Programação Orientada a Objetos em

Flávio Miguel Varejão

Departamento de Informática

UFES

Preâmbulo

- Formato do Curso
  - Aulas expositivas
    - Ritmo definido em conjunto
  - Leitura da apostila
    - Conteúdo amplo
  - Programação é FUNDAMENTAL!
    - Exercícios e Trabalhos
    - Aulas de Laboratório
    - Extra Classe

Preâmbulo

- Enquete
  - Programador Experiente em Java
  - Programador Iniciante em Java
  - Programador Experiente em C
  - Programador Experiente em outra Linguagem Orientada a Objetos (Delphi, C#, C++)
  - Programador Experiente em outra Linguagem Imperativa (Pascal, Visual Basic, Perl)
  - Nenhuma das Opções Anteriores

Preâmbulo

- Heterogeneidade da Turma
  - Necessidade de alcançar a todos (iniciantes e experientes)
  - Experientes necessitam ter paciência e se manterem concentrados
  - Perguntas são importantes e recomendadas

Conteúdo

- Introdução
- Tipos, Variáveis e Operadores
- Estruturas de Controle e Programação Básica
- Orientação a Objetos
- Reuso de Classes
- Polimorfismo

Introdução

- Motivações para Uso da Orientação a Objetos
  - uso de abstração de dados simplifica o processo de modelagem
  - paradigma único para análise, projeto e implementação
  - redução de custos de desenvolvimento através do reuso de código
  - redução de custos de manutenção através da separação entre interface e implementação

Introdução

- Motivações para o Uso de Java
  - Propósito geral
  - Bem projetada
  - Orientada a objetos
  - Confiável
  - Fácil (relativamente)
  - Segura

Objetivos do Curso

- Ensinar Conceitos Fundamentais de OO
  - Tipos Abstratos de Dados
  - Encapsulamento
  - Ocultamento de Informação
  - Composição e Herança
  - Polimorfismo

Objetivos do Curso

- Ensinar programação em Java
  - Primitivas da Linguagem
    - Quais são
    - Como usá-las para construir programas
    - Como e porque são incluídas
    - Como são implementadas
  - Programador pode ser mais produtivo

Conceitos Básicos de Linguagens de Programação

- Objetivo
  - Facilitar entendimento dos conceitos de Java
  - Análise crítica de Java
- Tópicos
  - Propriedades Desejáveis
  - Tradução de Programas
  - Alocação Dinâmica de Memória
  - Conceito de Abstração

Propriedades Desejáveis em LPs

- Legibilidade
  - bloco monolítico
  - programação macarrônica
- Redigibilidade
  - necessidade de explicitar qq conversão de tipos
  - tratamento de erros
- Confiabilidade
  - digitação de nomes
  - uso de ponteiros

Propriedades Desejáveis em LPs

- Eficiência
  - verificação dinâmica de acesso a vetores
  - recursividade
- Facilidade de Aprendizado
  - excesso de conceitos
  - uso ostensivo de ponteiros
- Reusabilidade
  - subprogramas sem parâmetros
  - algoritmos e estruturas específicas para tipo

Propriedades Desejáveis em LPs

- Flexibilidade
  - ausência de modularização
  - mistura de implementação e uso de código
- Portabilidade
  - ausência de padrão
  - liberdade para a implementação do tradutor
- Harmonia com metodologia de projeto
  - análise OO e programação estruturada

Tradução de Programas

- Hardware capaz de executar conjunto de instruções extremamente básicas
- Programação muito trabalhosa e difícil neste nível
  - Baixa produtividade dos programadores (caro)
- Uso de LPs de Alto Nível

```
int sum = 0;
```

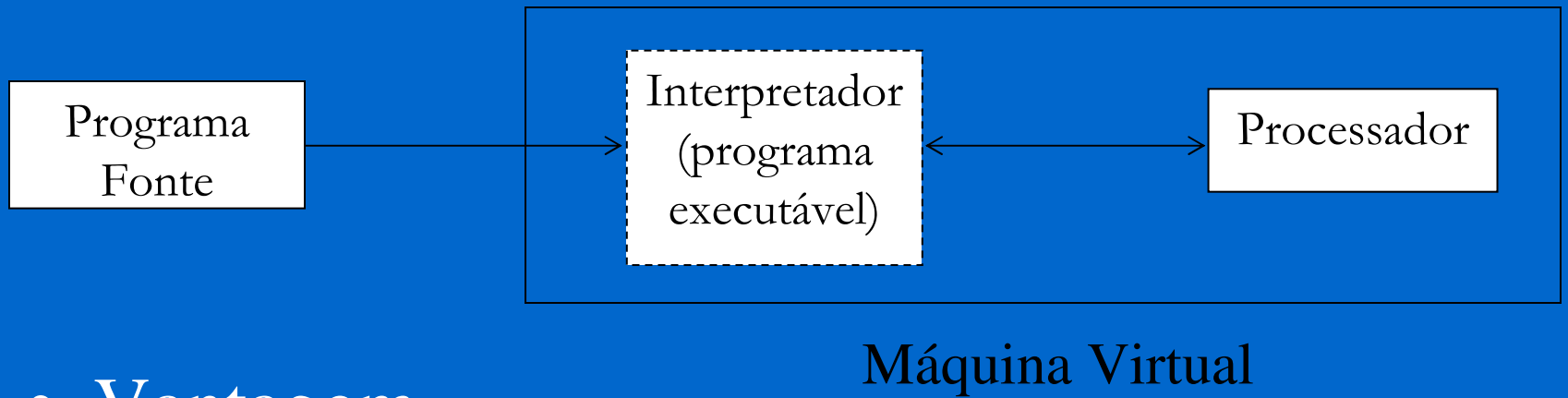
Tradução de Programas

- Hardware não é capaz de executar programas escritos nas LPs
- Necessário traduzir instruções das LPs para o conjunto de instruções básicas do hardware
- Modos de Tradução
  - Compilação
  - Interpretação

Compilação

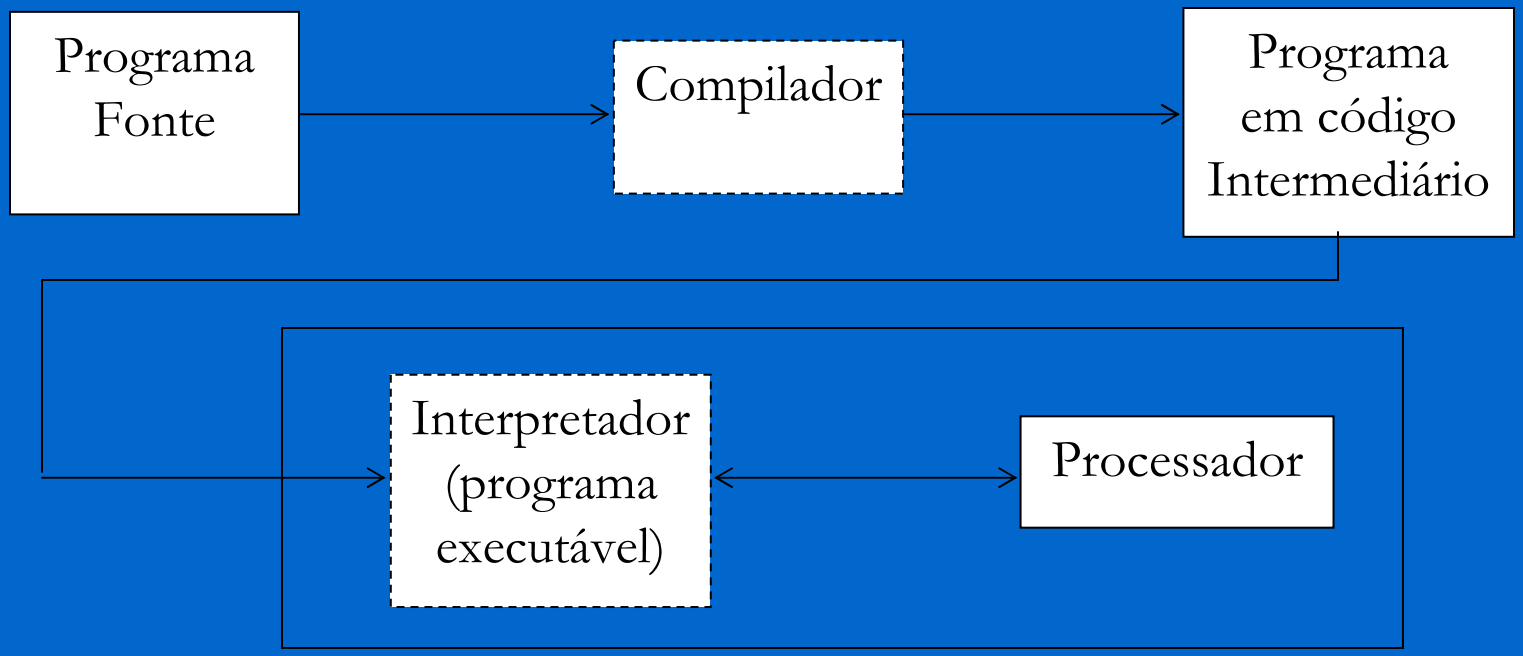
- Vantagem
  - Execução rápida
- Desvantagem
  - Falta de portabilidade

Interpretação



- Vantagem
  - Portabilidade
- Desvantagem
  - Execução lenta

Modelo Híbrido



- Combina compilação e interpretação
- Vantagens
  - Execução rápida
  - Portabilidade

Alocação de Memória

- Memória como vetor contíguo de células
- Memória para execução de programas
  - código executável
  - dados
- Diferentes políticas para alocação de memória de dados

Alocação Estática de Memória

- Alocação em tempo de carga do programa
  - Fortran
  - Desvantagens
    - Aproveitamento insatisfatório da memória disponível
      - reserva-se mais espaço do que necessário
        - » aloca tamanho máximo, mas normalmente usa menos
        - » nem todos os subprogramas são usados
      - estimativas de tamanho máximo podem falhar
    - Impede recursividade

Alocação Dinâmica de Memória

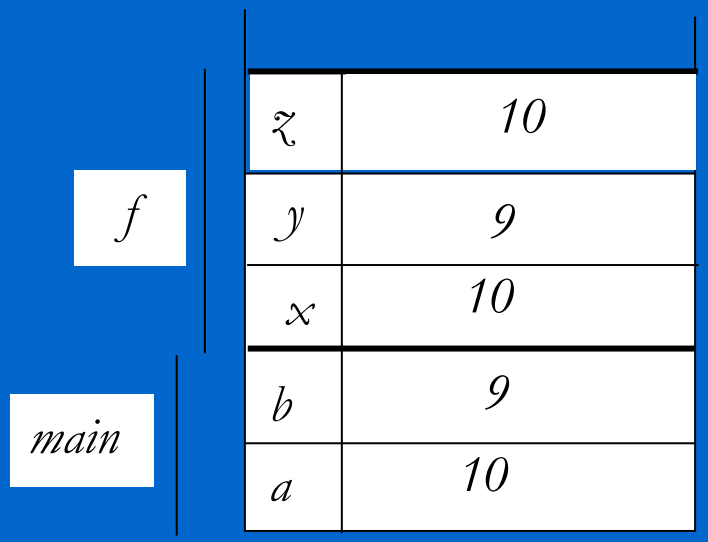
- Alocação em tempo de execução
  - na medida que a memória se faz necessária
  - na medida que o tamanho é conhecido
- Alocação contígua não é apropriada
  - requer número acentuado de movimentações dos elementos do vetor de memória
    - tamanho da variável pode mudar
    - tempo de vida pode ser intervalo menor
  - frequente atualização de endereços no código

Alocação Dinâmica de Memória

- Pilha
  - armazena espaço para variáveis locais e parâmetros dos subprogramas
  - resolve problema da recursividade
  - espaço só é alocado durante a execução do subprograma
  - eficiente, pois não requer número significativo de movimentações
  - problema de crescimento dinâmico das variáveis persiste

Pilha

```
f(int x, int y){
   int z;
   …
}
main () {
  int a, b;
   … f(a,b); …
}
```

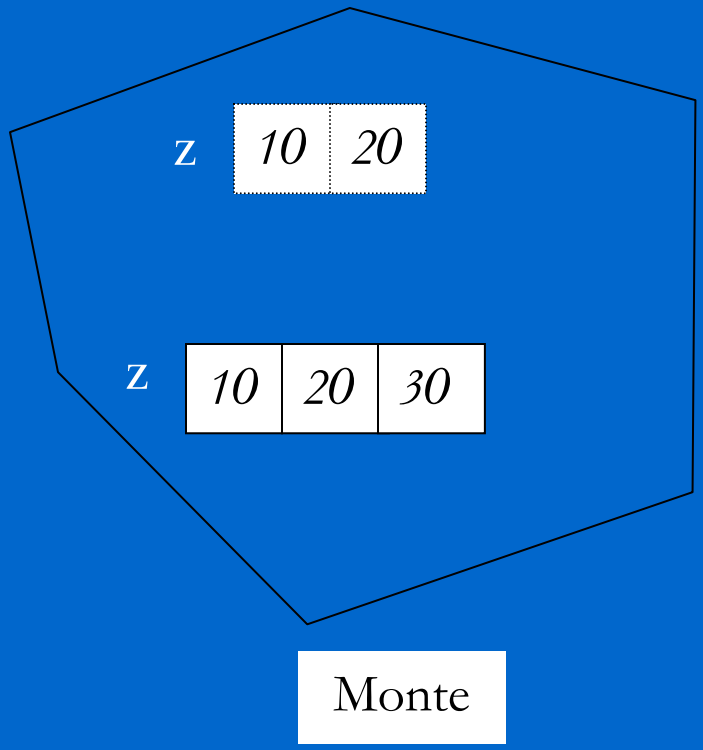


Alocação Dinâmica de Memória

- Monte
  - Alocação não contígua de memória
    - onde há espaço
    - crescimento dinâmico é tratado através de cópia
  - Problemas
    - cópia ainda é pouco eficiente
    - ocorre fragmentação da memória
    - frequente atualização de endereços no código
    - requer estrutura para gerenciamento da memória

Monte

```
main ( ) {
   …
   z = {10, 20};
   …
   …
   z = z + {30};
   …
}
```



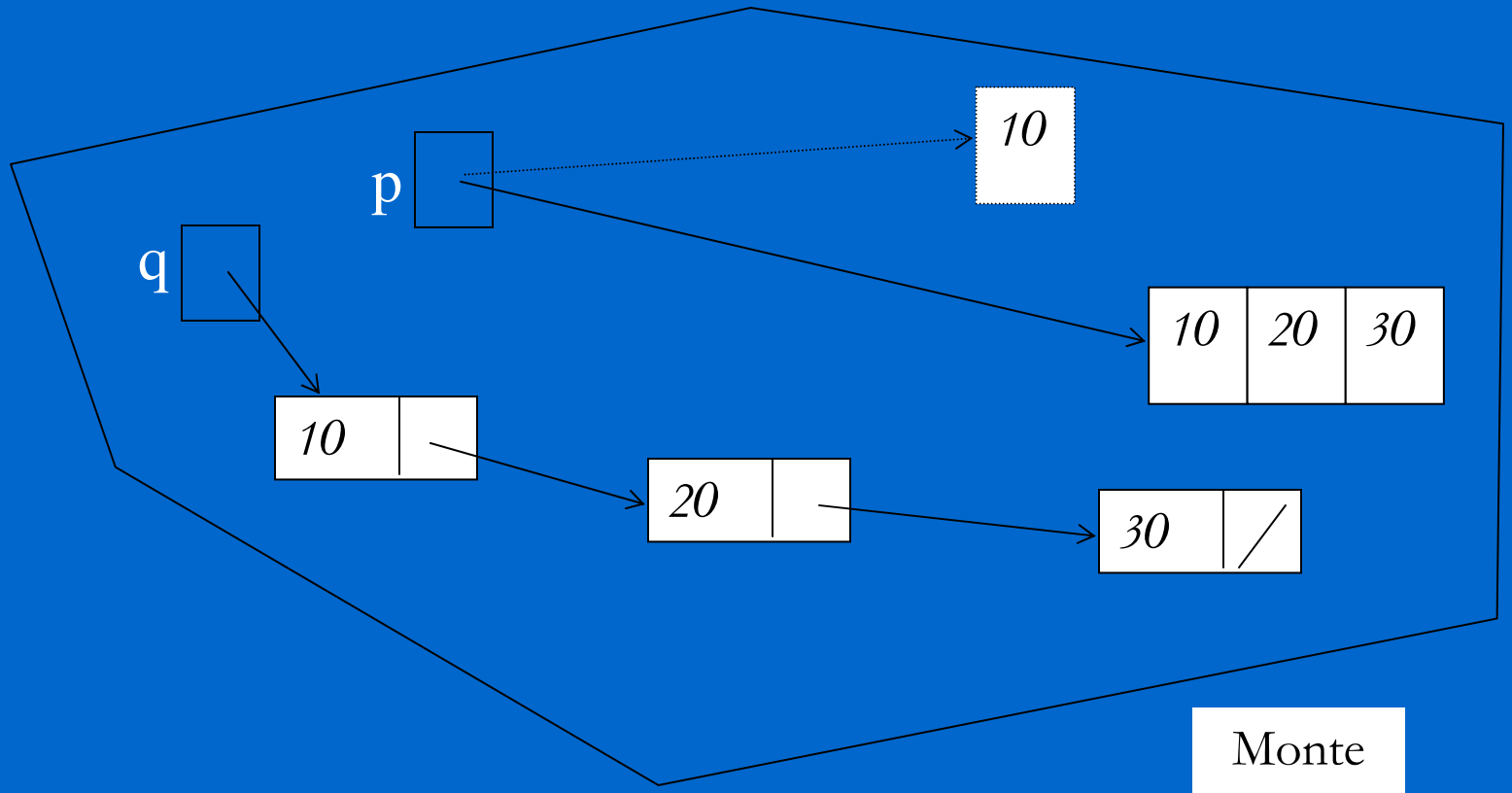
Gerenciamento da Memória

- Mapeamento da Memória do Programa
  - Mapa de Espaços Disponíveis
  - Mapa de Espaços Ocupados
  - Alocação
    - remove de disponíveis e inclui em ocupados
  - Desalocação
    - remove de ocupados e inclui em disponível
- Estruturas Dinâmicas

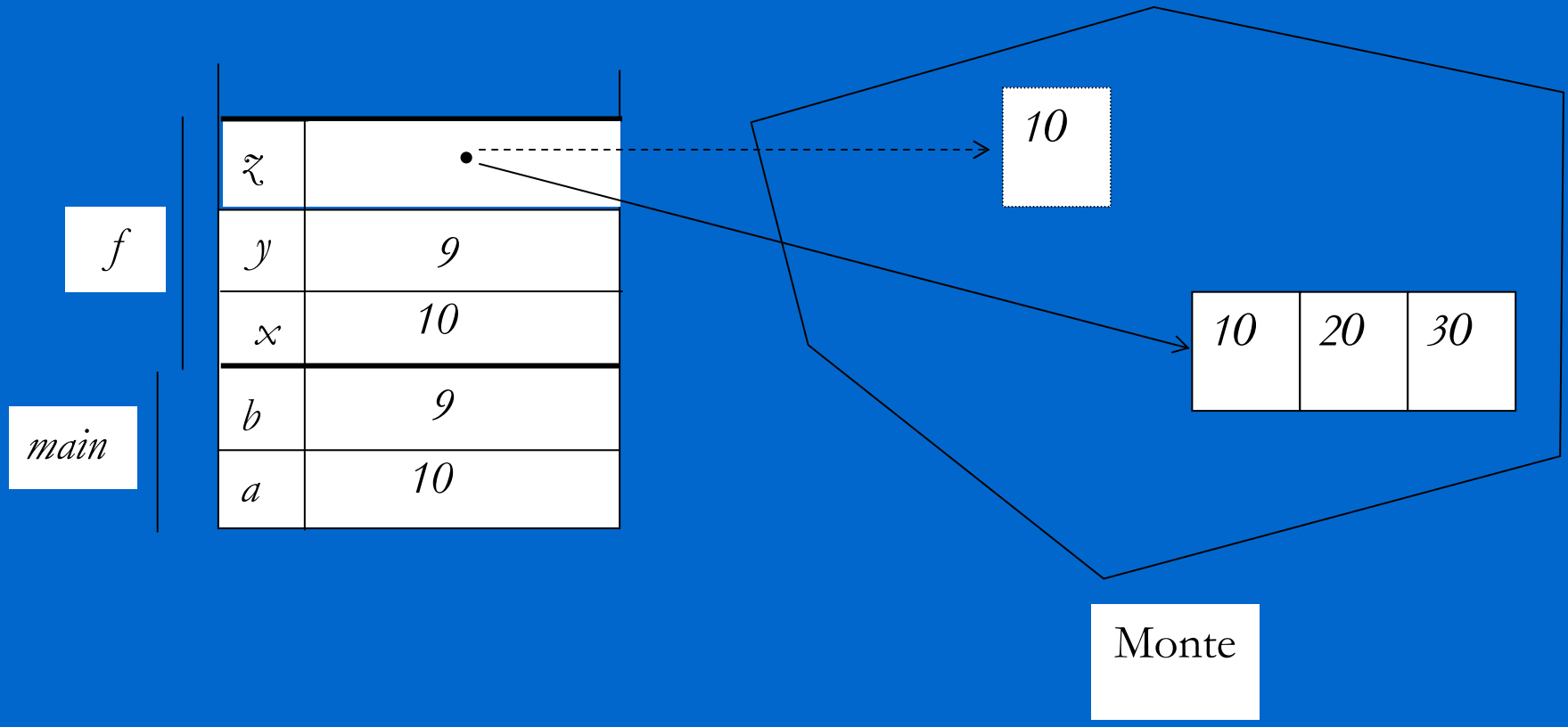
Ponteiros

- Variáveis que armazenam endereços de memória
  - usadas na implementação de estruturas dinâmicas
  - eliminam necessidade de atualização de endereços
  - reduzem a necessidade de cópia

Ponteiros



Pilha e Monte



Gerenciamento de Memória

- Controle do Programador
  - Eficiente
  - Trabalhosa
    - programador deve alocar e desalocar explicitamente
  - Uso de ponteiros provoca erros

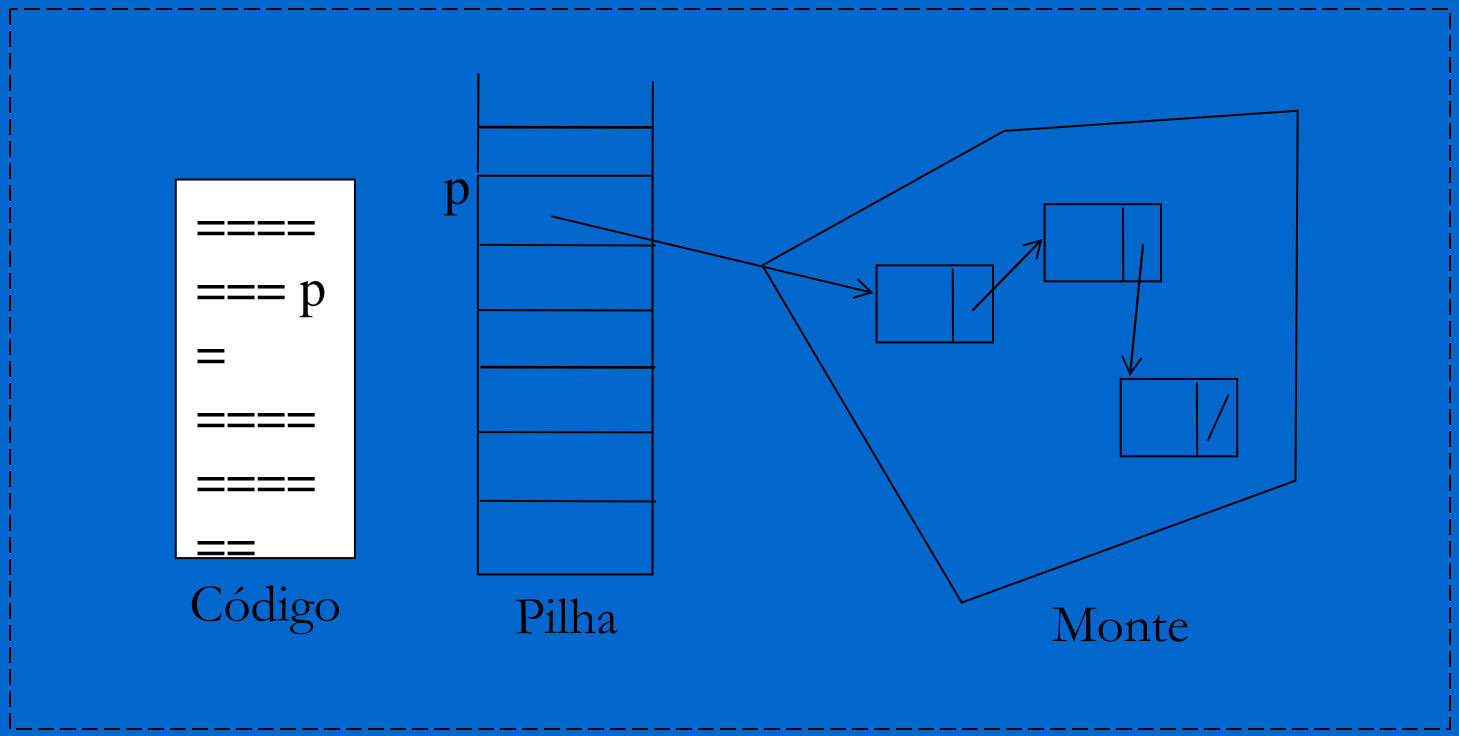
Problemas com Ponteiros

- Violação de Memória
  - Não Inicialização
  - Referências Pendentes
- Vazamento de Memória
  - Objetos Pendentes

Gerenciamento da Memória

- Controle da Linguagem
  - Confiabilidade
  - Menos eficiente
  - Coletor de Lixo

Modelo de Memória de Programas



Maiores Informações Sobre LPs

- Linguagens de Programação - Conceitos e Técnicas
  - Editora Elsevier (Campus)
  - Coleção Campus-SBC
  - Flávio Varejão
  - 2004

Abstração em LPs

- Assembler é abstração da linguagem de máquina
- LPs imperativas são abstrações dos assemblers
  - Requerem que se pense na arquitetura do computador ao invés da estrutura do problema
- LPs orientadas a objetos oferecem abstrações para modelagem do problema

Plataformas de Computação

- Plataformas (SO + Hardware)
  - Windows/PC
  - Solaris/Sun
  - AIX/IBM RS 6000
  - Linux/PC
  - Linux/Sun
- Plataforma Java (Máquina Virtual)

A Plataforma Java

- Nível acima das outras
- Baseia-se na JVM
- Executáveis escritos em bytecodes
- Idéia fundamental:

  "write once, run anywhere"
- Linguagem Java é a parte central da plataforma

Pequeno Histórico

- Projeto Green da Sun (1991)
  - sistemas embutidos portáteis
  - uso inicial de C++ não satisfez
  - criação da linguagem interpretada Oak
- Oak é rebatizado como Java (1993)
  - problemas de copyright

Pequeno Histórico

- Java apresentada formalmente pela Sun (1995)
  - sintaxe parecida com C/C++
  - distribuição gratuita, mas permanece proprietária
  - JDK 1.0
- JDK 1.1 (1997)
  - novo modelo de eventos

Pequeno Histórico

- JDK 1.2 (1999)
  - JFC e Swing
  - Java 2
- JDK => SDK
- J2SE 5.0 => JDK 1.5.0
  - Generics

Características de Java

- Orientada a Objetos
  - com exceção dos tipos primitivos, tudo é classe ou objeto
  - não existe o conceito de funções ou procedimentos independentes
  - reúso de código através de bibliotecas de classes
  - bem mais simples que C++

Características de Java

- Portável
  - Arquitetura baseada na JVM
  - Padrão rígido da linguagem
    - programas fontes são os mesmos
      - fixa o tamanho e intervalos dos tipos básicos
      - fixa a ordem de avaliação de expressões
      - biblioteca de classes padronizada

Características de Java

- Confiável
  - orientação a objetos induz o programador a práticas de programação confiáveis
  - ausência de ponteiros (não permite a manipulação direta de endereços de memória)
  - conversão de tipos regulada
  - mecanismo de tratamento de exceções
  - verificação extensiva em tempo de compilação e de execução
- Não elimina necessidade de boa prática

Características de Java

- Dinâmica
  - classes são carregadas sob demanda
- Distribuída
  - suporte através de bibliotecas de alto nível
  - invocação de métodos de objetos remotos
  - *Applets*
    - programas especiais carregados e executados através de *Web browsers*
    - atualização de código sem distribuição

Características de Java

- Segura
  - Várias camadas de controle de segurança contra código malicioso
    - não permite ao programador manipular endereços
    - interpretador Java realiza verificação de bytecodes (se não contém código ilegal) sempre que código não confiável é carregado
    - uso de caixa de areia (acesso limitado aos recursos locais)
    - assinatura digital criptografada
  - Segurança não é totalmente garantida

Características de Java

- Eficiente
  - Embora interpretada
  - 20 vezes mais lenta que equivalente em C
    - satisfaz grande parte das aplicações
  - Interpretadores Java podem incluir compiladores de bytecodes JIT
  - Pode-se escrever partes do programa em C/C++
  - Pode-se usar compilador específico para plataforma de execução
- Nunca será igual a C

Características de Java

- Suporta Concorrência
  - rotinas múltiplas concorrentes (threads)
  - elementos para sincronização

Recursos Necessários

- JDK 1.5.0

  http://java.sun.com/j2se
- Navegador com suporte a Java
- Instalação do JDK

  http://java.sun.com/j2se/1.5.0/install-linux.html
- Instalação da documentação

  http://java.sun.com/j2se/1.5.0/install-docs.hml

Recursos Disponíveis

- Site da Sun

  http://java.sun.com

  http://java.sun.com/docs/books/tutorial

- Pacotes, aplicações, exemplos

  http://www.developer.com/java

  http://www.javacoffeebreak.com

- Dúvidas comuns

  http://www.ibiblio.org/javafaq/javafaq.html

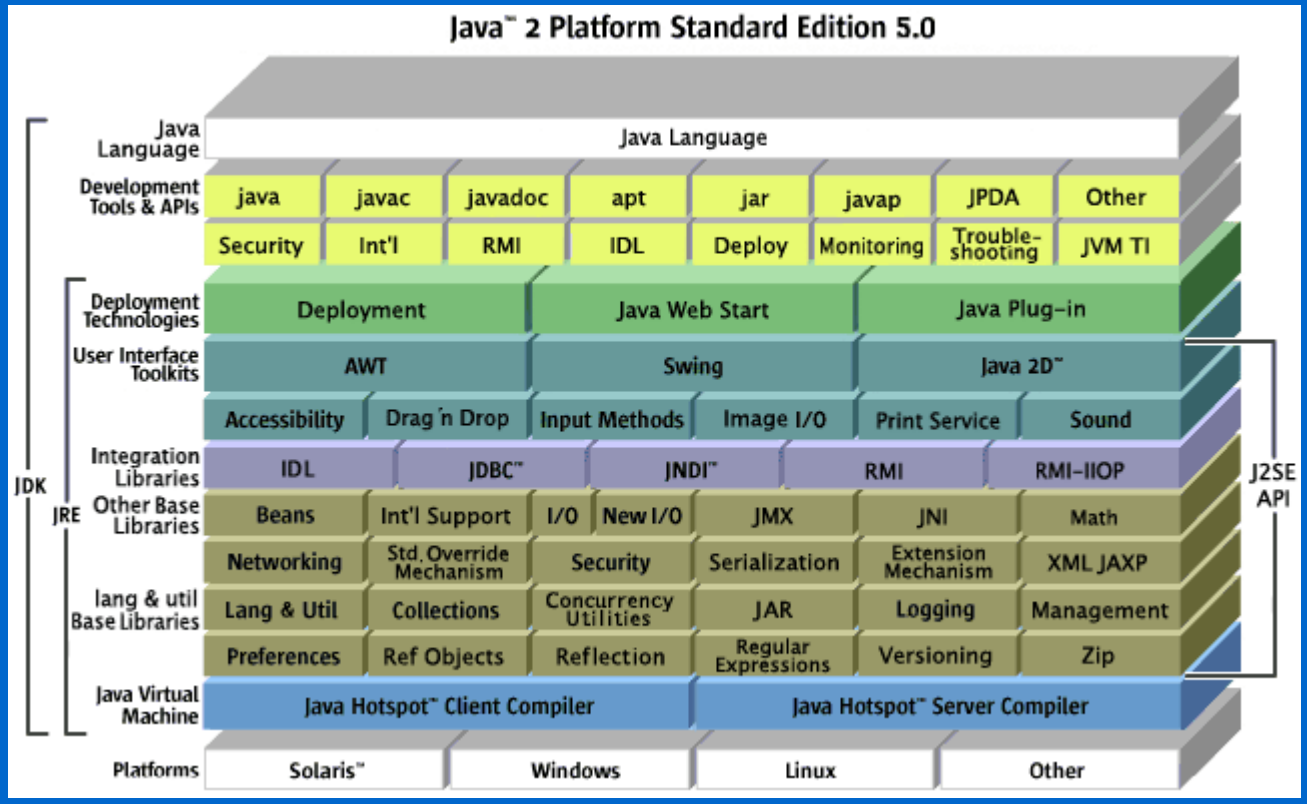
- Livro Thinking in Java

  http://www.bruceeckel.com

Componentes do JDK

- Compilador (javac)
- Interpretador (java)
- Bibliotecas (APIs)
- Gerador de documentação (javadoc)
- Depurador de programas (jdb)
- Ambiente de execução (jre)

Componentes do JDK - J2SE 5.0



Primeiro Exemplo

```
//Eco.java
import java.io.*;
public class Eco {
  public static void main(String[] args) {
    for (int i=0; i < args.length; i++)
    System.out.print(args[i] + " ");
    System.out.println();
  }
}
```

- Aplicação
- Salvar como arquivo Eco.java

Primeiro Exemplo

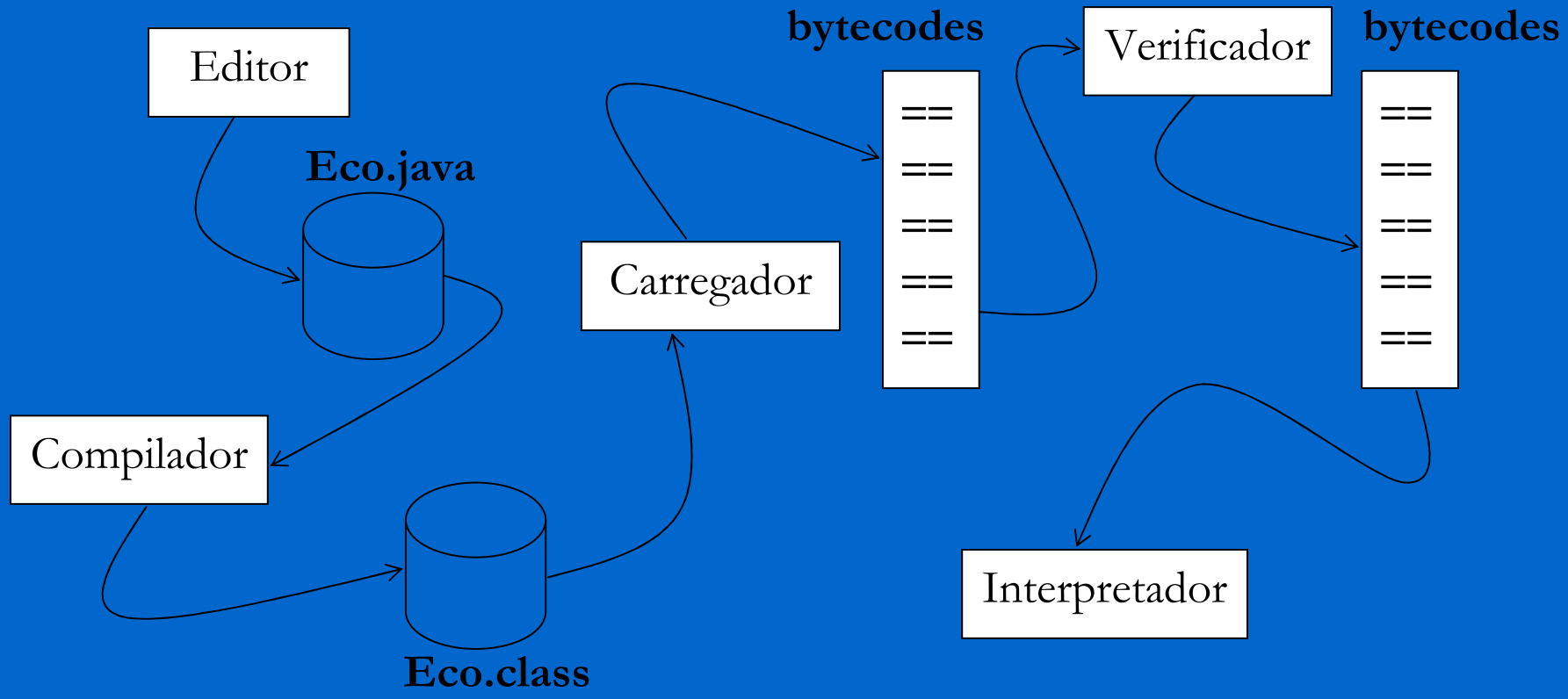
- Compilar com javac

```
javac Eco.java
```

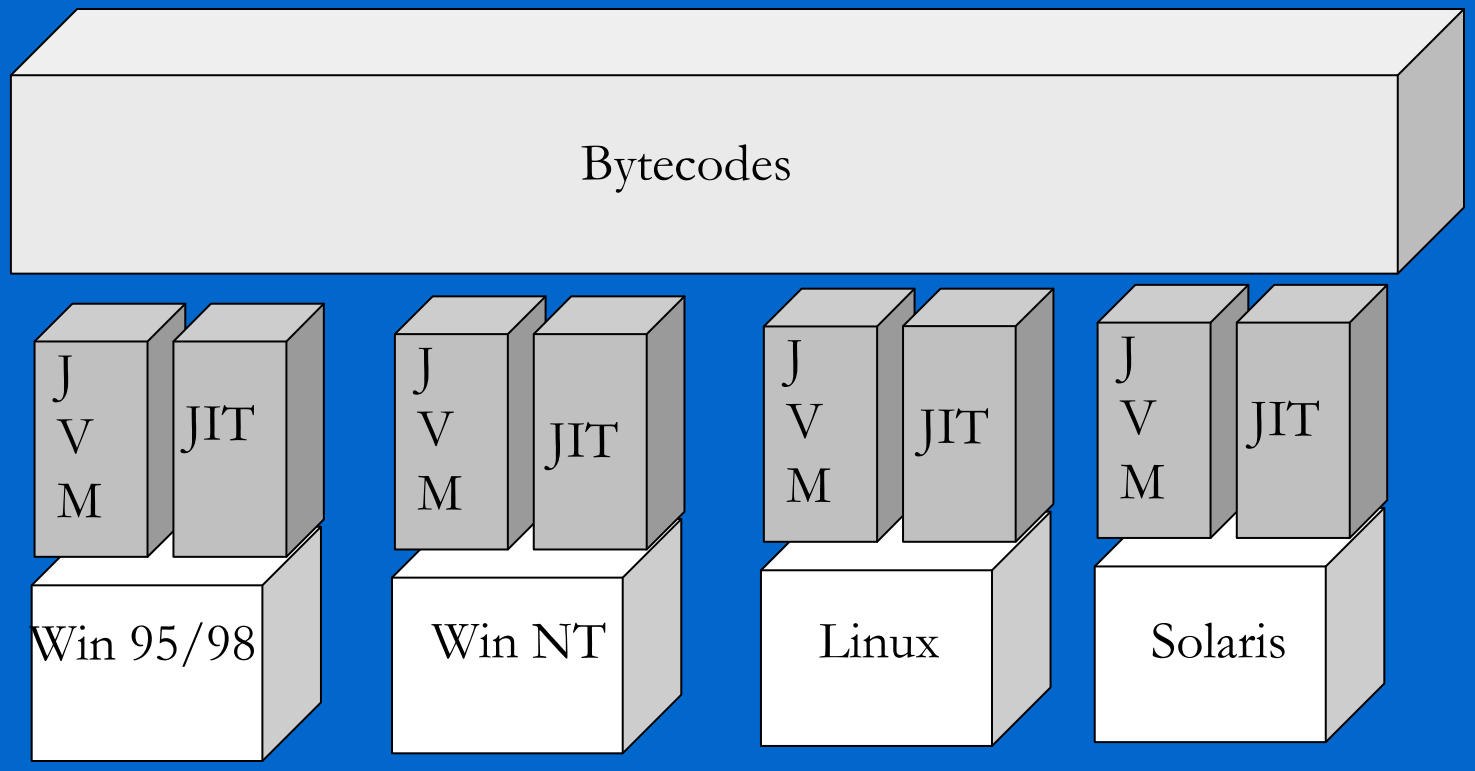
- Executar através de java

```
java Eco
java Eco Testando 1 2 3
```

Ambiente de Desenvolvimento



Ambiente de Execução



Tipos, Variáveis e Operadores

- Tipos primitivos e compostos
- Variáveis
  - declaração
  - escopo
  - tempo de vida
- Operadores
  - quais são e como operam
  - precedência e associatividade

Tipos Primitivos

- Não são considerados objetos
- Alocados na pilha
- Possuem mesmo tamanho e características, independentemente de plataforma
- 8 tipos:
  - boolean, char
  - byte, short, int, long
  - float, double

Tipos Inteiros

Tipo	Tamanho	Alcance
Byte	8 bits	-128 até 127
short	16 bits	-32.768 até 32.767
int	32 bits	-2.147.483.648 até 2.147.483.647
long	64 bits	-9223372036854775808 até 9223372036854775807

- Todos tipos possuem sinal
- Cuidado com valores atribuídos

Tipos Inteiros - Exemplo

```
// precisaoByte.java
public class precisaoByte {

 public static void main(String[] args) {
  byte i = 127;
  System.out.println("i: " + i);
  i++;
  System.out.println("i: " + i);
 }
}
```

Saída:

i: 127

i: -128

Tipos Ponto Flutuante

Tipo	Tamanho	Alcance
float	4 bytes	aprox.±3.402823 E+38F
double	8 bytes	aprox.±1.79769313486231 E+308

- Seguem padrão IEEE754
- Notação Científica

  `1.44E6 (= 1.44 x 10^6 = 1,440,000) ou`

  `3.4254e-2 (= 3.4254 x 10^-2 = 0.034254)`
- Precisão
  - float: até 7 casas decimais
  - double: até 15 casas decimais

Tipos Ponto Flutuante - Exemplo

```
public class precisaoFloat {

 public static void main(String[] args) {
  double d = 1.230000875132520000092785E+2;
  System.out.println(d);
  float f = (float)d;
  System.out.println(f);
 }
}
```

Saída:

```
1.23000087513252
1.23000084
```

Tipo Caractere

- Representação de Caracteres Individuais
- Tamanho de 16 bits
- Tabela Unicode
  - código numérico sem sinal (até 32768 caracteres)
  - Internacionalização
  - compatível com a tabela ASCII

Tipo Caractere

- valor literal de um caractere
  - limitado por aspas simples: ‘a’
- cuidado com uso de escapes

```
String s = "\"";
String s1 = "\u0022";          // """
String s2 = "\u005c\u0022"     // "\""
```

- caracteres especiais

Representação	**Significado**
\n	Pula linha *(newline* ou *linefee*d)
\r	Retorno de carro *(carriage retur*n)
\b	Retrocesso *(backspac*e)
\t	Tabulação *(horizontal tabulatio*n)
\f	Nova página *(formfee*d)
\’	Apóstrofe
\”	Aspas
\\	Barra invertida

Tipo Caractere - Exemplo I

```
public class charTeste {

 public static void main(String[] args) {
  char i = 'a';
  System.out.println("i: " + i);
  i++;
  System.out.println("i: " + i);
  i++;
  System.out.println("i: " + i);
  i = (char)(i * 1.2);
  System.out.println("i: " + i);
  int j = i;
  System.out.println("j: " + j);
 }
}
```

Saída:

```
i: a
i: b
i: c
i: v
i: 118
```

Tipo Caractere - Exemplo II

```
public class charTeste1 {

 public static void main(String[]   args) {
  char i = 'a';
  int j =  i;
  System.out.println("i: " + i);
  System.out.println("j: " + j);
  i = 97;
  j = i;
  System.out.println("i: " + i);
  System.out.println("j: " + j);
  i = '\u0061';
  j = i;
  System.out.println("i: " + i);
  System.out.println("j: " + j);
  i = '\141';
  j = i;
 }
}
```

Tipo Booleano

- Valores
  - true
  - false
- Condições devem ser tipos booleanos
- Não há equivalência com inteiros

Tipo Booleano - Exemplo

```
public class testeInvalido {

 public static void main(String[]  args) {
  int boolInt = 1;
  boolean bool = true;

  if (bool) System.out.println(bool); // imprime true
  if (boolInt) // ERRO: Só valores booleanos!
  {
      System.out.println("Olá Mundo!");
  }
  while (i = 5) // ERRO: só expressões booleanas!
  {
      System.out.println("Isto é um teste")
  }
 }
}
```

Declaração de Variáveis

- Identificadores
  - caracteres alfabéticos e numéricos
  - iniciados por letra ou _ ou $
  - podem ser de qualquer tamanho
  - somente diferencia os 32 primeiros caracteres
  - diferencia letras maiúsculas e minúsculas

  Exemplos:

a	total	x2	$mine
_especial	TOT	Total	ExpData
1x	Total geral	numero-minimo	void

Palavras Reservadas

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

Declaração de Variáveis

- Formato
  - Tipo seguido por lista de identificadores
- Três tipos
  - instância, classe e local
- Inicialização
  - pode ser feita na declaração
  - necessário inicializar variável local antes de usar valor
  - variáveis de classe e instância são inicializadas automaticamente

Declaração de Variáveis

- Pode ser feita em qualquer ponto do programa
- Visibilidade
  - a partir de onde foi declarada
  - visível nos blocos internos
    - blocos delimitados por chaves: { e }
  - não pode haver variáveis com mesmo nome no mesmo escopo

Variáveis - Exemplo I

```
public class variavel {

 public static int c = 10; // variável de classe
 public int i;  // variável de instância

 public int func()
  {
   int n = 5;  // variável local
   c++;
   i = 2*n*c;
   int z;
   if (i < 50) z = 10;
   z *= 10;
   return (i + c);
  }
}
```

Variáveis - Exemplo II

```
public class  testaVisibilidade {

 public static void main(String[] args) {
  int x = 10;  // definição da variável x
   int y = 100; // definição da variável y
   while (x < 100)
     {
      int z = 0; // válido: z visível apenas neste bloco
      int y = x; // inválido: redefinição da variável y
      x++; // válido: x também é visível neste bloco
     }
  }
}
```

Conversões entre Tipos Numéricos

- Pode misturar tipos
  - Se algum dos operandos for do tipo double, então o outro operando será convertido em um double
  - Caso contrário, se algum dos operandos for do tipo float, o outro operando será convertido em um float
  - Caso contrário, se algum dos operandos for do tipo long, o outro operando será convertido em um long
  - forma análoga para os tipos inteiros: int, short e byte

Conversões entre Tipos Numéricos

- Conversões nas quais pode haver perda de informação devem ser feitas explicitamente através do operador de cast
- Promoção
  - Ocorre quando são feitas operações com tipos inteiros (byte, short e char) menores que int
  - Resultados das operações são do tipo int
  - Racional: maoir probabilidade de ocorrência de overflow nestes tipos, uma vez que o intervalo de valores é pequeno

Conversões Numéricas - Exemplo I

```
public class testaConversoes {
 static public void main (String[] args)
 {
  char c = 'a';
  System.out.println("c: " + c);
  int i = c;   // conversão explícita desnecessária
  System.out.println("i: " + i);
  float f = c; // conversão explícita desnecessária
  System.out.println("f: " + f);
  int j = 101;
  System.out.println("j: " + j);
  char d = (char)j; // conversão explícita obrigatória
  System.out.println("d: " + d);
  boolean b = true;
  int k = (int)b; // ERRO: Os tipos são incompatíveis
 }
}
```

Conversões Numéricas - Exemplo II

```
public class testaConversoesI {
 static public void main (String[] args)
 {
    double x = 9.997;
    int nx = (int)x;
    System.out.println(" nx: " + nx);
    nx = (int)Math.round(x);
    System.out.println("nx: " + nx);
 }
}
```

Promoção

```
void  testeShort(short x, short y) {
    x = (short)(x * y);  // conversão obrigatória
    x = (short)(x / y);  // conversão obrigatória
    x = (short)(x % y);  // conversão obrigatória
    x = (short)(x + y);  // conversão obrigatória
    x = (short)(x - y);  // conversão obrigatória
    x++;  // conversão desnecessária
    x--;  // conversão desnecessária
}
```

Variáveis Final ou Constantes

```
public class Constantes {
 public static final double G = 9.81; // constante
 static public void main (String [] args)
 {
  final double CM_POR_POLEGADA = 2.54; // constante
  double larguraPalel = 8.5;
  double alturaPapel = 11;
  System.out.println(G + " metros por segundo " +
    "ao quadrado");
  System.out.println("Tamanho em centimetros: " +
    (larguraPalel * CM_POR_POLEGADA) + " por " +
    (alturaPapel * CM_POR_POLEGADA));
 }
}
```

Comentários

- Três tipos
  - resto de linha //
  - múltiplas linhas /* */
  - geração de documentação /** */
    - uso de javadoc
    - gera páginas html no padrão fornecido no JDK
    - comentários imediatamente antes do elemento
    - uso de tags html
      - formatação do texto, imagens, links, tabelas
    - tags especiais javadoc @
      - links para outros elementos e formatação padronizada

Comentários

```
/** Comentario de classe
* Classe destinada ao armazenamento de arquivos
* ou diretórios.
* <p> Pode ser usada para armazenar árvores de
* diretórios.
* @author Carlos da Silva
* @see java.io.File
*/
public class FileData {
 /** Comentario de atributo
 * Essa variavel e inutil
 */
 public int i = 10;     // inutil
 /** Comentario de metodo
 * Construtor
 * @param filename nome do arquivo
 */
 public FileData(String filename) {
 /* comentario de multiplas linhas
 nao farao parte da documentacao gerada pelo
 javadoc */
 }
}
```

Comentários

- Geração de documentação

```
javadoc FileData.java
```

  - Espaços e *s iniciais são descartados
  - Tags javadoc
    - @see
    - @author
    - @version
    - @param
    - @return
    - @exception
    - @deprecated

Operadores Aritméticos

Operador	Significado	Exemplo
+	Adição	a + b
-	Subtração	a- b
*	Multiplicação	a * b
/	Divisão	a / b
%	Resto da divisão inteira	a % b
-	Sinal negativo (unário)	- a
+	Sinal positivo (+ unário)	+a
++	Incremento unário	++a ou a++
--	Decremento unário	-- aou a-

Operadores Aritméticos

```
public class Aritmetica {
  public static void main  ( String[] args)
  {
    short x = 6;
    int y = 4;
    float a = 12.5;
    float b = 7;

    System.out.println ( "x é " + x + ", y é " + y );
    System.out.println ( "x + y = " + (x + y) );
    System.out.println ( "x - y = " + (x - y) );
    System.out.println ( "x / y = " + (x / y) );
    System.out.println ( "x % y = " + ( x % y ) );
    System.out.println ( "a é " + a + ", b é " + b );
    System.out.println ( " a / b = " + ( a / b ) );
    a = x / y;
    System.out.println ( " a = " + a );
  }
}
```

Operadores Relacionais

```
public class Relacional {
 static public void main (String[] args)
 {
  int a = 15;
  int b = 12;

  System.out.println("a = " + a);
  System.out.println("b = " + b);

  System.out.println("a == b -> " + (a == b));
  System.out.println("a != b -> " + (a != b));
  System.out.println("a < b -> " + (a < b));
  System.out.println("a > b -> " + (a > b));
  System.out.println("a <= b -> " + (a <= b));
  System.out.println("a >= b -> " + (a >= b));
  System.out.println("a = b -> " + (a = b));
 }
}
```

Operadores Lógicos

```
public class Logico {
 static public void main (String[] args)
 {
  System.out.println("True & True = " + (true & true) );
  System.out.println("True && True = " + (true && true) );
  System.out.println("True & False = " + (true & false) );
  System.out.println("True && False = " +(true && false) );
  System.out.println("True | True = " + (true | true) );
  System.out.println("True || True = " + (true || true) );
  System.out.println("True | False = " + (true & false) );
  System.out.println("True || False = " + (true & false) );
  System.out.println("Not True = " + (!true) );
  System.out.println("Not False = " + (!false) );
 }
}
```

Operadores de Atribuição

- Básico

```
int x = 5;        // inicializacao
x = x + 2;        // atribuicao
x = y = z = 7; // encadeada
```

- Compostos

Expressão	Significado
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`

Operadores de Atribuição

- Incremento

```
int x, y, z;
x = 42;
y = x++;
z = ++x;
x = ++z + z;
```

- Decremento

```
int x, y, z;
x = 42;
y = x--;
z = --x;
x = z-- + z;
```

Operador Ternário

- Formato

  <expressão_booleana> ? <valor_1> : <valor_2>

- Exemplo

```
public class ternario {
 static public void main (String[] args)
 {
  int i = 7;
  int j = i < 10 ? (2*i) : (i);  // faz j = 2*i
   System.out.println("i: " + i); // "i: 7"
   System.out.println("j: " + j); // "j: 14"
 }
}
```

Avaliação de Expressões

- Precedência de Operadores
  - tabela de precedências

```
c = a + b * c;
```

- Associatividade de Operadores
  - esquerda para direita, exceto atribuição

```
c = a * b * c;
c = a = b = 10;
```

- Associatividade de Operandos
  - esquerda para direita

```
c = b++ + b;
```

Precedência de Operadores

Nível	Operadores
1	`.(seleção) [] ()`
2	`++ -- ~ instanceof new -(unário)`
3	`* / %`
4	`+ -`
5	`<< >> >>>`
6	`< > <= >=`
7	`== !=`
8	`&`
9	`^`
10	`|`
11	`&&`
12	`||`
13	`?:`
14	`= += -= *= /=`
15	`,`

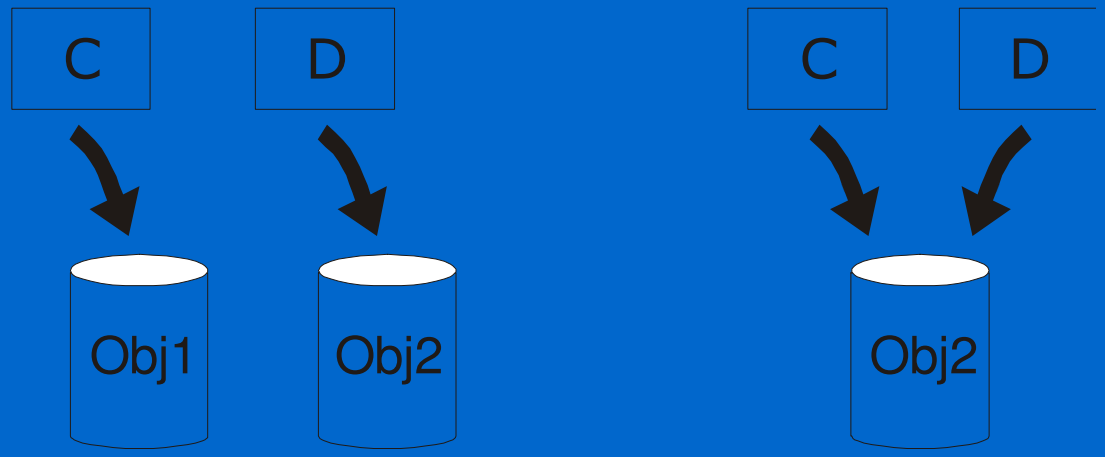
Tipos Compostos

- Objetos
  - único tipo composto
  - vetores, strings, matrizes, registros e arquivos são objetos
  - alocados no monte
  - variáveis primitivas possuem tipo, objetos possuem classe
- Exemplo

```
class coordenadas {
        public int x;
        public int y;
        public int z;
}
```

Tipos Compostos

- Tipos Primitivos
  - alocação na pilha
  - atribuição por cópia
- Objetos
  - alocação no monte
  - atribuição por referência



Atribuição de Tipos Compostos

```
class Numero {
 int i;
}
public class  Atribuicao {
 public static void main(String[]   args) {
  Numero n1 = new Numero();
  Numero n2 = new Numero();
  n1.i = 9;
  n2.i = 47;
  System.out.println("1: n1.i: " + n1.i +", n2.i: " +
                     n2.i);
  n1 = n2;
  System.out.println("2: n1.i: " + n1.i +", n2.i: " +
                     n2.i);
  n1.i = 27;
  System.out.println("3: n1.i: " + n1.i +", n2.i: " +
                     n2.i);
 }
}
```

Objetos Especiais - Strings

```
public class testaString {
  public static void main (String[] args) {
     int nascido = 1965;
     String s = "Fernando ";
     s += "Diniz";
     System.out.println (s);
     s = "Flavio";
     s += " Miguel Varejao";
     System.out.println (s);
     s = s + " nasceu em " + nascido + " e tem " +
        (2001 - nascido) + " anos";
     System.out.println (s);
  }
}
```

Estruturas de Controle e Programação Básica

- Repetições
  - Definidas
  - Condicionais
- Desvios
  - Condicionais
  - Incondicionais
- Modularização
- Entrada e Saída via Console

Comandos ou Diretivas

- Sequenciais
- Separadas por ponto e vírgula (;)
- Bloco { } permite agrupar diretivas

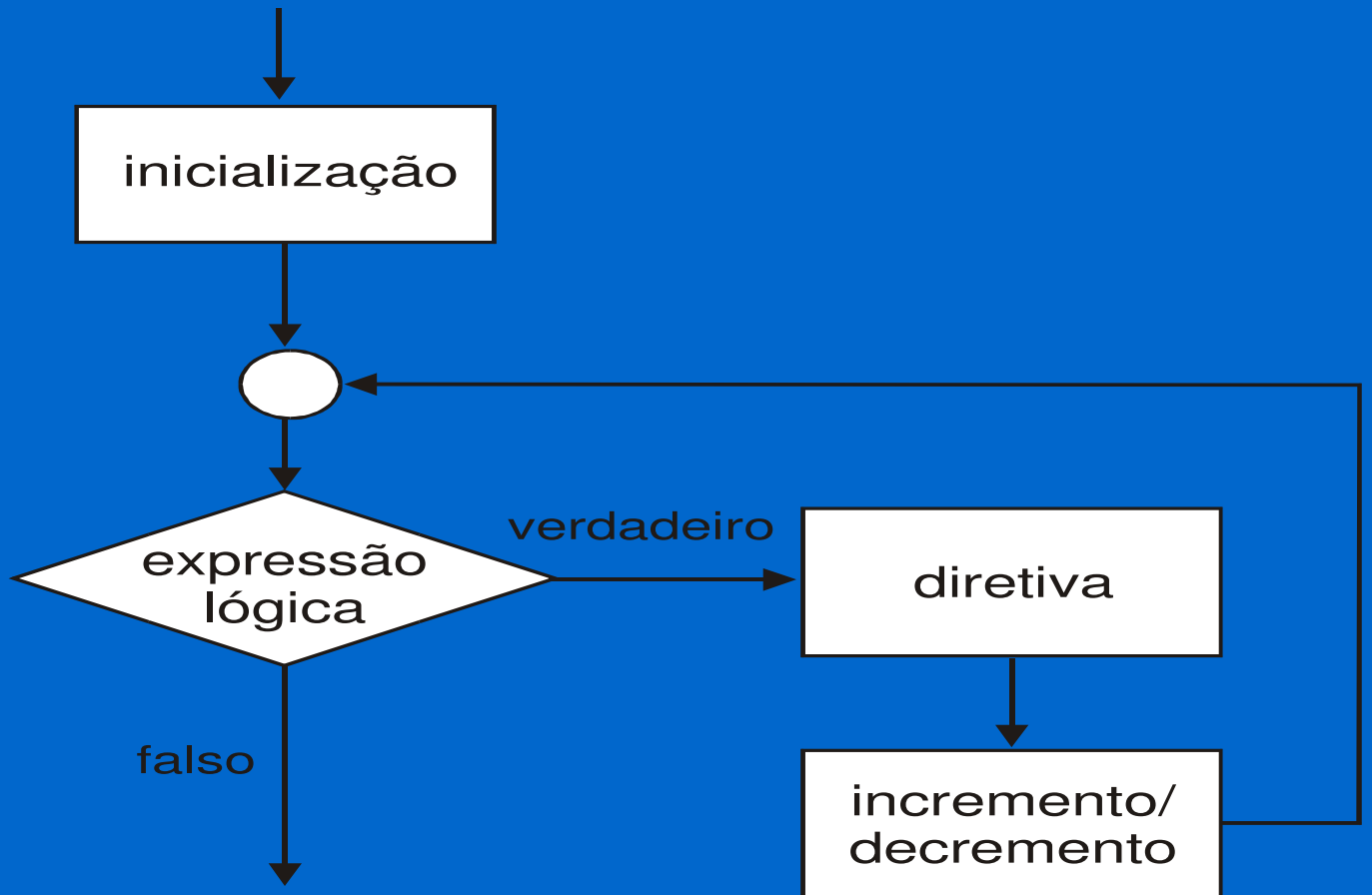
for

- Formato

```
for (inicialização; condição de execução;
  incremento/decremento) diretiva;
```

- Todos campos opcionais
- Super comando de repetição
  - definida
  - condicional
- Múltiplas variáveis de controle

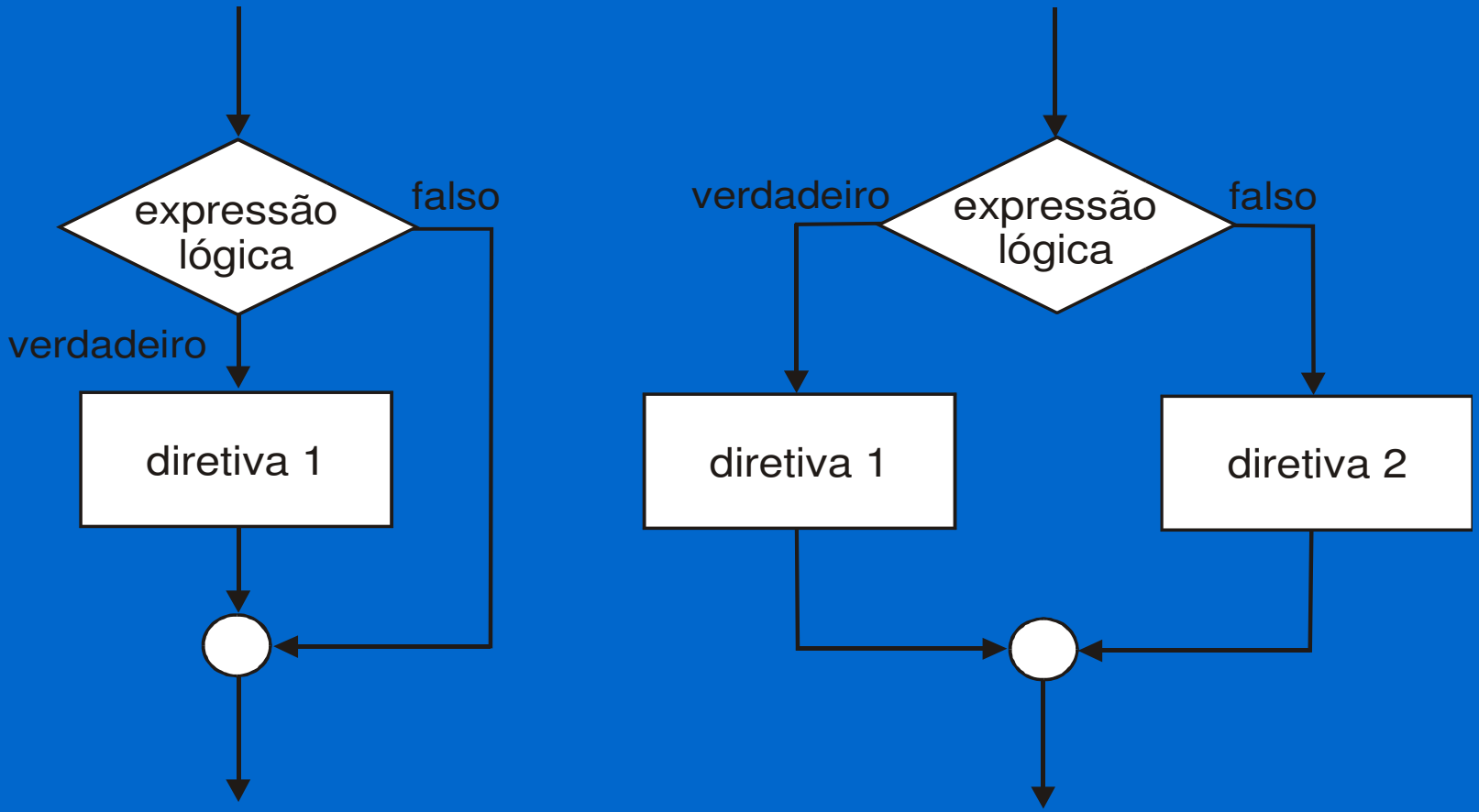
for - Estrutura Geral



for

```
public class exemploFor {
 public static void main (String[] args) {
    int j;
    for (j=0; j<10; j++)
    {
     System.out.println(“” + j);
    }
    for(int i = 1, j = i + 10; i < 5; i++, j = i * 2)
     {
      System.out.println("i= " + i + " j= " + j);
     }
 }
}
```

if - Estrutura Geral



if

```
public class exemploIf {
public static void main (String[] args) {
   if (args.length > 0)
   {
    for (int j=0; j<Integer.parseInt(args[0]); j++)
        {
         System.out.print("" + j + " ");
        }
    System.out.println("\nFim da Contagem");
   }
   System.out.println("Fim do Programa");
 }
}
```

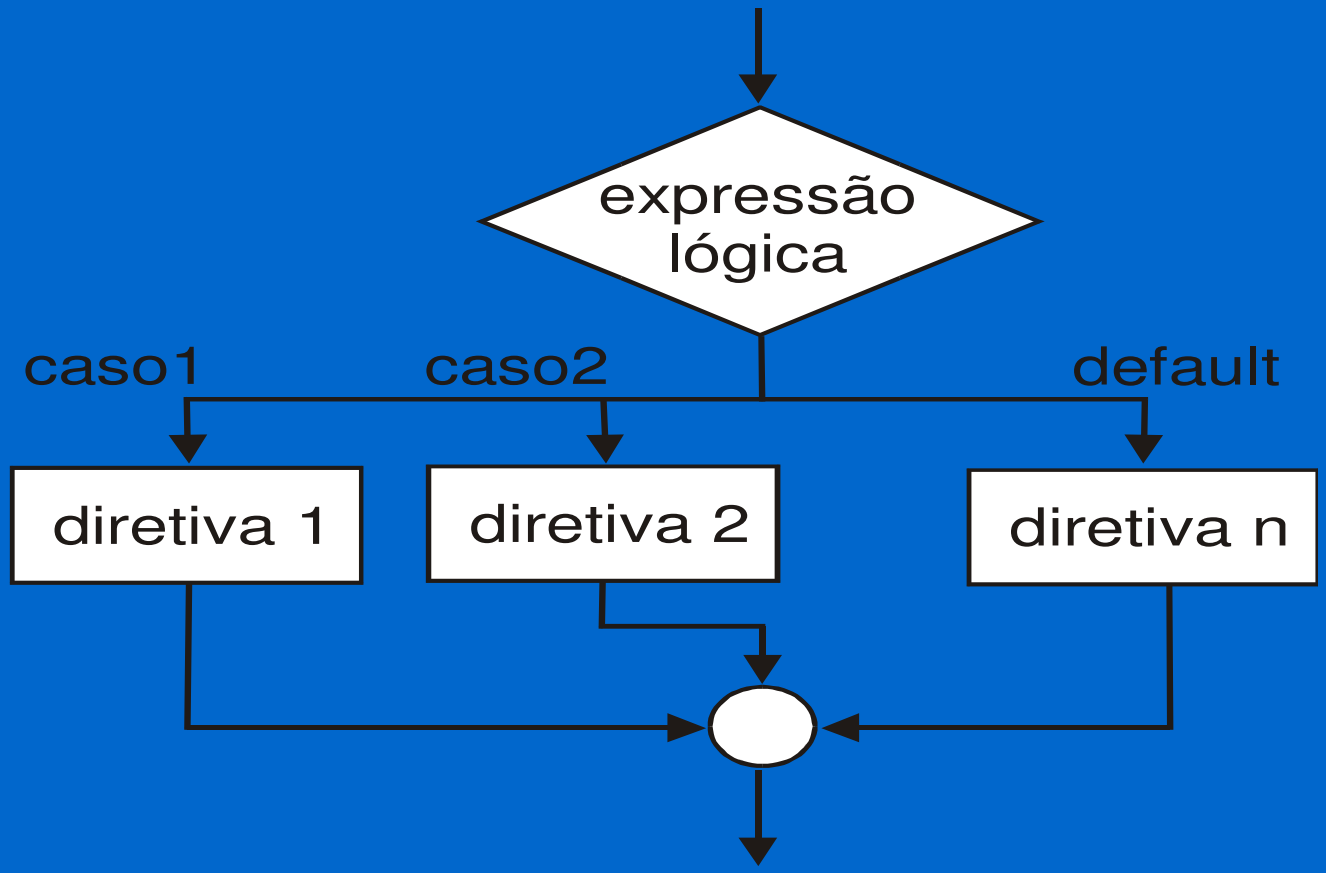
switch

- Formato

```
switch (expressão_ordinal){
 case ordinal1: diretiva3;
  break;
 case ordinal2: diretiva2;
  break;
 default: diretiva_default;
}
```

- uso de break
- default opcional

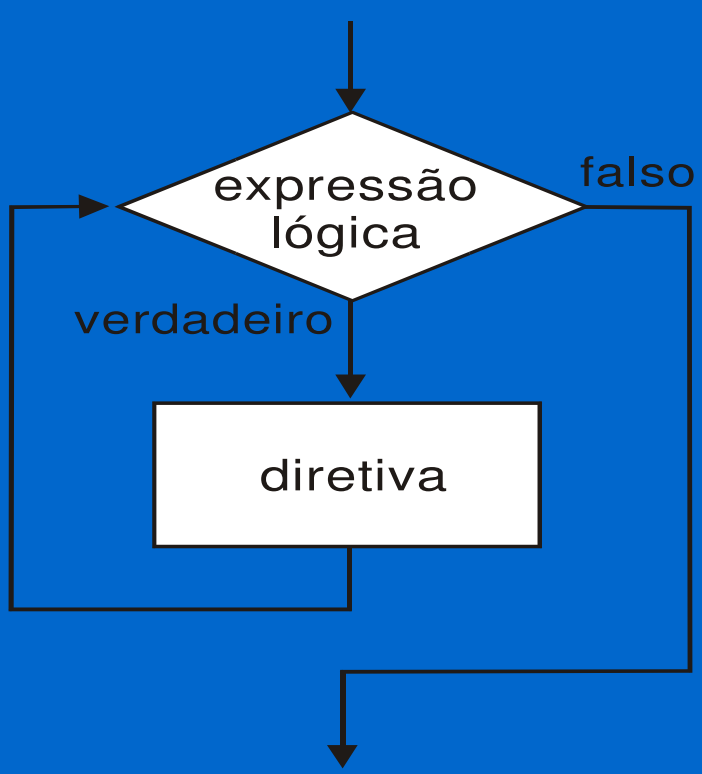
switch


expressão
lógica
caso1
caso2
default
diretiva 1
diretiva 2
diretiva n

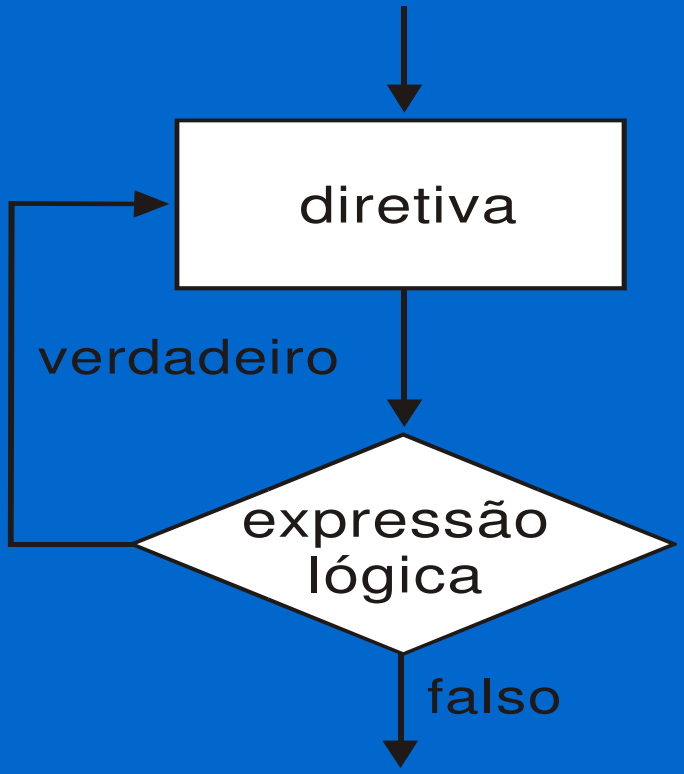
switch

```
public class   exemploSwitch {
public static void main (String[]      args) {
 if ( args.length > 0){
     switch( args[0]. charAt(0)) {
       case 'a':
       case 'A':   System.out.println("   Vogal A");
       break;
       case 'e':
       case 'E':   System.out.println("   Vogal E");
       break;
       case ' i':
       case 'I':   System.out.println("   Vogal I");
       break;
       default: System.out.println("Não é uma vogal");
      }
   } else {
    System.out.println("Não foi fornecido argumento");
   }
 }
}
```

while e do-while



while



do-while

while

```
public class    exemploWhile {
public static void main (String          args[]) {
 int j = 10;
 while (j >    Integer.parseInt(    args[0]))
       {
        System.out.println    (""+j);
        j--;
       }
 }
}
```

do-while

```
public class  exemploDoWhile {
public static void main (String[] args) {
 int min =  Integer.parseInt( args[0]);
 int max =  Integer.parseInt( args[1]);
 do {
      System.out.println ("" + min + " < " + max);
      min++; max--;
    } while (min < max);
 System.out.println(""+ min +"<"+ max +"condicao
invalida");
 }
}
```

Desvios

```
public class      BreakContinue {
   public static void main(String[]              args) {
      for(  int   i = 0; i < 100; i++) {
         if(i == 74) break; // saída do loop
         if(i % 9 != 0) continue; // próxima iteração
         System.out.println(i);
       }
     int i = 0;
     // Um Loop infinito
      for(;;) {
       i++;
       if(j == 9621) break; // saida 1 do loop
       int j = i * 27;
       if(j == 1269) break; // saida 2 do loop
       if(i % 10 != 0) continue; // Topo do loop
       System.out.println(i);
      }
   }
}
```

Desvios Rotulados

```
public class breakRotulado {
  public static void main(String[] args) {
    int i = 0;
    externo:
    for(; true ;) { // laço infinito
      interno:
      for(; i < 10; i++) {
        System.out.println("i = " + i);
        if(i == 2) {
          prt("continue");
          continue;
        }
        if(i == 3) {
          System.out.println("break");
          i++;
          break;
        }
```

Desvios Rotulados

```
if(i == 7) {
  System.out.println ("continue externo");
  i++;
   continue externo;
}
 if(i == 8) {
  System.out.println ("break externo");
   break externo;
 }
```

Desvios Rotulados

```
        for(int k = 0; k < 5; k++) {
          if(k == 3) {
            System.out.println ("continue interno");
            continue interno;
            }
          }
        }
      }
      // não é possível usar break ou continue
      // para os rótulos neste ponto do programa
  }
  static void prt(String s) {
    System.out.println(s);
  }
}
```

Saída no Console

- Associada ao monitor
- Uso de stream out de java.lang.System
- Stream de saída no console é aberta automaticamente
- out é da classe java.io.PrintStream
  - print (char)
  - println (char)
  - print (int)
  - println (int)

Saída no Console no J2SE 5.0

- Método printf em PrintStream
  - Semelhante a printf de C

```
System.out.printf("contagem de nome%n");
System.out.printf("%s %5d%n", nome, total);
```

  - \n X %n
  - Especificadores de Formato

%[indice_argumento$][flags][tamanho][.precisao]conversao

```
System.out.printf ("inteiro: %2$5.2d%n float: %1$5.2f%n", x, i);
```

Saída no Console no J2SE 5.0

Conversão	Tipo do Argumento	Descrição
'b', 'B'	Boolean ou boolean	Imprime true ou false
's', 'S'	Geral	Imprime como string
'c', 'C'	caractere	O resultado é um caractere Unicode
'd'	inteiro	O resultado é formatado com um inteiro decimal
'e', 'E'	Ponto flutuante	O resultado é formatado como um número decimal em notação científica
'f'	Ponto flutuante	O resultado é formatado como um número decimal
'g', 'G'	Ponto flutuante	O resultado é formatado como um número decimal em notação científica ou como um número decimal
'%'	Caractere percentagem	O resultado é o caractere '%'
'n'	Separador de linha	O resultado é o separador de linha específico da plataforma

Entrada pelo Console

- Associada ao teclado
- Uso de stream in de java.lang.System
- Stream de entrada no console é aberta automaticamente
- Métodos oferecidos são pouco úteis para entrada via Console
- Classe Console para ajudar

A Classe Console

Método	**Descrição**
`readBoolean()`	Lê um valor *boolean* da entrada padrão.
`readByte()`	Lê um valor *byte* da entrada padrão.
`readShort()`	Lê um valor *short* da entrada padrão.
`readInteger()`	Lê um valor *int* da entrada padrão.
`readLong()`	Lê um valor *long* da entrada padrão.
`readFloat()`	Lê um valor *float* da entrada padrão.
`readDouble()`	Lê um valor *double* da entrada padrão.
`readChar()`	Lê um valor *char* da entrada padrão.
`readString()`	Lê uma *string* da entrada padrão.

A Classe Console

```
public static int readInteger () {
      try {
         BufferedReader br = new BufferedReader (
                new InputStreamReader (System.in) );
         String s = br.readLine ();
         return Integer.parseInt (s);
      } catch (IOException e) {
         return 0;
      } catch (NumberFormatException e) {
         return 0;
      }
  }
```

Entrada pelo Console no J2SE 5.0

- Uso da classe java.util.Scanner
- Métodos next e hasNext para cada tipo e para linhas
- Separador default é espaço em branco

```
Scanner sc = new Scanner(System.in);
while (sc.hasNextLong()) {
    long aLong = sc.nextLong();
}
```