignorado pelo compilador Java; este é o formato que utilizamos neste capítulo e que utilizaremos quase sempre neste livro.
A segunda forma é através dos símbolos `/*` (que indica o início de um comentário) e `*/` (que indica o final de um comentário). Neste caso, um comentário pode ocupar várias linhas. Esta técnica é também útil para desprezarmos temporariamente um pedaço do nosso programa. Por exemplo, se queremos testar se uma classe funciona mesmo se removermos um de seus métodos, não é necessário apagar o código do método. Basta acrescentar um `/*` na linha anterior ao início do método, acrescentar um `*/` logo após a última linha do método e recompilar a classe. Posteriormente, se quiseremos reinserir o método, basta remover os símbolos de comentários.
Outro uso para este último tipo de comentários é quando desejamos testar uma versão alternativa de um método. Comentamos a versão antiga do método, inserimos a versão nova e testamos a classe. Podemos aí decidir qual versão utilizar e, então, remover a que não mais será usada.

Agora, vamos treinar o que aprendemos com alguns exercícios. Em particular, os exercícios 3, de refatoração, e 4, que indica quais casos devem ser testados, são muito importantes.

Exercícios

1. Escreva uma classe `TestaConversor3` para testar a classe `Conversor3`.

2. **Importante**: Refatore a classe `TestaConversor4` de modo a eliminar as partes repetidas do código. *Dica:* Crie um método que realiza o teste das duas funções.

3. **Importante**: Podemos criar testes para objetos de uma classe que não sabemos como foi implementada. Para tal, basta conhecermos suas entradas e saídas. Escreva um teste automatizado para a seguinte classe:

```
class Contas
{
  double calculaQuadrado(double x);
  double calculaCubo(double x);
}
```

   Ao escrever o teste, pense em testar vários casos com características diferentes. Em particular, não deixe de testar os casos limítrofes que é, geralmente, onde a maioria dos erros tendem a se concentrar. Na classe `Contas` acima, os casos interessantes para teste seriam, por exemplo:

   - $x = 0$;
   - x é um número positivo pequeno;
   - x é um número negativo pequeno;
   - x é um número positivo grande;
   - x é um número negativo grande.

   Além disso, seria interessante que, em alguns dos casos acima, x fosse um número inteiro e em outros casos, um número fracionário.

4. *Programas confiáveis:* Se conhecemos uma implementação e sabemos que a mesma funciona de maneira confiável, ela pode servir de base para o teste de outras implementações. Caso duas implementações diferentes produzam resultados distintos para os mesmos dados de entrada, podemos dizer que pelo menos uma das duas está errada;
   Supondo que a classe `Contas` do exercício anterior é confiável, escreva um teste automatizado que utiliza esta classe para testar a classe `ContasNaoConfiavel` dada abaixo:

```
class ContasNãoConfiável
{
  double calculaQuadrado(double x);
  double calculaCubo(double x);
}
```

Resoluções

- Exercício 2

```
class TesteConversor4Refatorado
{
  int testePontual(double celsius, double fahrenheit)
  {
    Conversor c4 = new Conversor4();

    if (c4.celsiusParaFahrenheit(celsius) != fahrenheit)
      System.out.println("C–> F não funciona para " + celsius);
    if (c4.fahrParaCelsius(fahrenheit) != celsius)
      System.out.println("F–> C não funciona para " + fahrenheit);
```

```
    return 0;
  }

  int testaTudo()
  {
    double celsius = 10.0;
    double fahrenheit = 50.0;
    testePontual(celsius, fahrenheit);
    celsius = 20.0;
    fahrenheit = 68.0;
    testePontual(celsius, fahrenheit);
    celsius = 101.0;
    fahrenheit = 213.8;
    testePontual(celsius, fahrenheit);
    System.out.println("Final dos testes");
    return 0;
  }
}
```

- Exercício 3

```
class TesteContas
{
  int teste()
  {
    Contas contas = new Contas();
    double valor = 4.0;

    if (contas.calculaQuadrado(valor) != 16.0)
      System.out.println("Não funciona para calcular o quadrado de 4");
    if (contas.calculaCubo(valor) != 64.0)
      System.out.println("Não funciona para calcular 4 ao cubo");
    return 0;
  }
}
```

Capítulo 5

Métodos com Vários Parâmetros

Quais novidades veremos neste capítulo?

- Métodos com vários parâmetros;
- Atributos;
- Métodos que devolvem nada (`void`);
- Escolha de bons nomes.

No último capítulo, vimos um método que recebe vários parâmetros. Apesar de não estarmos apresentando nenhuma novidade conceitual, vamos reforçar a possibilidade da passagem de mais de um parâmetro.

Ao chamarmos um método que recebe múltiplos parâmetros, assim como no caso de métodos de um parâmetro, devemos passar valores do mesmo tipo que aqueles que o método está esperando. Por exemplo, se um objeto (ator) tem um método (entende uma mensagem) que tem como parâmetro um número inteiro, não podemos enviar a este objeto um número `double`. Apesar de ser intuitivo, também vale ressaltar que a ordem dos parâmetros é importante, de modo que na chamada de um método devemos respeitar a ordem que está na sua definição.

Vamos começar com o cálculo da área de uma circunferência, onde é necessário apenas um parâmetro, o raio:

```
class Círculo1
{
  double calculaÁrea(double raio)
  {
    return 3.14159 * raio * raio;
  }
}
```

Nota Linguística: Em Java, o nome de variáveis, classes e métodos pode conter caracteres acentuados. Muitos programadores evitam o uso de acentuação mesmo ao usar nomes em português. Este costume advém de outras linguagens mais antigas, como C, que não permitem o uso de caracteres acentuados. Você pode usar caracteres acentuados se quiser mas lembre-se que, se você tiver uma variável chamada `área`, ela será diferente de outra chamada `area` (sem acento); ou seja, se você decidir usar acentos, deverá usá-los consistentemente sempre.

Podemos sofisticar o exemplo calculando também o perímetro, com o seguinte método:

```
double calculaPerímetro(double raio)
{
  return 3.14159 * 2.0 * raio;
}
```

O seguinte trecho de código calcula e imprime a área e o perímetro de uma circunferência de raio 3.0:

```
Círculo1 c = new Círculo1();

System.out.println(c.calculaÁrea(3.0));
System.out.println(c.calculaPerímetro(3.0));
```

Vejamos agora uma classe `Retângulo` que define métodos para o cálculo do perímetro e da área de um retângulo. Neste caso são necessários dois parâmetros.

```
class Retângulo1
{
  int calculaÁrea(int lado1, int lado2)
  {
    return lado1 * lado2;
  }
  int calculaPerímetro(int lado1, int lado2)
  {
    return 2 * (lado1 + lado2);
  }
}
```

As chamadas para os métodos podem ser da seguinte forma:

```
Retângulo1 r = new Retângulo1();

System.out.println(r.calculaÁrea(2, 3));
System.out.println(r.calculaPerímetro(2, 3));
```

5.1 Atributos

Assim como no exemplo do teatrinho dos objetos onde os jogadores tinham como característica o time e o número da camisa (atributos), também podemos ter algo semelhante para o caso dos retângulos. Podemos fazer com que os dados sobre os lados sejam características dos objetos. Isto é feito da seguinte forma:

```
class Retângulo2
{
  int lado1;
  int lado2;

  int calculaÁrea()
  {
    return lado1 * lado2;
  }
  int calculaPerímetro()
  {
    return 2 * (lado1 + lado2);
  }
}
```

Na classe acima, `lado1` e `lado2` são atributos que farão parte de todas as instâncias da classe `Retângulo2`. Um atributo, por ser definido fora dos métodos, pode ser acessados a partir de qualquer método da classe onde ele é definido. Atributos são também chamados de propriedades ou variáveis de instância.

No entanto, no exemplo acima, ficou faltando algum método para carregar os valores dos lados nas variáveis `lado1` e `lado2`. Poderíamos escrever um método para cada atributo ou então, apenas um método para definir o valor de ambos como no exemplo seguinte:

```
void carregaLados(int l1, int l2) // este método não devolve nenhum valor
{
  lado1 = l1;
  lado2 = l2;
}
```

Nota Linguística: `void` em inglês significa vazio, nada ou nulidade. Assim, quando temos um método que não devolve nenhum valor, como, por exemplo, o método `carregaLados` acima, colocamos a palavra `void` antes da definição do método para indicar que ele não devolverá nada. Neste caso, não é necessário incluir o comando `return` no corpo do método. Mesmo assim, podemos usar um comando de retorno vazio para provocar o fim prematuro da execução do método, como em: `return ;` . O uso do `return` vazio nem sempre é recomendado pois, em alguns casos, pode gerar código pouco claro, mas é uma opção permitida pela linguagem.

O funcionamento de um objeto da classe `Retângulo2` pode ser verificado com o código abaixo:

```
Retângulo2 r = new Retângulo2();

r.carregaLados(3, 5);
System.out.println(r.calculaÁrea());
System.out.println(r.calculaPerímetro());
```

Para não perdermos o hábito, também veremos como testar esta classe. Neste caso, temos duas opções: criar diversos objetos do tipo `Retângulo2`, um para cada teste, ou carregar diversos lados no mesmo objeto. Abaixo, a cada chamada de `TestePontual`, um novo `Retângulo2` é criado:

```
class TestaRetângulo
{
  void testePontual(int l1, int l2)
  {
    Retângulo2 r = new Retângulo2();

    r.carregaLados(l1, l2);
    if (r.calculaÁrea() != l1 * l2)
      System.out.println("Não funciona área para lados "
                         + l1 + " e " + l2);
    if (r.calculaPerímetro() != 2 * (l1 + l2))
      System.out.println("Não funciona perímetro para lados "
                         + l1 + " e " + l2);
  }

  void testaTudo ()
  {
    testePontual(10, 20);
    testePontual(1, 2);
    testePontual(3, 3);
  }
}
```

Da mesma forma que os lados foram atributos para a classe `Retângulo2`, podemos fazer o mesmo para a classe `Círculo1`.

```
class Círculo2
{
  double raio;

  void carregaRaio(double r)
  {
    raio = r;
  }
  double calculaÁrea()
  {
    return 3.14159 * raio * raio;
  }
  double calculoPerímetro()
  {
    return 3.14159 * 2.0 * raio;
  }
}
```

Assim como vimos anteriormente podemos também utilizar objetos de uma classe sem conhecer a sua implementação. Por exemplo, suponha que temos acesso a uma classe `Cálculo` que possui o seguinte método:

```
int calculaPotência(int x, int n);
```

Para calcular a potência poderíamos ter o seguinte trecho de código:

```
Cálculo c = new Cálculo();

System.out.println("2 elevado a 5 é igual a: " + c.calculaPotência(2, 5));
```

5.2 A importância da escolha de bons nomes

A característica mais importante em um programa de computador é a clareza do seu código. Quanto mais fácil for para um programa ser entendido por um ser humano que o leia, melhor será o código. Portanto, é fundamental que os nomes das variáveis, classes e métodos sejam escolhidos com muito cuidado e critério. Programadores inexperientes (e alguns não tão inexperientes assim mas descuidados) tendem a não dar importância a este ponto que é essencial para o desenvolvimento de software de boa qualidade.

Os nomes das variáveis devem indicar claramente o seu significado, isto é, devem explicar o que é o conteúdo que elas guardam. Por exemplo, na classe `Círculo1` descrita no início deste capítulo, a escolha do nome `raio` para a variável é perfeita porque ela não deixa a menor dúvida sobre o que a variável irá guardar. Se ao invés de `raio`, o programador tivesse escolhido `x` como nome da variável, isto seria péssimo pois o nome `x` não traria informação nenhuma para o leitor do programa sobre o seu significado. Se o programador tivesse dado o nome `r` para a variável, a escolha não chegaria a ser péssima, pois o `r` traz alguma informação, mesmo que sutil. Mas, mesmo assim, não seria uma solução tão boa quanto nomear como `raio`, pois este não deixa a menor dúvida.

Portanto, é importante não ter preguiça de pensar em bons nomes e de escrever nomes um pouco mais longos. Por exemplo, o nome `númeroDeAlunos` é muito melhor do que `nda` e melhor do que `numAlun`. Quanto menor o esforço mental do leitor para compreender o programa, maior serão os ganhos a médio e longo prazos. A economia de alguns segundos que obtemos com o uso de nomes abreviados pode facilmente se tornar em horas de dor de cabeça no futuro.

A escolha dos nomes de classes e métodos também deve ser feita criteriosamente. Normalmente as classes representam tipos de objetos do mundo real ou do mundo virtual que estamos criando dentro do computador. Portanto, o mais comum é usarmos substantivos como nome para classes (embora esta não seja uma regra obrigatória). Os nomes das classes tem que explicar muito bem qual o tipo de objeto que ela irá representar. Por exemplo, chamar uma classe de `Aluno` e depois usar objetos desta classe para guardar as notas de um aluno e calcular sua média, não é uma boa escolha. Provavelmente, neste caso, `NotasDeAluno` seria um nome melhor.

Os métodos representam ações que são realizadas, normalmente manipulando os atributos da classe. Portanto, normalmente utiliza-se verbos no imperativo para nomear os métodos (novamente, esta não é uma regra obrigatória). Por exemplo, os métodos `calculaÁrea`, `calculaPerímetro`, `carregaLados` e `testaTudo` atendem bem a este critério.

Portanto, antes de escrever sua próxima classe, método ou variável, não se esqueça: **nomes são importantes!**

Exercícios

1. Neste exercício, construiremos uma classe que calcula a média aritmética de 4 notas e diz se o dono das notas foi aprovado, está de recuperação ou foi reprovado. Por exemplo,

   - Entrada:

   ```
   8.7, 7.2, 9.3, 7.4
   5.2, 3.4, 6.5, 2.1
   3.4, 5.1, 1.1, 2.0
   ```

- Saída:

```
Média: 8.15 -> aprovado.
Média: 4.3 -> recuperação.
Média: 2.9 -> reprovado.
```

Para isso, crie uma classe `Aluno` com métodos que carreguem as 4 notas em variáveis `p1`, `p2`, `p3`, `p4` e um método responsável por calcular a média aritmética das notas e dar o "veredito".

2. Escreva uma classe `Olá` com um único método `cumprimenta` que, a cada chamada, cumprimenta o usuário de uma entre 3 maneiras diferentes. *Dica:* use um atributo para, dependendo de seu valor, escolher qual das maneiras será usada; depois de imprimir a mensagem, altere o valor do atributo.

3. Construa a classe `Inteiro` que representa um número inteiro. Essa classe deve ter os seguintes atributos e métodos:

Classe `Inteiro`

– Atributos:

  * `int valor`
    Valor do inteiro representado.

– Métodos para interação com o usuário da classe:

  * `void carregaValor(int v)`
    Muda o valor representado por este objeto. O novo valor deve ser `v`.
  * `int devolveValor()`
    Devolve o valor representado por este objeto.
  * `int devolveValorAbsoluto()`
    Devolve o valor absoluto do valor representado por este objeto.
  * `void imprime()`
    Imprime algo que representa este objeto. Sugestão: imprima o seu valor.

Exemplo de uso no DrJava:

```
> Inteiro i = new Inteiro();
> i.carregaValor(14);
> i.devolveValor()
14
> i.carregaValor(-473158);
> i.devolveValor()
-473158
> i.devolveValorAbsoluto()
473158
> i.imprime();
```

```
Valor: -473158.
```

4. Acrescente à classe `Inteiro` algumas operações matemáticas, implementando os seguintes métodos:

Classe `Inteiro`

– Mais métodos para interação com o usuário da classe:

* `int soma(int v)`
  Soma v ao valor deste objeto (valor + v). Este objeto passa a representar o novo valor, que também deve ser devolvido pelo método.
* `int subtrai(int v)`
  Subtrai v do valor deste objeto (valor - v). Este objeto passa a representar o novo valor, que também deve ser devolvido pelo método.
* `int multiplicaPor(int v)`
  Multiplica o valor deste objeto por v (valor * v). Este objeto passa a representar o novo valor, que também deve ser devolvido pelo método.
* `int dividePor(int divisor)`
  Verifica se divisor é diferente de zero. Se não, imprime uma mensagem de erro e não faz nada (devolve o valor inalterado). Se for, divide o valor deste objeto por v (valor / divisor, divisão inteira). Este objeto passa a representar o novo valor, que também deve ser devolvido pelo método.

Exemplo de uso no DrJava:

```
> Inteiro i = new Inteiro();
> i.carregaValor(15);
> i.subtrai(20)
-5
> i.devolveValor()
-5
```

Se quiser, você também pode fazer versões desses métodos que não alteram o valor representado, apenas devolvem o valor da conta.

5. Crie uma classe `TestaInteiro` que verifica o funcionamento correto da classe `Inteiro` em diversas situações. Lembre-se de verificar casos como a divisão por diversos valores. Ao escrever os testes, você notará que a classe `Inteiro` tem uma limitação importante no método `dividePor`.

6. Crie uma classe `Aluno2` para calcular a média aritmética de 4 notas. Mas no lugar de um método que recebe 4 parâmetros, a classe deve conter 4 métodos `recebeNotaX`, onde `X` = 1, 2, 3 ou 4, para receber as notas das provas, de modo que cada método receba apenas uma nota. A classe deve conter ainda um método `imprimeMedia` que imprime a média final do aluno, dizendo se ele foi aprovado ou reprovado. Em seguida, escreva uma classe `TestaAluno2` que verifica se a classe `Aluno2` calcula as médias corretamente.

Exemplo de utilização da classe:

```
> Aluno2 aluno = new Aluno2();
> aluno.recebeNota1(5.0);
> aluno.recebeNota2(7.0);
> aluno.recebeNota3(9.0);
> aluno.recebeNota4(7.0);
> aluno.imprimeMedia();
Média: 7.0 -> aprovado.
```

Capítulo 6

`if else` Encaixados

Quais novidades veremos neste capítulo?

- novidade: `if else` encaixados;
- exercício para reforçar o que aprendemos até agora.

No Capítulo 4, vimos pela primeira vez o conceito de desvio condicional através dos comandos `if` e `else`. Entretanto, vimos exemplos iniciais onde apenas um comando simples era executado, no caso, comandos de impressão. Na prática, comandos `if` e `else` podem ser encaixados de forma a criar estruturas muito complexas. Para isto, veremos inicialmente como criar blocos de comandos através do seguinte exemplo:

```
if (CONDIÇÃO)
{  // início do bloco
   COMANDO-1;
   COMANDO-2;
      ...
   COMANDO-n;
}
```

Neste trecho de código, caso a `CONDIÇÃO` seja verdadeira, os comandos, de 1 a n, são executados seqüencialmente. Veremos a seguir que é comum que alguns destes comandos sejam também comandos de desvio condicional.

Vamos iniciar programando uma classe para representar um triângulo retângulo. Ela contém um método que, dados os comprimentos dos lados do triângulo, verifica se o mesmo é retângulo ou não.

```
class TrianguloRetângulo
{
  void verificaLados(int a, int b, int c)
  {
    if (a * b * c != 0) // nenhum lado pode ser nulo
    {
      if (a*a == b*b + c*c)
        System.out.println("Triângulo retângulo.");
      if (b*b == a*a + c*c)
        System.out.println("Triângulo retângulo.");
```

```
      if (c*c == a*a + b*b)
        System.out.println("Triângulo retângulo.");
    }
  }
}
```

O método acima pode ser chamado da seguinte forma:

```
TrianguloRetângulo r = new TrianguloRetângulo();
r.verificaLados(1, 1, 1);
r.verificaLados(3, 4, 5);
```

Limitações:

1. mesmo que um `if` seja verdadeiro, ele executa os outros `if`s. Em particular, se tivéssemos um triângulo retângulo para o qual vários desses `if`s fossem verdadeiros, ele imprimiria esta mensagem várias vezes (neste exemplo específico, isto não é possível);

2. este método só imprime uma mensagem se os dados correspondem às medidas de um triângulo retângulo, se não é um triângulo retângulo, ele não imprime nada. Através do uso do `else` podemos imprimir mensagens afirmativas e negativas:

```
class TrianguloRetângulo2
{
  void verificaLados(int a, int b, int c)
  {
    if (a * b * c != 0) // nenhum lado pode ser nulo
    {
      if (a*a == b*b + c*c)
        System.out.println("Triângulo retângulo.");
      else if (b*b == a*a + c*c)
        System.out.println("Triângulo retângulo.");
      else if (c*c == a*a + b*b)
        System.out.println("Triângulo retângulo.");
      else
        System.out.println("Não é triângulo retângulo.");
    }
    else
      System.out.println("Não é triângulo pois possui lado de comprimento nulo.");
  }
}
```

Caso sejam necessários outros métodos, como um para o cálculo de perímetro, é interessante colocar os lados como atributos da classe.

```
class TrianguloRetângulo3
{
  int a, b, c;
  void carregaLados(int l1, int l2, int l3)
  {
    a = l1;
    b = l2;
    c = l3;
  }
```

```
  int calculaPerímetro()
  {
    return a + b + c;
  }
  void verificaLados()
  {
    if (a * b * c != 0) // nenhum lado pode ser nulo
    {
      if (a*a == b*b + c*c)
        System.out.println ("Triângulo retângulo.");
      else if (b*b == a*a + c*c)
        System.out.println ("Triângulo retângulo.");
      else if (c*c == a*a + b*b)
        System.out.println ("Triângulo retângulo.");
      else
        System.out.println("Não é triângulo retângulo.");
    }
    else
      System.out.println("Não é triângulo pois possui lado de comprimento nulo.");

  }
}
```

Exercícios

1. Você foi contratado por uma agência de viagens para escrever uma classe em Java para calcular a conversão de reais para dólar de acordo com a taxa de compra e a taxa de venda. Para isso, escreva uma classe `ConversorMonetário` que inclua os seguintes métodos:

   (a) `defineTaxaCompra()` e `defineTaxaVenda()`;

   (b) `imprimeTaxas()` que imprime o valor das 2 taxas de conversão;

   (c) `vendeDólar()` que recebe uma quantia em dólares e devolve o valor correspondente em reais;

   (d) `compraDólar()` que recebe uma quantia em dólares e devolve o valor correspondente em reais.

   Em seguida, escreva uma classe `TestaConversorMonetário` que define diferentes taxas de compra e venda de dólares e, para cada taxa de conversão, realiza operações de compra e venda.

2. Escreva uma classe `Baskara` que possui 3 atributos do tipo `double` correspondentes aos coeficientes a, b e c de uma equação do segundo grau. Escreva um método para carregar valores nestes atributos e, em seguida, escreva os 4 métodos seguintes:

   (a) `delta()` deve calcular o δ da fórmula de Baskara;

   (b) `númeroDeRaízesReais()` deve devolver um inteiro indicando quantas raízes reais a equação possui;

   (c) `imprimeRaízesReais()` deve imprimir as raízes reais;

   (d) `imprimeRaízesImaginárias()` deve imprimir as raízes imaginárias.

Para calcular a raiz quadrada, você pode utilizar o método `java.lang.Math.sqrt(double x)`, que recebe um `double` como parâmetro e devolve outro `double`. Você pode supor que o primeiro coeficiente, *a*, é diferente de 0.

3. Crie uma classe contendo um método que, dado um ponto determinado pelas suas coordenadas `x` e `y`, reais, imprime em qual quadrante este ponto está localizado. O primeiro quadrante corresponde aos pontos que possuem `x` e `y` positivos, o segundo quadrante a `x` positivo e `y` negativo e assim por diante. Para resolver este exercício, será necessário utilizar os operadores < e >. Sua utilização é similar à do operador == utilizado até agora.

4. **[Desafio!]** *São apresentadas a você doze esferas de aparência idêntica. Sabe-se que apenas uma delas é levemente diferente das demais em massa, mas não se sabe qual e nem se a massa é maior ou menor. Sua missão é identificar essa esfera diferente e também dizer se ela é mais ou menos pesada. Para isso você tem apenas uma balança de prato (que só permite determinar igualdades/desigualdades). Ah, sim, pequeno detalhe: o desafio é completar a missão em não mais que três pesagens.*

   Escreva um programa que resolve esse desafio. O seu programa deve dar uma resposta correta sempre e também informar as três ou menos pesagens que permitiram concluir a resposta. Como esse problema é um tanto complicado, recomendamos que você implemente o modelo descrito no quadro a seguir.

   Cada esfera deve ser representada por um inteiro entre 1 e 12. Para representarmos a esfera diferente, usaremos, além do identificador inteiro, uma variável booleana que receberá o valor true se a esfera for mais pesada ou o valor false se a esfera for mais leve.

   Importante: note que, para que o problema tenha sentido, o método resolveDesafioDasDozeEsferas não deve de modo algum acessar os atributos esferaDiferente e maisPesada para inferir a resposta. Quem dá pistas para este método sobre o valor desses atributos são os métodos pesa#x#.

   Lembre-se de que você também pode implementar métodos adicionais, se achar necessário ou mais elegante. Ou ainda, se você já se sente seguro(a), você pode implementar a(s) sua(s) própria(s) classe(s).

   Exemplo de uso no DrJava:

```
> DesafioDasEsferas dde = new DesafioDasEsferas();
> dde.defineEsferaDiferente(4, false);
Início do desafio: esfera 4 mais leve.
> dde.resolveDesafioDasDozeEsferas();
Pesagem 1: 1 2 3 4 5 x 6 7 8 9 10.
Resultado: (1) lado direito mais pesado.
Pesagem 2: 1 2 x 3 4.
Resultado: (-1) lado esquerdo mais pesado.
Pesagem 3: 1 x 2.
Resultado: (0) balança equilibrada.
Resposta: esfera 3 mais leve.
[Resposta errada!]
```

   Note que a resposta está errada! (Além disso, as pesagens não permitem dar a resposta certa.)

Classe `DesafioDasEsferas`

– Atributos:

* `int esferaDiferente`
  Identificador da esfera com peso diferente.
* `boolean maisPesada`
  Diz se a esfera diferente é ou não mais pesada que as demais (isto é, true para mais pesada e false para mais leve).
* `int numPesagens`
  Representa o número de pesagens realizadas.

– Métodos para interação com o usuário:

* `void defineEsferaDiferente(int esfera, boolean pesada)`
  Determina qual é a esfera diferente (parâmetro esfera), e se ela é mais pesada ou não (parâmetro pesada). Além disso, reinicia as pesagens, isto é, o número de pesagens realizadas volta a ser zero.
* `void resolveDesafioDasDozeEsferas()`
  Resolve o Desafio das Doze das Esferas. Este método deve imprimir as 3 (ou menos) pesagens feitas, e no final a resposta correta. Este método deve se utilizar dos métodos para uso interno descritos abaixo. Dica: na implementação deste método você também usará uma quantidade grande de ifs e elses encaixados.

– Métodos para uso interno da classe:

* `int pesa1x1(int e1, int d1)`
* `int pesa2x2(int e1, int e2, int d1, int d2)`
* `int pesa3x3(int e1, int e2, int e3, int d1, int d2, int d3)`
* `int pesa4x4(int e1, int e2, int e3, int e4, int d1, int d2, int d3, int d4)`
* `int pesa5x5(int e1, int e2, int e3, int e4, int e5, int d1, int d2, int d3, int d4, int d5)`
* `int pesa6x6(int e1, int e2, int e3, int e4, int e5, int e6, int d1, int d2, int d3, int d4, int d5, int d6)`

  Os métodos acima (no formato pesa#x#) funcionam de forma semelhante. Eles representam as possíveis pesagens e devolvem o resultado. Os parâmetros representam as esferas que são pesadas, os começados por e (ou seja, e1, e2, ...) representam esferas que vão para o prato esquerdo e os começados por d (d1, d2, ...) são as do prato direito. Lembrando, cada esfera é representada por um inteiro entre 1 e 12. Então, por exemplo, para comparar as esferas 1, 7 e 3 com as esferas 4, 5 e 6, basta chamar pesa3x3(1,7,3,4,5,6). Os métodos devem devolver -1 se a balança pender para o lado esquerdo, 0 se os pesos forem iguais ou 1 se a balança pender para o lado direito. Esses métodos também devem incrementar o número de pesagens realizadas.