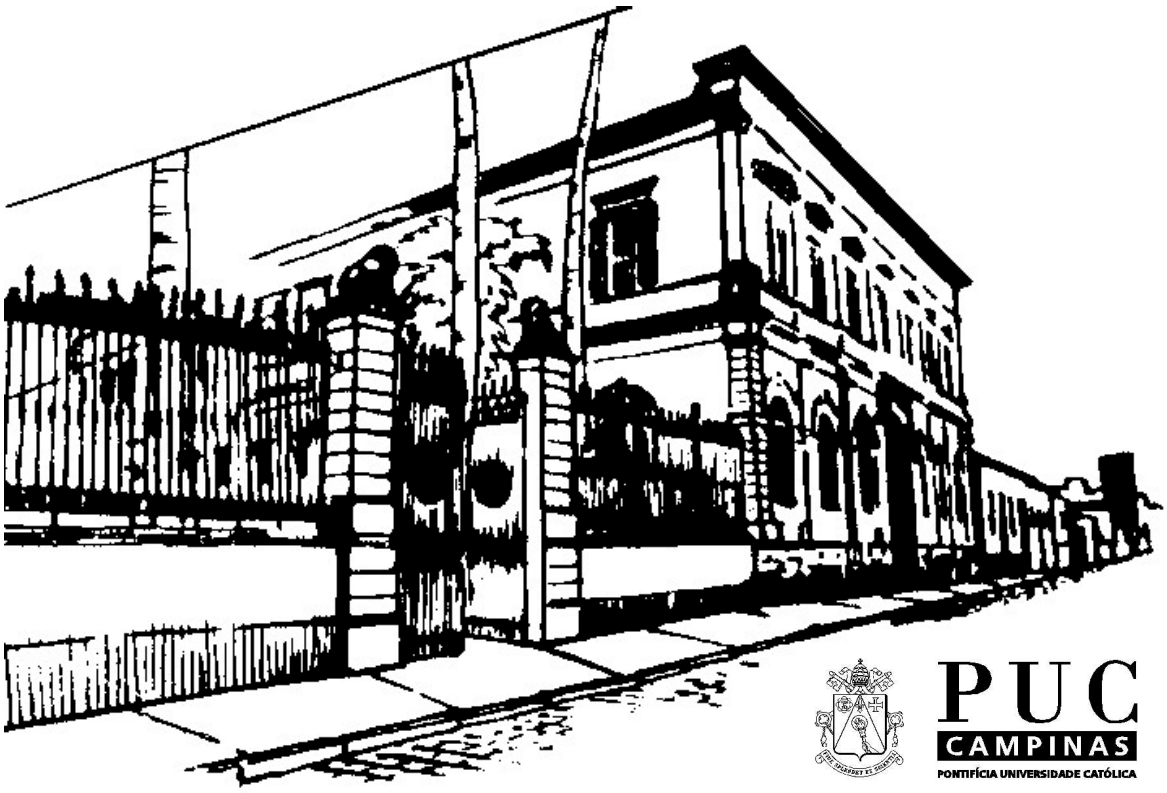

Centro de Ciências Exatas, Ambientais e de Tecnologias

Faculdade de Engenharia de Computação

Paradigmas de Linguagens de Programação I

1º Semestre de 2003
Volume 1

Prof. André Luís dos R.G. de Carvalho

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS
INSTITUTO DE INFORMÁTICA
PLANO DE ENSINO

Faculdade:	Disciplina:	Docente:	C.h. Semanal	Período Letivo:
Engenharia de Computação	Paradigmas de Linguagens de Programação I	André Luís dos Reis Gomes de Carvalho	06 (quatro horas aula)	1º Semestre de 2003

Objetivo Geral da Disciplina:

- Estudar o paradigma de programação orientada a objetos, bem como uma linguagem representativa deste paradigma.
- Estudar o meta-paradigma declarativo, e suas variantes, o paradigma funcional e o paradigma lógico, bem como linguagens representativas de cada um destes paradigmas.

Avaliação:

- 70% da nota será dada pela média ponderada de duas provas, respectivamente com pesos 2 e 1.
- 30% da nota será dada pela média ponderada de uma série de trabalhos a serem desenvolvidos durante o semestre, com pesos compatíveis com sua complexidade.

Freqüência:

- Nenhum abono de falta será concedido pelo professor. Qualquer problema neste sentido deverá ser encaminhada ao PA que tomará todas as providências no sentido de encaminhar a solução do mesmo.
- Para fins de controle de freqüência, não será permitido que alunos desta turma assistam aulas em outras turma, e nem o contrário.

Bibliografia Utilizada :

- Concepts of Programming Languages
  Sebesta, R.W.
- Conceitos de Linguagens de Programação
  Ghesi, C.; e Jazayeri, M.
- Programming Languages: Design and Implementation
  Pratt, T.W.
- Object Orientede System Analysis: Modelling the World in Data
  Shlaer, S.; e Mellor, S.J.
- Object Lifecycles: Modelling the World in States
  Shlaer, S.; e Mellor, S.J.
- Introdução à Programação Orientada a Objetos
  Takahashi, T.
- Programação Orientada a Objetos
  Takahashi T.; e Liesemberg, H.K.
- Programação Orientada para Objeto
  Cox, B.J.
- The TAO of Objects: A Beginner's Guide to Object Oriented Programming
  Entsminger, G.
- A Complete Object Oriented Design Example
  Richardson, J.E.; Schulz, R.C.; e Berard, E.V.
- Java – Como Programar
  Deitel, H.M. e Deitel, P.J.
- Java Unleashed
  Morrison, M.; December, J.; et alii
- Dominando o Java
  Naughton, P.Introdução à Programação Funcional
  Meira, S.R.L.
- Apprendre LISP
  Gnosis
- The XLISP Primer
  Bonnie J.F.
- Programação em Lógica e a Linguagem PROLOG
  Casanova, M.A.; Giorno, F.A.c.; e Furtado, A.L.
- Using Turbo Prolog
  Robinson, P.R.
- Using Turbo PROLOG
  Robinson, P.R.
- PROLOG Programming for Artificial Intelligence
  Bratko, I.
- Artigos selecionados e disponibilizados para xerox pelo professor.

Nº aulas	Conteúdos selecionados

04	O PARADIGMA DE ORIENTAÇÃO A OBJETOS:
	- INTRODUÇÃO;
	- CONCEITOS BÁSICOS:
	  - Objeto → (1) estado interno; (2) comportamento: métodos / mensagens.
	  - Classes → (1) estado; (2) comportamento: métodos / mensagens; (3) instâncias de classe.
	- FUNDAMENTOS:
	  - Encapsulamento.
	  - Herança.
	  - Polimorfismo.
	- CONCLUSÃO.

20	A LINGUAGEM DE PROGRAMAÇÃO JAVA (ASPECTOS BÁSICOS):
	
	- INTRODUÇÃO À JAVA
	  - Execução de Conteúdo → (1) o que pode-se fazer com Java: o que é java / o que é execução de conteúdo / como Java muda a WWW; (2) origens e futuro de Java: o estado corrente de Java / as possibilidades futuras de Java; (3) potencial da linguagem Java: animação / interação / interatividade e computação / comunicação / aplicações e handlers; (4) o que se torna possível com Java.
	  - Projeto Flexível e Dinâmico → (1) primeiro contato com Java: conexao com a WWW / alguns programas simples; (2) visão geral de Java: suporte a comunicação de rede / características de Java enquanto uma linguagem de programação / HotJava / Java em ação / componentes de software de Java / especificação da máquina virtual de Java / segurança em Java.
	  - Impactos na WWW → (1) visão geral da WWW: ideias que levaram à WWW / uma definição de WWW; (2) como Java transforma a WWW: suporte à interatividade / eliminação da necessidade aplicações auxiliares; (3) contribuições à comunicação na WWW; (4) impactos no potencial da WWW.
	  - Páginas Animadas → (1) applets em movimento; (2) animação; (3) sites comerciais.
	  - Páginas Interativas → (1) jogos interativos; (2) aplicações educacionais.
	  - Distribuição de Conteúdo → (1) significado de distribuição e recuperação em rede; (2) como lidar com novos protocolos e formatos; (3) recuperação e compartilhamento de informações em rede.
	
	- PRIMEIRO CONTATO

	- Ferramentas → (1) visão geral; (2) navegadors; (3) ambientes de desenvolvimento; (4) bibliotecas de programação; (5) recursos online. - O Kit de Desenvolvimento Java → (1) como obter a última versão; (2) visão geral; (3) o compilador; (4) o interpretador para a execução; (5) o visualizador de applets; (6) o depurador; (7) engenharia reversa em arquivos de classe; (8) o gerador de arquivos header e strub; (9) o gerador de documentação. - Outros Ambientes e Ferramentas → (1) ambientes de desenvolvimento; (2) bibliotecas para programação. - A LINGUAGEM DE PROGRAMAÇÃO JAVA - Fundamentos da Linguagem Java → (1) um primeor programa; (2) tokens: identificadores / palavras chave / literais / operadores / separadores / comentarios / espaços em branco; (3) tipos de dados: inteiros / reais / logico / caractere; (4) conversão de tipo; (5) blocos e escopo; (6) vetores; (7) strings. - Expressões, Operadores e Estruturas de Controle → (1) expressões e operadores: precedência de operadores / operadores de inteiros / operadores de reais / operadores lógicos / operadores de strings / o operador de atribuição; (2) estruturas de controle: seleções / repetições / break e continue. - Classes, Pacotes e Interfaces → (1) revisão de conceitos de programação orientada a objetos: objetos / encapsulamento / mensagens / classes / herança; (2) classes: declaração / derivação / redefinição de métodos / sobrecarga de métodos / modificadores de acesso / classes e métodos abstratos; (2) criação de objetos: o método de criação / o operador new; (3) destruição de objetos; (4) pacotes: declaração / importação / visibilidade das classes; (5) interfaces: declaração / implementação.
02	PRIMEIRA PROVA
22	A LINGUAGEM DE PROGRAMAÇÃO JAVA (ASPECTOS AVANÇADOS): - RESISTÊNCIA A FALHAS - Tratamento de Excessões → (1) programação em alto nível de abstração; (2) programação em baixo nível de abstração; (3) limitações do programador; (4) a cláusula finally. - AS BIBLIOTECAS DE CLASSES DA LINGUAGEM JAVA - Visão Geral das Bibliotecas de Classes → (1) o pacote Language; (2) o pacote Utilities; (3) o pacote I/O; (4) classes relacionadas com threads; (10) classes relacionadas com tratamento de erros; (5) classes relacionadas com processos. - O Pacote Language → (1) a classe Object; (2) classes relacionadas com os tipos de dados: a classe Boolean / a classe character / classes relacionadas com inteiros / classes relacionadas com reais; (3) a classe Math; (4) classes relacionadas com Strings: a classe String / a classe

	StringBuffer; (5) a classe System; (6) a classe Runtime; (7) a classe Class; (8) a classe ClassLoader. - O Pacote Utilities → (1) interfaces: enumeration / observer; (2) classes: BitSet / Date / Random / StringTokenizer / Vector / Stack / Dictionary / Hashtable / Properties / Observable. - O Pacote I/O → (1) classes de entrada: a classe InputSream / o objeto System.in / a classe BufferedInputStream / a classe DataInputStream / a classe FileInputStream / a classe StringBufferInputStream; (2) classes de saída: a classe OutputStream / a classe PrintSream / o objeto System.out / a classe BufferedOutputStream / a classe DataOutputSream / a classe FileOutputStream; (3) classes relacionadas com arquivos: a classe File / a classe RandomAccessFile. - PROGRAMAÇÃO DE APPLETS - Visão Geral de Programação de Applets → (1) o que é uma applet: applets e a WWW / diferença entre applets e aplicações; (2) os limites das applets: limites funcionais / limites impostos pelo navegador; (3) noções básicas sobre applets: herança da classe Applet / HTML; (4) exemplos básicos de applets. - O Pacote AWT (abstract window toolkit) → (1) uma applet AWT simples; (2) tratamento de eventos: tratamento de eventos em detalhes / handleEvent () ou action () / geração de eventos; (3) componentes: componentes de interface / containers / métodos comuns a todos os componentes; (4) como projetar uma interface de usuário. - O Pacote *Applets e Gráficos* → (1) características das applets; o ciclo de vida de uma applet: init () / start () / stop () / destroy (); (2) como explorar o navegador: como localizar arquivos / imagens / como usar o MediaTracker / audio; (3) contextos de uma applet: showDocument () / showStatus () / como obter parâmetros; (4) gráficos; (5) uma applet simples. - Como programar Applets → (1) projeto básico de applets: interface do usuário / projeto das classes; (2) applets no munto real: applets devem ser pequenas / como responder ao usuário. - MULTIPROGRAMAÇÃO - Threads e Multithreading → (1) o que são threads e para que elas servem; (2) como escrever applets com threads; (3) o problema do conceito de paralelismo; (4) como pensar em termos de multithreads; (5) como criar e usar threads; (6) como saber que uma thread parou; (7) thread scheduling: preemptivo X não preemptivo; como testar. - ACESSO A BASES DE DADOS - O Pacote SQL → (1) drivers; (2) classes: Connection, Statement, ResultSet. - PROGRAMAÇÃO DISTRIBUÍDA - ServerSockets e Sockets.

Nº aulas	Conteúdos selecionados
2	- Meta-Paradígma de programação declarativo (objetivos, limitações, considerações de hardware, considerações de software, motivação).
2	- Paradígma de programação funcional (funções matemáticas, funções matemáticas vs. funções de linguagens de programação, características das linguagens funcionais).
8	- M-Expressões (átomos, S-expressões, listas, funções primitivas, predicados, formas, expressões condicionais e regras de avaliação, definição de funções, invocação de funções).
2	- Linguagem de programação Lisp
2	- Paradígma de programação lógico (o que é, bancos de conhecimento, resolvedores de problemas, raciocínio reverso, lógica simbólica formal, cálculo de predicados, proposições simples, proposições compostas, quantificadores, forma clausal, resolução, cláusulas de Horn).
8	- Linguagem de programação Prolog (objetos, relações, fatos, regras, o ambiente turbo, divisões de um programa, depuração, comentários, predicados, domínios, execução, operadores relacionais, operadores aritméticos, funções, recursão, estratégia de busca, .predicados especiais, instanciação e binding, functors, listas).

Índice

ÍNDICE 7

CAPÍTULO I: O PARADIGMA DE ORIENTAÇÃO A OBJETOS 10

INTRODUÇÃO 11

MOTIVAÇÃO 11

INSTÂNCIAS 12

CLASSES 13

OBJETOS 13

MEMBROS DE CLASSE E DE INSTÂNCIA 13

ENCAPSULAMENTO 14

HERANÇA 14

POLIMORFISMO 15

SOBRECARGA 15

ASSINATURA 16

HERANÇA MÚLTIPLA 16

CLASSES ABSTRATAS 17

CONCLUSÃO 17

REFERÊNCIAS 18

BIBLIOGRAFIA 19

CAPÍTULO II: FUNDAMENTOS DA LINGUAGEM DE PROGRAMAÇÃO JAVA 20

INTRODUÇÃO 21

CARACTERÍSTICAS ENQUANTO LINGUAGEM DE PROGRAMAÇÃO 21

O JAVA DEVELOPMENT KIT (JDK) 24

IDENTIFICADORES 24

PALAVRAS RESERVADAS 25

OPERADORES 25

OUTROS SEPARADORES 25

COMENTÁRIOS 26

SAÍDA DE DADOS **26**

CLASSES **26**

COMPILAÇÃO E EXECUÇÃO **27**

TIPOS BÁSICOS **27**

CLASSES WRAPPER **29**

A CLASSE NUMBER 29
A CLASSE INTEGER 29
A CLASSE LONG 33
A CLASSE FLOAT 36
A CLASSE DOUBLE 38
A CLASSE BOOLEAN 40
A CLASSE CHARACTER 41

CONSTANTES **43**

CONSTANTES LITERAIS 43
CONSTANTES SIMBÓLICAS 45

CADEIAS DE CARACTERES **45**

A CLASSE STRING 45
A CLASSE STRINGBUFFER 50
A CLASSE STRINGTOKENIZER 54

FUNÇÕES **55**

VARIÁVEIS **56**

ACESSIBILIDADE DE MEMBROS **57**

ENTRADA DE DADOS **58**

A CLASSE MATH **62**

EXPRESSÕES **65**

OPERADORES ARITMÉTICOS CONVENCIONAIS 66
OPERADORES DE INCREMENTO E DECREMENTO 66
OPERADORES RELACIONAIS 67
OPERADORES LÓGICOS 68
OPERADOR DE ATRIBUIÇÃO COM OPERAÇÃO EMBUTIDA 68
EXPRESSÕES CONDICIONAIS 69
CONVERSÕES DE TIPO 69
OPERADORES DE BIT 70

A CLASSE BITSET **70**

COMANDOS **74**

O COMANDO DE ATRIBUIÇÃO 74

BLOCOS DE COMANDO **74**

O COMANDO IF **75**

O COMANDO `SWITCH` 75
O COMANDO `WHILE` 78
O COMANDO `DO-WHILE` 79

O COMANDO FOR 80
O COMANDO CONTINUE 82
O COMANDO BREAK 83

RECURSÃO 85

MEMBROS DE CLASSE E DE INSTÂNCIA 86

CRIAÇÃO DE INSTÂNCIAS 89

ORGANIZAÇÃO DE PROGRAMA 92

MODULARIZAÇÃO 92
PACOTES 96
Classes Públicas 96
Membros Default (ou de Pacote) 96
JARS 100

SOBRECARGA 105

CRIAÇÃO E DESTRUIÇÃO 111

VETORES 119

CONSTANTES VETOR 119
TAMANHO DAS DIMENSÕES DE UM VETOR 119
A CLASSE JAVA.UTIL.VECTOR 126

CLASSES MEMBRO 130

HERANÇA 130

MAIS SOBRE CONVERSÕES DE TIPO 131
VERIFICAÇÃO DE TIPO EM JAVA 131
MEMBROS PROTEGIDOS 132
JAVA.LANG.COMPARABLE 132
A CLASSE JAVA.LANG.OBJECT 142

INTERFACES 143

HERANÇA MÚLTIPLA 143

CLASSES ABSTRATAS 149

EXERCÍCIOS 154

I. CLASSES E OBJETOS 154

II. PACOTES 163

III. HERANÇA 164

BIBLIOGRAFIA 169

Capítulo I:
O Paradigma de Orientação a Objetos

Introdução

Apesar de ser um paradigma relativamente novo, o paradigma de orientação a objetos já pode ser considerado um clássico.

Seu uso vem aceleradamente aumentando com o tempo, tendo se firmado praticamente como um "modismo" dos dias atuais. Parece a concretização da previsão anunciada em [Takahashi88], que dizia que a orientação a objetos viria a ser nos anos 80 e 90 aquilo que programação estruturada foi nos anos 70.

Como praticamente todos os paradigmas de programação, o paradigma de orientação a objetos também surgiu juntamente com o desenvolvimento de uma linguagem de programação.

A primeira linguagem a implementar conceitos de orientação a objetos foi a linguagem Símula, proposta por Dahl e Nygaard em 1966. Posteriormente o conceito foi refinado e desenvolvido na linguagem SmallTalk, proposta por Alan Kay em 1972.

A linguagem SmallTalk não somente norteou o desenvolvimento do paradigma, mas, ainda hoje, é considerada a linguagem que suporta de forma mais completa e pura os seus conceitos.

Motivação

Podemos dizer que um programa de computador implementa uma solução computacional para um problema do mundo real.

Assim sendo, construir um programa de computador envolve vários processos de abstração e concretização entre dois mundos, o real (onde existe o problema, a solução, e o procedimento para se chegar do problema à solução) e o virtual (onde existe uma abstração do problema, uma abstração da solução, e um procedimento abstrato para chegar de uma na outra).

Naturalmente, existe uma distância conceitual inevitável entre este dois mundos. Diminuir esta distância é justamente uma das principais propostas deste paradigma.

Para tanto, o paradigma parte da premissa de que o mundo real não é composto nem por dados (como pretendem os paradigmas orientados a dados), nem por processos (como pretendem os

paradigmas orientados a processos), e sim por entidades semi-autônomas que trabalham e interagem cooperativamente.

Tais entidades são os elementos constituintes fundamentais do paradigma e recebem o nome de objeto.

Instâncias

Todo instância possui um estado interno para registrar e se lembrar do efeito de sua operação. O estado interno de uma instância é representado por uma área de memória local ao instância, que é inacessível e indevassável.

Todo instância possui ainda um comportamento, que é representado por um repertório de operações que o instância dispõe para responder a mensagens externas ou mesmo internas à própria instância.

O resultado da operação de uma instância depende não só da mensagem que ele recebeu, mas também de seu estado interno.

Mensagens são enviadas de uma instância a outro com a finalidade de que a instância receptora produza algum resultado desejado.

A natureza das operações que a instância receptora executa para produzi-lo é ela quem determina, e poderá provocar alterações em seu estado interno, o envio de novas mensagens para outras instâncias e mesmo para si própria, e até a criação de novas instâncias.

Parâmetros podem acompanhar as mensagens dirigidas a uma instância. Tais parâmetros, por sua vez, também são instâncias, e poderão ter algum efeito sobre as operações que a instância receptora executará quando receber a mensagem que elas acompanham.

A descrição das operações que uma instância executa quando recebe uma mensagem é chamada método. O conceito de método é muito parecido com o conceito de procedimento.

A relação que existe entre mensagens e métodos em uma instância é sempre biunívoca, i.e., uma mesma mensagem somente poderia resultar em métodos diferentes quando enviada para instâncias diferentes.

A associação entre mensagens e métodos é sempre pré-definida e estática, i.e., não pode ser alterada em tempo de execução.

Classes

Uma classe é um modelo de instância, i.e., consiste de descrições de estado, métodos, e associações entre mensagens e métodos, que todas as instâncias pertencentes àquela classe irão possuir.

O conceito de classe é muito parecido com o conceito de tipo abstrato de dados, na medida que uma classe define uma estrutura interna e um conjunto de operações que todas as instâncias daquela classe irão possuir.

Classes também podem receber mensagens, ter métodos e ter um estado interno.

Objetos

Todo objeto tem uma classe e serve ao propósito de armazenar instâncias da classe à qual pertence.

Instâncias podem ser criadas a partir de uma classe e, no ato de sua criação, ser solicitadas através de uma mensagem a prestar algum serviço, o que será feito pelas execução do método associado àquela mensagem.

Instâncias somente podem ser solicitadas através de uma mensagem a prestar algum serviço, em um momento posterior àquele de sua criação, quando estiver armazenada em um objeto, que lhe provê, com seu nome, uma forma de ser referenciada.

Assim sendo, se não estiverem armazenadas em um objeto, as instâncias tem uma sobrevida breve, já que perdem a utilidade logo após sua criação.

Membros de Classe e de Instância

Todos os dados e métodos definidos em uma classe podem ser ditos membros daquela classe. Podemos classificar os membros de uma classe em (1) dados e métodos de classe; e (2) dados e métodos de instância.

Entendemos que membros de classe, sejam eles dados ou métodos, são membros relativos à uma classe como um todo e não a nenhum instância individual da classe. Membros de classe expressam propriedades e ações aplicáveis a toda uma classe de instâncias e não a uma instância específica.

Entendemos que membros de instância de classe, sejam eles dados ou métodos, são membros relativos à instâncias individuais e não à classe como um todo. Membros de instância expressam propriedades e ações que são aplicáveis às instâncias de uma classe individualmente e não à classe de modo geral.

Encapsulamento

O conceito de encapsulamento de certa forma resume tudo o que foi dito até agora sobre o modelo de computação proposto pelo paradigma.

Ele determina que uma instância pode ser vista como o encapsulamento de seu estado interno e de seu comportamento. O mesmo ocorre com uma classe.

Herança

O conceito de herança se baseia em uma estrutura hierárquica de classes. Em tal estrutura de classes, cada classe pode ter zero ou mais subclasses e zero ou mais superclasses.

Uma dada subclasse herda todos os componentes (representação de estado interno e comportamento) de suas superclasses. A classe, em adição às propriedades que herda, pode definir componentes próprios, além de poder redefinir componentes que recebeu por herança.

Vale observar que os componentes de uma subclasse recebidos por herança que não forem redefinidos, comportar-se-ão exatamente como componentes definidos na própria classe, e isto sem a necessidade de nenhuma definição ou indicação especial.

Esta propriedade encoraja a definição de novas classes, sem no entanto, requerer uma extensiva duplicação de código [Dershem90].

No paradigma imperativo, o conceito mais próximo de herança é o de subtipos. Da mesma forma que uma subclasse herda as propriedades de suas superclasses, um subtipo herda as de seu supertipo.

O conceito de subtipo, no entanto, é muito mais limitado do que o conceito de herança, porque é aplicável somente a um conjunto limitado de tipos básicos, em geral, aos tipos escalares.

O paradigma imperativo não generaliza esta definição para contemplar tipos abstratos de dados, o que faz do conceito de herança uma grande contribuição do paradigma de orientação a instâncias.

Polimorfismo

Polimorfismo é a propriedade que possibilita que se envie uma mesma mensagem a diferentes instâncias, provocando em cada uma delas uma operação diferente.

Isto acontece porque cada uma das diversas instâncias receptoras potenciais de uma determinada mensagem possui um método próprio para responder à chegada daquela mensagem, dependendo da classe à qual ela pertence.

É importante ressaltar que, ao se enviar uma mensagem, não há a necessidade de se saber nem a classe da instância receptora, nem a forma que esta utilizará para responder à mesma.

No paradigma imperativo, o conceito mais próximo de polimorfismo é o de sobrecarga de operadores e de procedimento.

O conceito de polimorfismo estabelece que a determinação do método a ser invocado, quando da recepção de uma mensagem por uma instância, é feita com base na classe à qual a instância receptora pertence.

Sobrecarga

O conceito de sobrecarga estabelece a possibilidade de se definir 2 ou mais funções com o mesmo nome de forma que, quando ocorrer a chamada de uma função, dentre as diversas

funções de mesmo nome, vai-se determinar aquele que será executado pelo número, tipo e ordem dos parâmetros reais fornecidos por ocasião de sua chamada.

A diferença entre este conceito e o conceito de polimorfismo reside no tempo em que é feita a associação do procedimento a ser executado com a chamada de procedimento.

No polimorfismo, a associação ocorre dinamicamente, uma vez que a classe à qual um instância pertence não é conhecida senão em tempo de execução. Na sobrecarga, tudo se passa como se o número e o tipo dos parâmetros constituíssem uma extensão do nome do procedimento, e a associação pode ser estática.

Assinatura

O nome de um método, juntamente com o conjunto ordenados dos tipos dos parâmetros que ele recebe, costuma ser chamado de assinatura .

Herança Múltipla

Herança Múltipla é um recurso previsto pelo paradígma de orientação a objetos cuja finalidade é permitir que classe herde características não de uma, mas de várias outras classes.

Embora não seja um recurso do qual se precise lançar mão com tanta freqüência, trata-se de um recurso que pode por vezes ser muito útil.

Por exemplo: mesmo dispondo de um conjunto amplo de classes base desenvolvidas, pode ser que, num dado momento, nos vejamos confrontados com a necessidade de um novo tipo de objeto.

Pode ser ainda que este novo tipo de objeto guarde muita semelhança, não com uma, mas com muitas das classes que já temos. Trata-se de um caso típico a ser resolvido com herança múltipla.

Quando uma classe é derivada de mais de uma classe, ela pode ser usada em todos os lugares onde for esperada uma de suas classes base. Trata-se de uma extensão do conceito de polimorfismo.

Classes Abstratas

Numa situação convencional de herança, tem-se que classes bases compartilham com suas classes derivadas, tanto a seu comportamento, quanto sua implementação.

Entretanto, podem haver situações nas quais quase não exista, de fato, o compartilhamento de implementação, embora haja um efetivo compartilhamento de comportamento.

Geralmente não é possível escrever código para muitos dos métodos dessas classes, que, apesar de declarados, não são de fato definidos, existindo sem um corpo onde ocorra sua implementação.

Chamamos este tipo de método de Métodos Abstratos e classes que contem um ou mais métodos abstratos são chamadas de Classes Abstratas.

Não é permitido criar instâncias de classes abstratas, mas, apesar disso, elas podem ser muito úteis por propiciarem o polimorfismo.

Conclusão

As linguagens orientadas a objetos nos apresentam uma nova forma de organizar e estruturar programas e isto leva à necessidade de desenvolver uma nova forma de pensar em resolução computacional de problemas.

O Paradigma de Orientação a Objetos conduz naturalmente à implementação de módulos fortemente coesos e fracamente acoplados. Isto torna os programas orientados a objetos mais manuteníveis e suas partes mais reusáveis, facilitando o que em Engenharia de Software chamamos de Peça de Software.

Por tudo isso, cada vez mais encontramos o Paradigma de Orientação a Objetos incorporado nos mais diversos ambientes de computação de uso contemporâneo.

Anexo I

Referências

Dershem, H.L.; and Jipping, M.J., "Programming Languages: Structures and Models", Wadsworth, Inc., 1990.

Takahashi, T., "Introdução à Programação Orientada a Objetos", III EBAI - Escola Brasileiro-Argentina de Informática, 1988.

Anexo II

Bibliografia

Sebesta, R.W., “Concepts of Programming Languages”, The Benjamin/Cummings Publishing Company, Inc., 1989.

Ghezzi, Java., Jazayeri, M., “Programming Languages Concepts”, John Wiley & Sons, Inc., 1982.

Dershem, H.L.; and Jipping, M.J., “Programming Languages: Structures and Models”, Wadsworth, Inc., 1990.

Pratt, T.W., Programming Languages: Design and Implementation.

Takahashi T.; e Liesemberg, H.K.; "Programação Orientada a Objetos"

Entsminger, G.; "The TAO of Objects: A Beginner's Guide to Object Oriented Programming"

Cox, B.J., Programação Orientada para Instância.Morrison, M.; December, John; et alii, “Java Unleashed”, Sams.net Publishing, 1996.

Richardson, J.E.; Schulz, R.C.; e Berard, E.V.; "A Complete Object Oriented Design Example".

Capítulo II: Fundamentos da Linguagem de Programação Java

Introdução

Em meados dos anos 90 a WWW transformava o mundo online. Com um sistema de hipertexto, usuários da WWW se tornaram capazes de selecionar e visualizar informações de toda parte do mundo.

Apesar permitir a seus usuários um alto nível de seletividade de informação, a WWW se limitava a distribuir arquivos de texto, imagem e som, não lhes fornecendo condições para interagir adequadamente com a informação recebida.

Em outras palavras, faltava interatividade verdadeira, em tempo real, dinâmica e visual entre usuários e aplicações WWW.

Java vem suprir esta lacuna, tornando possível a distribuição de aplicações ativas através da WWW, aplicações que podem interagir continuamente com os usuários através do mouse e do teclado e dar-lhes respostas imediatas.

Características enquanto Linguagem de Programação

É sabido que linguagens de programação que encorajam o desenvolvimento de programas robustos, habitualmente, impõem ao programador boa dose de disciplina na escrita de código fonte. A lingaugem C é notoriamente falha neste sentido.

A linguagem C++, que deriva da linguagem C, foi projetada para ser menos permissiva que sua predecessora (e de fato é) mas ainda guarda uma influência excessiva desta linguagem. Isso é um problema e, neste sentido, a linguagem Java é mais rigorosa.

Sabe-se ainda que C++ é uma poderosa linguagem de programação orientada a objetos e, como acontece com a maioria das linguagens projetadas para serem poderosas, ela apresenta uma série de características que propiciam condições favoráveis para o programador desatento cometer erros de programação.

Apesar de se basear na linguagem C++, a Java é uma linguagem de programação estruturalmente bastante mais simples que sua predecessora, já que de seu projeto foram

elimidadas características que em C++ eram raramente usadas (ou mal usadas) e que podiam levar o programador a incorrer em erros de programação.

Especificamente, Java difere de C++ nos seguintes pontos:

1. Java não suporta structs, unions e ponteiros;
2. Java não suporta typedef nem #define;
3. Java apresenta diferenças em certos operadores e não suporta sobrecarga de operadores;
4. Java não suporta herança múltipla;
5. Java lida com parâmetros da linha de comando de forma diversa daquela utilizada por C ou por C++;
6. Java tem *strings* como parte do pacote Java.lang, o que difere dos vetores terminados com null das linguagens C e C++;
7. Java tem um sistema automático para alocar e liberar memória (coleta de lixo), o que torna desnecessário o uso de funções de alocação e desalocação de memória como é preciso em C e C++.

A linguagem Java suporta multithreading, i.e., oferece facilidades para o programador implementar aplicações que apresentam diversas linhas de execução (aplicações nas quais podem ser encontradas diversas tarefas em processo de execução num mesmo instante).

Diferentemente das linguagens C e C++, a linguagem Java foi especificamente projetada para trabalhar em ambiente de rede, possuindo uma vasta biblioteca de classes para comunicação através de protocolos para Internet da família TCP/IP (HTTP, FTP, etc).

Por esta razão, Java é capaz de manipular recursos via URLs com tão facilmente quanto linguagens como C ou C++ acessam um sistema de arquivos local.

É importante ressaltar que, ao mesmo tempo que essas características representam facilidades para a programação de sistemas distribuídos, representam também uma intensa preocupação com a questão da segurança computacional no sentido mais amplo da expressão.

Java é uma linguagem de arquitetura neutra e é implementada através de uma técnica que leva o nome de interpretação impura.

Tal técnica de implementação de linguagens de programação se caracteriza, principalmente, por lançar mão de um compilador e de um interpretador articulados da seguite forma: o compilador compila o programa fonte mas, em vez de gerar código executável por uma máquina real, ele gera código apra uma máquina virtual implementada em software, ou seja, pelo interpretador.

Em outras palavras, para poder ser executado, o código gerado pelo compilador deverá ser interpretado por um interpretador que implementa a máquina virtual para a qual o compilador gerou código.

Assim, o código compilado pode rodar em qualquer máquina real, naturalmente, desde que haja para tal máquina um interpretador que implemente nela a máquina virtual da linguagem Java.

A linguagem intermediária, na qual o compilador escreve o programa a ser interpretado, chama-se ByteCode. E virtude do processo de interpretação, um programa em Java nem sempre tem o mesmo desempenho que um programa equivalente compilado para uma particular plataforma de hardware.

Por isso, Java prevê a possibilidade de traduzir ByteCodes para a linguagem de máquinas reais, ensejando assim condições para se alcançar a mesma eficiência de um processo de compilação tradicional.

É importante ressaltar que, dizer que uma linguagem é de arquitetura neutra, é muito mais forte e contundente do que dizer que uma linguagem é portavel.

Sabemos que é possível acontecer de linguagens como C e C++ (reconhecidamente portáveis) gerarem programas que rodem de maneira ligeiramente diferente em plataformas de hardware diferentes.

Tal se deve, principalmente, à diferença de tamanho da palavra de memória nas diferentes arquiteturas de máquina, o que leva variáveis de um mesmo tipo, em diferentes plataformas, terem capacidades diferentes de representação de valores.

Em Java isso não acontece. Internamente, os dados são representados sempre da mesma forma, não importando a plataforma de hardware. Isso faz com que seja possível ter a certeza de que programas Java produzem sempre o mesmo resultado, não importando a plataforma na qual executem.

O Java Development Kit (JDK)

Distribuído pela SUN Microsystems, o JDK engloba, entre outras ferramentas:

1. Um compilador Java (javac);
2. Um interpretador de ByteCode que interpreta aplicações para console (java);
3. Um interpretador de ByteCode que interpreta aplicações para ambiente de janelas (javaw);
4. Um interpretador de ByteCode que interpreta aplicações integradas em uma página da WEB (appletviewer);
5. Um depurador (jdb);
6. Um compactador (jar);
7. Um utilitário para extrair informações sobre dados e funções, públicos ou não, de um arquivo de ByteCode (javap);
8. Um utilitário gerador de header files (.h) para integrar ByteCodes em aplicações feitas em linguagem C ou C++ (javah);
9. Um gerador de documentação (javadoc).

Identificadores

Identificadores introduzem nomes no programa. Em Java podem ser uma seqüência arbitrariamente longa de letras, dígitos, sublinhados (_) e cifrões ($). O primeiro caractere de um identificador deve necessariamente ser uma letra, sublinhado (_) ou cifrão ($).

É importante ressaltar que, diferentemente de outras linguagens de programação, a linguagem Java diferencia letras maiúsculas de letras minúsculas. Em Java, identificadores que são lidos

da mesma forma, mas que foram escritos de forma diferente no que tange ao emprego de letras maiúsculas e minúsculas, são considerados identificadores diferentes.

Palavras Reservadas

Estas palavras são identificadores predefinidos e reservados pela linguagem Java para propósitos específicos.

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	final
finally	float	for	goto
if	implements	import	instanceof
int	interface	long	native
new	null	package	private
protected	public	return	short
static	super	switch	synchronized
this	throw	throws	transient
try	void	volatile	while

Operadores

Os caracteres e as combinações de caracteres abaixo são reservados para uso como operadores e não podem ser utilizados com outra finalidade. Cada um deles deve ser considerado como um único símbolo.

+	-	*	/	%	&	|
^	~	&&	||	!	<	>
<=	>=	<<	>>	>>>	=	?
++	--	==	+=	-=	*=	/=
%=	&=	|=	^=	!=	<<=	>>=
>>>=	.	[	]	(	)	

Outros Separadores

Os caracteres abaixo são reservados para uso como sinais de pontuação e não podem ser utilizados com outra finalidade.

{ } ; , :

Comentários

Em Java existem três formas de escrever comentários:

- Iniciando o comentário com os caracteres /* e terminando com os caracteres */
- Iniciando o comentário com os caracteres // e terminando no final da linha
- Iniciando o comentário com os caracteres /** e terminando com os caracteres */

Este último tipo de comentário permite sua extração para fins de geração automática de documentação pelo programa javadoc.

Saída de Dados

Para escrever na saída padrão, basta o envio da mensagem System.out.print (para escrever sem pular de linha) ou System.out.println (para escrever e em seguida pular para a próxima linha). Essas mensagens aceitam um único parâmetro (aquilo que desejamos escrever).

Para escrever na saída padrão de erros, basta o envio da mensagem System.err.print (para escrever sem pular de linha) ou System.err.println (para escrever e em seguida pular para a próxima linha). Essas mensagens aceitam um único parâmetro (aquilo que desejamos escrever).

Cadeias de caracteres podem ser escritas delimitadas por aspas (“”). Se desejarmos escrever vários ítens com o envio de uma única mensagem, basta separá-los por um sinal de mais (+).

Classes

Classes são definidas mencionando a palavra chave class, seguida pelo identificador da classe, seguido por uma série de definições de variáveis (que servem ao propósito de definir a estrutura necessária para representar o estado interno da classe e de suas instâncias) e funções (que servem ao propósito de definir as ações de que dispõem a classe e suas instâncias para responder mensagens que lhes sejam enviadas) delimitados por um par de chaves ({}). Chamamos essas variáveis e funções de membros da classe.

Toda classe que deve ser capaz de ser executada deve possuir uma função de nome main que representa o local de início da execução do programa. Esta função deverá receber um vetor de objetos da classe String de tamanho indefinido (cada objeto do vetor representa um parâmetro fornecidos ao programa na linha de comando) e não deverá possuir retorno.

[C:\ExplsJava\Expl_01\Primeiro.java]

```
class BoasVindas
{
    public static void main(String Args [])
    {
        System.out.println ();
        System.out.println ("Bem vindos ao estudo de JAVA!");
        System.out.println ();
    }
}
```

Compilação e Execução

O programa acima, para poder ser executado, deverá primeiro ser compilado através do seguinte comando:

C:\ExplsJava\Expl_01> javac -classpath . Primeiro.java

O compilador Java gera, para cada classe presente no arquivo compilado, um arquivo com os ByteCodes resultantes de sua compilação. Tais arquivos recebem sempre o mesmo nome da classe à qual se referem e têm extensão .class.

Assim, após ter sido dado o comando acima, terá sido gerado em disco um arquivo de nome BoasVindas.class. Para executar seu programa, basta dar o seguinte comando:

C:\ExplsJava\Expl_01> java -classpath . -classpath . BoasVindas

Isso produzirá no console a seguinte saída:

Bem vindos ao estudo de JAVA!

Tipos Básicos

1. **Inteiros:**

Os tipos inteiros permitem a declaração de variáveis capazes de armazenar números inteiros. Existem em quatro tamanhos, a saber:

- byte: 8 bits (de -128 a 127);
- short: 16 bits (de -32768 a 32767);
- int: 32 bits (de -2147483648 a 2147483647);
- long 64 bits (de -9223372036854775808 a 9223372036854775807).

2. Reais:

Os tipos reais permitem a declaração de variáveis capazes de armazenar números reais. Existem em dois tamanhos, a saber:

- float: 32 bits (de $3,4 \times 10^{-38}$ a $3,4 \times 10^{+38}$);
- double: 64 bits (de $1,7 \times 10^{-308}$ a $1,7 \times 10^{+308}$).

3. char:

O tipo char permite a declaração de variáveis que ocupam 16 bits e que são capazes de armazenar um único caractere. Java não emprega o conjunto ASCII, mas sim o conjunto de caracteres UNICODE, no qual existem 65536 caracteres diferentes.

4. boolean:

O tipo boolean permite a declaração de variáveis booleanas (ou lógicas). Variáveis desse tipo são capazes de armazenar valores que expressam a verdade ou a falsidade.

5. void:

O tipo void especifica um conjunto vazio de valores. Ele é usado como tipo de retorno para funções que não retornam nenhum valor. Não faz sentido declarar variáveis do tipo void.

Classes Wrapper

A Classe Number

A classe Number é uma classe que oferece uma interface com todos os tipos escalares padrão: int, long, float e double. Possui métodos que retornam o valor potencialmente arredondados do objeto em cada um desses tipos escalares.

Veja abaixo a interface que a classe especifica para ser compartilhada por todas aquelas classe que dela derivarem:

- Métodos:
  - **int byteValue ():**
    Retorna um byte;
  - **double doubleValue ():**
    Retorna um double;
  - **float floatValue ():**
    Retorna um float;
  - **int intValue ():**
    Retorna um int;
  - **long longValue ():**
    Retorna um long;
  - **short shortValue ():**
    Retorna um short.

A Classe Integer

Derivada da classe Number, a classe Integer possui também outros membros além daqueles herdados de sua classe base.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**
  - **static final int MIN_VALUE:**
    Representa o valor do menor inteiro que uma instância da classe Integer é capaz de representar;
  - **static final int MAX_VALUE:**
    Representa o valor do maior inteiro que uma instância da classe Integer é capaz de representar;
- **Construtores:**
  - **Integer (int N):**
    Constroi uma instância da classe Integer a partir do valor N do tipo int;
  - **Integer (String S):**
    Constroi uma instância da classe Integer a partir da instância S da classe String;
- **Métodos:**
  - **byte byteValue ():**
    Retorna o valor deste Integer na forma de um byte;
  - **int compareTo (Integer N):**
    Compara este Integer com o Integer N; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo;
  - **static Integer decode (String S):**
    Retorna uma instância da classe Integer que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, octal ou hexadecimal, caso contrário a exceção NumberFormatException será lançada; o prefixo '0' indica octal; os prefixos '#', "0x" ou "0X" indicam hexadecimal; a ausência de prefixo indica decimal; o caractere '-' no início do String S indica um valor negativo.

- **double doubleValue ():**
  Retorna o valor deste Integer na forma de um double;
- **float floatValue ():**
  Retorna o valor deste Integer na forma de um float;
- **static Integer getInteger (String S):**
  Retorna uma instância da classe Integer que representa o valor da propriedade do sistema chamada S;
- **static Integer getInteger (String S, int D):**
  Retorna uma instância da classe Integer que representa o valor da propriedade do sistema chamada S (retorna o valor padrão Integer (D) no caso da referida propriedade não existir);
- **static Integer getInteger (String S, Integer D):**
  Retorna uma instância da classe Integer que representa o valor da propriedade sistêmica de nome S (retorna o valor padrão D no caso da referida propriedade nao existir);
- **int intValue ():**
  Retorna o valor deste Integer na forma de um int;
- **long longValue ():**
  Retorna o valor deste Integer na forma de um long;
- **static int parseInt (String S):**
  Retorna um int que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;
- **static int parseInt (String S, int B):**
  Retorna um int que representa o valor numérico expresso na base B pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido na base B, caso contrário a exceção NumberFormatException será lançada;

- **short shortValue ():**
  Retorna o valor deste Integer na forma de um short;

- **static String toBinaryString (int N):**
  Retorna um String que contém somente caracteres que representam dígitos binários que expressam o valor numérico de N;

- **static String toHexString (int N):**
  Retorna um String que contém somente caracteres que representam dígitos hexadecimais que expressam o valor numérico de N;

- **static String toOctalString (int N):**
  Retorna um String que contém somente caracteres que representam dígitos octais que expressam o valor numérico de N;

- **static String toString (int N):**
  Retorna um String que contém somente caracteres que representam dígitos decimais que expressam o valor numérico de N;

- **static String toString (int N, int B):**
  Retorna um String que contém somente caracteres que representam dígitos validos na base b que expressam o valor numérico de N na base B;

- **static Integer valueOf (String S):**
  Retorna uma instância da classe Integer que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;

- **static Integer valueOf (String S, int B):**
  Retorna uma instância da classe Integer que representa o valor numérico expresso na base B pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido na base B, caso contrário a exceção NumberFormatException será lançada.

A Classe Long

Derivada da classe Number, a classe Long possui também outros membros além daqueles herdados de sua classe base.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**
  - **static double MIN_VALUE:**
    Representa o valor do menor inteiro longo que uma instância da classe Long é capaz de representar;
  - **static double MAX_VALUE:**
    Representa o valor do maior inteiro longo que uma instância da classe Long é capaz de representar;
- **Construtores:**
  - **Long (long N):**
    Constroi uma instância da classe Long a partir do valor N do tipo long;
  - **Long (String S):**
    Constroi uma instância da classe Long a partir da instância S da classe String;
- **Métodos:**
  - **byte byteValue ():**
    Retorna o valor deste Long na forma de um byte;
  - **int compareTo (Long N):**
    Compara este Long com o Long N; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo;
  - **static Long decode (String S):**
    Retorna uma instância da classe Long que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico

válido em decimal, octal ou hexadecimal, caso contrário a exceção NumberFormatException será lançada; o prefixo '0' indica octal; os prefixos '#', "0x" ou "0X" indicam hexadecimal; a ausência de prefixo indica decimal; o caractere '-' no início do String S indica um valor negativo.

- **double doubleValue ():**
  Retorna o valor deste Long na forma de um double;
- **float floatValue ():**
  Retorna o valor deste Long na forma de um float;
- **static Long getLong (String S):**
  Retorna uma instância da classe Long que representa o valor da propriedade do sistema chamada S;
- **static Long getLong (String S, int D):**
  Retorna uma instância da classe Long que representa o valor da propriedade do sistema chamada S (retorna o valor padrão Long (D) no caso da referida propriedade não existir);
- **static Long getLong (String S, Long D):**
  Retorna uma instância da classe Long que representa o valor da propriedade sistêmica de nome S (retorna o valor padrão D no caso da referida propriedade nao existir);
- **int intValue ():**
  Retorna o valor deste Long na forma de um int;
- **long longValue ():**
  Retorna o valor deste Long na forma de um long;
- **static long parseLong (String S):**
  Retorna um long que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;
- **static long parseLong (String S, int B):**
  Retorna um long que representa o valor numérico expresso na base B pelos caracteres

de S; o String S deve conter caracteres que expressam um valor numérico válido na base B, caso contrário a exceção NumberFormatException será lançada;

- **short shortValue ():**
  Retorna o valor deste Long na forma de um short;

- **static String toBinaryString (long N):**
  Retorna um String que contém somente caracteres que representam dígitos binários que expressam o valor numérico de N;

- **static String toHexString (long N):**
  Retorna um String que contém somente caracteres que representam dígitos hexadecimais que expressam o valor numérico de N;

- **static String toOctalString (long N):**
  Retorna um String que contém somente caracteres que representam dígitos octais que expressam o valor numérico de N;

- **static String toString (long N):**
  Retorna um String que contém somente caracteres que representam dígitos decimais que expressam o valor numérico de N;

- **static String toString (long N, int B):**
  Retorna um String que contém somente caracteres que representam dígitos validos na base b que expressam o valor numérico de N na base B;

- **static Long valueOf (String S):**
  Retorna uma instância da classe Long que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;

- **static Long valueOf (String S, int B):**
  Retorna uma instância da classe Long que representa o valor numérico expresso na base B pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido na base B, caso contrário a exceção NumberFormatException será lançada.

A Classe Float

Derivada da classe Number, a classe Float possui também outros membros além daqueles herdados de sua classe base.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**
  - **static float MIN_VALUE:**
    Representa o valor do menor real que uma instância da classe Float é capaz de representar;
  - **static float MAX_VALUE:**
    Representa o valor do maior real que uma instância da classe Float é capaz de representar;
  - **static float NEGATIVE_INFINITY:**
    Representa o valor especial -∞;
  - **static float POSITIVE_INFINITY:**
    Representa o valor especial +∞;
  - **static float NaN:**
    Representa o valor especial que simboliza algo que não é um número (*not a number*);
- **Construtores:**
  - **Float (double N):**
    Constroi uma instância da classe Float a partir do valor N do tipo double;
  - **Float (float N):**
    Constroi uma instância da classe Float a partir do valor N do tipo float;
  - **Float (String S):**
    Constroi uma instância da classe Float a partir da instância S da classe String;
- **Métodos:**

- **byte byteValue ():**

  Retorna o valor deste Float na forma de um byte;

- **int compareTo (Float N):**

  Compara este Float com o Float N; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo;

- **double doubleValue ():**

  Retorna o valor deste Float na forma de um double;

- **static int floatToIntBits (float N):**

  Retorna em um int a seqüência de bits que forma N;

- **float floatValue ():**

  Retorna o valor deste Float na forma de um float;

- **static float intBitsToFloat (int N):**

  Retorna em um float a seqüência de bits que forma N;

- **int intValue ():**

  Retorna o valor deste Float na forma de um int;

- **boolean isInfinite ():**

  Verifica se este Float vale infinito;

- **static boolean isInfinite (float N):**

  Verifica se N vale infinito;

- **boolean isNaN ():**

  Verifica se este Float vale NaN (not-a-number);

- **static boolean isNaN (float N):**

  Verifica se N vale NaN (not-a-number);

- **long longValue ():**

  Retorna o valor deste Float na forma de um long;

- **static float parseFloat (String S):**
  Retorna um float que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;

- **short shortValue ():**
  Retorna o valor deste Float na forma de um short;

- **static String toString (float N):**
  Retorna um String que contém somente caracteres que representam dígitos decimais que expressam o valor de N;

- **static Float valueOf (String S):**
  Retorna uma instância da classe Float que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada.

A Classe Double

Derivada da classe Number, a classe Double possui também outros membros além daqueles herdados de sua classe base.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**

  - **static double MIN_VALUE:**
    Representa o valor do menor real de dupla precisão que uma instância da classe Double é capaz de representar;

  - **static double MAX_VALUE:**
    Representa o valor do maior real de dupla precisão que uma instância da classe Double é capaz de representar;

  - **static double NEGATIVE_INFINITY:**
    Representa o valor especial -∞;

- **static double POSITIVE_INFINITY:**

  Representa o valor especial +∞;

- **static double NaN:**

  Representa o valor especial que simboliza algo que não é um número (*not a number*);

- **Construtores:**

  - **Double (double N):**

    Constroi uma instância da classe Double a partir do valor N do tipo double;

  - **Double (String S):**

    Constroi uma instância da classe Double a partir da instância S da classe String;

- **Métodos:**

  - **byte byteValue ():**

    Retorna o valor deste Double na forma de um byte;

  - **int compareTo (Double N):**

    Compara este Double com o Double N; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo;

  - **static long doubleToLongBits (double N):**

    Retorna em um long a seqüência de bits que forma N;

  - **double doubleValue ():**

    Retorna o valor deste Double na forma de um double;

  - **float floatValue ():**

    Retorna o valor deste Double na forma de um float;

  - **int intValue ():**

    Retorna o valor deste Double na forma de um int;

  - **boolean isInfinite ():**

    Verifica se este Double vale infinito;

- **static boolean isInfinite (double N):**
  Verifica se N vale infinito;
- **boolean isNaN ():**
  Verifica se este Double vale NaN (not-a-number);
- **static boolean isNaN (double N):**
  Verifica se N vale NaN (not-a-number);
- **static double longBitsToDouble (long N):**
  Retorna em um double a seqüência de bits que forma N;
- **long longValue ():**
  Retorna o valor deste Double na forma de um long;
- **static double parseDouble (String S):**
  Retorna um double que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada;
- **short shortValue ():**
  Retorna o valor deste Double na forma de um short;
- **static String toString (double N):**
  Retorna um String que contém somente caracteres que representam dígitos decimais que expressam o valor de N;
- **static Double valueOf (String S):**
  Retorna uma instância da classe Double que representa o valor numérico expresso pelos caracteres de S; o String S deve conter caracteres que expressam um valor numérico válido em decimal, caso contrário a exceção NumberFormatException será lançada.

A classe Boolean

Veja abaixo a interface que a classe Boolean especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**
  - **static Boolean FALSE:**
    Representa uma instância da classe Boolean que vale o correspondente a false;
  - **static Boolean TRUE:**
    Representa uma instância da classe Boolean que vale o correspondente a true;
- **Construtores:**
  - **Boolean (boolean V):**
    Constroi uma instância da classe Boolean a partir do valor V do tipo boolean;
  - **Boolean (String S):**
    Constroi uma instância da classe Boolean a partir da instância S da classe String;
- **Métodos:**
  - **boolean booleanValue ():**
    Retorna o valor deste Boolean na forma de um boolean;
  - **static boolean getBoolean (String S):**
    Retorna um valor boolean que representa o valor da propriedade do sistema chamada S;
  - **static Boolean valueOf (String S):**
    Retorna uma instância da classe Boolean que representa o valor lógico expresso pelos caracteres de S; o String S deve conter apenas "true" ou "false".

A classe Character

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **Character (char C):**
    Constroi uma instância da classe Character a partir do valor C do tipo char;
- **Métodos:**

- **char charValue ():**

  Retorna o valor deste Character na forma de um char;

- **int compareTo (Boolean B):**

  Compara este Boolean com o Boolean B; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo;

- **static int digit (char C, int B):**

  Retorna o valor numérido do caractere C enquanto dígito da base B;

- **static char forDigit (int D, int B):**

  Retorna o caractere que, enquanto dígito da base B, corresponde ao valor numérido D;

- **static int getNumericValue (char C):**

  Retorna o valor numérico que correspondo ao caractere C na tabela UNICODE;

- **static int getType (char C):**

  Retorna o valor numérico indicando classe à qual pertence C;

- **static boolean isDefined (char C):**

  Verifica se o caractere C está definido na tabela UNICODE;

- **static boolean isDigit (char C):**

  Verifica se o caractere C é um digito;

- **static boolean isDefined (char C):**

  Verifica se o caractere C está definido na tabela UNICODE;

- **static boolean isLetter (char C):**

  Verifica se o caractere C é uma letra;

- **static boolean isLetterOrDigit (char C):**

  Verifica se o caractere C é uma letra ou então um dígito;

- **static boolean isLowerCase (char C):**

  Verifica se o caractere C é um caractere minúsculo;

- **static boolean isSpaceChar (char C):**
  Verifica se o caractere C é um espaço;
- **static boolean isUpperCase (char C):**
  Verifica se o caractere C é um é um caractere maiúsculo;
- **static boolean isWhiteSpace (char C):**
  Verifica se o caractere C é um espaço;
- **static char toLowerCase (char C):**
  Retorna o correspondente minúsculo do char C;
- **static char toUpperCase (char C):**
  Retorna o correspondente maiúsculo do char C;

Constantes

Constantes são instâncias de tipos das linguagens. Temos duas variedades de constantes, a saber, constantes literais e constantes simbólicas.

Constantes Literais

Constantes literais são aquelas que especificam literalmente uma instância de um tipo da linguagem. A seguir apresentaremos as constantes literais existentes na linguagem Java.

1. Constantes Literais do Tipo Inteiro

Uma constante inteira é constituída por uma seqüência de dígitos (eventualmente precedida por um hífen indicando sinal negativo).

Se terminar com letra l (ou L) será considerada do tipo long, caso contrário, será considerada do tipo int.

Se não começar com um dígito 0, será considerada decimal.

Se for precedida por 0x (ou 0X) será considerada hexadecimal. Neste caso, além dos dígitos, também as letras de a (ou A) até h (ou H) podem ser empregadas.

Se for precedida por 0, será considerada octal. Neste caso os dígitos 8 e 9 não podem ser empregados, já que não são dígitos octais válidos.

2. Constantes Literais do Tipo Real

Uma constante real é constituída por uma constante inteira (indicando a parte inteira da constante real), seguido pelo caractere ponto (indicando o ponto decimal), seguido por outra constante inteira (indicando a parte fracionária da constante real), seguido pelo caractere e (ou E) e por outra constante inteira (indicando a multiplicação por uma potência de 10).

Se terminar com letra f (ou F) será considerada do tipo float, caso contrário, será considerada do tipo double. Também será considerada do tipo double se terminar com a letra d (ou D).

Tanto a parte inteira como a parte fracionária podem ser omitidas (mas não ambas). Tanto o ponto decimal seguida da parte fracionária como o `e` (ou `E`) e o expoente podem ser omitidos (mas não ambos). O sufixo de tipo também pode ser omitido.

3. Constantes Literais do Tipo char

Uma constantes caractere é constituída por um único caracteres delimitado por apostrofes (‘c’). Constantes caractere são do tipo char.

Certos caracteres não visíveis, o apóstrofe (‘), as aspas (“) e a barra invertida (\) podem ser representados de acordo com a seguinte tabela de seqüências de caracteres:

‘\b’	retrocesso (backspace)
‘\t’	tabulação (tab)
‘\n’	nova linha (new line)
‘\f’	avanço de formulário (form feed)
‘\r’	retorno do carro (carriage return)
‘\’’	apóstrofe (single quote)
‘\”’	aspas (double quote)
‘\\’	barra invertida (backslash)
‘\uHHHH’	caractére unicode (HHHH em hexa)

Uma seqüência como estas, apesar de constituídas por mais de um caractere, representam um único caractere.

4. **Constantes do Tipo boolean**

   Existem somente duas constantes booleanas, a saber: true e false.

5. **Constantes do Tipo String**

   Toda constante literal do tipo String é constituída por uma seqüência de zero ou mais caracteres delimitados por aspas (“”).

Constantes Simbólicas

Constantes simbólicas são aquelas que associam um nome a uma constante literal, de forma que as referenciamos no programa pelo nome, e não pela menção de seu valor literal.

Sendo Tipo um tipo, $Const_i$ nomes de identificadores e $Expr_i$ expressões que resultam em valores do tipo Tipo, temos que a forma geral de uma declaração de constantes simbólicas é como segue:

$$\text{final Tipo } Const_1 = Expr_1, Const_2 = Expr_2, \ldots, Const_n = Expr_n;$$

Cadeias de Caracteres

A Classe String

Esta classe implementa objetos que representam cadeias de caracteres. Tais objetos são inalteráveis uma vez criados.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **String ():**
    Constroi um *string* vazio;
  - **String (byte VB []):**
    Constroi um *string* a partir do vetor de caracteres VB;

- **String (byte [], int P, int Q):**
  Constroi um *string* a partir do vetor de bytes VB (para tanto, usa a partir da posição P de VC, Q bytes);
- **String (char VC []):**
  Constroi um *string* a partir do vetor de caracteres VC;
- **String (char VC [], int P, int Q):**
  Constroi um *string* a partir do vetor de caracteres VC (para tanto, usa a partir da posição P de VC, Q caracteres);
- **String (String S):**
  Constroi um *string* a partir de uma instância da classe String;
- **String (StringBuffer B):**
  Constroi um *string* a partir de uma instância da classe StringBuffer B.

- **Métodos:**
  - **char charAt (int P):**
    Retorna o caractere que esta na posição P deste String (é importante ressaltar que, em Java, os *string*s começam da posição zero);
  - **int compareTo (String S):**
    Retorna (1) um número negativo se este String for menor que o String S; (2) zero, se o este String for igual ao *string* S; e (3) um número positivo se este String for maior que o String S;
  - **int compareToIgnoreCase (String S):**
    Ignorando diferenças entre minúsculas e maiúsculas, retorna (1) um número negativo se este String for menor que o String S; (2) zero, se o este String for igual ao *string* S; e (3) um número positivo se este String for maior que o String S;
  - **String concat (String S):**
    Retorna a concatenação deste String com o String S;

- **static String copyValueOf (char VC []):**
  Retorna um String construído a partir do vetor de caracteres VC;

- **static String copyValueOf (char VC [], int P, int Q):**
  Retorna um String *tring* a partir do vetor de caracteres VC (para tanto, usa a partir da posição P de VC, Q caracteres);

- **boolean endsWith (String S):**
  Verifica se este String termina com sufixo S;

- **boolean equalsIgnoreCase (String S):**
  Ignorando diferenças entre minúsculas e maiúsculas, verifica se este String é igual ao String S;

- **byte[] getBytes ():**
  Retorna a conversão deste String em um vetor de bytes;

- **void getChars (int IO, int FO, char[] D, int ID):**
  Instancia e preenche o vetor D, a partir da posição ID, com os caracteres deste String compreendidos entre IO e FO;

- **int indexOf (char C):**
  Retorna a primeira posição deste String onde encontramos o caractere C (retorna –1 caso C não ocorra neste String);

- **int indexOf (char C, int P):**
  Retorna a primeira posição a partir da posição P deste String onde encontramos o caractere C (retorna –1 caso C não ocorra neste String);

- **int indexOf (String S):**
  Retorna a primeira posição deste String onde se encontramos o substring S (retorna –1 caso S não seja substring deste String);

- **int indexOf (String S, int P):**
  Retorna a primeira posição a partir da posição P deste String onde se encontramos o substring S (retorna –1 caso S não seja substring deste String);

- **int lastIndexOf (char C):**
  Retorna a última posição deste String onde encontramos o caractere C (retorna –1 caso C não ocorra neste String);

- **int lastIndexOf (char C, int P):**
  Retorna a última posição a partir da posição P deste String onde encontramos o caractere C (retorna –1 caso C não ocorra neste String);

- **int lastIndexOf (String S):**
  Retorna a última posição do *string* onde se encontramos o substring S (retorna –1 caso S não seja substring deste String);

- **int lastIndexOf (String S, int P):**
  Retorna a última posição a partir da posição P deste String onde se encontramos o substring S (retorna –1 caso S não seja substring deste String);

- **int length ():**
  Retorna o tamanho deste String;

- **boolean regionMatches (boolean IC, int P, String S, int PS, int Q):**
  Verifica se, a partir da posição P, Q caracteres deste String, são iguais a Q caracteres de S, a partir da posição PS; ignorar-se-á diferenças de minúsculas e maiúsculas se IC for true, e considerar-se-á tais diferenças, caso contrário;

- **boolean regionMatches (int P, String S, int PS, int Q):**
  Verifica se, a partir da posição P, Q caracteres deste String, são iguais a Q caracteres de S, a partir da posição PS;

- **String replace (char V, char N):**
  Retorna um String em tudo idêntico a este String, exceto pelo fato de que, no primeiro, todos as aparições do caractere V foram trocadas pelo caractere N;

- **boolean startsWith (String S):**
  Verifica se este String tem o String S como prefixo;

- **boolean startsWith (String S, int P):**
  Verifica se este String tem o String S como substring a partir da posição P;

- **String substring (int I):**

  Retorna o substring deste String se inicia na posição I e vai até a última posição deste String;

- **String substring (int I, int F):**

  Retorna o substring deste String que se inicia na posição I e vai até a posição F;

- **char[] toCharArray ():**

  Retorna, num vetor de char, todos os caracteres deste String;

- **String toLowerCase ():**

  Retorna um String em tudo idêntico a este String, exceto pelo fato de que, no primeiro, todas as letras serão transformadas em minúsculas;

- **String toUpperCase ():**

  Retorna um String em tudo idêntico a este String, exceto pelo fato de que, no primeiro, todas as letras serão transformadas em maiúsculas;

- **String trim ():**

  Retorna um String em tudo idêntico a este String, exceto pelo fato de que, no primeiro, todos os espaços iniciais e finais terão sido removidos;

- **static String valueOf (boolean B):**

  Retorna um String que representa o boolean B;

- **static String valueOf (char C):**

  Retorna um String que representa o char C;

- **static String valueOf (char VC []):**

  Retorna um String que representa os caracteres de VC;

- **static String valueOf (char VC [], int P, int Q):**

  Retorna um String que representa Q caracteres de VC a partir da posição P;

- **static String valueOf (int I):**

  Retorna um String que representa o int I;

- **static String valueOf (double D):**
  Retorna um String que representa o double D;
- **static String valueOf (float F):**
  Retorna um String que representa o float F;
- **static String valueOf (long L):**
  Retorna um String que representa o long L;
- **static String valueOf (Object O):**
  Retorna um String que representa o Object O; o objeto O deve possuir uma função membro de instância chamada toString () quer resulta nessa representação;

A Classe StringBuffer

Esta classe implementa objetos que representam cadeias de caracteres alteráveis após terem sido criadas.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **StringBuffer ():**
    Constroi um StringBuffer vazio;
  - **StringBuffer (int T):**
    Constroi um StringBuffer de tamanho T;
  - **StringBuffer (String S):**
    Constroi um StringBuffer a partir de uma instância da classe String;
- **Métodos:**
  - **void append (boolean B):**
    Acrescenta ao StringBuffer o boolean B representado sob a forma de um String;
  - **void append (char C):**
    Acrescenta ao StringBuffer o caractere C;

- **void append (char VC []):**
  Acrescenta ao StringBuffer os caracteres do vetor de caracteres VC;

- **void append (char VC [], int I, int Q):**
  Acrescenta ao StringBuffer Q caracteres a partir da posição I do vetor de caracteres VC;

- **void append (double D):**
  Acrescenta ao StringBuffer o real de dupla precisão D representado sob a forma de um String;

- **void append (float F):**
  Acrescenta ao StringBuffer o real F representado sob a forma de um String;

- **void append (int I):**
  Acrescenta ao StringBuffer o inteiro I representado sob a forma de um String;

- **void append (long L):**
  Acrescenta ao StringBuffer o inteiro longo L representado sob a forma de um String;

- **void append (Object O):**
  Acrescenta ao StringBuffer o Object O representado sob a forma de um String (o Object O deve possuir uma função membro de instância chamada toString () que resulta nessa representação);

- **void append (String S):**
  Acrescenta ao StringBuffer o String S;

- **int capacity ():**
  Retorna a capacidade do StringBuffer;

- **char charAt (int P):**
  Retorna o caractere que esta na posição P do StringBuffer (é importante ressaltar que, em Java, os StringBuffer começam da posição zero);

- **StringBuffer delete (int I, int F):**
  Retorna um StringBuffer idêntico a este StringBuffer, exceto pelo fato de que os

caracteres localizados entre as posições entre I e F deste StringBuffer foram removidos, i.e., não estão presentes no resultado;

- **StringBuffer deleteCharAt (int P):**
  Retorna um StringBuffer idêntico a este StringBuffer, exceto pelo fato de que o caractere localizado na posição P deste StringBuffer foi removido, i.e., não está presente no resultado;

- **void EnsureCapacity (int M):**
  Assegura que no mínimo a capacidade deste StringBuffer seja no mínimo igual a M posições;

- **void getChars (int IO, int FO, char[] D, int ID):**
  Instancia e preenche o vetor D, a partir da posição ID, com os caracteres deste StringBuffer compreendidos entre IO e FO;

- **void insert (int P, boolean B):**
  Insere na posição P deste StringBuffer o boolean B representado sob a forma de um String;

- **void insert (int P, char C):**
  Insere o caractere C na posição P deste StringBuffer;

- **void insert (int P, char VC []):**
  Insere os caracteres do vetor VC na posição P deste StringBuffer;

- **void insert (int P, char VC [], int I, int Q):**
  Insere na posição P deste StringBuffer Q caracteres a partir da posição I do vetor VC;

- **void insert (int P, double D):**
  Insere na posição P deste StringBuffer o double D representado sob a forma de um String;

- **void insert (int P, float F):**
  Insere na posição P deste StringBuffer o float F representado sob a forma de um String;

- **void insert (int P, int I):**
  Acrescenta ao StringBuffer o inteiro I representado sob a forma de um String;
- **void insert (int P, long L):**
  Insere na posição P deste StringBuffer o long L representado sob a forma de um String;
- **void insert (int P, Object O):**
  Insere na posição P deste StringBuffer o Object O representado sob a forma de um String (o Object O deve possuir uma função membro de instância chamada toString () que resulta nessa representação);
- **void insert (int P, String S):**
  Insere na posição P deste StringBuffer o String S;
- **int length ():**
  Retorna o tamanho do StringBuffer;
- **void replace (int I, int F, String S):**
  Substitui os caracteres localizados entre as posições I e F deste StringBuffer pelo String S;
- **void reverse ():**
  Inverte a seqüência dos caracteres deste StringBuffer;
- **char setCharAt (int P, char C):**
  Coloca o caractere C na posição P do StringBuffer (é importante ressaltar que, em Java, os StringBuffer começam da posição zero);
- **void setLength (int T):**
  Ajusta o tamanho do StringBuffer para T;
- **String substring (int P):**
  Retorna um String contendo os caracteres a partir da posição P deste StringBuffer;

- **String substring (int I, int F):**
  Retorna um String contendo os caracteres compreendidos entre as posições I e F deste StringBuffer.

A Classe StringTokenizer

Esta classe implementa métodos capazes de quebrar um *string* em *tokens*. O delimitador pode ser especificado quando da instanciação da classe ou a cada *token*. Tem-se que o delimitador default é o caractere espaço em branco.

Classes que fazem uso desta classe em sua implementação, devem ter logo no início do arquivo onde forem definidas a seguinte diretiva:

```
import java.util.StringTokenizer;

ou

import java.util.*;
```

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **StringTokenizer (String S):**
    Constroi uma nova instância da classe StringTokenizer para extrair *tokens* do String S (considera o caractere em branco como delimitador);
  - **StringTokenizer (String S, String D):**
    Constroi uma nova instância da classe StringTokenizer para extrair *tokens* do String S, considerando como delimitadores os caracteres presentes no String D;
  - **StringTokenizer (String S, String D, boolean DT):**
    Constroi uma nova instância da classe StringTokenizer para extrair *tokens* do String S, considerando como delimitadores os caracteres presentes no String D; se DT for true, os delimitadores também serão considerados *tokens* e retornados como Strings de comprimento unitário e se DT for false, os delimitadores serão ignorados;

- **Métodos:**
  - **int countTokens ():**
    Retorna a quantidade de *tokens* disponíveis para serem extraídos;
  - **boolean hasMoreElements ():**
    Como a classe StringTokenizer implementa a interface Enumeration, este método existe e implementa a mesma funcionalidade que o método hasMoreTokens;
  - **boolean hasMoreTokens ():**
    Verifica se existem mais *tokens* para serem estraídos do StringTokenizer;
  - **boolean nextElement ():**
    Como a classe StringTokenizer implementa a interface Enumeration, este método existe e implementa a mesma funcionalidade que o método nextToken.
  - **String nextToken ():**
    Retorna o próximo *token* disponível.

Funções

Convencionalmente entendemos que funções são abstrações de processo, i.e., pelo nome de uma função identificamos e ativamos o processo que ela abstrai, e através de seu retorno e de seus eventuais efeitos colaterais obtemos os resultados de seu processamento.

Num programa orientado a objetos, no entanto, entendemos que as funções representam a descrição das ações que uma classe ou um objeto dispões a fim de responder mensagens que lhes são dirigidas.

Por isso entendemos que toda função está vinculada à uma classe ou a um objeto, i.e., não faz sentido imaginar uma função "avulsa" como temos em um programa convencional.

Além disso, entendemos que a chamada de uma função representa o envio de uma mensagem à classe ou ao objeto ao qual ela se vincula, e que seu retorno representa a resposta àquela mensagem.

Por isso não faz sentido simplesmente chamar uma função como fazemos numa linguagem convencional. Já que entendemos uma chamada de função como o envio de uma mensagem, é preciso mencionar o destinatário dessa mensagem, i.e., a classe ou o objeto que deverá recebê-la.

Funções são definidas mencionando o tipo do retorno da função, seguido pelo identificador da função, seguido por um par de parênteses (()) contendo, opcionalmente, a lista dos parâmetros formais da função, seguido por um bloco de comandos representando o corpo da função.

Uma lista de parâmetros formais nada mais é do que uma série de definições de parâmetros formais separadas por vírgulas (,). A definição de um parâmetro formal é feita mencionando o nome do tipo do parâmetro e em seguida o identificador do parâmetro formal. É importante ressaltar que, em Java, a única mecanismo de passagem de parâmetro é a passagem por valor.

Não existe em Java o conceito de procedimento, muito embora possamos definir funções que não retornam valor nenhum, o que dá no mesmo. Isto pode ser feito dizendo que o tipo do retorno da função é void.

O comando return seguido pela expressão cujo valor se de deseja retornar é empregado para fazer o retorno do resultado da função a seu chamante.

Variáveis

Convencionalmente entendemos que variáveis são células de memória capazes de armazenarem valores para serem futuramente recuperados.

Num programa orientado a objetos, no entanto, o principal papel das variáveis é o de representar o estado interno de uma classe ou de um objeto. Em Java poderemos ainda encontrar variáveis locais servindo a propósitos temporários. Não encontramos variáveis globais em Java.

Variáveis são definidas quando mencionamos o nome de um tipo, e em seguida uma série de identificadores separados por vírgulas (,) e tendo no final um ponto-e-vírgula (;). Cada um dos identificadores será uma variável do tipo que encabeçou a definição.

Sendo Tipo um tipo e Var_i nomes de identificadores, temos que a forma geral de uma declaração de variáveis é como segue:

$$Tipo\ Var_1, Var_2, \ldots, Var_n;$$

Variáveis podem ser iniciadas no ato de sua definição, i.e., podem ser definidas e receber um valor inicial. Isto pode ser feito acrescentando um sinal de igual (=) e o valor inicial desejado imediatamente após o identificador da variável que desejamos iniciar.

Sendo Tipo um tipo, Var_i nomes de identificadores e $Expr_i$ expressões que resultam em valores do tipo Tipo, temos que a forma geral de uma declaração de variáveis iniciadas é como segue:

$$Tipo\ Var_1 = Expr_1, Var_2 = Expr_2, \ldots, Var_n = Expr_n;$$

Acessibilidade de Membros

Podemos estabelecer três níveis de proteção de acesso aos membros de uma classe através do uso ou não de qualificadores específicos. Nesta seção estudaremos dois desses modificadores: o modificador public e o modificador private.

Membros qualificados com public são acessíveis a qualquer função, seja ela também membro da classe ou não. Membros qualificados como private são acessíveis somente às demais funções membro da classe.

Um programador disciplinado evita qualificar dados variáveis membro com public, já que o principal papel destes é descrever o estado interno de classes o objetos (e como o próprio nome diz, o estado interno é interno). O mesmo não diríamos com respeito aos dados constantes (qualificados com final).

Um programador disciplinado também evita qualificar indiscriminadamente funções membro com public. Apenas as funções que descrevem as ações que servem para responder a mensagens externas à classe devem ser públicas.

Entrada de Dados

Para que seja possível fazer entrada de dados, é preciso, antes de mais nada, importar, da biblioteca de entrada e saída, as classes necessárias à realização da referida operação. Para tanto, os comandos abaixo devem ser incluídos no ínicio do arquivo que contém a função que realizará operações de entrada:

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

ou

import java.io.*;
```

Mais adiante, neste texto, teremos a oportunidade de compreender melhor as razões da exigência que ora será feita, bem como de conhecer alternativas que permitiriam descumprí-la. No momento, diremos que é preciso que toda função que realiza operações de entrada, assim como toda função que a ativar direta ou indiretamente uma função que realize operações de entrada, tenham acrescentado no final do seu cabeçalho a expressão:

```
throws IOException
```

Deve-se então declarar um objeto da classe BufferedReader que atuará como um prestador de serviços de entrada de dados. Isto pode ser feito como segue:

```
BufferedReader Entrada = new BufferedReader (new InputStreamReader (System.in));
```

Objetos da classe BufferedReader, como é o caso do objeto Entrada, somente podem ser usados para ler cadeias de caracteres. A solicitação deste serviço pode ser feita como no comando abaixo:

```
String S = Entrada.readLine ();
```

Cadeias de caracteres que contém somente caracteres que conformam constantes literais dos tipos primitivos da linguagem Java, podem ser convertidas nesses mesmos tipos conforme segue:

- **Conversão de String para char:**

  char C = S.charAt (0);

- **Conversão de String para byte:**

  byte b = Byte.parseByte (S);

  ou, se preferirmos,

  byte b = new Byte (S).byteValue ();

  ou, se preferirmos,

  byte b = Byte.valueOf (S).byteValue ();

- **Conversão de String para short:**

  short s = Short.parseShort (S);

  ou, se preferirmos,

  short s = new Short (S).shortValue ();

  ou, se preferirmos,

  short s = Short.valueOf (S).shortValue ();

- **Conversão de String para int:**

  int i = Integer.parseInt (S);

  ou, se preferirmos,

  int i = new Integer (S).intValue ();

  ou, se preferirmos,

  int i = Integer.valueOf (S).intValue ();

- **Conversão de String para long:**

  long l = Long.parseLong (S);

  ou, se preferirmos,

  long l = new Long (S).longValue ();

  ou, se preferirmos,

  long l = Long.valueOf (S).longValue ();

- **Conversão de String para float:**

  float f = Float.parseFloat (S);

  ou, se preferirmos,

float f = new Float (S).floatValue ();

ou, se preferirmos,

float f = Float.valueOf (S).floatValue ();

- **Conversão de String para double:**

  double d = Double.parseDouble (S);

  ou, se preferirmos,

  double d = new Double (S).doubleValue ();

  ou, se preferirmos,

  double d = Double.valueOf (S).doubleValue ();

- **Conversão de String para boolean:**

  boolean b = Boolean.parseBoolean (S);

  ou, se preferirmos,

  boolean b = new Boolean (S).booleanValue ();

  ou, se preferirmos,

  boolean b = Boolean.valueOf (S).booleanValue ().

[C:\ExplsJava\Expl_02\BoasVindas.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class BoasVindas
{
    public static void main (String Args []) throws IOException
    {
        InputStreamReader FonteDeDados;
        BufferedReader    Entrada;

        FonteDeDados = new InputStreamReader (System.in);
        Entrada      = new BufferedReader    (FonteDeDados);

        System.out.println ();

        System.out.print   ("Qual o seu nome? ");
        String Nome        = Entrada.readLine ();

        System.out.println ("Bem vindo(a) ao estudo de JAVA, " +
                            Nome +
                            "!");

        System.out.println ();

        System.out.print   ("Com que nota voce pretende passar? ");
```

```
        String Nota1;
        Nota1 = Entrada.readLine ();

        Float Nota2;
        Nota2 = new Float (Nota1);

        float Nota3 = Nota2.floatValue ();

        System.out.println ();

        System.out.println (Nome +
                            ", desejo sucesso a voce;");

        System.out.println ("que voce passe com " +
                            Nota3 +
                            " ou mais!");

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_02> javac -classpath . BoasVindas.java
C:\ExplsJava\Expl_02> java -classpath . -classpath . BoasVindas
```

Isto poderia produzir no console a seguinte interação:

```
Qual o seu nome? Jose
Bem vindo ao estudo de JAVA, Jose!

Com que nota voce pretende passar? 7.0
Jose, desejo sucesso a voce;
que voce passe com 7.0 ou mais!
```

[C:\ExplsJava\Expl_03\BoasVindas.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class BoasVindas
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader (
                                 new InputStreamReader (
                                 System.in));

        System.out.println ();

        System.out.print   ("Qual o seu nome? ");
        String Nome        = Entrada.readLine ();

        System.out.println ("Bem vindo(a) ao estudo de JAVA, " +
```

```
                            Nome +
                            "!");

        System.out.println ();

        System.out.print   ("Com que nota voce pretende passar? ");

        float Nota = new Float (Entrada.readLine ()).floatValue ();

        System.out.println ();

        System.out.println (Nome +
                            ", desejo sucesso a voce;");

        System.out.println ("que voce passe com " +
                            Nota +
                            " ou mais!");

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_03> javac -classpath . BoasVindas.java
C:\ExplsJava\Expl_03> java -classpath . -classpath . BoasVindas
```

Isto poderia produzir no console a seguinte interação:

```
Qual o seu nome? Jose
Bem vindo ao estudo de JAVA, Jose!

Com que nota voce pretende passar? 7.0
Jose, desejo sucesso a voce;
que voce passe com 7.0 ou mais!
```

A Classe Math

A classe Math contém muitas funções matemáticas úteis, bem como muitas constantes também úteis. A classe Math não deve ser instanciada e, como trata-se de uma classe final, não pode ser usada como base para derivação de outra classe.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Campos:**
  - **PI:**

    Representa a contante matemática Pi (3.141593…);

- **E:**

  Representa a contante matemática E, tambem conhecida como número de Neper (2.718282…).

- **Métodos:**

  - **static double abs (double D):**

    Retorna o valor absoluto do double D;

  - **static float abs (float F):**

    Retorna o valor absoluto do float F;

  - **static int abs (int I):**

    Retorna o valor absoluto do int I;

  - **static long abs (long L):**

    Retorna o valor absoluto do long L;

  - **static double acos (double D):**

    Retorna o arco cujo cosseno é D;

  - **static double asin (double D):**

    Retorna o arco cujo seno é D;

  - **static double atan (double D):**

    Retorna o arco cuja tangente é D;

  - **static double atan2 (double X, double Y):**

    Retorna o θ da coordenada polar (R,θ) equivalente à cordenada carteziana (X,Y);

  - **static double ceil (double D):**

    Retorna o menor número inteiro igual ou superior a D;

  - **static double cos (double D):**

    Retorna o cosseno de D;

  - **static double exp (double D):**

    Retorna e^{D};

- **static double floor (double D):**
  Retorna o maior número inteiro igual ou inferior a D;
- **static double IEEEremainder (double N, double D):**
  Retorna o resto da divisão inteira de N por D (conforme recomendação do padrão IEEE 754);
- **static double log (double D):**
  Retorna ln (D);
- **static double max (double D1, double D2):**
  Retorna o máximo entre D1 e D2.
- **static float max (float F1, float F2):**
  Retorna o máximo entre F1 e F2;
- **static int max (int I1, int I2):**
  Retorna o máximo entre I1 e I2;
- **static long max (long L1, long L2):**
  Retorna o máximo entre L1 e L2;
- **static double min (double D1, double D2):**
  Retorna o mínimo entre D1 e D2;
- **static float min (float F1, float F2):**
  Retorna o mínimo entre F1 e F2;
- **static int min (int I1, int I2):**
  Retorna o mínimo entre I1 e I2;
- **static long min (long L1, long L2):**
  Retorna o mínimo entre L1 e L2;
- **static double pow (double B, double P):**
  Retorna B^P;

- static double random ():

  Retorna um número aleatório R tal que $0.0 \leq R < 1.0$;

- **static double rint (double D):**

  Retorna em um double o inteiro mais próximo de D;

- **static long round (double D):**

  Retorna o inteiro mais próximo de D;

- **static int round (float F):**

  Retorna o inteiro mais próximo de F;

- **static double sin (double D):**

  Retorna o seno de D;

- **static double sqrt (double D):**

  Retorna a raiz quadrada de D;

- **static double tan (double D):**

  Retorna a tangente de D;

- **static toDegrees (double A):**

  Retorna em graus o valor angular A fornecido em radianos;

- **static toRadians (double A):**

  Retorna em radianos o valor angular A fornecido em graus.

Expressões

Uma expressão é uma seqüência de operadores e operandos que especifica um computação e resulta um valor.

A ordem de avaliação de subexpresssões é determinada pela precedência e pelo agrupamento dos operadores. As regras matemáticas usuais para associatividade e comutatividade de operadores podem ser aplicadas somente nos casos em que os operadores sejam realmente associativos e comutativos.

A ordem de avaliação dos operandos de operadores individuais é indefinida. Em particular, se um valor é modificado duas vezes numa expressão, o resultado da expressão é indefinido, exceto no caso dos operadores envolvidos garantirem alguma ordem.

Operadores Aritméticos Convencionais

Os operadores aritméticos da linguagem Java são, em sua maioria, aqueles que usualmente encontramos nas linguagens de programação imperativas. Todos eles operam sobre números e resultam números. São eles:

1. + (soma);
2. - (subtração);
3. * (multiplicação);
4. / (divisão);
5. % (resto da divisão inteira); e
6. - (menos unário).

Não existe em Java um operador que realize a divisão inteira, ou, em outras palavras, em Java uma divisão sempre resulta um número real. Para resolver o problema, podemos empregar um conversão de tipo para forçar o resultado da divisão a ser um inteiro. Neste caso, o real resultante terá truncada a sua parte fracionária, se transformando em um inteiro.

O operador % (resto da divisão inteira) opera sobre dois valores integrais. Seu resultado também será um valor integral. Ele resulta o resto da divisão inteira de seu primeiro operando por seu segundo operando.

O operador - (menos unário) opera sobre um único operando, e resulta o número que se obtém trocando o sinal de seu operando.

Operadores de Incremento e Decremento

Os operadores de incremento e decremento da linguagem Java não são usualmente encontrados em outras linguagens de programação. Todos eles são operadores unários e operam sobre variáveis inteiras e resultam números inteiros. São eles:

1. ++ (préfixo);
2. -- (préfixo);
3. ++ (pósfixo);
4. -- (pósfixo).

O operador ++ (préfixo) incrementa seu operando e produz como resultado seu valor (já incrementado).

O operador -- (préfixo) decrementa seu operando e produz como resultado seu valor (já decrementado).

O operador ++ (pósfixo) incrementa seu operando e produz como resultado seu valor original (antes do incremento).

O operador -- (pósfixo) decrementa seu operando e produz como resultado seu valor original (antes do decremento).

Observe, abaixo, uma ilustração do uso destes operadores:

```
...
int a=7, b=13, c;
System.out.println (a+" "+b);       // 7 13

a--; b++;
System.out.println (a+" "+b);       // 6 14

--a; ++b;
System.out.println (a+" "+b);       // 5 15

c = a++ + ++b;
System.out.println (a+" "+b+" "+c); // 6 16 21
...
```

Operadores Relacionais

Os operadores relacionais da linguagem Java são, em sua maioria, aqueles que usualmente encontramos nas linguagens de programação imperativas. Todos eles operam sobre números e resultam valores lógicos. São eles:

1. == (igualdade);
2. != (desigualdade);

3. < (inferioridade);
4. <= (inferioridade ou igualdade);
5. > (superioridade); e
6. >= (superioridade ou igualdade).

Operadores Lógicos

Os operadores lógicos da linguagem Java são, em sua maioria, aqueles que usualmente encontramos nas linguagens de programação imperativas. Todos eles operam sobre valores lógicos e resultam valores lógicos. São eles:

1. && (conjunção ou *and*);
2. || (disjunção ou *or*); e
3. ! (negação ou *not*).

Operador de Atribuição com Operação Embutida

Os operadores aritméticos e de bit podem ser combinados com o operador de atribuição para formar um operador de atribuição com operação embutida.

Sendo Var uma variável, Opr um operador aritmético ou de bit e Expr uma expressão, temos que:

Var Opr= Expr;

é equivalente a

Var = Var Opr (Expr);

Observe, abaixo, uma ilustração do uso destes operadores:

```
...
short a, b, c, d, e;

a = a + (3 * b);              // a += 3*b;
b = b / (a - Math.sin (a));   // b /= a-Math.sin(a);
c = c << d;                   // c <<= d;
d = d & (a + b);              // d &= a+b;
e = e >> (d + 3);             // e >>= d+3;
...
```

Expressões Condicionais

Expressões condicionais representam uma forma compacta de escrever comandos a escolha de um dentre uma série possivelmente grande de valores. Substituem uma seqüência de ifs.

Sendo Cond uma condição booleana e $Expr_i$ expressões, temos que:

Condição? $Expr_1$: $Expr_2$

resulta $Expr_1$ no caso de Cond ser satisfeita, e $Expr_2$, caso contrário.

Observe, abaixo, uma ilustração do uso deste operador:

```
...
int a, b;
...
System.out.println (a>b?a:b);
...
```

equivale a

```
...
int a, b;
...
int maior;

if (a>b)
    maior = a;
else
    maior = b;

System.out.println (maior);
...
```

Conversões de Tipo

Expressões podem ser forçadas a resultar um certo tipo se precedidas pelo tipo desejado entre parênteses.

Sendo Tipo um tipo e Expr uma expressão que resulta um valor que não é do tipo Tipo, temos que:

(Tipo) Expr

resulta a expressão Expressão convertida para o tipo Tipo.

Observe, abaixo, uma ilustração do uso deste operador:

```
...
int a  = 65;
char c = (char)a;

System.out.println (c); // A
...
```

Operadores de Bit

Os operadores de bit da linguagem Java são, em sua maioria, aqueles que usualmente encontramos nas linguagens de programação imperativas. Todos eles operam sobre valores integrais e resultam valores integrais. São eles:

1. & (and bit a bit);
2. | (or bit a bit);
3. ^ (xor bit a bit);
4. ~ (not bit a bit);
5. << (shift left bit a bit); e
6. >> (shift right bit a bit).

Observe, abaixo, uma ilustração do uso destes operadores:

```
...
short a,
      b=0x7C3A, // 0111 1100 0011 1010
      c=ox1B3F; // 0001 1011 0011 1111

a = b & c;      // 0001 1000 0011 1010
a = b | c;      // 0111 1111 0011 1111
a = b ^ c;      // 0110 0111 0000 0101
a = ~b;         // 1000 0011 1100 0101
a = b << 3;     // 1110 0001 1101 0000
a = c >> 5;     // 0000 0000 1101 1001
...
```

A Classe BitSet

Esta classe implementa um tipo de dados que representa uma coleção de bits. A coleção crescerá dinamicamente a medida que mais bits forem necessários. Bits específicos são identificados por um inteiro não negativo. O primeiro bit sempre é o de ordem zero.

Classes que fazem uso desta classe em sua implementação, devem ter logo no início do arquivo onde forem definidas a seguinte diretiva:

```
import java.util.StringTokenizer;

ou

import java.util.*;
```

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **BitSet ():**
    Constroi uma instância vazia da classe BitSet;
  - **BitSet (int T):**
    Constroi uma instância da classe BitSet com capacidade para comportar até T bits;
- **Métodos:**
  - **void and (BitSet BS):**
    Faz um AND deste BitSet com o BS e deixa o resultado neste BitSet;
  - **void andNot (BitSet BS):**
    Torna 0 todos os bits deste BitSet cujo bit correspondente em BS vale 1;
  - **void clear (int P):**
    Torna 0 o bit da posição P deste BitSet;
  - **object clone ():**
    Retorna um clone deste BitSet;
  - **boolean get (int P):**
    Retorna true, se o bit da posição P deste BitSet for 1, e false, caso contrário;
  - **int length ():**
    Retorna o tamanho lógico deste BitSet, i.e., o número de ordem do bit mais significativo deste BitSet que vale 1 mais um;

- **void or (BitSet BS):**

  Faz um OR deste BitSet com o BS e deixa o resultado neste BitSet;

- **void set (int P):**

  Torna 1 o bit da posição P deste BitSet;

- **int size ():**

  Retorna o tamanho real deste BitSet, i.e., o tamanho total alocado a ele;

- **void xor (BitSet BS):**

  Faz um XOR deste BitSet com o BS e deixa o resultado neste BitSet.

[C:\ExplsJava\Expl_04\Media.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class Media
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        System.out.println ();

        System.out.print ("Qual o seu nome? ");
        String Nome = Entrada.readLine ();

        System.out.println ();

        System.out.print ("Quanto voce tirou na 1a Prova? ");
        float NtPrv1 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Quanto voce tirou na 2a Prova? ");
        float NtPrv2 = Float.parseFloat (Entrada.readLine ());

        float NtPrvs = (2*NtPrv1 + NtPrv2) / 3;

        System.out.println ();

        System.out.println ("Bem, " + Nome +
                            ", nao sei quantos trabalhos seu professor deu;");
        System.out.println ("acho 3 um bom numero... " +
                            "vamos considerar essa possibilidade...");

        System.out.println ();

        System.out.print ("Quanto voce tirou no 1o Trabalho? ");
        float NtTrb1 = Float.parseFloat (Entrada.readLine ());
```

```
        System.out.print ("Qual o peso do 1o Trabalho? ");
        float PsTrb1 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Quanto voce tirou no 2o Trabalho? ");
        float NtTrb2 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Qual o peso do 2o Trabalho? ");
        float PsTrb2 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Quanto voce tirou no 3o Trabalho? ");
        float NtTrb3 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Qual o peso do 3o Trabalho? ");
        float PsTrb3 = Float.parseFloat (Entrada.readLine ());

        float NtTrbs = (PsTrb1*NtTrb1 + PsTrb2*NtTrb2 + PsTrb3*NtTrb3) /
                       (PsTrb1        + PsTrb2        + PsTrb3       );

        System.out.println ();

        System.out.print ("Quantos PPs voce tem? ");
        int QtsPPs = Integer.parseInt (Entrada.readLine ());

        System.out.println ();

        float Media  = 0.7f*NtPrvs + 0.3f*NtTrbs + 0.1f*QtsPPs;

        System.out.println (Nome  + ", sua media foi " +
                            Media + "!");

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_04> javac -classpath . Media.java
C:\ExplsJava\Expl_04> java -classpath . -classpath . Media
```

Isto poderia produzir no console a seguinte interação:

```
Qual o seu nome? Jose

Quanto voce tirou na 1a Prova? 7
Quanto voce tirou na 2a Prova? 6

Bem, Jose, nao sei quantos trabalhos seu professor deu;
acho 3 um bom numero... vamos considerar essa possibilidade...

Quanto voce tirou no 1o Trabalho? 7
Qual o peso do 1o Trabalho? 1
Quanto voce tirou no 2o Trabalho? 6
Qual o peso do 2o Trabalho? 2
Quanto voce tirou no 3o Trabalho? 5
```

Qual o peso do 3o Trabalho? 3

Quantos PPs voce tem? 5

Jose, sua media foi 6.866667!

Comandos

Comandos são a unidade básica de processamento. Em Java não existe um repertório muito vasto de comandos. Muitos comandos que em outras linguagens existem, em Java são encontrados em bibliotecas de classe.

Comandos em Java, salvo raras exceções, são terminados por um ponto-e-vírgula (;).

O Comando de Atribuição

O comando de atribuição tem a seguinte forma básica: em primeiro lugar vem o identificador da variável receptora da atribuição, em seguida vem o operador de atribuição (=), em seguida vem a expressão a ser atribuída, em seguida vem um ponto-e-vírgula (`;`).

Sendo Var o identificador de uma variável e Expr uma expressão, temos abaixo a forma geral do comando de atribuição:

```
Var = Expr;
```

Blocos de Comando

Já conhecemos o conceito de bloco de comandos, que nada mais é do que uma série de comandos delimitada por um par de chaves (`{}`).

Blocos de comando não são terminados por ponto-e-vírgula (`;`).

Blocos de comando são muito úteis para proporcionar a execução de uma série de subcomandos de comandos que aceitam apenas um subcomando.

O Comando if

O comando `if` tem a seguinte forma básica: em primeiro lugar vem a palavra-chave `if`, em seguida vem, entre parênteses, a condição a ser avaliada, em seguida vem o comando a ser executado no caso da referida condição se provar verdadeira.

Sendo Cond uma condição booleana e Cmd um comando, temos abaixo a forma geral do comando if (sem else):

```
if (Cond)
        Cmd;
```

O comando `if` pode também executar um comando, no caso de sua condição se provar falsa. Para tanto, tudo o que temos que fazer é continuar o comando `if` que já conhecemos, lhe acrescentando a palavra chave `else`, e em seguida o comando a ser executado, no caso da referida condição se provar falsa.

Sendo Cond uma condição booleana e Cmd_i dois comandos, temos abaixo forma geral do comando if com else:

```
if (Cond)
        Cmd1;
else
        Cmd2;
```

Observe que, tanto no caso da condição de um comando `if` se provar verdadeira, quanto no caso dela se provar falsa, o comando `if` somente pode executar um comando. Isso nem sempre, ou melhor, quase nunca é satisfatório; é comum desejarmos a execução de mais de um comando.

Por isso, nos lugares onde se espera a especificação de um comando, podemos especificar um bloco de comandos.

O Comando `switch`

O comando `switch` tem a seguinte forma: em primeiro lugar vem a palavra-chave `switch`, em seguida vem, entre parênteses, uma expressão integral a ser avaliada, em seguida vem, entre chaves (`{}`), uma série de casos.

Um caso tem a seguinte forma básica: em primeiro lugar vem a palavra-chave `case`, em seguida vem uma constante integral especificando o caso, em seguida vem o caractere dois pontos (`:`), em seguida vem uma série de comandos a serem executados no caso.

Sendo Expr uma expressão, $Const_I$ constantes literais do mesmo tipo que Expr e Cmd_{IJ} comandos, temos abaixo uma das formas gerais do comando switch:

```
switch (Expr)
{
        case Const1:   Cmd1a;
                       Cmd1b;
                       …
        case Const2:   Cmd2a;
                       Cmd2b;
                       …
        case Const3:   Cmd3a;
                       Cmd3b;
                       …
        …
}
```

Se, em alguma circunstância, desejarmos executar uma mesma série de comandos em mais de um caso, tudo o que temos a fazer é especificar em seqüência os casos em questão, deixando para indicar somente no último deles a referida seqüência de comandos.

Sendo Expr uma expressão, $Const_{I,J}$ constantes literais do mesmo tipo que Expr e Cmd_{IJ} comandos, temos abaixo uma das formas gerais do comando switch:

```
switch (Expr)
{
        case Const1,1:
        case Const1,2:
        …              Cmd1a;
                       Cmd1b;
                       …
        case Const2,1:
        case Const2,2:
        …              Cmd2a;
                       Cmd2b;
                       …
        case Const3,1:
        case Const3,2:
        …              Cmd3a;
```

Cmd_{3b};
…
…
}

Se, em alguma circunstância, desejarmos executar uma série de comandos qualquer que seja o caso, tudo o que temos a fazer é especificar um caso da forma: em primeiro lugar vem a palavra-chave `default`, em seguida vem o caractere dois pontos (`:`), em seguida vem uma série de comandos a serem executados qualquer que seja o caso.

Sendo Expr uma expressão, $Const_{I,J}$ constantes literais do mesmo tipo que Expr e Cmd_{IJ} comandos, temos abaixo uma das formas gerais do comando switch:

switch (Expr)
{
case $Const_{1,1}$:
case $Const_{1,2}$:
… Cmd_{1a};
Cmd_{1b};
…
case $Const_{2,1}$:
case $Const_{2,2}$:
… Cmd_{2a};
Cmd_{2b};
…
…
default:
Cmd_{Da};
Cmd_{Db};
…
}

É importante ficar claro que o comando switch não se encerra após a execução da seqüência de comandos associada a um caso; em vez disso, todas as seqüências de comandos, associadas aos casos subseqüentes, serão também executadas.

Se isso não for o desejado, basta terminar cada seqüência (exceto a última), com um comando break.

Sendo Expr uma expressão, $Const_{I,J}$ constantes literais do mesmo tipo que Expr e Cmd_{IJ} comandos, temos abaixo uma das formas gerais do comando switch com breaks:

switch (Expr)
{
case $Const_{1,1}$:
case $Const_{1,2}$:
… Cmd_{1a};
Cmd_{1b};
…
break;
case $Const_{2,1}$:
case $Const_{2,2}$:
… Cmd_{2a};
Cmd_{2b};
…
break;
…
default:
Cmd_{Da};
Cmd_{Db};
…
}

Observe, abaixo, uma ilustração do uso deste comando:

```
...
int a, b, c;
char Operacao;
...
switch (Operacao)
{
    case '+': a = b + c;
              break;

    case '-': a = b - c;
              break;

    case '*': a = b * c;
              break;

    case '/': a = b / c;
}

System.out.println (a);
...
```

O Comando `while`

O comando `while` tem a seguinte forma básica: em primeiro lugar vem a palavra-chave `while`, em seguida vem, entre parênteses, a condição de iteração, em seguida vem o

comando a ser iterado. A iteração se processa enquanto a condição de iteração se provar verdadeira.

Sendo Cond uma condição booleana e Cmd um comando, temos abaixo a forma geral do comando while:

```
while (Cond)
    Cmd;
```

Observe que, conforme especificado acima, o comando `while` itera somente um comando. Isso nem sempre, ou melhor, quase nunca é satisfatório; é comum desejarmos a iteração de um conjunto não unitário de comandos.

Por isso, em lugar de especificar o comando a ser iterado, podemos especificar um bloco de comandos. Assim conseguiremos o efeito desejado.

O Comando `do-while`

O comando `do-while` tem a seguinte forma básica: em primeiro lugar vem a palavra-chave `do`, em seguida vem o comando a ser iterado, em seguida vem a palavra-chave `while`, em seguida vem, entre parênteses, a condição de iteração. A iteração se processa enquanto a condição de iteração se provar verdadeira.

A diferença que este comando tem com relação ao comando `while` é que este comando sempre executa o comando a ser iterado pelo menos uma vez, já que somente testa a condição de iteração após tê-lo executado. Já o comando `while` testa a condição de iteração antes de executar o comando a ser iterado, e por isso pode parar antes de executá-lo pela primeira vez.

Sendo Cond uma condição booleana e Cmd um comando, temos abaixo a forma geral do comando `do-while`:

```
do Cmd;
while (Cond);
```

Observe que, conforme especificado acima, o comando `do-while` itera somente um comando. Isso nem sempre, ou melhor, quase nunca é satisfatório; é comum desejarmos a iteração de um conjunto não unitário de comandos.

Por isso, em lugar de especificar o comando a ser iterado, podemos especificar um bloco de comandos. Assim, conseguiremos o efeito desejado.

O Comando for

O comando `for` tem a seguinte forma básica: em primeiro lugar vem a palavra-chave `for`, em seguida vem, entre parênteses e separadas por pontos-e-vírgulas (`;`), a sessão de iniciação, a condição de iteração, e a sessão de reiniciação. Em seguida vem o comando a ser iterado. A iteração se processa enquanto a condição de iteração se provar verdadeira.

No caso de haver mais de um comando na sessão de iniciação ou na sessão de reiniciação, estes devem ser separados por virgulas (`,`).

Sendo $Cmd_{I,i}$, $Cmd_{R,i}$ e Cmd comandos, e Cond uma condição booleana, temos abaixo a forma geral do comando `for` (os números indicam a ordem de execução das partes do comando `for`):

for ($Cmd_{I,1}$, $Cmd_{I,2,...}$; Cond; $Cmd_{R,1}$, $Cmd_{R,2,...}$) Cmd;

1	**2**		**3**
	5	**4**	**6**
	8	**7**	**9**
	11	**10**	**12**
	...		
	n	**n-1**	

Observe que, conforme especificado acima, o comando `for` itera somente um comando. Isso nem sempre, ou melhor, quase nunca é satisfatório; é comum desejarmos a iteração de um conjunto não unitário de comandos.

Por isso, no lugar onde se espera a especificação de um comando, podemos especificar um bloco de comandos. Assim, conseguiremos o efeito desejado.

Observe, abaixo, uma ilustração do uso deste comando, que escreve na tela 10 vezes a frase "Java e demais!":

```
...
for (int i=1; i<=10; i++)
    System.out.println ("Java e demais!");
...
```

equivale a

```
...
int i;

for (i=1; i<=10; i++)
    System.out.println ("Java e demais!");
...
```

equivale a

```
...
int i=1;

for (; i<=10; i++)
    System.out.println ("Java e demais!");
...
```

equivale a

```
...
int i=1;

for (; i<=10;)
{
    System.out.println ("Java e demais!");
    i++;
}
...
```

equivale a

```
...
int i=1;

for (;;)
{
    System.out.println ("Java e demais!");
    i++;
    if (i>10) break;
}
...
```

Observe, abaixo, outra ilustração do uso deste comando, que inverte um vetor v de 100 elementos:

```
...
for (int i=0, j=99; i<j; i++, j--)
{
    int temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
...
```

O Comando continue

O comando continue força o reinicio de uma nova iteração nos comandos while, do-while e for. Existem duas variações deste comando: a primeira, simplesmente

continue;

provoca o reinício imediato de uma nova iteração do comando iterativo, dentro do qual se encontra encaixado mais diretamente; e a segunda,

continue Rotulo;

provoca o reinício imediato de uma nova iteração do comando iterativo rotulado com Rotulo (para rotular um comando, basta preceder-lhe com o identificador do rotulo seguido pelo caractere dois-pontos).

Observe, abaixo, uma ilustração do uso deste comando, que escreve na tela os números de 1 a 7, inclusive, seguidos pelos números de 17 a 20, inclusive:

```
...
int i = 0;

while (i<20)
{
    i++;
    if (i>7 && i<17) continue;
    System.out.println (i);
}
...
```

Observe, abaixo, outra ilustração do uso deste comando:

```
...
loop_de_fora: while (cond1)
{
    ...
    while (cond2)
    {
        ...
        if (cond3) continue loop_de_fora;
        ...
    }
    ...
}
...
```

O Comando break

O comando break força a saída imediata dos comandos while, do-while, for e switch (veja abaixo os comandos while, do-while e for). Existem duas variações deste comando: a primeira, simplesmente

break;

provoca a saída imediata do comando while, do-while, for ou switch, dentro do qual se encontra encaixado mais diretamente; e a segunda,

break Rotulo;

provoca a saída imediata do comando while, do-while, for ou switch rotulado com Rotulo (para rotular um comando, basta preceder-lhe com o identificador do rotulo seguido pelo caractere dois-pontos).

Observe, abaixo, uma ilustração do uso deste comando, que escreve na tela os números de 1 a 7, inclusive:

```
...
int i = 0;

while (i<20)
{
    i++;
    if (i>7 && i<17) break;
    System.out.println (i);
}
...
```

Observe, abaixo, outra ilustração do uso deste comando:

```
...
loop_de_fora: while (cond1)
{
    ...
    while (cond2)
    {
        ...
        if (cond3) break loop_de_fora;
        ...
    }
    ...
}
...
```

[C:\ExplsJava\Expl_05\Media.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class Media
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        System.out.println ();

        System.out.print ("Qual o seu nome? ");
        String Nome = Entrada.readLine ();

        System.out.println ();

        System.out.print ("Quanto voce tirou na 1a Prova? ");
        float NtPrv1 = Float.parseFloat (Entrada.readLine ());

        System.out.print ("Quanto voce tirou na 2a Prova? ");
        float NtPrv2 = Float.parseFloat (Entrada.readLine ());

        float NtPrvs = (2*NtPrv1 + NtPrv2) / 3;

        System.out.println ();

        System.out.print ("Quantos trabalhos foram dados? ");
        int QtsTrbs = Integer.parseInt (Entrada.readLine ());

        System.out.println ();

        float NtTrbs = 0.0f,
              PsTrbs = 0.0f;

        for (int i = 1; i <= QtsTrbs; i++)
        {
            System.out.print ("Quanto voce tirou no " +
                              i + "o Trabalho? ");

            float NtTrb = Float.parseFloat (Entrada.readLine ());

            System.out.print ("Qual o peso do " +
                              i + "o Trabalho? ");

            float PsTrb = Float.parseFloat (Entrada.readLine ());

            NtTrbs += NtTrb * PsTrb;
            PsTrbs += PsTrb;
        }

        NtTrbs /= PsTrbs;

        System.out.println ();

        System.out.print ("Quantos PPs voce tem? ");
        int QtsPPs = Integer.parseInt (Entrada.readLine ());

        System.out.println ();

        float Media  = 0.7f*NtPrvs + 0.3f*NtTrbs + 0.1f*QtsPPs;
```

```
        System.out.println (Nome  + ", sua media foi " +
                            Media + "!");

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_05> javac -classpath . Media.java
C:\ExplsJava\Expl_05> java -classpath . -classpath . Media
```

Isto poderia produzir no console a seguinte interação:

```
Qual o seu nome? Jose

Quanto voce tirou na 1a Prova? 7
Quanto voce tirou na 2a Prova? 6

Quantos trabalhos foram dados? 3

Quanto voce tirou no 1o Trabalho? 7
Qual o peso do 1o Trabalho? 1
Quanto voce tirou no 2o Trabalho? 6
Qual o peso do 2o Trabalho? 2
Quanto voce tirou no 3o Trabalho? 5
Qual o peso do 3o Trabalho? 3

Quantos PPs voce tem? 5

Jose, sua media foi 6.866667!
```

Recursão

Sabemos que o conceito de recursão se encontra entre os mais poderosos recursos de programação de que se dispõe e a linguagem Java, sendo uma linguagem completa, não poderia deixar de suportá-lo.

Trata-se da possibilidade de escrever funções que ativam outras instâncias de execução de si próprias na implementação de suas funcionalidades

Como pressupomos um bom conhecimento de programação em alguma linguagem estruturada completa (como Pascal, por exemplo), entendemos que o estudo minucioso desta técnica de

programação foge ao escopo deste texto e, por isso, limitar-nos-emos a mencionar sua existência a assumir que o leitor saiba como empregá-la.

Membros de Classe e de Instância

Diferenciamos os membros de classe dos membros de instância através de um qualificador. Membros serão de classe se forem precedidos pelo qualificador static, e serão de instâncias caso contrário.

Existem 3 formas para acessar membros, a saber:

1. **Acesso Simples:**

   Chamamos de acesso simples aqueles acesso nos quais, para acessar um membro, simplesmente mencionamos o nome do membro. São simples os seguintes acessos:

   - Funções de uma classe acessando membros de classe de sua própria classe (neste caso o acesso também pode ser feito mencionando o nome de sua classe, seguido pelo caractere ponto, seguido pelo nome do membro que se deseja acessar);
   - Funções de uma instância de classe acessando membros de classe sua própria classe (neste caso o acesso também pode ser feito mencionando o nome de sua classe, seguido pelo caractere ponto, seguido pelo nome do membro que se deseja acessar);
   - Funções membro de uma instância de classe acessando membros de instância de classe de sua própria instância de classe (neste caso o acesso também pode ser feito mencionando a palavra chave this, seguida pelo caractere ponto, seguido pelo nome do membro que se deseja acessar).

2. **Acesso com Nome de Classe:**

   Todo acesso não simples a um membro de classe se faz mencionando o nome da classe eu questão, seguido do caractere ponto (.), seguido do nome do membro que se deseja acessar.

3. **Acesso com Nome de Instância de Classe:**

Todo acesso não simples a um membro de uma instância de classe se faz mencionando o nome do objeto questão, seguido do caractere ponto (.), seguido do nome do membro que se deseja acessar.

[C:\ExplsJava\Expl_06\Fracao.java]

```
class Fracao
{
    private static long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public static void AssumaValor (long N, long D)
    {
        if (D == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            Fracao.Num = -N;
            Fracao.Den = -D;
        }
        else
        {
            Fracao.Num = N;
            Fracao.Den = D;
        }

        Fracao.Erro = false;
    }

    public static String NaFormaDeString ()
    {
        Fracao.Erro = false;

        if (Fracao.Num == Fracao.Den)
            return "1";

        if (Fracao.Num + Fracao.Den == 0)
            return "-1";

        if (Fracao.Num == 0 || Fracao.Den == 1)
            return "" + Fracao.Num;

        return Fracao.Num + "/" + Fracao.Den;
    }

    public static void SomeSeCom (int I)
    {
        Fracao.Num = Fracao.Den * I + Fracao.Num;
```

```
        Fracao.Erro = false;
    }

    public static void SubtraiaDeSi (int I)
    {
        Fracao.Num = Fracao.Num - Fracao.Den * I;

        Fracao.Erro = false;
    }

    public static void MultipliqueSePor (int I)
    {
        Fracao.Num = Fracao.Num * I;

        Fracao.Erro = false;
    }

    public static void DividaSePor (int I)
    {
        if (I == 0)
        {
            Fracao.Erro = true;
            return;
        }

        Fracao.Den = Fracao.Den * I;

        if (Fracao.Den < 0)
        {
            Fracao.Num = -Fracao.Num;
            Fracao.Den = -Fracao.Den;
        }

        Fracao.Erro = false;
    }
}

class TesteDeFracao
{
    public static void main (String V [])
    {
        Fracao.AssumaValor (1,2);
        System.out.println  ();

        System.out.print   (Fracao.NaFormaDeString ());
        Fracao.SomeSeCom   (7);
        System.out.println (" + 7 = " + Fracao.NaFormaDeString ());

        System.out.print    (Fracao.NaFormaDeString ());
        Fracao.SubtraiaDeSi (7);
        System.out.println  (" - 7 = " + Fracao.NaFormaDeString ());

        System.out.print       (Fracao.NaFormaDeString ());
        Fracao.MultipliqueSePor (7);
        System.out.println      (" * 7 = " + Fracao.NaFormaDeString ());

        System.out.print   (Fracao.NaFormaDeString ());
        Fracao.DividaSePor (7);
        System.out.println (" / 7 = " + Fracao.NaFormaDeString ());
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_06> javac -classpath . Calculos.java
C:\ExplsJava\Expl_06> java -classpath . Calculos
```

Isto produziria no console a seguinte interação:

```
1/2 + 7 = 15/2
15/2 - 7 = 1/2
1/2 * 7 = 7/2
7/2 / 7 = 7/14
```

Criação de Instâncias

A declaração de uma variável de uma classe (objeto) não faz com que, automaticamente, seja criada uma instância da classe para ser armazenada no objeto declarado. Isso somente é conseguido com o operador new.

O operador new e um operador que opera sobre uma classe, criando uma instância da mesma. Sendo Classe o identificador de uma classe, temos que a forma geral de uma expressão de criação de uma instância é a seguinte:

```
new Classe ()
```

A expressão de criação de uma instância pode ser usada em qualquer lugar onde um objeto da classe Classe é esperado.

[C:\ExplsJava\Expl_07\Fracao.java]

```
class Fracao
{
    private int Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public void AssumaValor (int N, int D)
    {
        if (D == 0)
        {
            Fracao.Erro = true;
            return;
        }
```

```
        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
    }

    public String NaFormaDeString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";

        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;

        return this.Num + "/" + this.Den;
    }

    public void SomeSeCom (int I)
    {
        this.Num = this.Den * I + this.Num;

        Fracao.Erro = false;
    }

    public void SubtraiaDeSi (int I)
    {
        this.Num = this.Num - this.Den * I;

        Fracao.Erro = false;
    }

    public void MultipliqueSePor (int I)
    {
        this.Num = this.Num * I;

        Fracao.Erro = false;
    }

    public void DividaSePor (int I)
    {
        if (I == 0)
        {
            Fracao.Erro = true;
            return;
        }

        this.Den = this.Den * I;

        if (this.Den < 0)
        {
```

```
            this.Num = -this.Num;
            this.Den = -this.Den;
        }

        Fracao.Erro = false;
    }
}

class TesteDeFracao
{
    public static void main (String V [])
    {
        Fracao F1 = new Fracao (),
               F2 = new Fracao ();

        F1.AssumaValor (1,2);
        F2.AssumaValor (3,5);

        System.out.println  ();

        System.out.print   (F1.NaFormaDeString ());
        F1.SomeSeCom       (7);
        System.out.println (" + 7 = " + F1.NaFormaDeString ());

        System.out.print    (F2.NaFormaDeString ());
        F2.SubtraiaDeSi     (7);
        System.out.println  (" - 7 = " + F2.NaFormaDeString ());

        System.out.print    (F1.NaFormaDeString ());
        F1.MultipliqueSePor (7);
        System.out.println  (" * 7 = " + F1.NaFormaDeString ());

        System.out.print   (F2.NaFormaDeString ());
        F2.DividaSePor     (7);
        System.out.println (" / 7 = " + F2.NaFormaDeString ());
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_07> javac -classpath . Fracao.java
C:\ExplsJava\Expl_07> java -classpath . TesteDeFracao
```

Isto produziria no console a seguinte interação:

```
1/2 + 7 = 15/2
15/2 - 7 = 1/2
1/2 * 7 = 7/2
7/2 / 7 = 7/14
```

Organização de Programa

Modularização

Embora não seja obrigatório particionar os programas em módulos, recomenda-se fortemente que isso seja feito, especialmente programas de um porte maior. Isso fará com que o programa se torne mais manutenível, aumenta as possibilidade de reuso, além de minimizar a perda de tempo com compilações, já que módulos são unidades de compilação separada.

Em Java, módulos são arquivos com uma coleção de classes. Recomenda-se colocar em um mesmo módulo somente classes afins. É importante ressaltar que, se diversas classes forem escritas em um mesmo arquivo fonte, somente poderá ser utilizada por outras classes aquela que tiver o mesmo nome do arquivo fonte, se houver uma classe nessas condições, e retirada a extensão .java, naturalmente. As demais poderão apenas serem utilizadas por suas companheiras com quem compartilha o arquivo fonte.

[C:\ExplsJava\Expl_08\Fracao.java]

```
class Fracao
{
    private long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public void AssumaValor (long N, long D)
    {
        if (D == 0)
        {

            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
```

```
    }

    public String toString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";

        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;

        return this.Num + "/" + this.Den;
    }

    public Fracao Mais (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den +
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Menos (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den -
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Vezes (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Num;
        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao DivididoPor (Fracao F)
    {
        if (F.Num == 0)
        {
            Fracao.Erro = true;
            return null;
        }
```

```
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den;
        R.Den = this.Den * F.Num;

        if (R.Den < 0)
        {
            R.Num = -R.Num;
            R.Den = -R.Den;
        }

        Fracao.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_08\TesteDeFracao.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class TesteDeFracao
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        Fracao F2 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 1a fracao: ");
            long NF2 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 1a fracao: ");
            long DF2 = Long.parseLong (Entrada.readLine ());

            F2.AssumaValor (NF2, DF2);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        Fracao F3 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 2a fracao: ");
            long NF3 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 2a fracao: ");
```

```
            long DF3 = Long.parseLong (Entrada.readLine ());

            F3.AssumaValor (NF3, DF3);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        System.out.println ();

        Fracao F1;

        F1 = F2.Mais (F3);
        System.out.println (F1 + " = " + F2 + " + " + F3);

        F1 = F2.Menos (F3);
        System.out.println (F1 + " = " + F2 + " - " + F3);

        F1 = F2.Vezes (F3);
        System.out.println (F1 + " = " + F2 + " * " + F3);

        F1 = F2.DivididoPor (F3);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + F3);

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_08> javac -classpath . TesteDeFracao.java
C:\ExplsJava\Expl_08> java -classpath . TesteDeFracao
```

Isto poderia produzir no console a seguinte interação:

```
Entre  com  o  numerador da 1a fracao: 2
Entre com o denominador da 1a fracao: 5

Entre  com  o  numerador da 2a fracao: 1
Entre com o denominador da 2a fracao: -2

-1/10 = 2/5  +  -1/2
9/10 = 2/5  -   -1/2
-2/10 = 2/5  *   -1/2
-4/5 = 2/5  /    -1/2
```

Pacotes

Como vimos, um programa em Java pode se compor de diversos arquivos. Podem também usar arquivos de dados, imagens, sons, etc. Projetos maiores podem de fato ficar muito desorganizados se todos estes arquivos residirem juntos em um mesmo diretório.

Para resolver este problema, introduzimos o conceito de pacote. Pacotes definem uma hierarquia de espaços onde as classes podem ser armazenadas. Todo pacote pode Ter zero ou mais subpacotes, todo subpacote pode ter zero ou mais subsubpacotes, e assim por diante. O compilador Java usa diretórios do sistema de arquivos para armazenar pacotes.

É importante ressaltar o fato de que deve existir uma corrersponděncia biunívoca entre a hierarquia de pacotes e a hierarquia de diretórios que fica abaixo do diretório raiz do diretório onde está armazenado o programa.

É preciso também chamar a atenção para o fato de ques as letras minúsculas e maiúsculas que constituem os nomes dos diretórios devem corresponder exatamente às letras que constituem o nome dos patotes.

Arquivos fonte de classes que residem em um pacote devem começar pelo comando:

package NomPac1[.NomPac2[...]];

onde NomPac1 é o nome do pacote onde reside a classe em questão, NomPac2 é o nome do subpacotete do pacote NomPac1 onde ela reside, e assim sucessivamente.

Classes Públicas

Classes que residem dentro de um mesmo nó da hierarquia de pacotes podem empregar-se mutuamente em suas implementações, ao passo que classes que residem em um determinado nó da hierarquia de pacotes somente podem fazer uso de classes que residem em um outro nó da hierarquia de pacotes se estas últimas forem qualificas com o qualificador public.

Membros Default (ou de Pacote)

Conforme sabemos, existem 2 tipos de membros que podem ser definidos em uma classe, a saber: os membros públicos (que são acessíveis para qualquer método que tenha acesso à classe em que foram definidos) e os membros privativos (que são acessíveis somente nos métodos definidos em sua classe).

Ficaremos sabendo agora da existência de um terceiro tipo de membro, os membros default (ou de pacote). Esse tipo de membro é declarado sem nenhuma qualificação, i.e., sem serem precedidos pela palavra public, pela palavra private, ou por qualquer outra palavra especialmente definida para qualifica-los.

Membros default (ou de pacote) são acessíveis para todas os métodos definidos nas classes que integram o pacote onde foi definida sua classe.

Convem ressaltar que, quando não dividimos classes em pacotes, Java coloca todas as classes que definimos em um único pacote.

[C:\ExplsJava\Expl_09\BibMat\Fracao.java]

```
package BibMat;

public class Fracao
{
    private long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public void AssumaValor (long N, long D)
    {
        if (D == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
    }

    public String toString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";
```

```
        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;

        return this.Num + "/" + this.Den;
    }

    public Fracao Mais (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den +
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Menos (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den -
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Vezes (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Num;
        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao DivididoPor (Fracao F)
    {
        if (F.Num == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den;
        R.Den = this.Den * F.Num;

        if (R.Den < 0)
        {
            R.Num = -R.Num;
            R.Den = -R.Den;
```

```
        }

        Fracao.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_09\TesteDeFracao.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import BibMat.Fracao;

class TesteDeFracao
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        Fracao F2 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 1a fracao: ");
            long NF2 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 1a fracao: ");
            long DF2 = Long.parseLong (Entrada.readLine ());

            F2.AssumaValor (NF2, DF2);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        Fracao F3 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 2a fracao: ");
            long NF3 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 2a fracao: ");
            long DF3 = Long.parseLong (Entrada.readLine ());

            F3.AssumaValor (NF3, DF3);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
```

```
        }

        System.out.println ();

        Fracao F1;

        F1 = F2.Mais (F3);
        System.out.println (F1 + " = " + F2 + " + " + F3);

        F1 = F2.Menos (F3);
        System.out.println (F1 + " = " + F2 + " - " + F3);

        F1 = F2.Vezes (F3);
        System.out.println (F1 + " = " + F2 + " * " + F3);

        F1 = F2.DivididoPor (F3);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + F3);

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_09> javac -classpath . -classpath . TesteDeFracao.java
C:\ExplsJava\Expl_09> java -classpath .  -classpath . TesteDeFracao
```

Isto poderia produzir no console a seguinte interação:

```
Entre  com  o  numerador da 1a fracao: 2
Entre com o denominador da 1a fracao: 5

Entre  com  o  numerador da 2a fracao: 1
Entre com o denominador da 2a fracao: -2

-1/10 = 2/5  +  -1/2
9/10 = 2/5  -   -1/2
-2/10 = 2/5  *   -1/2
-4/5 = 2/5  /    -1/2
```

JARs

JARs, ou Java Archives, são repositórios com formato independente de plataforma, onde jazem divesos arquivos, por vezes diversos diretórios contendo diversos arquivos.

Além de serem úteis para a distribuição de aplicações Java, os JARs podem ser também empregados para construir pacotes de software contendo muitas applets, bem como todos os eventuais arquivos necessários a elas.

Tais pacotes podem ser baixados por um browser em uma única transação HTTP, melhorando significativamente a velocidade de carga de uma página. Além disso, o formato JAR também suporta compressão de dados, o que reduz também o tamanho do arquivo, o que faz cair ainda mais o tempo gasto na transação.

Além disso, o autor das applets pode assinar digitalmente cada entrada em um JAR para de forma a autenticar sua origem.

[C:\ExplsJava\Expl_10\Fracao.java]

```
class Fracao
{
    private long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public void AssumaValor (long N, long D)
    {
        if (D == 0)
        {

            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
    }

    public String toString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";

        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;
```

```
        return this.Num + "/" + this.Den;
    }

    public Fracao Mais (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den +
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Menos (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den -
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Vezes (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Num;
        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao DivididoPor (Fracao F)
    {
        if (F.Num == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den;
        R.Den = this.Den * F.Num;

        if (R.Den < 0)
        {
            R.Num = -R.Num;
            R.Den = -R.Den;
        }

        Fracao.Erro = false;
        return R;
    }
```

```
}
```

[C:\ExplsJava\Expl_10\TesteDeFracao.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class TesteDeFracao
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        Fracao F2 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 1a fracao: ");
            long NF2 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 1a fracao: ");
            long DF2 = Long.parseLong (Entrada.readLine ());

            F2.AssumaValor (NF2, DF2);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        Fracao F3 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 2a fracao: ");
            long NF3 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 2a fracao: ");
            long DF3 = Long.parseLong (Entrada.readLine ());

            F3.AssumaValor (NF3, DF3);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        System.out.println ();

        Fracao F1;
```

```
        F1 = F2.Mais (F3);
        System.out.println (F1 + " = " + F2 + " + " + F3);

        F1 = F2.Menos (F3);
        System.out.println (F1 + " = " + F2 + " - " + F3);

        F1 = F2.Vezes (F3);
        System.out.println (F1 + " = " + F2 + " * " + F3);

        F1 = F2.DivididoPor (F3);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + F3);

        System.out.println ();
    }
}
```

[C:\ExplsJava\Expl_10\FcaoPrin.MF]

```
Main-Class: TesteDeFracao
```

Podemos observar que o exemplo acima apresenta um arquivo de nome FcaoPrin.MF. Este arquivo será empregado no momento de criar o JAR e tem por função indicar qual é a classe principal do JAR, i.e., a classe a partir da qual a execução terá início.

Para compilar este programa, procederemos como temos procedido com todos os programas compilados anterirormente, ou seja, daremos o seguinte comando:

C:\ExplsJava\Expl_10> javac -classpath . TesteDeFracao.java

Para criar o JAR, basta dar o seguinte comando:

C:\ExplsJava\Expl_10> jar fmc Fracao.jar FcaoPrin.MF Fracao.class TesteDeFracao.class

Poderemos observar que foi criado o arquivo Fracao.jar. Para executá-lo, basta dar o seguinte comando:

C:\ExplsJava\Expl_10> java -classpath . -jar Fracao.jar

Isto poderia produzir no console a seguinte interação:

Entre com o numerador da 1a fracao: 2
Entre com o denominador da 1a fracao: 5

Entre com o numerador da 2a fracao: 1

Entre com o denominador da 2a fracao: -2

-1/10 = 2/5 + -1/2
9/10 = 2/5 - -1/2
-2/10 = 2/5 * -1/2
-4/5 = 2/5 / -1/2

Sobrecarga

Java permite sobrecarga de nomes de função, i.e., um único nome para diversas funções diferentes.

Cada operação deve ter tipos de parâmetros diferentes. Java seleciona uma operação com base nos tipos dos argumentos fornecidos.

O conceito de sobrecarga, além de possibilitar nomes de função mais significativos, ainda possibilita a escrita de código polimórfico.

[C:\ExplsJava\Expl_11\Fracao.java]

```
class Fracao
{
    private long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public void AssumaValor (long N, long D)
    {
        if (D == 0)
        {

            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
    }
```

```
public void AssumaValor (String S)
{
    if (S.length () == 0)
    {
        Fracao.Erro = true;
        return;
    }

    if (S.charAt (0) == '/' || S.charAt (S.length () - 1) == '/')
    {
        Fracao.Erro = true;
        return;
    }

    if (S.length () == 1 &&
        (S.charAt (0) == '+' || S.charAt (0) == '-'))
    {
        Fracao.Erro = true;
        return;
    }

    if (S.length () >= 2)
        if ((S.charAt (0) == '+' || S.charAt (0) == '-') &&
            (S.charAt (1) <  '0' || S.charAt (1) >  '9'))
    {
        Fracao.Erro = true;
        return;
    }

    int    I     = 0;
    String Slong = "";

    if (S.charAt (I) == '+')
        I++;
    else
        if (S.charAt (I) == '-')
            Slong = Slong + S.charAt (I++);

    while (I < S.length ())
    {
        if (S.charAt (I) == '/')
            break;

        if (S.charAt (I) < '0' || S.charAt (I) > '9')
        {
            Fracao.Erro = true;
            return;
        }

        Slong = Slong + S.charAt (I++);
    }

    this.Num = Long.valueOf (Slong).longValue ();

    if (I++ == S.length ())
    {
        this.Den = 1;

        Erro = false;
        return;
    }
```

```
        Slong = "";

        if (S.charAt (I) == '+')
            I++;
        else
            if (S.charAt (I) == '-')
                Slong = Slong + S.charAt (I++);

        while (I < S.length ())
        {
            if (S.charAt (I) < '0' || S.charAt (I) > '9')
            {
                Fracao.Erro = true;
                return;
            }

            Slong = Slong + S.charAt (I++);
        }

        this.Den = Long.valueOf (Slong).longValue ();

        if (this.Den == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (this.Den < 0)
        {
            this.Num = -this.Num;
            this.Den = -this.Den;
        }

        Fracao.Erro = false;
    }

    public String toString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";

        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;

        return this.Num + "/" + this.Den;
    }

    public Fracao Mais (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den +
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
```

```
    }


    public Fracao Mais (long L)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num +
                this.Den * L;

        R.Den = this.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Menos (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den -
                this.Den * F.Num;

        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Menos (long L)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num -
                this.Den * L;

        R.Den = this.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Vezes (Fracao F)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * F.Num;
        R.Den = this.Den * F.Den;

        Fracao.Erro = false;
        return R;
    }

    public Fracao Vezes (long L)
    {
        Fracao R = new Fracao ();

        R.Num = this.Num * L;
        R.Den = this.Den;

        Fracao.Erro = false;
        return R;
    }
```

```
    public Fracao DivididoPor (Fracao F)
    {
        if (F.Num == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        Fracao R = new Fracao ();

        R.Num = this.Num * F.Den;
        R.Den = this.Den * F.Num;

        if (R.Den < 0)
        {
            R.Num = -R.Num;
            R.Den = -R.Den;
        }

        Fracao.Erro = false;
        return R;
    }

    public Fracao DivididoPor (long L)
    {
        if (L == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        Fracao R = new Fracao ();

        R.Num = this.Num;
        R.Den = this.Den * L;

        if (R.Den < 0)
        {
            R.Num = -R.Num;
            R.Den = -R.Den;
        }

        Fracao.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_11\TesteDeFracao.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class TesteDeFracao
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
```

```
                                (System.in));

        Fracao F2 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 1a fracao: ");
            long NF2 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 1a fracao: ");
            long DF2 = Long.parseLong (Entrada.readLine ());

            F2.AssumaValor (NF2, DF2);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        Fracao F3 = new Fracao ();

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre com a 2a fracao: ");
            F3.AssumaValor  (Entrada.readLine ());

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        System.out.print ("Entre com um  inteiro: ");
        long L = Long.parseLong (Entrada.readLine ());

        System.out.println ();

        Fracao F1;

        F1 = F2.Mais (F3);
        System.out.println (F1 + " = " + F2 + " + " + F3);

        F1= F2.Mais (L);
        System.out.println (F1 + " = " + F2 + " + " + L);

        System.out.println ();

        F1 = F2.Menos (F3);
        System.out.println (F1 + " = " + F2 + " - " + F3);

        F1= F2.Menos (L);
        System.out.println (F1 + " = " + F2 + " - " + L);

        System.out.println ();

        F1 = F2.Vezes (F3);
        System.out.println (F1 + " = " + F2 + " * " + F3);
```

```
        F1= F2.Vezes (L);
        System.out.println (F1 + " = " + F2 + " * " + L);

        System.out.println ();

        F1 = F2.DivididoPor (F3);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + F3);

        F1 = F2.DivididoPor (L);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + L);

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_11> javac -classpath . TesteDeFracao.java
C:\ExplsJava\Expl_11> java -classpath . TesteDeFracao
```

Isto poderia produzir no console a seguinte interação:

```
Entre  com o  numerador da 1a fracao: 2
Entre com o denominador da 1a fracao: 5

Entre com a 2a fracao: 1/-2
Entre com um  inteiro: 7

-1/10 = 2/5 + -1/2
37/5 = 2/5 + 7

9/10 = 2/5 - -1/2
-33/5 = 2/5 - 7

-2/10 = 2/5 * -1/2
14/5 = 2/5 * 7

-4/5 = 2/5 / -1/2
2/35 = 2/5 / 7
```

Criação e Destruição

O autor de uma classe pode controlar o que deve acontecer por ocasião da criação e da destruição de instâncias da classe escrevendo funções construtoras e destrutoras.

Construtores servem basicamente para iniciar os membros da instância que está sendo criada. Quanto aos destrutores, como Java tem um mecanismo automático de coleta de lixo, não é preciso, como em outras linguagens, se preocupar com liberação de memória. Por esta razão, é muito raro ser preciso utilizar um destrutor em Java. Mas se a destruição de um objeto tiver implicações outras que a liberação de memória, neste caso empregar um destrutor pode ser interessante.

Quando uma instância é criada, aloca-se memória para a instância e em seguida é chamado o construtor da classe (se houver um). Quando uma instância é destruída, o destrutor é chamado (se houver um) e em seguida libera-se a memória alocada para a instância.

Construtores são batizados com o nome da classe a que se referem e destrutores são batizados com o nome de finalize .

Como os construtores e os destrutores são chamados automaticamente, não faz sentido que eles tenham retorno. Não indicamos o tipo retornado por um construtor (nem mesmo void). O tipo de retorno de um destrutores é sempre void.

Destrutores não podem ter parâmetros, o que significa que toda classe terá no máximo um destrutor. Não é possível precisar o momento da execução de um destrutor, já que não se sabe em que momento o mecanismo de coleta de lixo vai liberar a memória associada aos objetos e vetores inutilizados.

Construtores podem ter parâmetros, o que significa que podemos ter vários deles em uma mesma classe, cada qual realizando a iniciação do objeto em processo de criação de uma maneira diferente.

No caso de uma classe ter vários construtores, se na implementação de um deles for desejável chamar um outro, poderemos fazê-lo através da palavra chave this. Em vez ser utilizado o nome da classe para chamar o construtor, seria utilizada a palavra this eventualmente seguida dos argumentos a serem passados ao construtor separados por virgulas (,) e entre parênteses (()).

[C:\ExplsJava\Expl_12\Fracao.java]

```
class Fracao
```

```
{
    private long Num, Den;
    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Fracao.Erro;
    }

    public Fracao (long N, long D)
    {
        if (D == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (D < 0)
        {
            this.Num = -N;
            this.Den = -D;
        }
        else
        {
            this.Num = N;
            this.Den = D;
        }

        Fracao.Erro = false;
    }

    public Fracao (String S)
    {
        if (S.length () == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (S.charAt (0) == '/' || S.charAt (S.length () - 1) == '/')
        {
            Fracao.Erro = true;
            return;
        }

        if (S.length () == 1 &&
            (S.charAt (0) == '+' || S.charAt (0) == '-'))
        {
            Fracao.Erro = true;
            return;
        }

        if (S.length () >= 2)
            if ((S.charAt (0) == '+' || S.charAt (0) == '-') &&
                (S.charAt (1) <  '0' || S.charAt (1) >  '9'))
        {
            Fracao.Erro = true;
            return;
        }

        int    I    = 0;
        String Slong = "";
```

```
        if (S.charAt (I) == '+')
            I++;
        else
            if (S.charAt (I) == '-')
                Slong = Slong + S.charAt (I++);

        while (I < S.length ())
        {
            if (S.charAt (I) == '/')
                break;

            if (S.charAt (I) < '0' || S.charAt (I) > '9')
            {
                Fracao.Erro = true;
                return;
            }

            Slong = Slong + S.charAt (I++);
        }

        this.Num = Long.valueOf (Slong).longValue ();

        if (I++ == S.length ())
        {
            this.Den = 1;

            Erro = false;
            return;
        }

        Slong = "";

        if (S.charAt (I) == '+')
            I++;
        else
            if (S.charAt (I) == '-')
                Slong = Slong + S.charAt (I++);

        while (I < S.length ())
        {
            if (S.charAt (I) < '0' || S.charAt (I) > '9')
            {
                Fracao.Erro = true;
                return;
            }

            Slong = Slong + S.charAt (I++);
        }

        this.Den = Long.valueOf (Slong).longValue ();

        if (this.Den == 0)
        {
            Fracao.Erro = true;
            return;
        }

        if (this.Den < 0)
        {
            this.Num = -this.Num;
            this.Den = -this.Den;
        }
```

```
        Fracao.Erro = false;
    }

    public String toString ()
    {
        Fracao.Erro = false;

        if (this.Num == this.Den)
            return "1";

        if (this.Num + this.Den == 0)
            return "-1";

        if (this.Num == 0 || this.Den == 1)
            return "" + this.Num;

        return this.Num + "/" + this.Den;
    }

    public Fracao Mais (Fracao F)
    {
        long N = this.Num * F.Den +
                 this.Den * F.Num;

        long D = this.Den * F.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao Mais (long L)
    {
        long N = this.Num +
                 this.Den * L;

        long D = this.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao Menos (Fracao F)
    {
        long N = this.Num * F.Den -
                 this.Den * F.Num;

        long D = this.Den * F.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao Menos (long L)
    {
        long N = this.Num -
                 this.Den * L;

        long D = this.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }
```

```
    public Fracao Vezes (Fracao F)
    {
        long N = this.Num * F.Num;
        long D = this.Den * F.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao Vezes (long L)
    {
        long N = this.Num * L;
        long D = this.Den;

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao DivididoPor (Fracao F)
    {
        if (F.Num == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        long N = this.Num * F.Den;
        long D = this.Den * F.Num;

        if (D < 0)
        {
            N = -N;
            D = -D;
        }

        Fracao.Erro = false;
        return new Fracao (N,D);
    }

    public Fracao DivididoPor (long L)
    {
        if (L == 0)
        {
            Fracao.Erro = true;
            return null;
        }

        long N = this.Num;
        long D = this.Den * L;

        if (D < 0)
        {
            N = -N;
            D = -D;
        }

        Fracao.Erro = false;
        return new Fracao (N,D);
    }
}
```

[C:\ExplsJava\Expl_12\TesteDeFracao.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class TesteDeFracao
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        Fracao F2;

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre  com o  numerador da 1a fracao: ");
            long NF2 = Long.parseLong (Entrada.readLine ());

            System.out.print ("Entre com o denominador da 1a fracao: ");
            long DF2 = Long.parseLong (Entrada.readLine ());

            F2 = new Fracao (NF2, DF2);

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        Fracao F3;

        for (;;)
        {
            System.out.println ();

            System.out.print ("Entre com a 2a fracao: ");
            F3 = new Fracao (Entrada.readLine ());

            if (Fracao.DeuErro ())
                System.err.println ("Denominador nao pode ser zero!");
            else
                break;
        }

        System.out.print ("Entre com um  inteiro: ");
        long L = Long.parseLong (Entrada.readLine ());

        System.out.println ();

        Fracao F1;

        F1 = F2.Mais (F3);
        System.out.println (F1 + " = " + F2 + " + " + F3);

        F1= F2.Mais (L);
        System.out.println (F1 + " = " + F2 + " + " + L);
```

```
        System.out.println ();

        F1 = F2.Menos (F3);
        System.out.println (F1 + " = " + F2 + " - " + F3);

        F1= F2.Menos (L);
        System.out.println (F1 + " = " + F2 + " - " + L);

        System.out.println ();

        F1 = F2.Vezes (F3);
        System.out.println (F1 + " = " + F2 + " * " + F3);

        F1= F2.Vezes (L);
        System.out.println (F1 + " = " + F2 + " * " + L);

        System.out.println ();

        F1 = F2.DivididoPor (F3);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + F3);

        F1 = F2.DivididoPor (L);
        if (Fracao.DeuErro ())
            System.err.println ("Nao se pode dividir por zero!");
        else
            System.out.println (F1 + " = " + F2 + " / " + L);

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_12> javac -classpath . TesteDeFracao.java
C:\ExplsJava\Expl_12> java -classpath . TesteDeFracao
```

Isto poderia produzir no console a seguinte interação:

```
Entre  com o  numerador da 1a fracao: 2
Entre com o denominador da 1a fracao: 5

Entre com a 2a fracao: 1/-2
Entre com um  inteiro: 7

-1/10 = 2/5 + -1/2
37/5 = 2/5 + 7

9/10 = 2/5 - -1/2
-33/5 = 2/5 - 7

-2/10 = 2/5 * -1/2
```

14/5 = 2/5 * 7

-4/5 = 2/5 / -1/2
2/35 = 2/5 / 7

Vetores

Um vetor é uma coleção de informações de mesma natureza. Vetores podem ter um número arbitrariamente grande de subscritos. Subscritos identificam de forma única um elemento dentro da coleção. Os subscritos em Java são sempre números naturais, sendo zero o primeiro deles.

A declaração de um vetor tem a seguinte forma: primeiramente vem o tipo dos elementos do vetor, em seguida vem o identificador do vetor, em seguida vem uma série pares de colchetes ([]), um para cada dimensão do vetor. Sendo Tipo um tipo e Vetor o identificador do vetor a ser declarado, temos abaixo a declaração de um vetor de elementos do tipo Tipo:

Tipo Vetor [ ][ ][ ]...[ ] ou Tipo [ ][ ][ ]...[ ] Vetor

Como acontece com os objetos, os vetores antes de serem usados precisam ser instanciados. Faz-se isso com o operador new. Sendo Vetor o identificador de um vetor do tipo Tipo previamente declarado com k dimensões e Ni números naturais, temos:

$$Vetor = new\ Tipo\ [N_1][N_2]...[N_k];$$

Para acessar um elemento de um vetor, basta mencionar o seu nome e, entre colchetes o subscrito desejado.

Constantes Vetor

Uma constante vetor consiste dos valores dos seus elementos separados por vírgulas (,) e delimitados por chaves ({}). Podem somente ser usadas para iniciá-lo (não se pode atribuir uma constante vetor a um vetor).

Tamanho das Dimensões de um Vetor

O tamanho de cada uma das dimensões de um vetor pode ser obtido através da propriedade length. Assim, sendo Vetor o identificador de um vetor e i, j e k seus subscritos, temos que:

1. Vetor.length → Dá o tamanho da primeira dimensão do vetor;
2. Vetor[i].length → Dá o tamanho da segunda dimensão do vetor;
3. Vetor[i][j].length → Dá o tamanho da terceira dimensão do vetor;
4. Vetor[i][j][k].length → Dá o tamanho da quarta dimensão do vetor;
5. Etc.

[C:\ExplsJava\Expl_13\Agenda.java]

```
class Agenda
{
    private int     QtosContatos = 0;
    private boolean Erro         = false;

    private String  Nome     [],
                    Telefone [];

    private int OndeEsta (String N)
    {
        int Inicio = 0, Final = this.QtosContatos - 1, Onde;

        while (Inicio <= Final)
        {
            Onde = (Inicio + Final) / 2;

            int Comparacao = N.compareTo (this.Nome [Onde]);

            if (Comparacao == 0)
                return Onde;
            else
                if (Comparacao < 0)
                    Final = Onde - 1;
                else
                    Inicio = Onde + 1;
        }

        return -1;
    }

    private int OndeGuardar (String N)
    {
        int Inicio     = 0,
            Final      = this.QtosContatos - 1,
            Onde       = 0,
            Comparacao = -1;

        while (Inicio <= Final)
        {
            Onde       = (Inicio + Final) / 2;
            Comparacao = N.compareTo (this.Nome [Onde]);

            if (Comparacao == 0)
                return -1;
            else
```

```
                if (Comparacao < 0)
                    Final = Onde - 1;
                else
                    Inicio = Onde + 1;
            }

            return Comparacao < 0? Onde: Onde + 1;
        }

        public Agenda (int C)
        {
            if (C <= 0)
            {
                this.Erro = true;
                return;
            }

            this.Nome      = new String [C];
            this.Telefone  = new String [C];

            this.Erro      = false;
        }

        public boolean DeuErro ()
        {
            return this.Erro;
        }

        public int Capacidade ()
        {
            this.Erro = false;
            return this.Nome.length;
        }

        public int QuantosContatos ()
        {
            this.Erro = false;
            return this.QtosContatos;
        }

        public boolean HaRegistroDoContato (String N)
        {
            this.Erro = false;
            return this.OndeEsta (N) != -1;
        }

        public void RegistreOContato (String N, String T)
        {
            if (this.QtosContatos == this.Nome.length)
            {
                this.Erro = true;
                return;
            }

            int Posicao = this.OndeGuardar (N);

            if (Posicao == -1)
            {
                this.Erro = true;
                return;
            }

            for (int I = this.QtosContatos - 1; I >= Posicao; I--)
```

```
        {
            this.Nome     [I+1] = this.Nome     [I];
            this.Telefone [I+1] = this.Telefone [I];
        }

        this.Nome     [Posicao] = N;
        this.Telefone [Posicao] = T;

        this.QtosContatos++;
        this.Erro = false;
    }

    public String DigaMeONome (int P)
    {
        if (P < 0 || P >= this.QtosContatos)
        {
            this.Erro = true;
            return null;
        }

        this.Erro = false;
        return this.Nome [P];
    }

    public String DigaMeOTelefone (int P)
    {
        if (P < 0 || P >= this.QtosContatos)
        {
            this.Erro = true;
            return null;
        }

        this.Erro = false;
        return this.Telefone [P];
    }

    public String DigaMeOTelefoneDe (String N)
    {
        int Posicao = this.OndeEsta (N);

        if (Posicao == -1)
        {
            this.Erro = true;
            return null;
        }

        this.Erro = false;
        return this.Telefone [Posicao];
    }

    public void DescarteOContato (String N)
    {
        if (this.QtosContatos == 0)
        {
            this.Erro = true;
            return;
        }

        int Posicao = this.OndeEsta (N);

        if (Posicao == -1)
        {
            this.Erro = true;
```

```
            return;
        }

        int I;

        for (I = Posicao; I < this.QtosContatos - 1; I++)
        {
            this.Nome     [I] = this.Nome     [I+1];
            this.Telefone [I] = this.Telefone [I+1];
        }

        this.Nome     [I] = null;
        this.Telefone [I] = null;

        this.QtosContatos--;
        this.Erro = false;
    }
}
```

[C:\ExplsJava\Expl_13\TesteDeAgenda.java]

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class TesteDeAgenda
{
    public static void main (String Args []) throws IOException
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        int Capacidade;

        for (;;)
        {
            System.out.println ();

            System.out.print ("Capacidade desejada para a Agenda: ");
            Capacidade = Integer.parseInt (Entrada.readLine ());

            if (Capacidade <= 0)
            {
                System.err.println ("Capacidade tem de ser natural e > 0!");
                continue;
            }

            break;
        }

        Agenda agenda  = new Agenda (Capacidade);
        String Nome, Telefone;
        char   Opcao;

        do
        {
            System.out.println ();

            System.out.print ("Digite sua Opcao (" +
```

```
                        "I=Incluir/" +
                        "C=Consultar/" +
                        "L=Listar/" +
                        "E=Excluir/" +
                        "S=Sair)" +
                        ": ");

            Opcao = (Entrada.readLine ()).charAt (0);

            switch (Opcao)
            {
                case 'i':
                case 'I':
                    if (agenda.QuantosContatos () == agenda.Capacidade ())
                    {
                        System.err.println ("Capacidade da agenda esgotada!");
                        System.out.println ();
                        break;
                    }

                    System.out.print ("Nome....: ");
                    Nome = Entrada.readLine ();

                    if (agenda.HaRegistroDoContato (Nome))
                        System.err.println ("Esse nome ja consta na agenda!");
                    else
                    {
                        System.out.print ("Telefone: ");
                        Telefone = Entrada.readLine ();

                        agenda.RegistreOContato (Nome, Telefone);
                    }

                    System.out.println ();
                    break;


                case 'c':
                case 'C':
                    if (agenda.QuantosContatos () == 0)
                        System.err.println ("A agenda esta vazia!");
                    else
                    {
                        System.out.print ("Nome....: ");
                        Nome = Entrada.readLine ();

                        Telefone = agenda.DigaMeOTelefoneDe (Nome);

                        if (agenda.DeuErro ())
                            System.err.println ("Nao consta na agenda!");
                        else
                            System.out.println ("Telefone: " +
                                                Telefone);
                    }

                    System.out.println ();
                    break;


                case 'l':
                case 'L':
                    if (agenda.QuantosContatos () == 0)
                        System.err.println ("A agenda esta vazia!");
```

```
                    else
                    {
                        String Tecla;

                        for (int I = 0;
                             I < agenda.QuantosContatos ();
                             I++)
                        {
                            System.out.println ("Nome: " +
                                                agenda.DigaMeONome (I) +
                                                " - Telefone: " +
                                                agenda.DigaMeOTelefone (I));

                            if (I%23 == 22 && I < agenda.QuantosContatos()-1)
                            {
                                System.out.println ();
                                System.out.print ("Tecle ENTER... ");
                                Tecla = Entrada.readLine ();
                                System.out.println ();
                            }
                        }
                    }

                    System.out.println ();
                    break;


                case 'e':
                case 'E':
                    if (agenda.QuantosContatos () == 0)
                        System.err.println ("A agenda esta vazia!");
                    else
                    {
                          System.out.print ("Nome....: ");
                          Nome = Entrada.readLine ();

                          agenda.DescarteOContato (Nome);

                          if (agenda.DeuErro ())
                              System.err.println ("Nao consta na agenda!");
                    }

                    System.out.println ();
                    break;


                case 's':
                case 'S':
                    break;


                default :
                    System.err.println ("Opcao invalida!");
                    System.err.println ();
            }
        }
        while ((Opcao != 's') && (Opcao != 'S'));
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_13> javac -classpath . TesteDeAgenda.java
C:\ExplsJava\Expl_13> java -classpath . TesteDeAgenda
```

Isto poderia produzir no console a seguinte interação:

```
Capacidade desejada para a Agenda: 100

Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): i
Nome....: Jose
Telefone: 211.4466


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): i
Nome....: Jose
Esse nome ja consta na agenda!


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): e
Nome....: Joao
Nao consta na agenda!


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): c
Nome....: Jose
Telefone: 211.4466


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): i
Nome....: Joao
Telefone: 202.9955


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): l
Nome: Joao - Telefone: 202.9955
Nome: Jose - Telefone: 211.4466


Digite sua Opcao (I=Incluir/C=Consultar/L=Listar/E=Excluir/S=Sair): s
```

A Classe java.util.Vector

Esta classe implementa um repositório de objetos. Tal repositório cresce dinamicamente a medida que mais objetos vão sendo armazenados nele.

Para otimizar o processo de crescimento, o repositório cresce sempre de várias unidades de armazenamento, e não apenas de uma. Ajustar a capacidade do repositório antes de inserir nele uma grande quantidade de objetos é uma boa ideia para minimizar o tempo gasto com realocações incrementais.

Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**
  - **Vector ():**
    Constroi uma nova instância vazia da classe Vector (capacidade e tamanho das alocações incrementais têm valores padrão);
  - **Vector (int C):**
    Constroi uma nova instância vazia da classe Vector, com capacidade de armazenamento C (tamanho das alocações incrementais tem valor padrão);
  - **Vector (int C, int T):**
    Constroi uma nova instância vazia da classe Vector, com capacidade de armazenamento C e tamanho das alocações incrementais T;
- **Métodos:**
  - **void add (Object O):**
    Acrescenta o Object O no final deste Vector;
  - **void add (int P, Object O):**
    Acrescenta o Object O na posição P deste Vector;
  - **void addElement (Object O):**
    Acrescenta o objeto O no final deste Vector;
  - **int Capacity ():**
    Retorna a capacidade deste Vector (quantos objetos podem ser armazenados nele e não quantos estão armazenados nele);

- **void clear ():**

  Remove todos os elementos deste Vector;

- **boolean contains (Object O):**

  Verifica se este Vector contem o Object O;

- **void copyInto (Object V []):**

  Copia os elementos deste Vector no vetor V;

- **Object elementAt (int P):**

  Retorna o Object que se encontra na posição P deste Vector;

- **void ensureCapacity (int C):**

  Faz crescer a capacidade deste Vector de forma a garantir que ele tenha ao menos a capacidade de armazenamento C;

- **Object firstElement ():**

  Retorna o primeiro elemento do Vector;

- **Object get (int P):**

  Retorna o Object que se encontra na posição P deste Vector;

- **int indexOf (Object O):**

  Retorna o número da primeira posição do Vector onde pode ser encontrado o objeto O;

- **int indexOf (Object O, int P):**

  Retorna o número da primeira posição do Vector (a partir da posição de número P) onde pode ser encontrado o objeto O;

- **void insertElementAt (Object O, int P):**

  Insere o Object O na posição P deste Vector;

- **boolean isEmpty ():**

  Verifica se este Vector está vazio;

- **Object lastElement ():**

  Retorna o último elemento do Vector;

- **int lastIndexOf (Object O):**

  Retorna o número da última posição do Vector onde pode ser encontrado o objeto O;

- **int indexOf (Object O, int P):**

  Retorna o número da última posição do Vector (anterior à posição de número P) onde pode ser encontrado o objeto O;

- **void remove (int P):**

  Remove deste Vector o elemento que se encontra na posição P;

- **void remove (Object O):**

  Remove deste Vector a primeira ocorrência do Object O;

- **void removeAllElements ():**

  Remove todos os elementos deste Vector;

- **void removeElement (Object O):**

  Remove deste Vector a primeira ocorrência do Object O;

- **void removeElementAt (int P):**

  Remove deste Vector o elemento que se encontra na posição P;

- **void set (int P, Object O):**

  Armazena o Object O na posição P deste Vector;

- **void setElementAt (Object O, int P):**

  Armazena o Object O na posição P deste Vector;

- **void setSize ():**

  Ajusta o tamanho deste Vector;

- **int Size ():**

  Retorna a quantidade de elementos deste Vector;

- **Object[] toArray ():**

  Retorna um vetor que contem os elementos deste Vector na ordem apropriada;

- **void toArray (Object V []):**
  Posiciona no vetor V os elementos deste Vector na ordem apropriada;

- **void trimToSize ():**
  Reduz a capacidade do Vector ao mínimo necessário a fim de armazenar os elementos que se encontram nele.

Classes Membro

Já estamos acostumados a ver, dentro da definição de classes, a definição de dados e funções membros de instância ou de classe; mas não são apenas os dados e as funções que podem ser membros, classes também podem.

Classes membro sempre serão membros de classe (mesmo não estando formalmente qualificadas como tal, i.e., nunca as qualificamos com static, mas, mesmo assim, sempre serão consideradas membros de classe) e podem ser qualificadas como public, private ou protected.

Herança

Um programa pode muitas vezes conter grupos de classes relacionadas (semelhantes, mas não idênticas). Com o conceito de herança, grupos de classes relacionadas podem ser criados de maneira simples, flexível, eficiente e elegante.

Características comuns são agrupadas em uma classe (classe base). Desta classe poderíamos derivar outras classes (classes derivadas), e destas ainda outras classes, e assim por diante. Classes derivadas herdam todos os membros da classe base. Elas podem acrescentar novos membros àqueles herdados, bem como redefinir aqueles membros.

Posto que classes derivadas herdam as características de sua classe base, a descrição de uma classe derivada deve se concentrar nas diferenças que esta apresenta com relação a classe base, adicionando ou modificando características.

As funções acrescentadas por uma classe derivada somente podem ser usadas a partir de objetos da classe derivada. As funções definidas numa classe base podem ser usadas em objetos da classe base ou em objetos de qualquer de suas classes derivadas.

Para redefinir uma função membro de sua classe base, a classe derivada deve implementar uma função membro de mesmo nome e com parâmetros do mesmo tipo. A redefinição não será possível se a função membro da classe base estiver qualificada pelo modificador final.

No caso de uma classe precisar chamar um construtor de sua classe base na redefinição de seu construtor, ela poderá fazê-lo através da palavra chave super. Em vez ser utilizado o nome da classe para chamar o construtor, seria utilizada a palavra super eventualmente seguida dos argumentos a serem passados ao construtor separados por virgulas (,) e entre parênteses (()).

Classes qualificadas como final não podem ser usadas como base para a derivação de outra classe.

Mais sobre Conversões de Tipo

Conversões de tipo que transformam um objeto de uma dada subclasse em uma de suas superclasses podem acontecer implícita ou explicitamente e são completamente confiáveis.

Conversões de tipo que transformam um objeto de uma dada superclasse em uma de suas subclasses não são completamente confiáveis e somente podem ocorrer explicitamente.

Conversões de tipo transformam um objeto de uma dada classe em outra classe que não está em linha direta de ascendência ou descendência em uma mesma hierarquia de classes não são permitidas.

Verificação de Tipo em Java

Em Java, sempre que um objeto de um certo tipo é esperado, é possível usar um objeto daquele mesmo tipo, ou então, em seu lugar, um objeto de qualquer classe derivada daquele tipo.

Esta característica da linguagem permite a criação de código polimórfico, o que permite a programação adequada tanto funções que precisam trabalhar especificamente com uma certa classe derivada, quanto funções que precisam trabalhar genericamente com qualquer classe da hierarquia.

A função a ser executada quando de uma chamada é determinada com base no tipo do objeto ao qual a chamada foi vinculada, no nome usado na chamada e nos tipos dos parâmetros fornecidos.

Se o tipo do objeto declarado não bate com o tipo do objeto fornecido, então, em tempo de execução, o objeto fornecido será examinado de modo a determinar qual função deve ser executada. Chamamos a isto Binding Dinâmico.

Membros Protegidos

Conforme sabemos, existem 3 tipos de membros que podem ser definidos em uma classe, a saber: os membros públicos (que são acessíveis para qualquer método que tenha acesso à classe em que foram definidos), os membros privativos (que são acessíveis somente nos métodos definidos em sua classe) e os membros default (ou de pacote, que são acessíveis para todas os métodos definidos nas classes que integram o pacote onde foi definida sua classe).

Ficaremos sabendo agora da existência de um quarto e último tipo de membro, os membros protegidos. Para declarar um membro protegido, basta preceder a declaração do membro com a palavra protected (em vez de usar a palavra public ou a palavra private).

Membros protegidos, além de serem, como os membros default (ou de pacote), acessíveis para todas os métodos definidos nas classes que integram o pacote onde foi definida sua classe, são também acessíveis para métodos definidos em uma classe derivada da classe onde foram definidos.

Convém ressaltar que, no caso da classe base e da classe derivada residirem em pacotes diferentes, a classe derivada somente terá acesso aos membros protegidos de sua classe e de instâncias de sua própria classe, não tendo acesso aos membros protegidos de sua classe base ou de instâncias de sua classe base.

java.lang.Comparable

Este tipo de dado representa qualquer dado que possui a propriedade de poder ser comparado com outro do mesmo tipo. Veja abaixo a interface que a classe especifica para comunicação e com instâncias deste tipo:

- **Métodos:**

  – **int compareTo (Object O):**

    Compara este Comparable com o Object O; retorna um número negativo no caso do primeiro ser menor que o segundo; retorna zero, no caso de ambos serem iguais; e retorna um número positivo no caso do primeiro ser maior que o segundo.

[C:\ExplsJava\Expl_14\ Listas\Lista.java]

```
package Listas;

public class Lista
{
    protected class Elem
    {
        private Comparable Info;
        private Elem       Prox;

        public Elem (Comparable I, Elem E)
        {
            this.Info = I;
            this.Prox = E;
        }

        public Comparable SuaInfo ()
        {
            return this.Info;
        }

        public Elem SeuProx ()
        {
            return this.Prox;
        }

        public void AssumaProx (Elem E)
        {
            this.Prox = E;
        }
    }

    protected Elem    Prim = null;
    protected boolean Erro = false;

    public boolean DeuErro ()
    {
        return this.Erro;
    }

    public boolean Vazia ()
    {
        this.Erro = false;

        return this.Prim == null;
    }

    public void GuardeEmOrdem (Comparable I)
```

```
    {
        Elem Novo = new Elem (I, null);

        if (this.Vazia ())
        {
            this.Prim = new Elem (I, null);
            return;
        }

        if (I.compareTo (this.Prim.SuaInfo ()) < 0)
        {
            this.Prim = new Elem (I, this.Prim);
            return;
        }

        Elem Ant, Atu;

        for (Ant = this.Prim, Atu = this.Prim.SeuProx ();
             ;
             Ant = Atu, Atu = Atu.SeuProx ())
        {
            if (Atu ==  null)
                break;

            if (I.compareTo (Atu.SuaInfo ()) < 0)
                break;
        }

        Ant.AssumaProx (new Elem (I, Atu));
    }

    public boolean Possui (Comparable I)
    {
        for (Elem E = this.Prim; E != null; E = E.SeuProx ())
            if (I.compareTo (E.SuaInfo ()) == 0)
                return true;

        this.Erro = false;
        return false;
    }

    public void Descarte (Comparable I)
    {
        Elem Ant, Atu;

        for (Ant = null, Atu = this.Prim;
             ;
             Ant = Atu, Atu = Atu.SeuProx ())
        {
            if (Atu == null)
                break;

            if (I.compareTo (Atu.SuaInfo()) == 0)
                break;
        }

        if (Atu == null)
        {
            this.Erro = true;
            return;
        }

        if (Ant == null)
```

```
            this.Prim = Atu.SeuProx ();
        else
            Ant.AssumaProx (Atu.SeuProx ());

        this.Erro = false;
    }

    public String toString ()
    {
        String R  = "";

        for (Elem E = this.Prim; E != null; E = E.SeuProx ())
           R = R + E.SuaInfo () + (E.SeuProx () == null? "": ", ");

        this.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_14\ Listas\ListaCompleta.java]

```
package Listas;

public class ListaCompleta extends Lista
{
    public Comparable Primeiro ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
            return null;
        }

        this.Erro = false;
        return this.Prim.SuaInfo ();
    }

    public void GuardeNoInicio (Comparable F)
    {
        this.Prim = new Elem (F, this.Prim);
        this.Erro = false;
    }

    public void DescartePrimeiro ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
            return;
        }

        this.Prim = this.Prim.SeuProx ();
        this.Erro = false;
    }

    public Comparable Ultimo ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
```

```
            return null;
        }

        Elem E;

        for (E = this.Prim; E.SeuProx () != null; E = E.SeuProx ());


        this.Erro = false;
        return E.SuaInfo ();
    }

    public void GuardeNoFinal (Comparable F)
    {

        this.Erro = false;

        if (this.Vazia ())
        {
            this.Prim = new Elem (F, null);
            return;
        }

        Elem E;

        for (E = this.Prim; E.SeuProx  ()!= null; E = E.SeuProx ());

        E.AssumaProx (new Elem (F, null));
    }

    public void DescarteUltimo ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
            return;
        }

        this.Erro = false;

        if (this.Prim.SeuProx () == null)
        {
            this.Prim = null;
            return;
        }

        Elem E;

        for (E = this.Prim; E.SeuProx().SeuProx () != null; E = E.SeuProx ());

        E.AssumaProx (null);
    }

    public String toString ()
    {
        return '{' + super.toString () + '}';
    }
}
```

[C:\ExplsJava\Expl_14\TesteDeLista.java]

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import Listas.*;

class TesteDeLista
{
    public static void main (String Args []) throws Exception
    {
        BufferedReader Entrada = new BufferedReader
                                 (new InputStreamReader
                                 (System.in));

        Lista L1 = new Lista ();

        Float F;

        System.out.println ();

        System.out.print   ("Teste de Lista em Lista -> ");
        System.out.println ("Entre com 10 reais: ");

        for (int I = 1; I <= 10; I++)
        {
            F = new Float (Entrada.readLine ());
            L1.GuardeEmOrdem (F);
        }

        System.out.println ();

        System.out.println ("Em ordem, os numeros digitados sao: " );
        System.out.println (L1);

        System.out.println ();

        System.out.print ("Entre com um real para ser excluido: ");
        F = new Float (Entrada.readLine ());

        L1.Descarte (F);

        if (L1.DeuErro ())
            System.err.println ("O real dado nao consta na lista!");
        else
        {
            System.out.println ();
            System.out.println ("Em ordem, apos excluir, a lista ficou: ");
            System.out.println (L1);
        }

        System.out.println ();

        System.out.print   ("Entre com dois reais ");
        System.out.println ("para ver se pertencem a lista: ");

        F = new Float (Entrada.readLine ());

        if (L1.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        F = new Float (Entrada.readLine ());
```

```
        if (L1.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        System.out.println ();

        L1 = new ListaCompleta ();

        System.out.println ();

        System.out.print   ("Teste de ListaCompleta em Lista -> ");
        System.out.println ("Entre com 10 reais: ");

        for (int I = 1; I <= 10; I++)
        {
            F = new Float (Entrada.readLine ());
            L1.GuardeEmOrdem (F);
        }

        System.out.println ();

        System.out.println ("Em ordem, os numeros digitados sao: " );
        System.out.println (L1);

        System.out.println ();

        System.out.print ("Entre com um real para ser excluido: ");
        F = new Float (Entrada.readLine ());

        L1.Descarte (F);

        if (L1.DeuErro ())
            System.err.println ("O real dado nao consta na lista!");
        else
        {
            System.out.println ();
            System.out.println ("Em ordem, apos excluir, a lista ficou: ");
            System.out.println (L1);
        }

        System.out.println ();

        System.out.print   ("Entre com dois reais ");
        System.out.println ("para ver se pertencem a lista: ");

        F = new Float (Entrada.readLine ());

        if (L1.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        F = new Float (Entrada.readLine ());

        if (L1.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        System.out.println ();

        ListaCompleta L2 = new ListaCompleta ();
```

```
        System.out.println ();

        System.out.print   ("Teste de ListaCompleta em ListaCompleta -> ");
        System.out.println ("Entre com 10 reais: ");

        for (int I = 1; I <= 10; I++)
        {
            F = new Float (Entrada.readLine ());
            L2.GuardeEmOrdem (F);
        }

        System.out.println ();

        System.out.println ("Em ordem, os numeros digitados sao: " );
        System.out.println (L2);

        System.out.println ();

        System.out.print ("Entre com um real para ser excluido: ");
        F = new Float (Entrada.readLine ());

        L2.Descarte (F);

        if (L2.DeuErro ())
            System.err.println ("O real dado nao consta na lista!");
        else
        {
            System.out.println ();
            System.out.println ("Em ordem, apos excluir, a lista ficou: ");
            System.out.println (L2);
        }


        System.out.println ();

        System.out.print   ("Entre com dois reais ");
        System.out.println ("para ver se pertencem a lista: ");

        F = new Float (Entrada.readLine ());

        if (L2.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        F = new Float (Entrada.readLine ());

        if (L2.Possui (F))
            System.out.println ("O numero pertence a lista");
        else
            System.out.println ("O numero nao pertence a lista");

        System.out.println ();

        System.out.println ("Entre com dois reais ");
        System.out.print   ("(um para ser incluido no inicio ");
        System.out.println ("e outro no final da lista): ");

        F = new Float (Entrada.readLine ());

        L2.GuardeNoInicio (F);
```

```
        F = new Float (Entrada.readLine ());

        L2.GuardeNoFinal (F);

        System.out.println ();

        System.out.println ("Apos as inclusoes, a lista ficou: ");
        System.out.println (L2);

        System.out.println ();

        System.out.print   ("Agora, o primeiro da lista e ");
        System.out.println (L2.Primeiro ());
        System.out.print   ("e o ultimo da lista e ");
        System.out.println (L2.Ultimo ());

        System.out.println ();

        L2.DescartePrimeiro ();
        L2.DescarteUltimo   ();

        System.out.print   ("Apos a exclusao do primeiro ");
        System.out.println ("e do ultimo, a lista ficou: ");
        System.out.println (L2);

        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_14> javac -classpath . -classpath . TesteDeLista.java
C:\ExplsJava\Expl_14> java -classpath . -classpath . TesteDeLista
```

Isto poderia produzir no console a seguinte interação:

```
Teste de Lista em Lista -> Entre com 10 inteiros:
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0

Em ordem, os numeros digitados sao:
0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0

Teste de ListaCompleta em Lista -> Entre com 10 inteiros:
9.0
```

```
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0

Em ordem, os numeros digitados sao:
{0.0 ,  1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0 ,  6.0 ,  7.0 ,  8.0 ,  9.0}

Teste de ListaCompleta em ListaCompleta -> Entre com 10 inteiros:
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0

Em ordem, os numeros digitados sao:
{0.0 ,  1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0 ,  6.0 ,  7.0 ,  8.0 ,  9.0}

Entre com um inteiro para ser excluido: 7.0

Em ordem, apos excluir, a lista ficou:
{0.0 ,  1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0 ,  6.0 ,  8.0 ,  9.0}

Entre com dois inteiros para ver se pertencem a lista:
3.0
O numero pertence a lista
7.0
O numero nao pertence a lista

Entre com dois inteiros
(um para ser incluido no inicio e outro no final da lista):
-13.0
13.0

Apos as inclusoes, a lista ficou:
{-13.0 ,  0.0 ,  1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0 ,  6.0 ,  7.0 ,  8.0 ,  9.0 ,  13.0}
```

```
Agora, o primeiro da lista e -13
e o ultimo da lista e 13

Apos a exclusao do primeiro e do ultimo, a lista ficou:
{0.0 ,  1.0 ,  2.0 ,  3.0 ,  4.0 ,  5.0 ,  6.0 ,  8.0 ,  9.0}
```

A Classe java.lang.Object

Esta classe é a raiz da hierarquia de classes, i.e., toda classe a tem como superclasse. Todos as classes, incluindo os vetores, implementam os métodos desta classe. Veja abaixo a interface que a classe especifica para comunicação com ela própria e com suas instâncias:

- **Construtores:**

– **Object ():**
  Constroi uma instância da classe Object;

- **Métodos:**

– **boolean equals (Object O):**
  Verifica se este Object é igual a O;

– **Class getClass ():**
  Retorna o descritor da classe deste Object;

– **int hashCode ():**
  Retorna o código de hash deste Object (código de hash é um número que identifica unicamente um objeto);

– **void notify ():**
  Acorda uma Thread que esteja esperando no monitor deste Object;

– **void notifyAll ():**
  Acorda todas as Thread's que estejam esperando no monitor deste Object;

– **String toString ():**
  Retorna uma instância da classe String que representa este Object;

- **void wait ():**

  Faz com que a Thread corrente espere até que uma outra Thread chame o método notify () ou notifyAll () para este Object;

- **void wait (long M):**

  Faz com que a Thread corrente espere, por no máximo M milisegundos, que uma outra Thread chame o método notify () ou notifyAll () para este Object;

- **void wait (long M, int N):**

  Faz com que a Thread corrente espere, por no máximo M milisegundos mais N nanosegundos, que uma outra Thread chame o método notify () ou notifyAll () para este Object.

Interfaces

Interfaces são, em certo sentido, muito parecidas com classes.Podemos pensar que interfaces sejam tão simplesmente classes totalmente abstratas, ou seja, classes cujos métodos, em sua totalidade, são abstratos e, em Java, devem ser qualificados com as palavras chave public e abstract.

Em vez de serem introduzidas pela palavra chave class, interfaces são introduzidas pela palavra chave interface.

Os dados de uma interface, se existirem, deverão ser todos qualificados com os modificadores static final; em outras palavras, os dados de uma interface, se existirem, deverão ser constantes de classe.

Herança Múltipla

A linguagem Java não implementa o recurso de herança múltipla de forma tão abrangente quanto fazem outras linguagens, e.g., C++. Veja abaixo as restrições que Java impõe à herança múltipla:

1. Interfaces não podem derivar de classes;

2. Interfaces podem derivar de múltiplas interfaces (sendo IB_i e ID nomes de interfaces, veja abaixo a forma geral do cabeçalho de uma interface que deriva de outras interfaces: interface ID extends IB1, IB2,..., IBn);

3. Classes podem derivar de apenas uma outra classe, mas podem derivar de múltiplas interfaces (sendo CB e CD nomes de classes, e IB_i nomes de interfaces, veja abaixo a forma geral do cabeçalho de uma classe que deriva de uma outra classe e de diversas interfaces: class CD extends CB implements IB_1, IB_2,..., IB_n).

[C:\ExplsJava\Expl_15\Lista.java]

```
class Lista
{
    private Object Info;
    private Lista  Resto;

    private static boolean Erro = false;

    public static boolean DeuErro ()
    {
        return Lista.Erro;
    }

    private Lista (Object I, Lista R)
    {
        this.Info  = I;
        this.Resto = R;

        Lista.Erro = false;
    }

    public static Object Primeiro (Lista L)
    {
        if (L == null)
        {
            Lista.Erro = true;
            return null;
        }

        Lista.Erro = false;
        return L.Info;
    }

    public static Lista ComNovoInicio (Object I, Lista L)
    {
        Lista.Erro = false;
        return new Lista (I, L);
    }

    public static Lista SemPrimeiro (Lista L)
    {
        if (L == null)
        {
            Lista.Erro = true;
```

```
            return null;
        }

        Lista.Erro = false;
        return L.Resto;
    }

    public static Object Ultimo (Lista L)
    {
        if (L == null)
        {
            Lista.Erro = true;
            return null;
        }

        Lista.Erro = false;

        if (L.Resto != null)
            return L.Info;
        else
            return Lista.Ultimo (L.Resto);
    }

    public static Lista ComNovoFinal (Object I, Lista L)
    {
        Lista.Erro = false;

        if (L == null)
            return new Lista (I, null);
        else
        {
            L.Resto = Lista.ComNovoFinal (I, L.Resto);
            return L;
        }
    }

    public static Lista SemUltimo (Lista L)
    {
        if (L == null)
        {
            Lista.Erro = true;
            return null;
        }

        Lista.Erro = false;

        if (L.Resto == null)
            return null;
        else
        {
            L.Resto = Lista.SemUltimo (L.Resto);
            return L;
        }
    }
}
```

[C:\ExplsJava\Expl_15\Pilha.java]

```
interface Pilha
{
```

```
    public abstract boolean Cheia ();
    public abstract boolean Vazia ();

    public abstract void  Empilhe (Object I);
    public abstract Object Desempilhe ();
}
```

[C:\ExplsJava\Expl_15\PilhaV.java]

```
class PilhaV implements Pilha
{
    private Object  Elem [];
    private int     Topo = -1;
    private boolean Erro = false;

    public PilhaV (int T)
    {
        if (T <= 0)
        {
            this.Erro = true;
            return;
        }

        this.Elem = new Object [T];
        this.Erro = false;
    }

    public PilhaV ()
    {
        this (25);
    }

    public boolean DeuErro ()
    {
        return this.Erro;
    }

    public boolean Cheia ()
    {
        this.Erro = false;
        return this.Topo+1 == this.Elem.length;
    }

    public boolean Vazia ()
    {
        this.Erro = false;
        return this.Topo == -1;
    }

    public void Empilhe (Object E)
    {
        if (this.Cheia ())

        {
            this.Erro = true;
            return;
        }

        this.Elem [++this.Topo] = E;
        this.Erro = false;
```

```
    }

    public Object Desempilhe ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
            return null;
        }

        this.Erro = false;
        return this.Elem [this.Topo--];
    }
}
```

[C:\ExplsJava\Expl_15\PilhaL.java]

```
class PilhaL implements Pilha
{
    private Lista   Elems = null;
    private boolean Erro  = false;

    public boolean DeuErro ()
    {
        return this.Erro;
    }

    public boolean Cheia ()
    {
        return false;
    }

    public boolean Vazia ()
    {
        return this.Elems == null;
    }

    public void Empilhe (Object E)
    {
        this.Elems = Lista.ComNovoInicio (E, Elems);
    }

    public Object Desempilhe ()
    {
        if (this.Vazia ())
        {
            this.Erro = true;
            return null;
        }

        Object R = Lista.Primeiro (this.Elems);
        this.Elems = Lista.SemPrimeiro (this.Elems);

        return R;
    }
}
```

[C:\ExplsJava\Expl_15\TesteDeInterface.java]

```
class TesteDeInterface
{
    public static void main (String Args [])
    {
        Pilha P;

        System.out.println ();

        System.out.println ("Teste de PilhaV ->");
        System.out.println ();
        System.out.println ("Empilhamos, nesta ordem, o seguinte:");

        P = new PilhaV ();

        for (float F = 0.0f; F <= 9.0f; F++)
        {
            P.Empilhe (new Float (F));
            System.out.print (F + " ");
        }

        System.out.println ();
        System.out.println ();
        System.out.println ("Desempilhamos, nesta ordem, o seguinte:");

        while (!P.Vazia ())
            System.out.print (P.Desempilhe () + " ");

        System.out.println ();
        System.out.println ();
        System.out.println ();
        System.out.println ();

        System.out.println ("Teste de PilhaL ->");
        System.out.println ();
        System.out.println ("Empilhamos, nesta ordem, o seguinte:");

        P = new PilhaL ();

        for (float F = 0.0f; F <= 9.0f; F++)
        {
            P.Empilhe (new Float (F));
            System.out.print (F + " ");
        }

        System.out.println ();
        System.out.println ();
        System.out.println ("Desempilhamos, nesta ordem, o seguinte:");

        while (!P.Vazia ())
            System.out.print (P.Desempilhe () + " ");

        System.out.println ();
        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_15> javac -classpath . TesteDeInterface.java
C:\ExplsJava\Expl_15> java -classpath . TesteDeInterface
```

Isto poderia produzir no console a seguinte interação:

Teste de PilhaV ->

Empilhamos, nesta ordem, o seguinte:
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0

Desempilhamos, nesta ordem, o seguinte:
9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0 0.0

Teste de PilhaL ->

Empilhamos, nesta ordem, o seguinte:
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0

Desempilhamos, nesta ordem, o seguinte:
9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0 0.0

Classes Abstratas

Conforme sabemos, a herança convencional representa a possibilidade de incorporar e extender as características (comportamento e implementação) de uma classe já existente.

Também sabemos da existência de interfaces, que definem comportamentos, sem implementá-los, que podem ser herdados por outras interfaces, bem como implementados por classes.

Classes abstratas nos oferecem um meio termo, i.e., a possibilidade de herdar alguns comportamentos implementados, outros por implementar. Os comportamentos não implementados são representados por métodos abstratos.

[C:\ExplsJava\Expl_16\Deposito.java]

```
abstract class Deposito
{
    protected float   Elem [];
    protected int     DeOndeTirar = 0,
                      QtosElems   = 0;
    protected boolean Erro        = false;

    public Deposito (int T)
    {
        if (T <= 0)
```

```
        {
            this.Erro = true;
            return;
        }

        this.Elem = new float [T];
        this.Erro = false;
    }

    public Deposito ()
    {
        this (25);
    }

    public boolean DeuErro ()
    {
        return this.Erro;
    }

    public boolean SemEspaco ()
    {
        this.Erro = false;
        return this.QtosElems == this.Elem.length;
    }

    public boolean SemElementos ()
    {
        this.Erro = false;
        return this.QtosElems == 0;
    }

    abstract public void Guarde (float N);
    abstract public float RetireElemento ();
}
```

[C:\ExplsJava\Expl_16\Pilha.java]

```
class Pilha extends Deposito
{
    public void Guarde (float N)
    {
        if (this.SemEspaco ())
        {
            this.Erro = true;
            return;
        }

        this.Elem [this.DeOndeTirar++] = N;
        this.QtosElems++;

        this.Erro = false;
    }

    public float RetireElemento ()
    {
        if (this.SemElementos ())
        {
            this.Erro = true;
            return Float.NaN;
        }
```

```
        float R = this.Elem [--this.DeOndeTirar];

        this.QtosElems--;

        this.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_16\Fila.java]

```
class Fila extends Deposito
{
    private int OndeGuardar = 0;

    public void Guarde (float N)
    {
        if (this.SemEspaco ())
        {
            this.Erro = true;
            return;
        }

        this.Elem [this.OndeGuardar] = N;

        this.OndeGuardar = (this.OndeGuardar + 1) % this.Elem.length;
        this.QtosElems++;

        this.Erro = false;
    }

    public float RetireElemento ()
    {
        if (this.SemElementos ())
        {
            this.Erro = true;
            return Float.NaN;
        }

        float R = this.Elem [this.DeOndeTirar];

        this.DeOndeTirar = (this.DeOndeTirar + 1) % this.Elem.length;
        this.QtosElems--;

        this.Erro = false;
        return R;
    }
}
```

[C:\ExplsJava\Expl_16\TesteDeClasseAbstrata.java]

```
class TesteDeClasseAbstrata
{
    public static void main (String Args [])
    {
```

```
        Deposito D;

        System.out.println ();

        System.out.println ("Teste de Pilha ->");
        System.out.println ();

        D = new Pilha ();

        System.out.println ("Empilhamos, nesta ordem, o seguinte:");

        for (float I = 0.0f; I <= 9.0f; I++)
        {
            D.Guarde (I);
            System.out.print (I + " ");
        }

        System.out.println ();
        System.out.println ();

        System.out.println ("Desempilhamos, nesta ordem, o seguinte:");

        while (!D.SemElementos ())
            System.out.print (D.RetireElemento () + " ");

        System.out.println ();
        System.out.println ();
        System.out.println ();
        System.out.println ();

        System.out.println ("Teste de Fila ->");
        System.out.println ();

        D = new Fila ();

        System.out.println ("Enfileiramos, nesta ordem, o seguinte:");

        for (float I = 0.0f; I <= 9.0f; I++)
        {
            D.Guarde (I);
            System.out.print (I + " ");
        }

        System.out.println ();
        System.out.println ();

        System.out.println ("Desenfileiramos, nesta ordem, o seguinte:");

        while (!D.SemElementos ())
            System.out.print (D.RetireElemento () + " ");

        System.out.println ();
        System.out.println ();
    }
}
```

Para compilar e executar este programa, daremos os seguintes comandos:

```
C:\ExplsJava\Expl_16> javac -classpath . TesteDeClasseAbstrata.java
C:\ExplsJava\Expl_16> java -classpath . TesteDeClasseAbstrata
```

Isto poderia produzir no console a seguinte interação:

```
Teste de Pilha ->

Empilhamos, nesta ordem, o seguinte:
0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0

Desempilhamos, nesta ordem, o seguinte:
9.0  8.0  7.0  6.0  5.0  4.0  3.0  2.0  1.0  0.0


Teste de Fila ->

Enfileiramos, nesta ordem, o seguinte:
0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0

Desenfileiramos, nesta ordem, o seguinte:
0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0
```

Anexo I

Exercícios

I. Classes e Objetos

1. Responda verdadeiro ou falso e justifique: a definição de uma classe reserva espaço de memória para conter todos os dados presentem em sua definição.

2. A finalidade de definir classes é:

   a) Reservar uma quantidade de memória;

   b) Indicar que o programa é orientado a objetos;

   c) Agrupar dados e funções protegendo-os do compilador;

   d) Descrever novos tipos abstratos de dados antes desconhecidos do compilador.

3. A relação entre classes, objetos e instâncias é a mesma existente entre:

   a) Tipos básicos, variáveis e valores desses tipos;

   b) Variáveis, funções e procedimentos;

   c) Constantes, variáveis e valores;

   d) Constantes, variáveis e tipos básicos.

4. Os membros de classe designados como privativos na definição de uma classe:

   a) São acessíveis por qualquer função do programa;

   b) São acessíveis por qualquer função de sua classe;

   c) São acessíveis apenas para a funções de classe de sua classe;

   d) São acessíveis apenas para a funções de instância de sua classe.

5. Os membros de instância designados como privativos na definição de uma classe:

   a) São acessíveis por qualquer função do programa;

   b) São acessíveis por qualquer função de sua classe;

   c) São acessíveis apenas para a funções de classe de sua classe;

   d) São acessíveis apenas para a funções de instância de sua classe.

6. Responda verdadeiro ou falso: membros privados definem a atividade interna da classe e de suas instâncias, enquanto os membros públicos definem o padrão de comunicação da classe e de suas instâncias com o mundo exterior a elas.

7. Para acessar um membro de classe presente na definição de uma classe, o operador ponto conecta:

   a) O nome da classe e o nome do membro;

   b) O nome do membro e o nome de um objeto da classe;

   c) O nome de um objeto da classe e o nome do membro;

   d) O nome da classe e o nome de um objeto da classe.

8. Para acessar um membro de instância presente na definição de uma classe, o operador ponto conecta:

   a) O nome da classe e o nome do membro;

   b) O nome do membro e o nome de um objeto da classe;

   c) Uma instância da classe, necessariamente em um objeto da classe, e o nome do membro.

   d) Uma instância da classe, eventualmente em um objeto da classe, e o nome do membro.

9. Métodos são:

   a) Classes;

   b) Dados-membro;

c) Funções-membro;

d) Chamadas a funções-membro.

10. Mensagens são:

a) Classes;

b) Dados-membro;

c) Funções-membro;

d) Chamadas a funções-membro.

11. Construtores são funções:

a) Que constroem classes;

b) Executadas automaticamente quando uma instância é criada;

c) Executadas automaticamente quando um objeto é declarado;

d) Executadas automaticamente quando uma instância é destruída.

12. O nome de um construtor é sempre _____.

13. Responda verdadeiro ou falso: numa classe, é possivel haver mais de um construtor de mesmo nome.

14. Responda verdadeiro ou falso: um construtor é do tipo retornado por meio do comando *return*.

15. Responda verdadeiro ou falso: um construtor não pode ter argumentos.

16. Destrutores são funções:

a) Que destroem classes;

b) Executadas automaticamente quando uma instância é criada;

c) Executadas automaticamente quando um objeto é declarado;

d) Executadas automaticamente quando uma instância é destruida.

17. O nome de um destrutor é sempre _____.

18. Responda verdadeiro ou falso: numa classe pode haver mais de um destrutor de mesmo nome.

19. Responda verdadeiro ou falso: um destrutor é do tipo retornado por meio do comando *return*.

20. Responda verdadeiro ou falso: um destrutor não pode receber argumentos.

21. Assuma que C1, C2 e C3 sejam objetos de uma mesma classe. Quais das seguintes instruções são válidas?

    a) C1 = C2;

    b) C1 = C2 + C3;

    c) C1 = C2 = C3;

    d) C1 = C2 + 7;

22. Uma função-membro de classe pode apenas acessar os dados:

    a) Da instância à qual está associada;

    b) Da classe onde foi definida;

    c) Da classe e de qualquer instância da classe onde foi definida;

    d) Da parte pública da classe onde foi definida.

23. Uma função-membro de instância pode apenas acessar os dados:

    a) Da instância à qual está associada;

    b) Da classe onde foi definida;

    c) Da classe e de qualquer instância da classe onde foi definida;

    d) Da parte pública da classe onde foi definida.

24. Suponha que cinco instâncias de uma classe sejam criadas. Quantas cópias dos dados-membro de classe de sua classe serão armazenadas na memória? Quantas cópias

dos dados-membro de instância de sua classe serão armazenadas na memória? Quantas cópias de suas funções-membro de classe de sua classe serão armazenadas na memória? Quantas cópias de suas funções-membro de instância de sua classe serão armazenadas na memória?

25. Escreva uma classe chamada Horario com 3 membros de instância inteiros chamados Horas, Minutos e Segundos.

    - Inclua um construtor que, recebendo como parâmetros 3 valores inteiros, inicie os dados internos da instância à qual se refere;
    - Crie uma função-membro de instância que, sem receber parâmetros, retorne uma cadeia de caracteres no formato HH:MM:SS que represente a instância à qual se refere;
    - Crie uma função-membro que, recebendo uma instância da classe Horario, some-a com a instância à qual se refere, retornando uma instância da classe Horario que represente o resultado;
    - Crie uma função-membro que, recebendo uma instância da classe Horario, o subtraia da instância à qual se refere, retornando uma instância da classe Horario que represente o resultado

26. Escreva uma classe chamada Polígono que descreva figuras poligonais convexas sem impor limitações na quantidade de lados. As instâncias da classe deverão ser capazes de armazenar a quantidade de vértices que o polígono que ela representa deve ter, a quantidade de vértices do polígono que efetivamente foram informados, e as coordenadas de cada um desses vértices.

    - Escreva um construtor que receba como argumento a quantidade de lados do polígono a ser criado, armazene internamente essa informação, e instancie os vetores para armazenar as coordenadas X e Y dos vértices do polígono.
    - Escreva uma função-membro de instância chamada IncluaVertice que, recebendo dois números float que representam respectivamente as coordenadas X e Y de um vértice do polígono, os armazena internamente.

- Escreva uma função-membro de instância chamada RemovaVertice que, recebendo o número de ordem de um vértice, o remove toma as atitudes necessárias para remove-lo da instância à qual se refere.

27. Escreva uma classe chamada Angulo. Suas instâncias deverão ser capazes de armazenar um valor angular expresso em graus.

    - Escreva um construtor que, recebendo um valor angular expresso em graus, o armazena internamente;

    - Escreva funções-membro de instância chamadas ValorEmGraus, ValorEmGrados e ValorEmRadianos que, sem receber parâmetros, retornam, respectivamente, o valor angular do objeto expresso em graus, grados e radianos.

28. Escreva uma classe chamada Circulo. Suas instâncias deverão ser capazes de armazenar as coordenadas do centro, bem como o valor do raio de um círculo.

    - Escreva um construtor que, recebendo números reais expressando as coordenadas e o valor do raio de um círculo, armazena essas informações internamente;

    - Escreva uma função-membro que, recebendo um objeto da classe Angulo, retorna o comprimento do arco do círculo com aquela varredura angular;

    - Escreva uma função-membro que, recebendo um objeto da classe Angulo, retorna a área do setor do círculo com aquela varredura angular.

29. Escreva uma classe chamada Lista cujas instâncias descrevam listas circulares duplamente ligadas de inteiros.

    - Escreva um construtor que, sem receber parâmetros, inicia a lista com vazio;

    - Escreva um construtor que, recebendo uma instância da classe Lista, a usa para iniciar a instância à qual se refere (lembre-se, tem que duplicar a lista);

    - Escreva uma função-membro de instância chamada Inclua que, recebendo um inteiro, o inclui na instância à qual se refere;

- Escreva uma função-membro de instância chamada Remova que, recebendo um número inteiro, remove todas as ocorrências do inteiro dado da instância à qual se refere;
- Escreva uma função-membro de instância chamada Existe que, recebendo um número inteiro, verifica se este existe na instância à qual se refere, retornando o valor lógico verdadeiro, em caso afirmativo, ou o valor lógico falso, caso contrário;
- Escreva uma função-membro de instância chamada Inverso que, sem receber parâmetros, produz e retorna uma instância da classe Lista que representa o inverso da instância à qual se refere;
- Escreva uma função-membro de instância chamada Concatenacao que, recebendo uma instância da classe Lista, produz e retorna uma instância da classe Lista que representa a concatenação da instância à qual se refere com a instância fornecida;
- Escreva uma função-membro de instância chamada Ordenação que, sem receber parâmetros, produz e retorna uma instância da classe Lista que representa a ordenação da instância à qual se refere;
- Escreva uma função-membro de instância chamada Merge que, recebendo uma instância da classe Lista, produz e retorna uma instância da classe Lista que representa a intercalação dos elementos da instância à qual se refere com os elementos da instância fornecida;
- Escreva uma função-membro de instância chamada Igual que, recebendo uma instância da classe Lista, retorna o valor lógico verdadeiro, caso a instância à qual se refere seja idêntica à instância recebida, e o valor lógico falso, caso contrário.

30. Escreva uma classe chamada Conjunto cujas instâncias descrevam conjuntos de números inteiros. Os valores pertencentes a um Conjunto deverão ser armazenados em uma instância da classe Lista que deverá existir em seu interior.

    - Escreva um construtor que, sem receber parâmetros, inicia a instância à qual se refere com o conjunto vazio.

- Escreva um construtor que, recebendo uma instância da classe Conjunto, inicia a instância à qual se refere com os mesmos elementos da instância recebida.
- Escreva uma função-membro de instância chamada Inclua que, recebendo um valor inteiro, promove a inclusão deste na instância à qual se refere.
- Escreva uma função-membro de instância chamada Elimine que, recebendo um valor inteiro, o remove da instância à qual se refere.
- Escreva uma função-membro de instância chamada Intersecao que, recebendo uma instância da classe Conjunto, produz e retorna uma instância da classe conjunto que representa a interseção da instância à qual se refere com a instância recebida.
- Escreva uma função-membro de instância chamada Uniao que, recebendo uma instância da classe Conjunto, produz e retorna uma instância da classe conjunto que representa a união da instância à qual se refere com a instância recebida.
- Escreva uma função-membro de instância chamada Pertence que, recebendo como parâmetro um valor inteiro, verifica se o mesmo pertence à instância à qual se refere, retornando o valor lógico verdadeiro, em caso afirmativo, ou o valor lógico falso, caso contrário.
- Implemente uma função-membro de instância chamado toString que, sem receber nenhum parâmetro, produz e retorna um String que representa textualmente a instância à qual se refere.

31. Suponha a existência uma classe Sensor que implementa funções de instância capazes de controlar um sensor e que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; e (2) um método de instância chamado Ativado sem parâmetros que retorna valor lógico que indica a detecção ou não de algo se movendo em seu "campo de visão".

    Suponha a existência uma classe Porta que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros

inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; (2) uma função (de nome AbraSe) sem parâmetros e sem retorno que, quando executada, faz com que a porta se abra; e (3) uma função (de nome FecheSe) sem parâmetros e sem retorno que, quando executada, faz com que a porta feche.

Valendo-se dessas duas classes, crie a classe PortaAuto, derivada das duas classes acima mencionadas, para representar uma porta automática como as que costumamos encontrar em aeroportos. Pede-se:

- Implemente um construtor para inicializar apropriadamente as instâncias das classes Sensor e Porta que existem no interior da instância à qual se refere;
- Implemente um método chamado EntreEmOperacao que, sem receber parâmetros e nem produzir retornos, faz com que a porta automática entre em operação ininterrupta.

ATENÇÃO: a porta pode ser considerada inicialmente fechada e nunca deve ser aberta, quando já se encontra aberta, e nem fechada, quando já se encontra fechada.

32. Suponha a existência uma classe Sensor que implementa funções de instância capazes de controlar um sensor de ficha e que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; e (2) um método de instância chamado Ativado sem parâmetros que retorna um valor lógico que, se for verdadeiro, indica que o sensor detectou o depósito de uma ficha, e, se for falso, indica que o sensor não detectou o depósito de uma ficha (essa função retorna verdadeiro apenas uma vez por ficha depositada).

    Suponha a existência uma classe Cancela que implementa funções capazes de controlar uma cancela eletrônica e que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; e (2) um método de instância chamado

AbraSe, sem parâmetros e sem retorno, que, quando executada, faz com que a cancela se abra para a passagem de um veículo, fechando-se automaticamente após a passagem deste.

O objetivo deste exercício é, valendo-se dessas duas classes, implementar, DE FORMA COMPLETA E ADEQUADA, a classe EstacaoDePedagio que implementa funções capazes de controlar uma estação automática de cobrança de pedágio. Pede-se:

- Declarar objetos privativos da classe Sensor e Cancela;
- Implementar um construtor que, recebendo 6 inteiros (3 deles para o construtor de Sensor e os outros 3 para o construtor de Cancela), inicia apropriadamente a instância à qual se refere; e
- Implementar um método chamado Funcione que, sem receber parâmetros e sem produzir retornos, faz com que a estação automática de cobrança de pedágio passe a funcionar de forma ininterrupta.

II. Pacotes

1. Assuma que as classes C1 e C2 estejam definidas em pacotes diferentes. Para que C1 possa ser empregada na definição de C2, é necessário que C1 seja _____.
2. Para que membros definidos em uma classe possam ser acessados em funções da definidas em uma classe que se encontra no mesmo pacote, eles devem ter sido declarados como:
   a) public ou private;
   b) protected ou private;
   c) public ou protected;
   d) Nenhuma das anteriores.
3. Para que membros definidos em uma classe publica possam ser acessados em funções da definidas em uma classe que se encontra em outro pacote, eles devem ter sido declarados como:
   a) public;

b) protected;

c) private;

d) Todas as anteriores.

III. Herança

1. Herança é um proceso que permite:

   a) A inclusão de um objeto dentro de outro;

   b) Transformar classes genéricas em classes mais específicas;

   c) Implementar uma classe semelhante a uma classe existente sem a necessidade de copiá-la ou reescrevê-la

   d) Relacionar objetos por meio de seus argumentos.

2. As vantagens do uso de herança incluem:

   a) Aproveitamento de classes existentes na elaboração de novas classes;

   b) Uso de bibliotecas;

   c) Concepção top-down de algorítmos;

   d) Melhor uso da memória.

3. Para derivar uma classe de outra já existente, deve-se:

   a) Alterar a classe existente;

   b) Reescrever a classe existente;

   c) Indicar que a nova classe incorpora as características da classe existente, acrescentando à nova classe novas características ou redefinindo na nova classe características herdadas;

   d) Nenhuma das anteriores.

4. Se em uma classe-base foi definida uma função de instância chamada F e em uma classe dela derivada não foi definida nenhuma função com este nome, responda: em que situação a referida função chamada F poderá ser aplicada a uma instância da classe derivada?

5. Responda verdadeiro ou falso: suponha que em uma classe-base foi definida uma função chamada F e em uma classe dela derivada também foi definida uma função chamada F, com os mesmos parâmetros e retorno. Uma outra função definida na classe-derivada pode empregar a função herdada apesar da redefinição.

6. Responda verdadeiro ou falso: suponha que em uma classe-base foi definida uma função chamada F e em uma classe dela derivada também foi definida uma função chamada F, com os mesmos parâmetros e retorno. Uma outra função definida fora da classe-derivada pode empregar a função herdada apesar da redefinição.

7. Classes derivadas incorporam:

   a) Todos os membros da classe-base;

   b) Somente os membros públicos da classe-base;

   c) Somente os membros protegidos da classe-base;

   d) O segundo e o terceiro item são verdadeiros.

8. Responda verdadeiro ou falso: se nenhum construtor existir na classe derivada, objetos desta classe usarão o construtor sem argumentos da classe-base.

9. Responda verdadeiro ou falso: se nenhum construtor existir na classe derivada, objetos desta classe poderão iniciar os dados herdados de sua classe-base usando o construtor com argumentos da classe-base.

10. Uma classe é dita abstrata quando:

    a) Nenhum objeto dela é declarado;

    b) É representada apenas mentalmente;

    c) Só pode ser usada como base para outras classes;

d) É definida de forma obscura.

11. A conversão de tipos implícita é usada para:

    a) Armazenar instâncias de uma classe-derivada em objetos de sua classe-base;

    b) Armazenar instâncias de uma classe-base em objetos de uma classe dela derivada;

    c) Armazenar instâncias de uma classe-derivada em objetos de sua classe-base e vice-versa;

    d) Nao pode ser usada para conversão de objetos.

12. Responda verdadeiro ou falso: uma classe derivada não pode servir de base para outra classe.

13. Responda verdadeiro ou falso: um objeto de uma classe pode ser membro de outra classe.

14. Suponha implementada uma classe chamada Pto para representar um ponto no plano cartesiano. As estruturas internas desta classe supostamente são desconhecidas e seus métodos públicos devem ser considerados como sendo os seguintes:

    - Um construtor que, recebendo um par de números reais, respectivamente as coordenadas de um ponto no plano cartesiano, inicie apropriadamente a instância à qual se refere;
    - Um método de instância chamado X que retorna a coordenada X da instância à qual se refere e um método de instância chamado Y que retorna a coordenada Y da instância à qual se refere.

    O objetivo desta questão é implementar uma classe chamada Ponto, derivada da classe Pto, para representar, de forma mais completa, um ponto no plano cartesiano. Sabendo que:

    - A distância entre os pontos $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ é dada pela seguinte expressão:

      $$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- A inclinação da reta que passa pelos pontos $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ é dada pela seguinte expressão:

Pede-se:

$$\frac{(y_2 - y_1)}{(x_2 - x_1)}$$

- Declare a classe Ponto e as estruturas internas que a definem;
- Implemente um método de instância chamado Distancia que, recebendo como parâmetro uma instância da classe Ponto, retorna a distância do ponto representado pela instância à qual se refere até o ponto representado pela instância recebida como parâmetro;
- Implemente um método de instancia chamado Inclinacao que retorna a inclinação da reta imaginária que une o ponto representado pela instancia à qual se refere à origem do sistema cartesiano.

15. Suponha a existência uma classe Sensor que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; e (2) uma função (de nome Velocidade) sem parâmetros que retorna um real que indica a velocidade do objeto detectado (indicará sempre zero no caso de nada estar sendo detectado).

    Suponha a existência uma classe Camera que dispõe de (1) um construtor que, recebendo um inteiro que representa o número de seu pedido de interrupção, bem como dois outros inteiros que representam, respectivamente, os limites, inferior e superior, do seu intervalo de E/S, inicia apropriadamente a instância à qual se refere; (2) uma função (de nome Fotografe) que recebe um parâmetro real e sem retorno que, quando executada, faz com que a câmera bata uma foto na qual aparecerá no canto inferior direito o horário em que a foto foi batida (a câmera tem um relógio interno) e o número real que vem a esta função como parâmetro.

    Pede-se:

- Derivada da classe Sensor e valendo-se da classe Camera, crie a classe Radar para representar um radar como estes que encontramos instalados nas principais ruas e avenidas das grandes cidades;
- Implemente um construtor com dois argumentos inteiros que representam os endereços das portas de hardware nas quais estarão conectados, respectivamente, um sensor e uma câmera automática. Esse construtor deverá iniciar apropriadamente as estruturas internas da classe Radar;
- Implemente uma função de instância pública, chamada EntreEmAção, que coloca em operação de forma ininterrupta uma instância da classe Radar.

Anexo II

Bibliografia

SUN Microsystems, "The Java™ Tutorial", (http://java.sun.com/docs/books/tutorial/).

SUN Microsystems, "The Java™ Language Specification", (http://java.sun.com/docs/books/jls/index.html).

SUN Microsystems, "The Java™ API Specification", (http://java.sun.com/products/jdk/1.2/docs/api/index.html).

SUN Microsystems, "The Java™ VM Specification", (http://java.sun.com/docs/books/vmspec/index.html).

Naughton, Patrick, "Dominando o Java", Makron Books, 1997.

Morrison, M.; "Java Unleashed", December, J.; et alii; SAMS Publishing.

Naughton, P.; "Dominando o Java"; Makron Books.

Linden, P.; "Just Java"; Makron Books.

Fraizer, C.; e Bond, J.; "API Java: Manual de Referência"; Makron Books.

Thomas, M.D.; Patel, P.r.; et alii; "Programando em Java para a Internet"; Makron Books.

Ritchey, T.; "Programando Java e JavaScript"; Quark Editora.