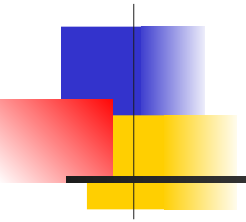
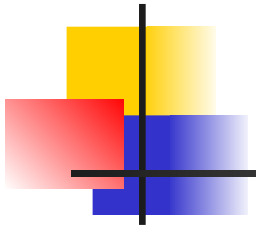
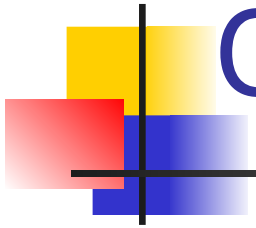
# Programação Orientada a Objetos





# Parte 1
## Orientação a objetos

# O que é Orientação a Objetos?



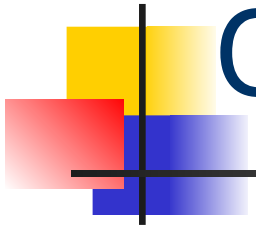
- Um paradigma de programação que permite representar programaticamente elementos do espaço de problemas.

# Contextualizando o Problema

O que leva um programador a mudar do paradigma procedimental para um novo?

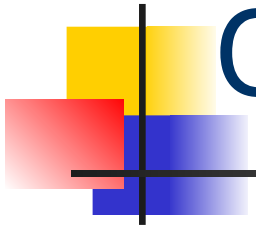
- Complexidade crescente dos sistemas
- Limitações da capacidade humana de compreensão de um sistema como um todo.

# Paradigma de Orientação a Objetos



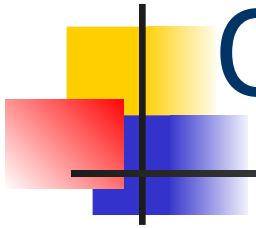
- O paradigma de Orientação a Objetos tem várias características marcantes:
  - Alta reutilização de software,
  - Desenvolvimento sempre evolutivo,
  - Construções complexas a partir de construções simples,
  - Manutenção tende a ser simples.

# Por que usar Orientação a Objetos?



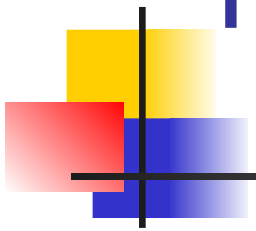
A orientação a objetos, quando bem utilizada, oferece ganhos em termos de rapidez, custo, confiabilidade, flexibilidade e facilidade de manutenção.

# Por que 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

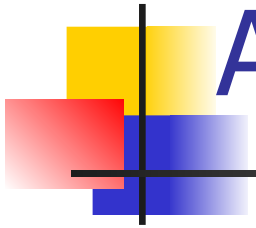
# Paradigma Único



- A nomenclatura e os métodos utilizados na produção de um software orientado a objetos permanecem praticamente os mesmos.

- Programadores, analistas, projetistas, usuários, etc. comunicam-se utilizando praticamente a mesma linguagem.

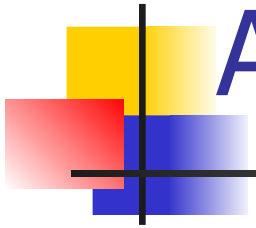
# Facilidade na Reutilização da Arquitetura e do Software



- O software produzido é encapsulado em unidades básicas denominadas classes.

- Essas classes possuem semânticas que podem ser reaproveitadas em novos softwares.

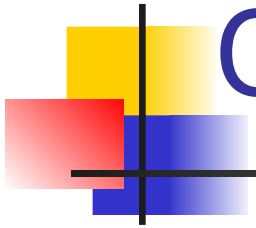
# Facilidade na Reutilização da Arquitetura e do Software



**Exemplo:** Uma classe denominada *Funcionário* poderia ser utilizada

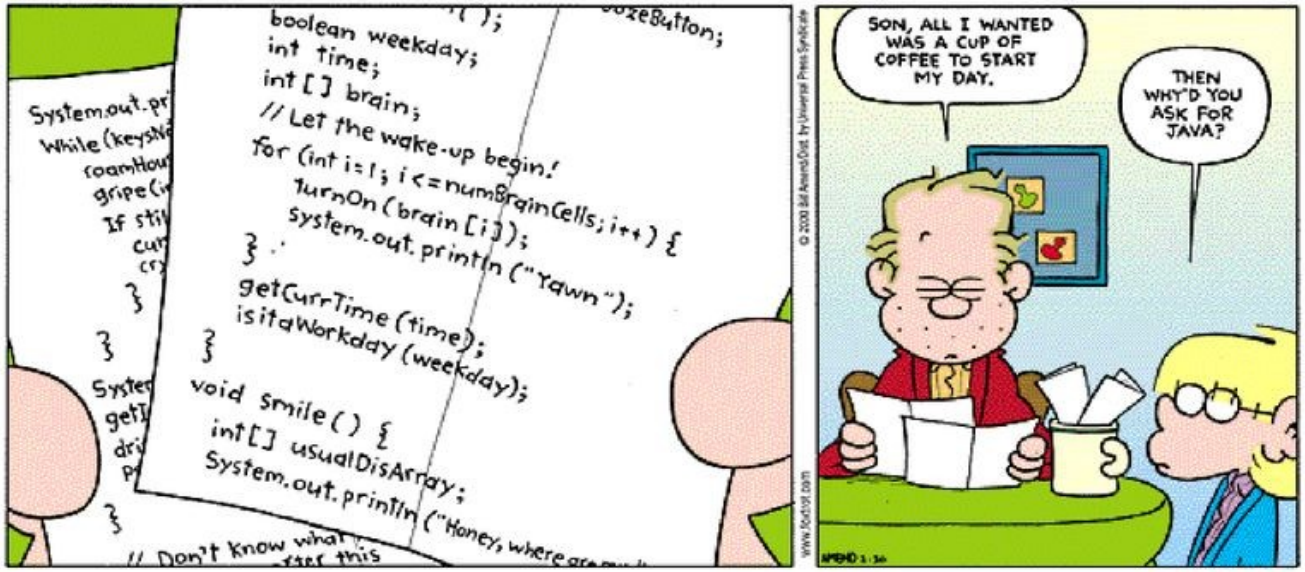
- em um controle de departamento pessoal
- em um software de ponto eletrônico.

# Conceito de Orientação a Objetos

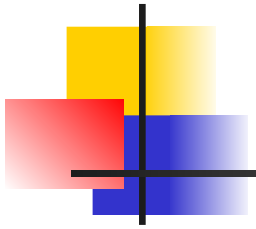


- No mundo real tudo é objeto
- Os objetos se relacionam entre si de diversas maneiras.

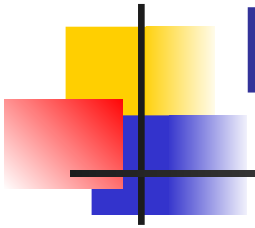
# Parte 2


boolean weekday;
int time;
int[] brain;
// Let the wake-up begin!
for (int i=1; i<=numBrainCells; i++) {
turnOn(brain[i]);
System.out.println("Yawn");
}
getCurrTime(time);
isItaWorkday(weekday);
}
void smile() {
int[] usualDisArray;
System.out.println("Honey, where
SON, ALL I WANTED WAS A CUP OF COFFEE TO START MY DAY.
THEN WHY'D YOU ASK FOR JAVA?

# Introdução



- Motivações para o Uso de Java
  - Propósito geral
  - Orientada a objetos
  - Baseada na linguagem C
    - Importante no início
  - Segura



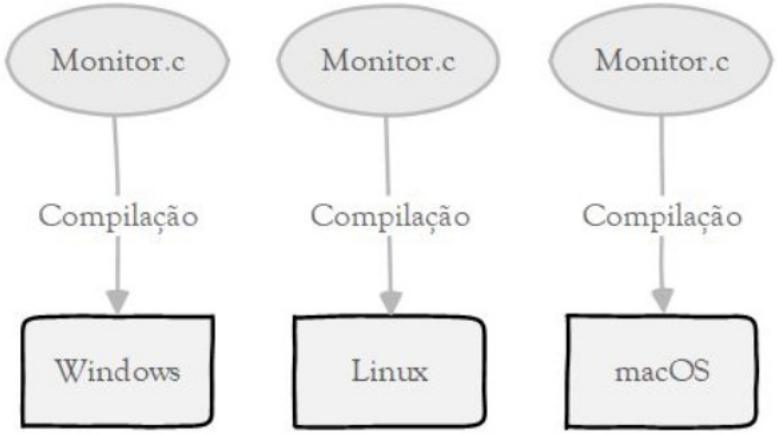
# Linguagem Java

- É uma linguagem de programação de propósito geral, orientada a objetos e adequada para a Internet.
  - *Utilizada para o desenvolvimento de qualquer tipo de software, com sintaxe similar ao C.*
  - *Modelo OO baseado em Smalltalk e Simula67*
  - *Applets JAVA*
- A linguagem JAVA foi projetada e implementada por um pequeno grupo pessoas, coordenado por **James Gosling**, na **Sun Microsystems** em Mountain View, Califórnia, em 1991.

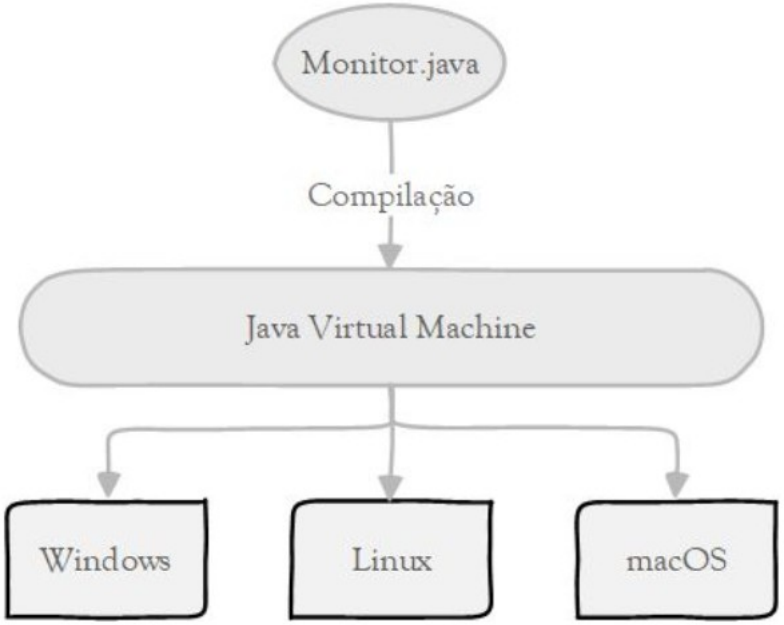
# Introdução

- Write once, run anywhere → JVM

C:

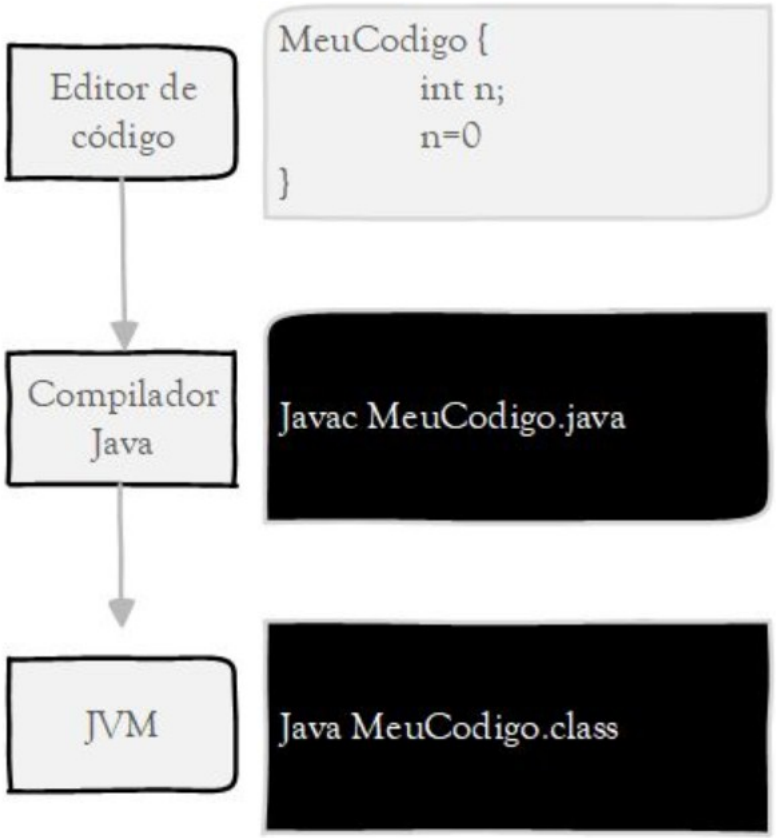


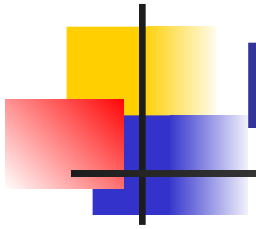

Java:



# JVM

- Combina compilação e interpretação





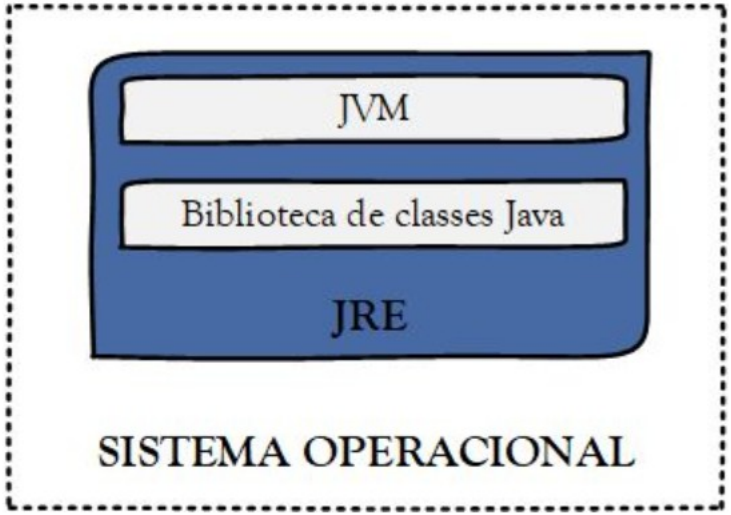
# Plataforma Java

JVM – plataforma virtual residente em RAM que executa/interpreta/traduz Bytecode em código de máquina.
Possui:

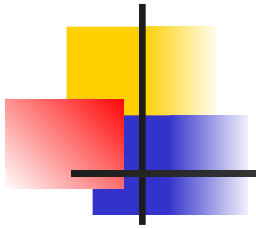
- *classloader*, que carrega o arquivo .class na RAM
- *Verificador de bytecode*, para ver se há violações de restrição de acesso em seu código.
- *Execution Engine* converte o Bytecode em código de máquina executável
- JIT - Compila uma seqüência de instruções bytecode em tempo de execução antes de executá-la nativamente.

# A Plataforma Java: JRE

- JRE: esse é o programa que você instala
  - ↳JVM
  - ↳Bibliotecas



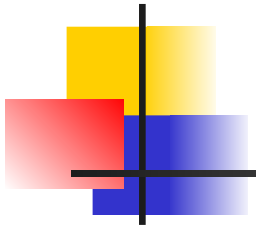
# A Plataforma Java: JDK



- Compilador (javac)
- Interpretador (java)
- Visualizador de applets (appletviewer)
- Bibliotecas (packages)
- Gerador de documentação (javadoc)
- Depurador de programas (jdb)
- Ambiente de execução (jre)

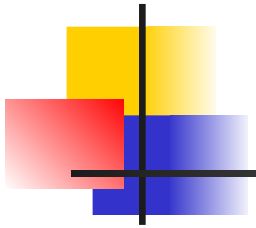
# Plataforma Java

# Características de Java



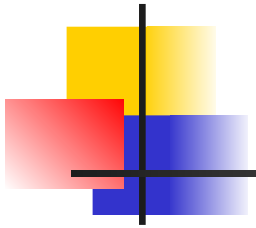
- Orientada a Objetos
  - com exceção dos tipos primitivos, tudo são classes e objetos
  - não existe o conceito de funções ou procedimentos independentes
  - reuso de código através de bibliotecas de classes
  - Versões mais novas permitem programação funcional

# Características de Java



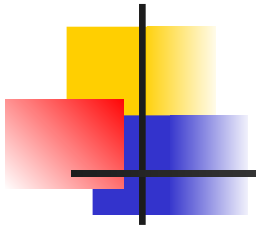
- Dinâmica
  - classes são carregadas sob demanda
- Distribuída
  - suporte através de bibliotecas de alto nível
  - invocação de métodos de objetos remotos
  - *Applets e Java Web Start*
    - programas especiais carregados e executados através de *Web browsers*
    - Descontinuados (Java 9 e 11, respectivamente)

# Características de Java



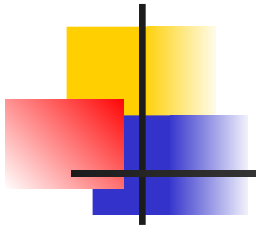
- Segura
  - Várias camadas de controle de segurança contra código malicioso
    - não permite ao programador manipular endereços
    - interpretador Java realiza verificação de bytecodes (se não contém código ilegal) sempre que código não confiável é carregado
    - uso de sandbox (acesso limitado aos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

# Características de Java



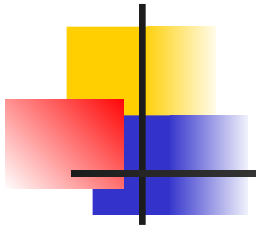
- Eficiente
  - Embora interpretada
  - App java tende a ser mais lenta que equivalente em C (mas mais rápido que a maioria das outras linguagens)
  - Interpretadores Java podem incluir compiladores de bytecodes JIT
  - Pode-se escrever partes do programa em C/C+

# Características de Java



- Suporte à Concorrência
  - rotinas múltiplas concorrentes (threads)
  - elementos para sincronização

# Java é Orientada a Objetos



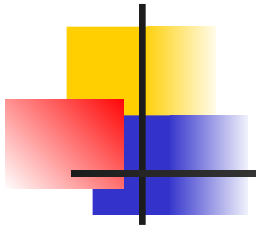
- Objetos e Classes
- Encapsulamento de dados e operações
- Subtipos e Herança
- Polimorfismo
- Ligações dinâmicas (*dynamic binding*)
- Criação e remoção dinâmica de objetos
- alocação e liberação automática de memória

# Primeiro Exemplo

```
public class HelloWorld {
      public static void main(String[] args) {
        System.out.print("Hello, World");
      }
    }
```

- Aplicação
- Salvar como arquivo HelloWorld.java

# Primeiro Exemplo



- Compilar com *javac*

```
javac HelloWorld.java
```

- Executar com *java*

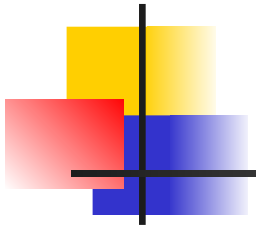
```
java HelloWorld
```

# Outro Exemplo (passando parâmetros)

```
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



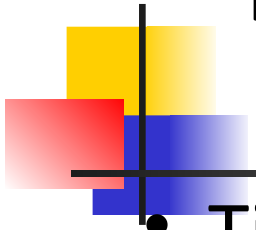
- Compilar com javac

```
javac Eco.java
```

- Executar com java

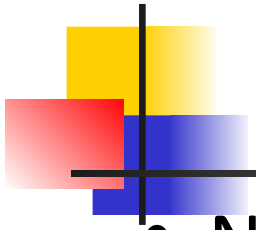
```
java Eco
java Eco Testando 1 2 3
```

# 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

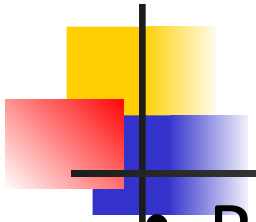
```
public class precisaoFloat {

 public static void main(String[] args) {
  double d = 1.2300008751325200092785E+2;
  System.out.println(d);
  float f = (float)d;
  System.out.println(f);
 }
}
```

*Saída:*

```
123.000087513252
123.000084
```

# 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

- Tabela Unicode
  - compatível com a tabela ASCII
  - valor literal de um caractere
    - limitado por aspas simples: ‘a’
  - caracteres especiais e sem representação visual
    - precedidos por barra invertida: ‘\n’, ‘\’’
  - uso de escapes

    \u0061  \141  ‘a’
  - uso como identificador (só escape \uxxxx)

```
int b\u0061 = 10;  // int ba = 10;
```

# Tipo Caractere

## Tabela Unicode

- 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 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: Java não é C!
  {
      System.out.println("Olá Mundo!");
  }
  while (boolInt = 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, segundo tabela a seguir*:

*No entanto, não é boa prática de programação contar com esses valores default

# Inicialização de Variáveis: Valores default

| Data Type | Default Value (for fields) |
|---|---|
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

# 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

# Literais

- Um literal é a representação do código-fonte de um valor fixo;
- literais são representados diretamente no seu código sem requerer computação.
- Como mostrado abaixo, é possível atribuir um literal a uma variável de um tipo primitivo:
  - boolean result = true;
  - char capitalC = 'C';
  - byte b = 100;
  - short s = 10000;
  - int i = 100000;
  - double d1 = 123.4;
  - float f1 = 123.4f;

# 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

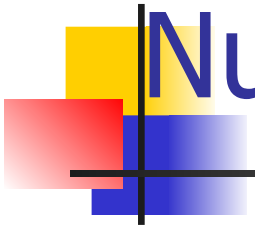
```
public class  testaVisibilidade {

 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

- Java permite misturar tipos numéricos em uma mesma expressão:
  - 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* onde 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, mas o resultado das operações é do tipo int

# Conversões Numéricas - Exemplo I

```
public class TestaConversoes {
 static public void main (String [] args)
 {
  char c = 'a';
  System.out.println("c: " + c);
  int i = c;   //char->int:conversão explícita desnecessária
  System.out.println("i: " + i);
  float f = c; //char->float:conversão explícita desnecessária
  System.out.println("f: " + f);
  int j = 101;
  System.out.println("j: " + j);
  char d = (char)j; //int->char: 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 i = (int)x;
     System.out.println("  i: " + i);
      i = (int)Math.round(x);
       System.out.println("i: " + i);
 }
}
```

# 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 larguraPapel = 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

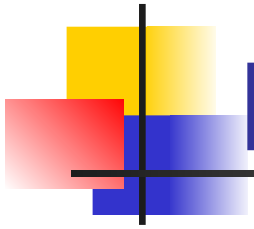
    @version

    @param

    @return

    @exception

    @deprecated



# println() vs. printf()

```
public static void printSomeStrings() {
    String nome = "José";
    String sobrenome = "Da Silva";
    int numAnimais = 7;
    String tipoDeAnimal = "gatos";
    System.out.printf("%s %s tem %s %s.%n", nome, sobrenome, numAnimais, tipoDeAnimal);
    System.out.println(nome + " " + sobrenome + " tem " + numAnimais + " " + tipoDeAnimal
}
```

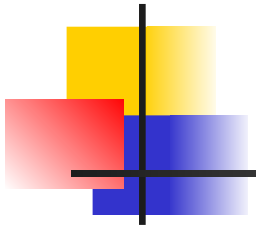
%d = número inteiro (byte, short, int, long)

%f = ponto flutuante (float, double)

%s = string

%n = nova linha

# 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 | --a ou 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

| | |
|---|---|
| > | Utilizado quando desejamos verificar se uma variável é maior que outra. |
| >= | Utilizado quando desejamos verificar se uma variável é maior ou igual a outra |
| < | Utilizado quando desejamos verificar se uma variável é menor que outra. |
| <= | Utilizado quando desejamos verificar se uma variável é menor ou igual a outra. |

- O resultado é sempre um valor lógico (booleano)
- Comparam valores de expressões

# 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 (ou booleanos)

- Resultado é sempre booleano
- Compara valores/ expressões booleanas
- curto-circuito: avaliação termina se resultado puder ser determinado antes de avaliar toda expressão

| | |
|---|---|
| **E** | **&&** |
| **Ou** | **\|\|** |
| **Não** | **!** |

# Operadores Lógicos (ou booleanos)

```
public class Logico {
 static public void main (String[] args)
 {

  System.out.println("True && True = " + (true && true) );

  System.out.println("True && False = " +(true && false) );

  System.out.println("True || True = " + (true || true) );

  System.out.println("True || False = " + (true || false) );

  System.out.println("Not False = " + (!false) );
 }
}
```

# Operadores Bitwise: & e |

```
int a = 4;
int b = 7;
System.out.println(a & b); // imprime 4
//em um sistema de 32 bits
// 00000000 00000000 00000000 00000100
// 00000000 00000000 00000000 00000111
// ================================
// 00000000 00000000 00000000 00000100
```

# 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;
```

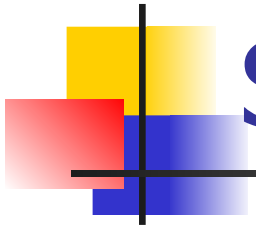
# Operador Ternário

- 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"
 }
}
```



# String

- String é uma sequência de caracteres entre aspas. Por exemplo: "Olá"
- Uma string é uma instância da classe **String**
  - portanto, strings são objetos.
- O processamento de strings é tão importante e frequente, que Java fornece suporte especial para strings para tornar seu uso mais fácil:
  - ex: instanciação, operador de concatenação nativo

# String

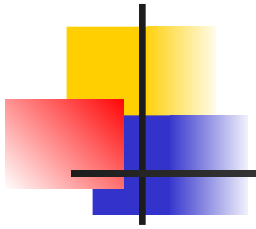
## Scanner

Uma das formas de se ler dados em Java é com objeto do tipo **Scanner:**

```
Scanner sc = new Scanner(System.in); //import java.util.Scanner;
int x = sc.nextInt();
String z = sc.next();
```

## Métodos úteis de String:

- Formatar: toLowerCase(), toUpperCase(), trim()
- Recortar: substring(inicio), substring(inicio, fim)
- Substituir: replace(char, char), replace(string, string)
- Buscar: indexOf, lastIndexOf
- str.split(" ")



# Matriz (ou Array) em Java

*tipo* variavel '[' ']' '=' '{' [ valor1 [ ',' ... ] ] '}";'

*tipo* '[' ']' variavel '=' 'new' *tipo* '[' tamanho ']' ';'

ex1: int [] notas = { 0, 8, 2, 3 };

ex2: int notas[] = new int [4];

notas[0] = 9;

Exercicio 1: crie uma matriz com os meses do ano e imprima na tela os elementos.

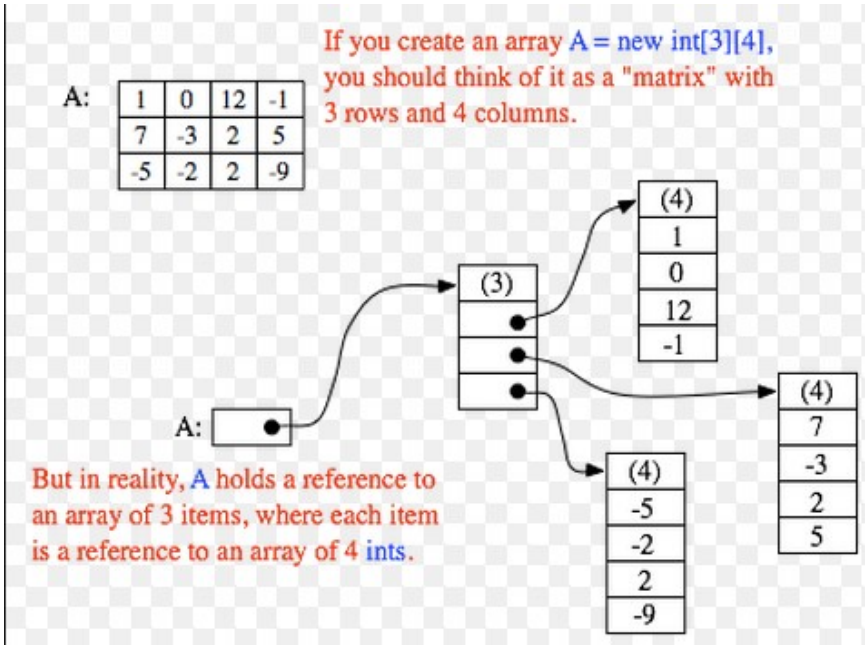
# Arrays de 1 e 2 dimensões

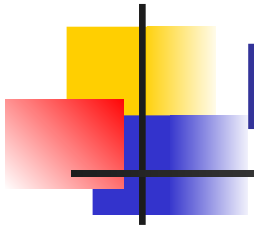
## One Dimensional array

Initialization `int a[] = new int [12];`

| Value | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] | a[10] | a11[11] |

`System.out.print(a[5]);` Output: 6



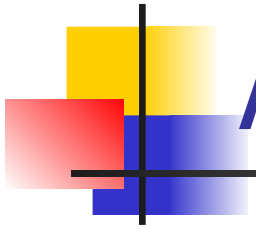


# Percorrendo um Array

```
public static void main(String args[]) {
   int[] idades = new int[10];
   for (int i = 0; i < 10; i++) {
     idades[i] = i * 10;
   }
  for (int i = 0; i < 10; i++) {
     System.out.println(idades[i]);
  }
}
```

# E quando não sabemos o tamanho do array?

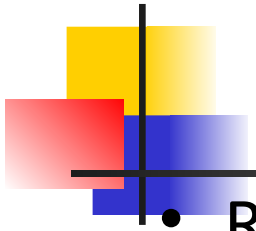
```
void imprimeArray(int[] array) {
  for (int i = 0; i < array.length; i++)        {
    System.out.println(array[i]);
  }
}
```



# Array no Java 5.0

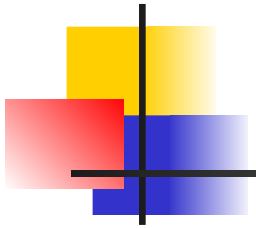
```
public static void main(String args[]) {
   int[] idades = new int[10];
   for (int i = 0; i < 10; i++) {
      idades[i] = i * 10;
   }
   for (int x : idades) {
       System.out.println(x);
   }
}
```

# Estruturas de Controle e Programação Básica



- Repetições
  - Definidas
  - Condicionais
- Desvios
  - Condicionais
  - Incondicionais
- Modularização
- Tratamento de Exceções
- Entrada e Saída Padrão

# Comandos ou Diretivas



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

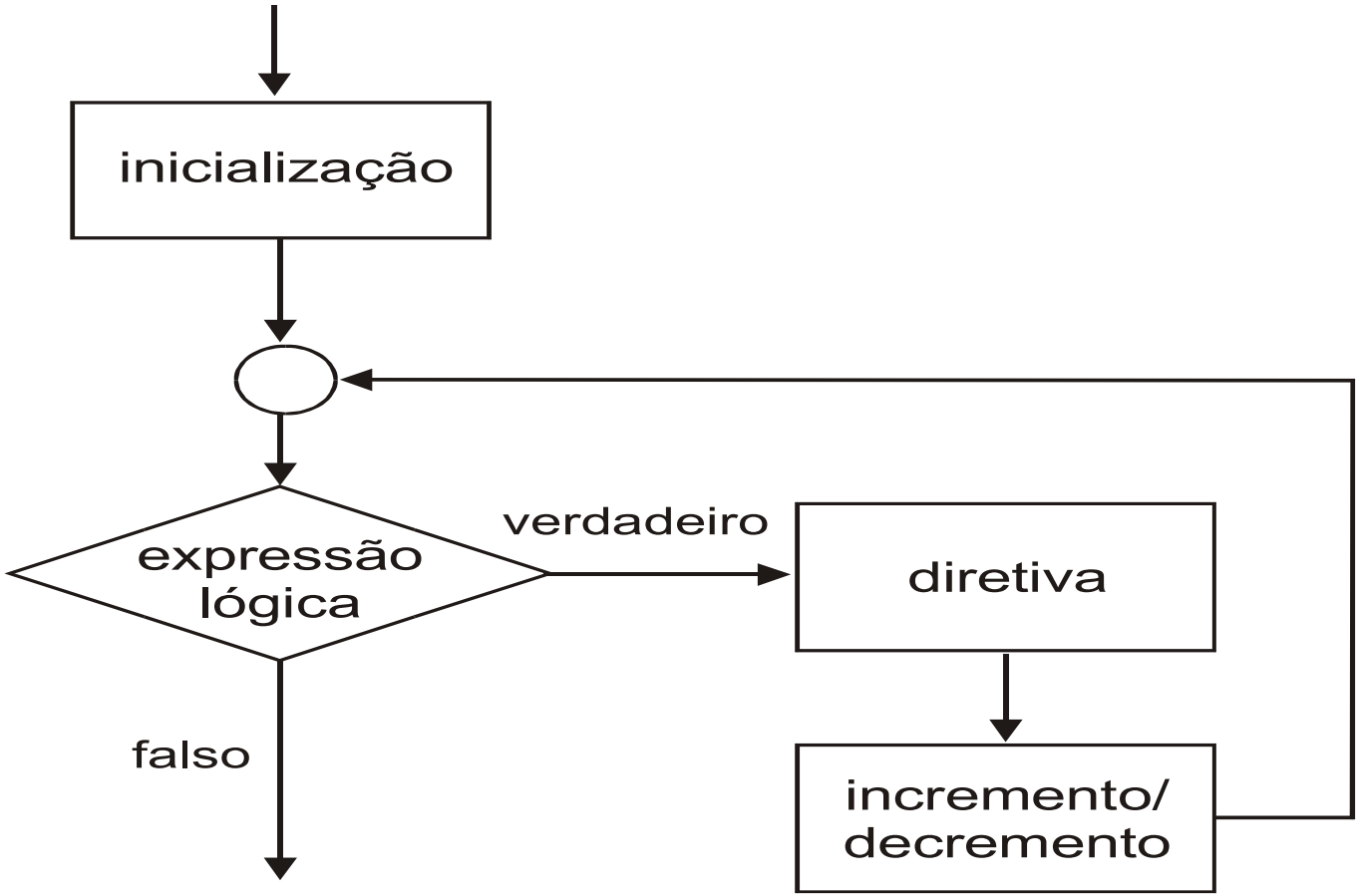
# for

- Formato

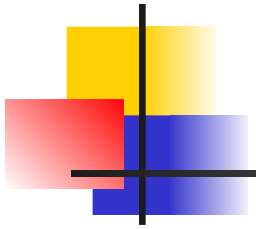
```
for (inicialização; condição de execução;
  incremento/decremento) diretiva;
```

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

# for - Estrutura Geral

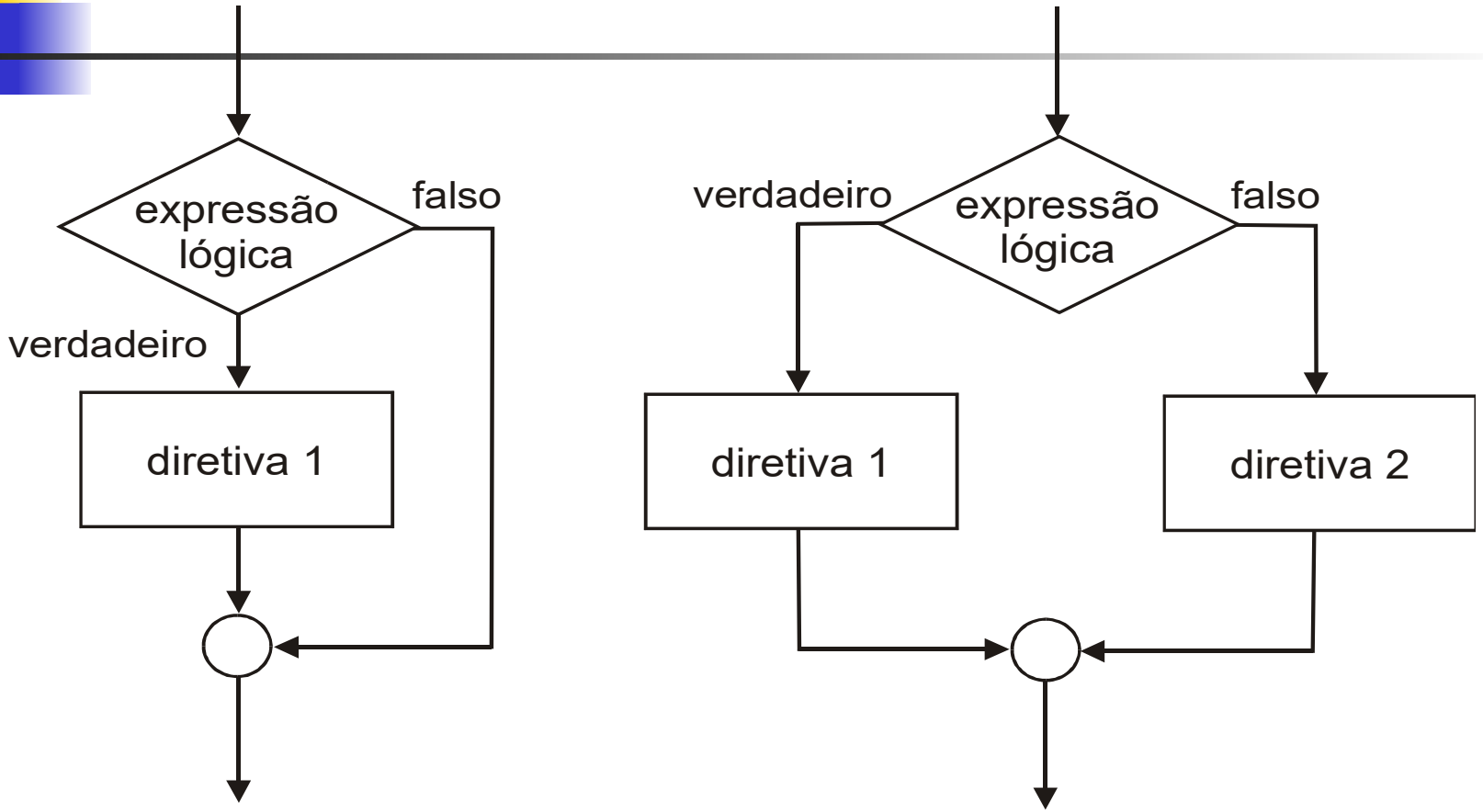


# for



```
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");
  }
}
```

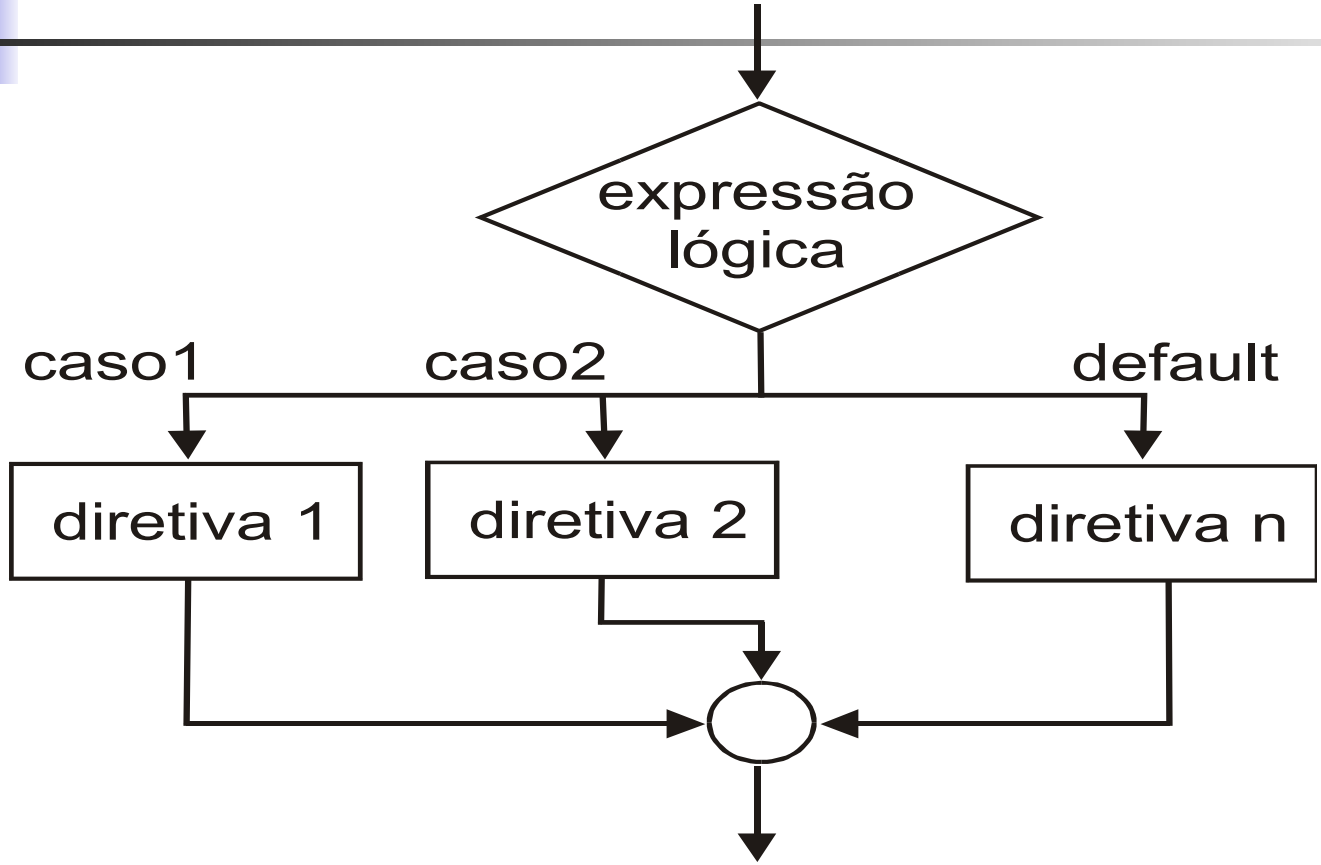
# switch

- Formato

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

- uso de break
- default opcional

# switch

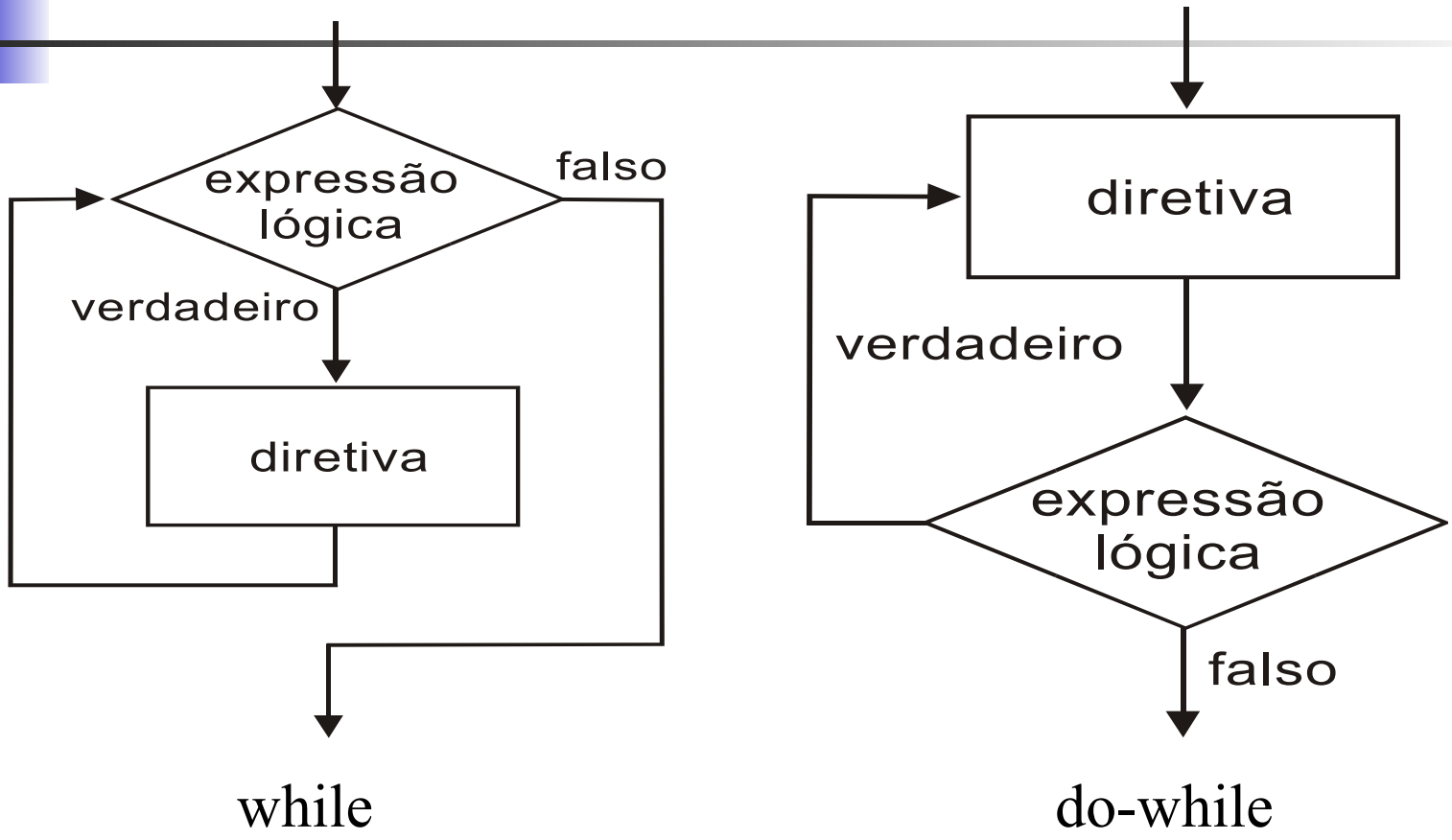

expressão
lógica
caso1
caso2
default
diretiva 1
diretiva 2
diretiva n

# switch

**Só funciona com int, short, char, enum, string e wrapper classes*

```
import java.util.Scanner;
public class   exemploSwitch {
   public static void main (String[]   args) {
     Scanner scan = new Scanner(System.in);
     System.out.println("entre com uma palavra: ");
     String entrada = scan.next();
     switch( entrada[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");
      }
   }
}
```

# while e do-while



# while

```
public class    ExemploWhile {
public static void main (String args[]) {
 Scanner scan = new Scanner(System.in);
 System.out.println("digite um número menor que 10");
 int i = scan.nextInt();
 int j = 10;
 while (j > i)
      {
       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");
 }
}
```

# Entrada de dados em Java

## Scanner

Uma das formas de se ler dados em Java é por meio de um objeto do tipo **Scanner**, o qual deve ser instanciado com o argumento **System.in**:

```
Scanner sc = new Scanner(System.in);
```

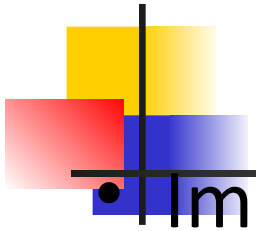
```
import java.util.Scanner;
```

faça sc.close() quando não precisar mais do objeto sc

```
int x = sc.nextInt();
double y = sc.nextDouble();
String z = sc.next();
```

**ATENÇÃO:**
Use Locale.*setDefault(Locale.US);*
**ANTES** de criar o Scanner para separador de decimais como ponto.

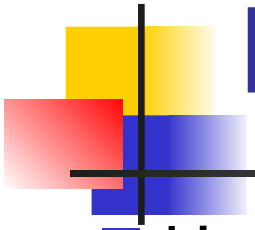
# Saída de dados em Java



- Imprimir na saída padrão (console):
  - System.out.print
    - Concatena strings com o operador “+”
  - System.out.println:
    - Igual ao anterior, mas pula linha ao final
  - System.out.printf:
    - sintaxe inspirada no printf do C:
      - System.out.printf([locale,] format string, arguments);
      - System.out.printf(Locale.US, "%,d %n", 10000);
        //10,000
      - System.out.printf(new Locale("pt","BR"), "%,d %n", 10000);  //10.000

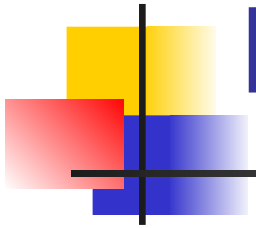

# Parte 3

## Programação Orientada a Objetos



# Histórico

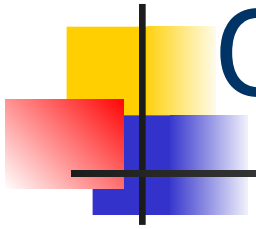
- Linguagem SIMULA (anos 60), derivada do ALGOL e desenvolvida no Centro Norueguês de Computação, por Kristen Nygaard e Ole-Johan Dahl.
- A SIMULA era utilizada para simulações e foi a pioneira na utilização de classes e subclasses, semelhantes às usadas atualmente em POO.
- SMALLTALK (anos 70), foi a primeira linguagem autêntica orientada a objetos. Foi desenvolvida por cientistas do Xerox Palo Alto Research Center (Xerox PARC).
- Cada elemento do SMALLTALK é tratado como sendo um objeto. Esta linguagem é puramente OO.



# Histórico

- C++, dos laboratórios da AT&T Bell, criada por Bjarne Stroustrup, uma expansão da linguagem C, muito popular e conhecida.
- Suporta POO, mantendo as características do C tradicional.
- Fácil transição do C para o C++.

# Elementos básicos em Orientação a Objetos



**Objetos**

**Classes**

**Instâncias**

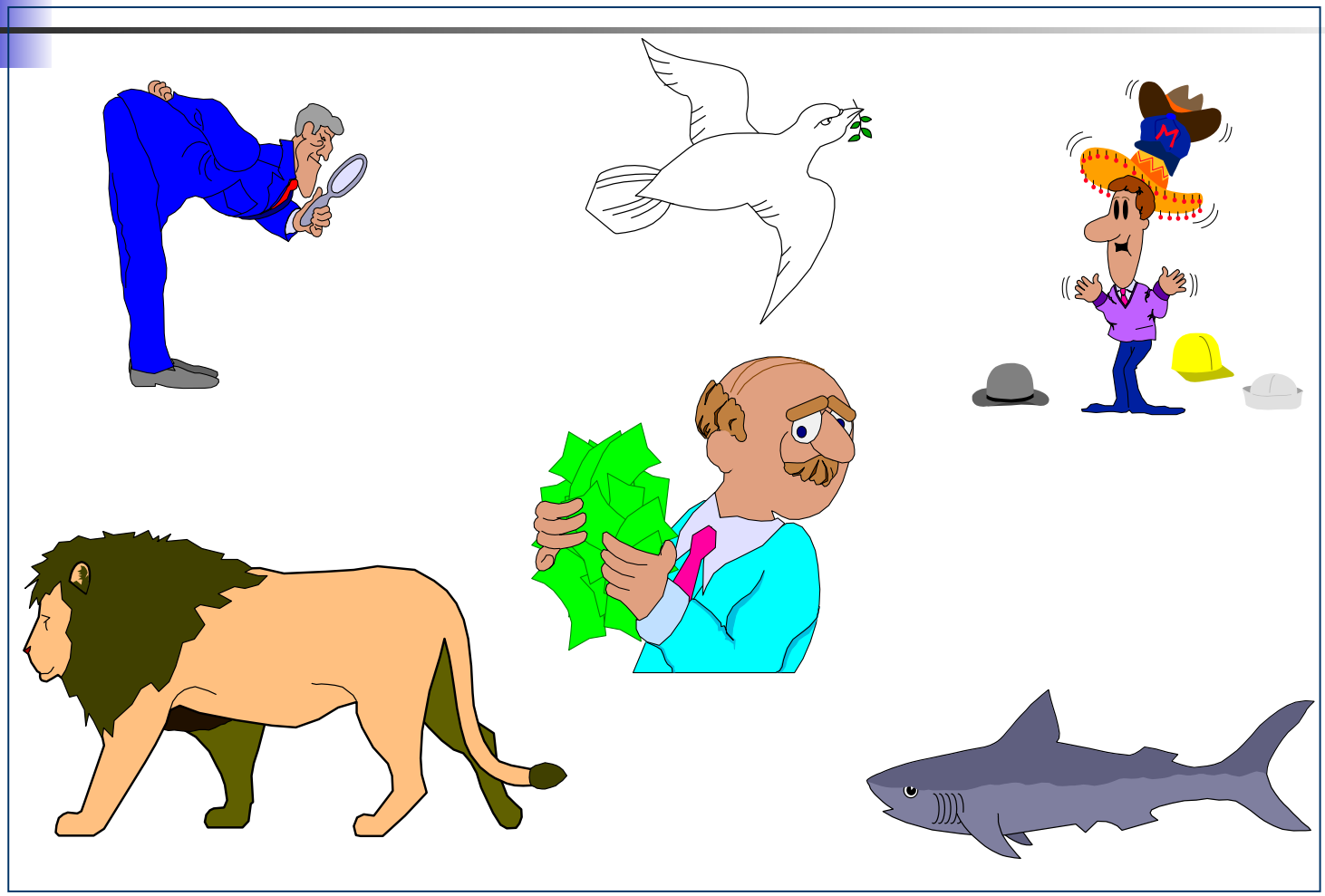
# Objeto

Um objeto é a entidade que realiza um tipo ou uma coleção de tipos (tipo composto) através de suas propriedades e comportamento.

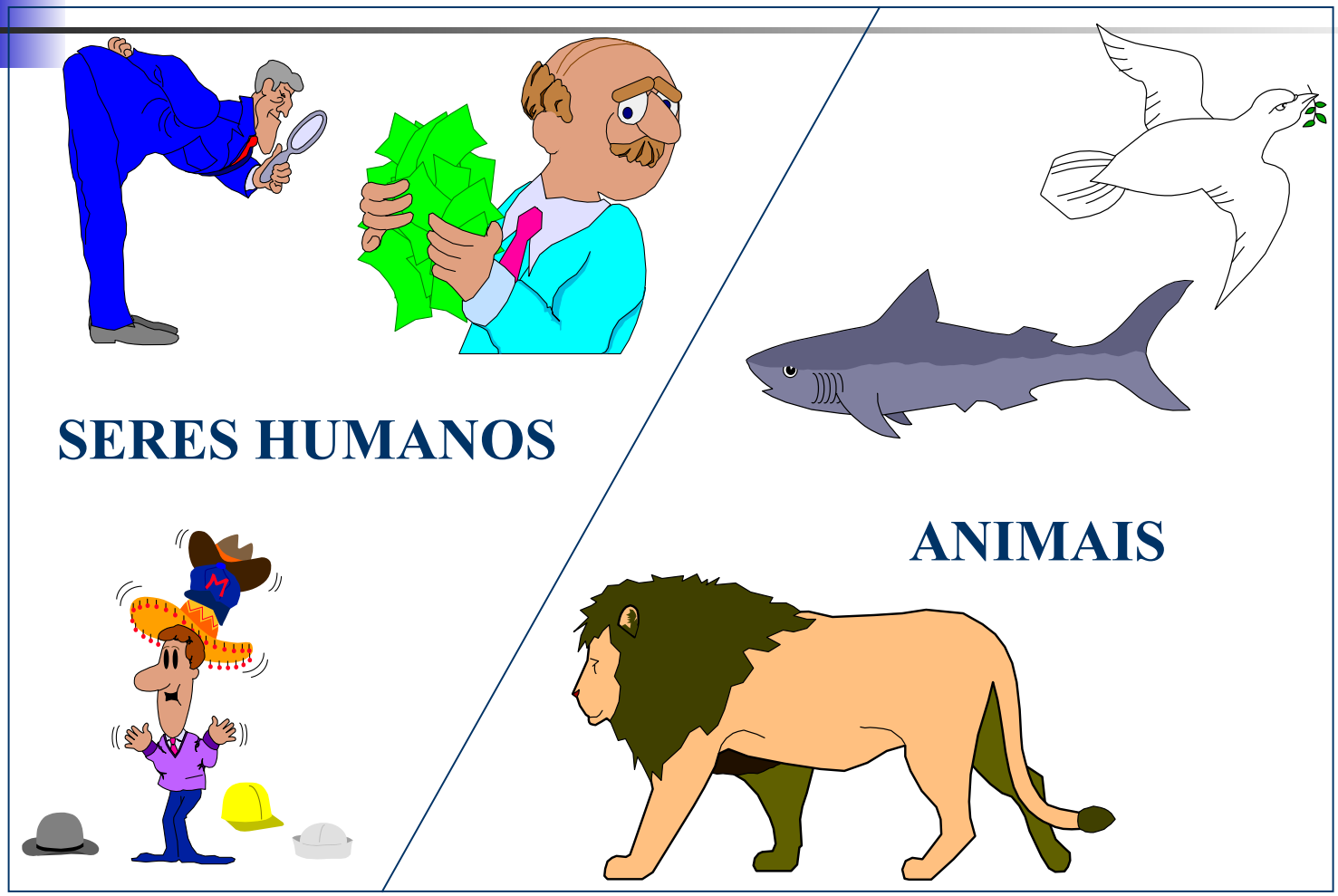
Um objeto representa um elemento do problema real modelado a partir dos tipos que implementa.

Um objeto é uma variável do tipo de uma classe definida pelo usuário. Ou seja, um objeto é uma instância de uma classe.

# Objetos



# Classificando os objetos...



# Objeto

Um objeto tem estado que é um elemento particular e personificado da classe.

Podemos dizer que um objeto possui três partes: estado, comportamento e identidade.

# Objeto - Estado

O estado de um objeto é uma das condições em que ele pode existir.

É uma característica transitória de um objeto.

Ele normalmente muda com o decorrer do tempo e é caracterizado pelos valores instantâneos dos seus atributos e de suas ligações e relacionamentos com outros objetos.

# Objeto – Estado - Exemplo

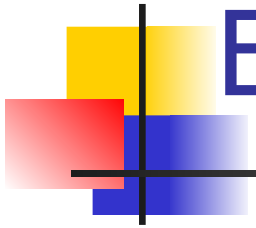
**Exemplo:** Um objeto lâmpada pode ter basicamente dois estados - acesa e apagada, que só podem ser alterados através das ações: acender ou apagar. Tais ações fariam com que o valor do atributo de nome *aceso* variasse entre verdadeiro, quando a lâmpada estivesse acesa e falso, quando estivesse apagada.

# Objeto – Comportamento

É definido pelo conjunto de seus métodos, ou seja, pelo conjunto das ações que este objeto pode executar e pela forma que ele responde às chamadas de outros objetos.

Ou seja, determina como o objeto age e reage às requisições de outros objetos.

# Objeto – Comportamento – Exemplo



**Exemplo:** Um objeto pessoa possui alguns comportamentos padrões como comer, beber, andar, etc.

Se você enviar uma mensagem para um objeto pessoa do tipo comer, ele saberá como responder e executar a ação. Mas se você mandar uma mensagem do tipo voar, esse objeto não irá responder, pois voar não faz parte de seu comportamento.

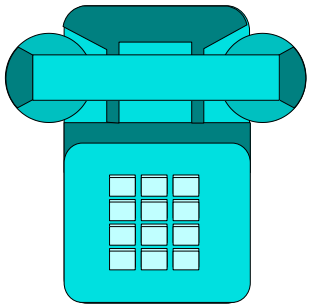
# Objeto - Identidade

Unicidade do objeto pelo simples fato de ele existir.

Um objeto sempre será único mesmo que exista um outro objeto com os valores dos atributos iguais.

# Objeto - Exemplos

*Identidade*: 'Telefone residencial'



*Características*:

cor azul
número 576-0989

*Comportamento*:

tocar
discar

*Identidade*: 'Ônibus escolar'

*Características*:

cor amarelo
placa LXY 7684
30 assentos
a diesel

*Comportamento*:

frear
andar
correr
buzinar
acelerar

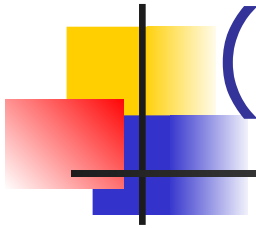
# Objeto – Outro Exemplo

Montagem de um computador

Composto por vários componentes:

- Placa-mãe
- CPU
- Placa de vídeo
- Disco rígido
- Teclado, etc.

# Objeto – Outro Exemplo (cont.)



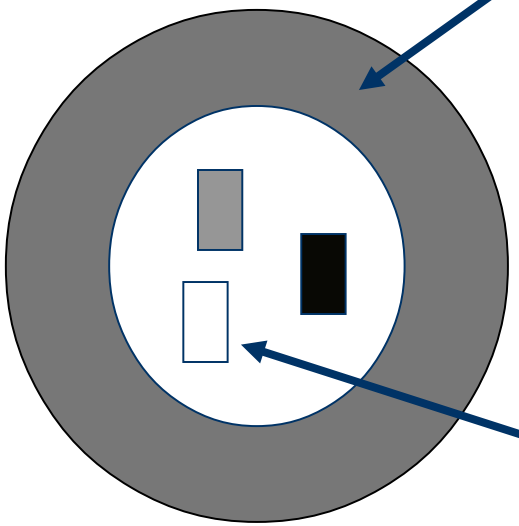
Cada componente é bastante sofisticado, mas o usuário não precisa saber como funciona internamente.

Cada componente é independente dos demais.

Para quem está montando, interessa apenas como os componentes interagem entre si:

- O monitor funciona com essa placa?
- A CPU é compatível com a placa-mãe?

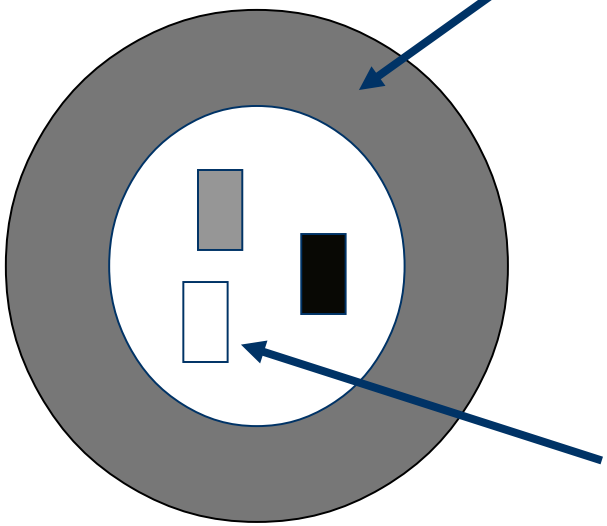
# Objeto - Representação



Operações/métodos/interface
(público)

Dados/propriedades/atributos
(privado)

# Exemplo – Lâmpada

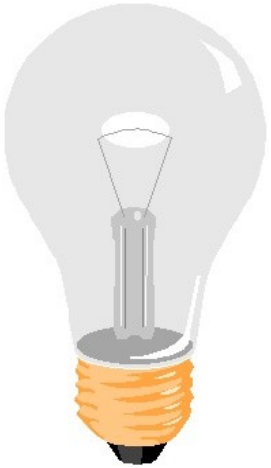


Operações:

- Ligar
- Desligar

Dados:

- Ligada (s/n)
- Potência
- Voltagem



# Classe

É um conjunto de objetos que compartilham uma estrutura e comportamento comuns.

É uma abstração para representar objetos com características e comportamentos comuns.

# Classe

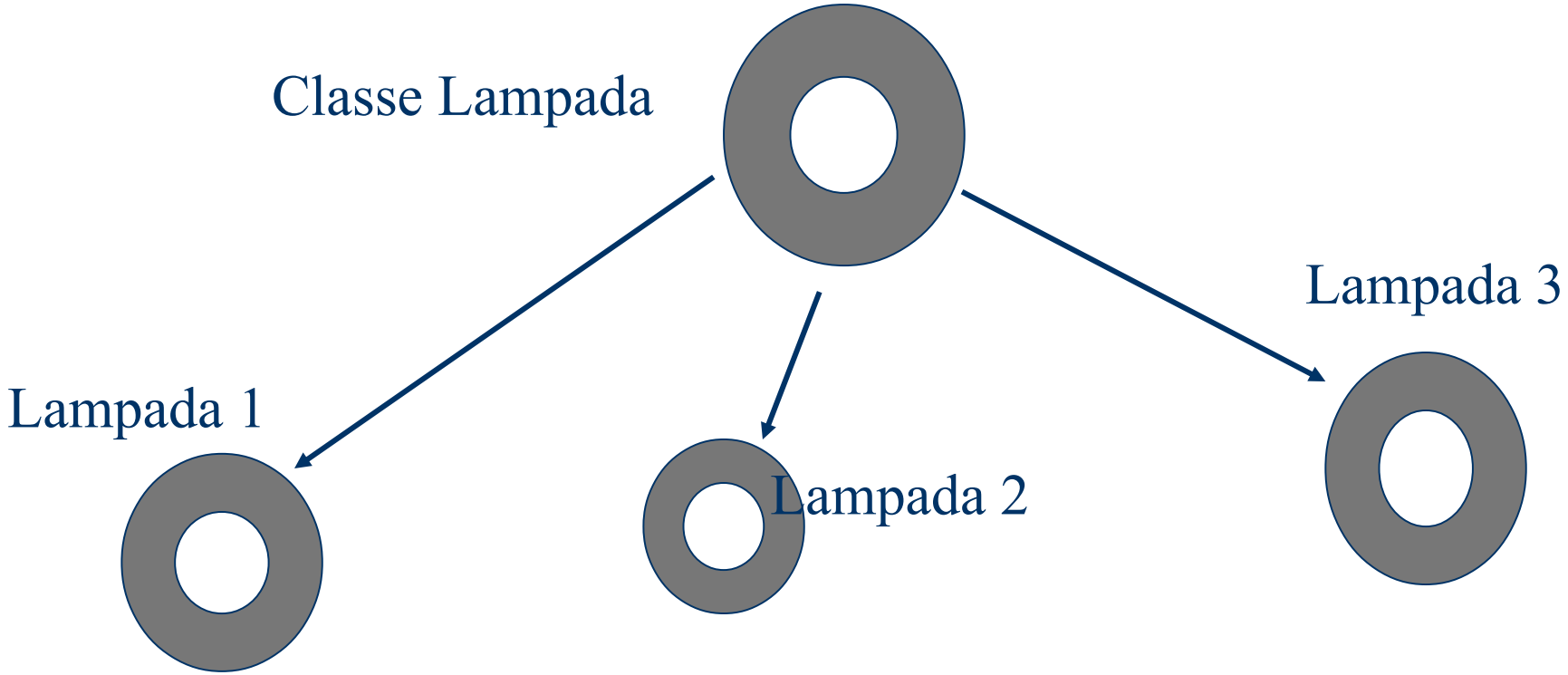
Uma classe pode ser vista como uma "fábrica de objetos".

Objetos de uma classe são denominados **instâncias**.

Todos os objetos são instâncias de alguma classe.

# Classe - Uma fábrica de objetos

Uma fábrica de objetos




Objetos ou instâncias da classe Lampada.

# Classe

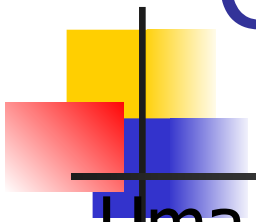
- A classe implementa um ou mais tipos , estabelecendo propriedades e comportamento.

Define o aspecto genérico de um objeto. Todo objeto pertence a uma classe.

A classe é a idéia a partir do qual o objeto se concretiza.

Uma classe pode especializar e agregar outras classes para formalizar uma idéia.

# Classe



Uma classe determina um conjunto de objetos com:

Propriedades semelhantes

Comportamentos semelhantes

Relacionamentos comuns com outros objetos

Em uma classe encontramos duas divisões:

Estrutura: as informações inerentes à classe.

Comportamento: as operações realizadas pela classe.

# Classes - Exemplos

Telefone

*Características*:
cor
número
discagem

*Comportamento*:
tocar
discar

*Identidade*: 'Telefone da minha casa'

*Características*:
cor: azul
número: 576-0989
discagem: tone

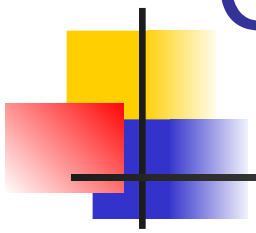
*Comportamento*:
tocar
discar

*Identidade*: 'Meu celular'

*Características*:
cor: preta
número: 99193467
discagem: tone

*Comportamento*:
tocar
discar

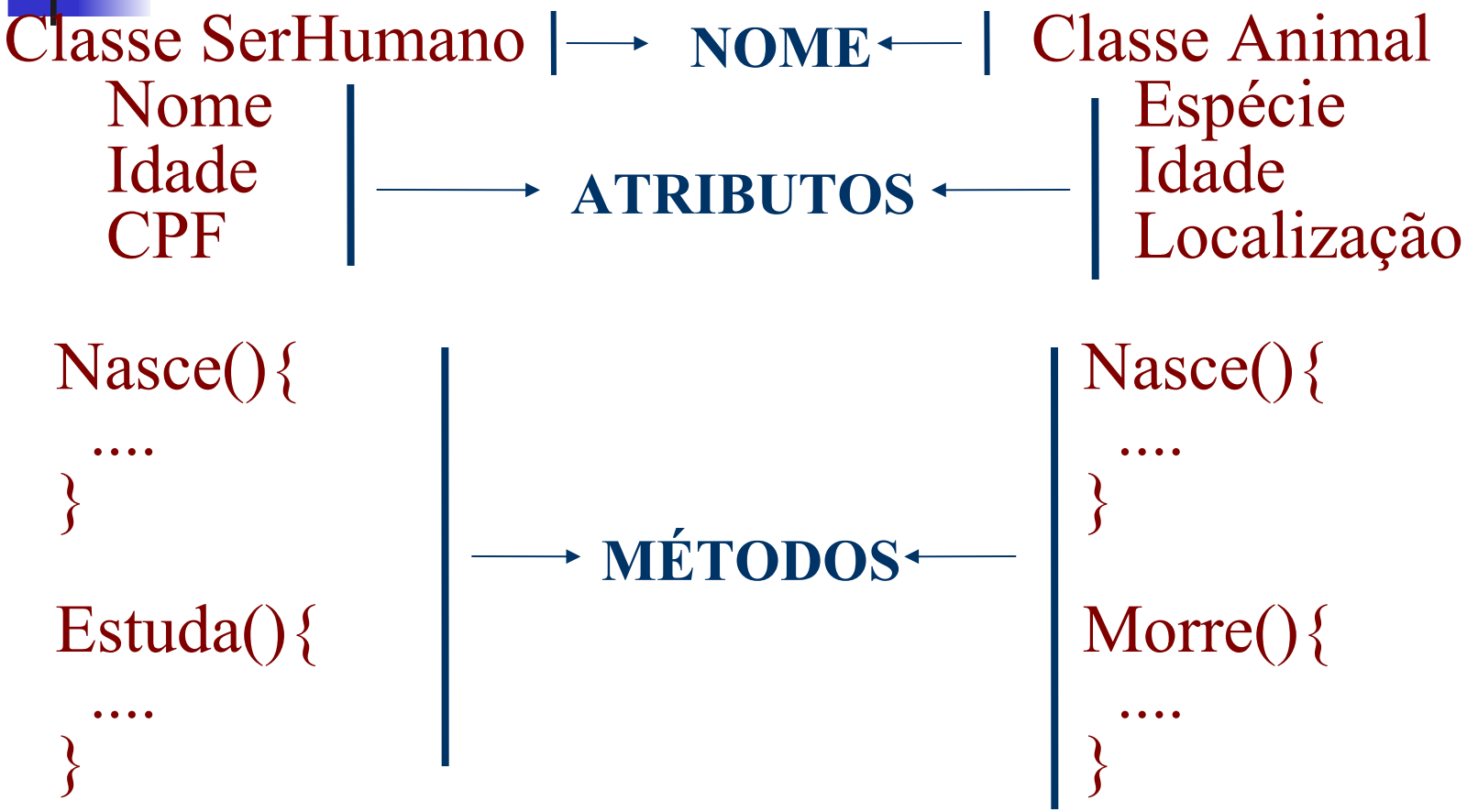
# Classe



Uma classe é formada:

- Por um nome
- Por atributos
- Por métodos

# Classes – Exemplos



# Classe - Exemplo

**Exemplo:** Uma pessoa caracterizada por uma classe.

Estrutura: Nome, Endereço, Peso e Altura

Comportamento: Andar, Comer, Beber e Dormir.

# Classe – Exemplo de Código

**Exemplo:**

```
class Pessoa {
    String nome, endereco;
    float peso, altura;
    void correr(){ ... }
        void comer(){ ... }
        void beber(){ ... }
        void dormir(){ ...}
     }
```

# Classe – Outro Exemplo de Código

**Exemplo:**

```
class Correntista extends Cliente
implements Depositante, Comprador{
   Investidor investidor;
   Conta conta;
   void deposita(doublequantia) {
      conta.deposita(quantia);
      }
       void compra(Acao acao){
          investidor.compra(acao);
      }
    }
```

# Pacotes

Os pacotes são unidades de agrupamentos de classes.

Classes são agrupadas em pacotes por vários motivos:

Por questão de organização,

Performance da aplicação,

Solução de problemas de escopo de variáveis,

Verticalização de uma entidade para comercializá-la como um componente, etc.

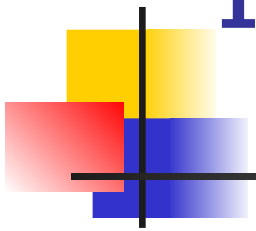
# Pacotes

Os pacotes podem possuir subpacotes de qualquer profundidade.

Exemplo: Pacote Pagamento pode conter subpacotes tais como Mensalidade, Professores e Funcionários.

Os pacotes servem para encapsular classes.

# Instância

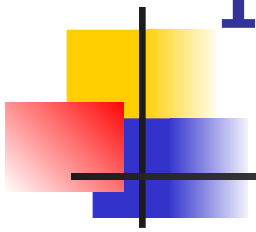


Instância é o mesmo que objeto.

Instanciação é o ato de criar um objeto.

Quando instanciamos uma classe, estamos particularizando, identificando um objeto.

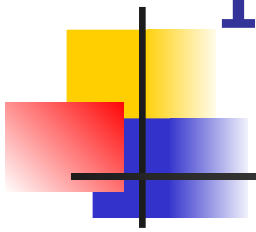
# Instância



Uma instância tem estado, comportamento e identidade.

A instância é o objeto nominal criado a partir de uma classe.

# Instância



A classe é apenas uma idéia, a instância é um objeto real e operante.

Um sistema OO em execução é composto de inúmeras instâncias de objetos.

Instâncias devem ser criadas a partir da classe que a define através de um método chamado construtor.

# Instância - Exemplo

Classificação

Instanciação

Classe
SerHumano
Nome
Idade
CPF

Nasce(){
....
}

Estuda(){
....
}

Objeto OBJ1

Nome=Mr.Jones
Idade=34

CPF=4567889484

Nasce(){
....
}

Estuda(){
....
}

Objeto OBJ2

Nome=Mr.Zoo
Idade=45

CPF=6786968696

Nasce(){
....
}

Estuda(){
....
}

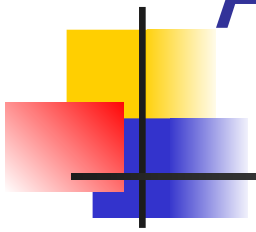
# Instância – Exemplo de código

Instâncias são invocadas para realizar seu trabalho a partir de referências.

No código Java temos o exemplo da criação de uma instância e associação a uma referência:

```
classe Cliente{
  Identificacao id = new
  Identificacao(“José das Couves”);
}
```

# Atributos



Atributos são informações de dados que pertencem a um determinado objeto.

Um atributo é uma característica de uma classe.

Atributos não possuem comportamento, pois eles não são objetos.

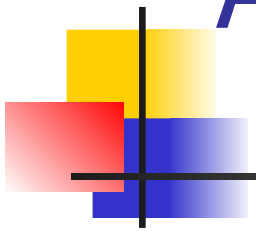
# Atributos

Enquanto na classe o atributo é uma especificação, na instância representa o estado do objeto, a própria identidade da instância.

As instâncias de um objeto têm em comum a classe que define o objeto. O que diferencia as instâncias entre si é o valor de seus atributos.

Atributos podem ser inicializados na classe e são modificados ao longo da vida útil da instância.

# Atributos - Exemplo

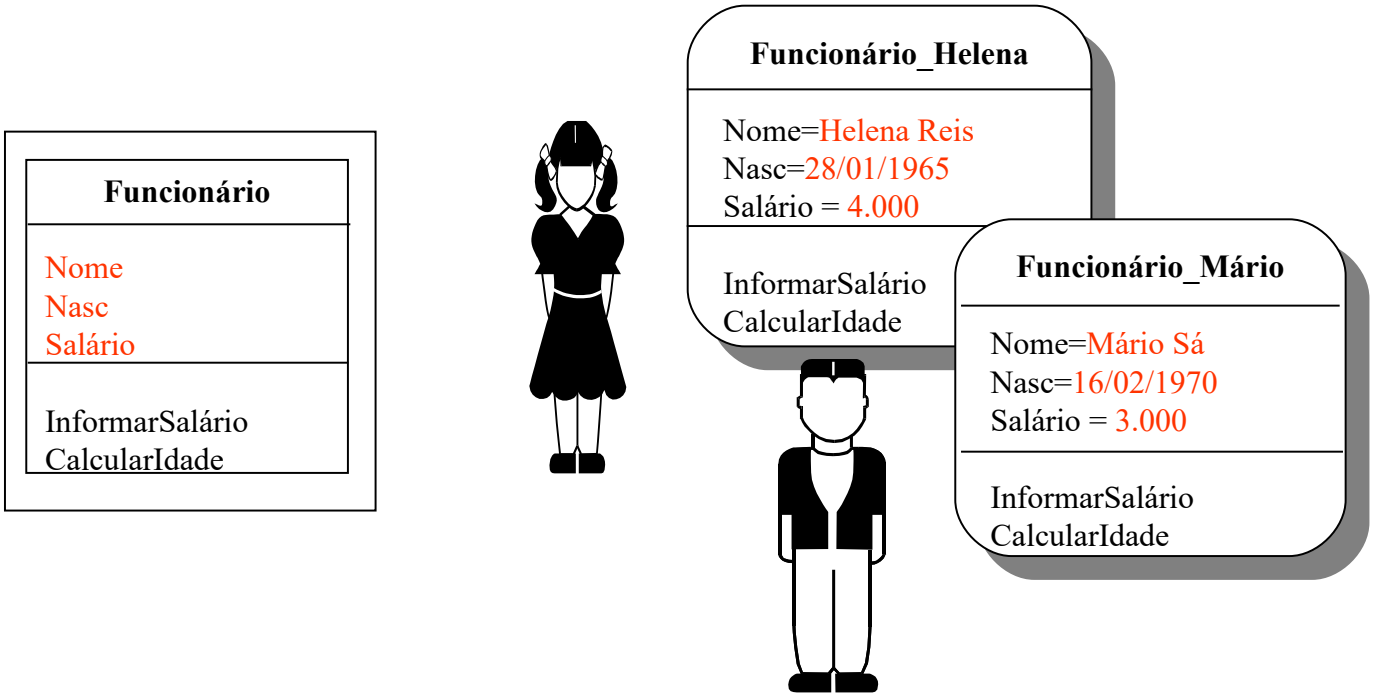


**Exemplo:**

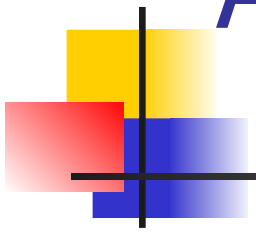
Classe: Triangulo

Atributos: base, altura, angulo1, angulo 2, angulo 3

# Atributo – Outro Exemplo



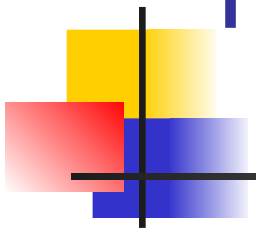
# Atributo – Exemplo de código



No código Java temos o exemplo dos atributos de uma classe:

```
classe ContaCorrente{
  double saldo = 0.0;
  double credito = 1000.0;
  Correntista titular;
 }
```

# Mensagens e Métodos



Para invocar um método, deve-se enviar uma mensagem para o objeto desejado.

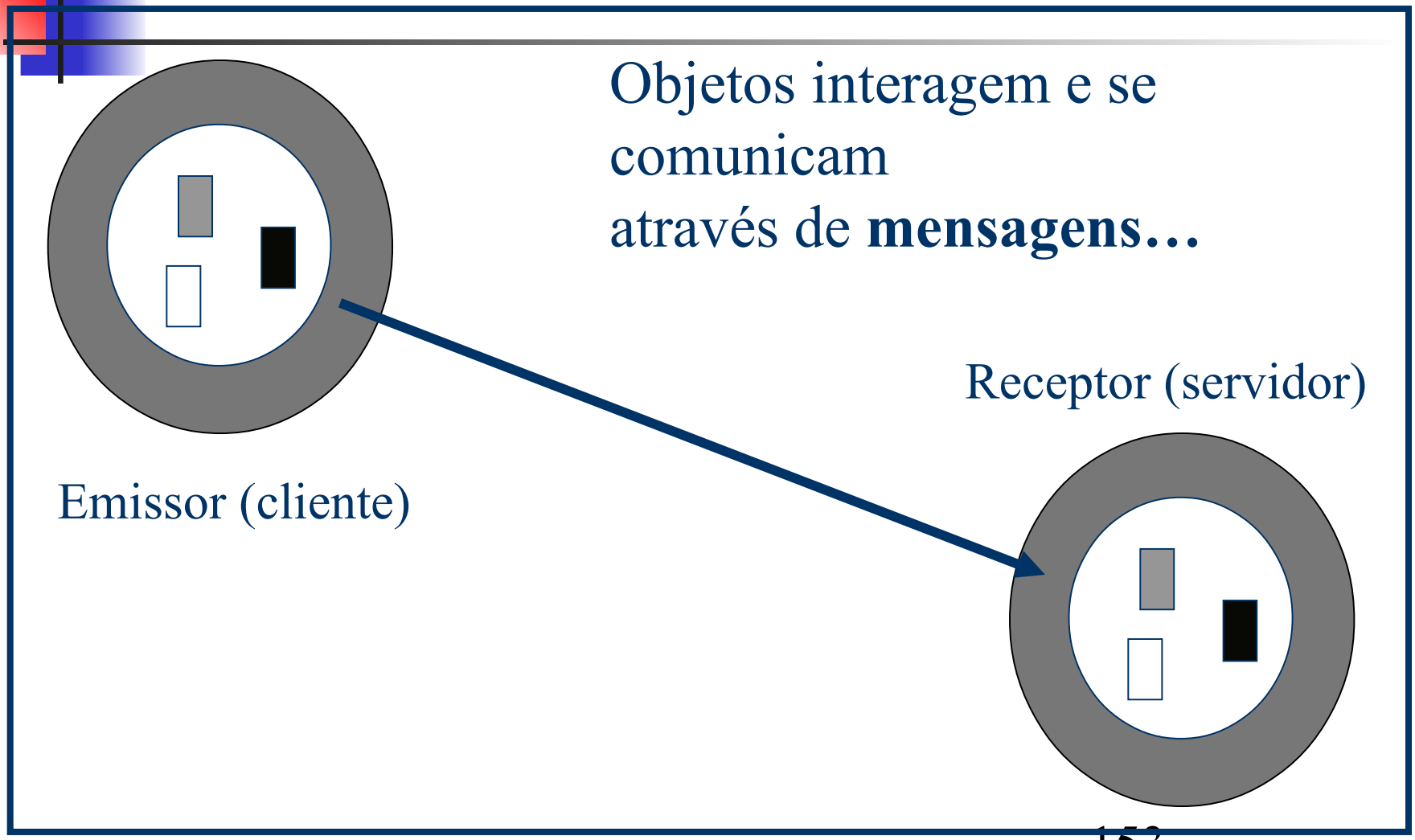
Para identificar uma mensagem, deve-se:

Identificar o **objeto** que receberá a mensagem.

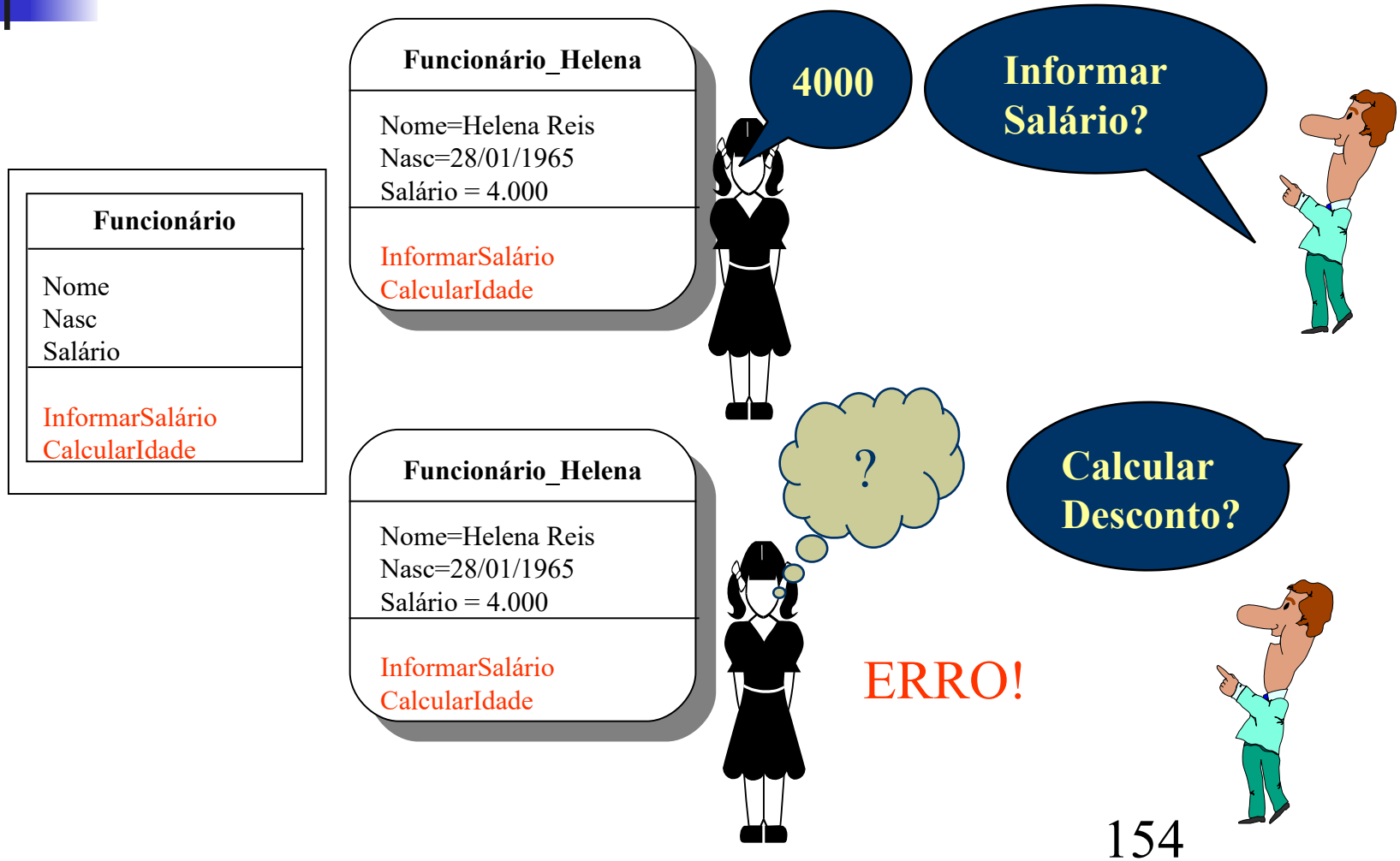
Itenficicar o **método** que o objeto deverá executar.

Passar os **argumentos** requeridos pelo método.

# Mensagens

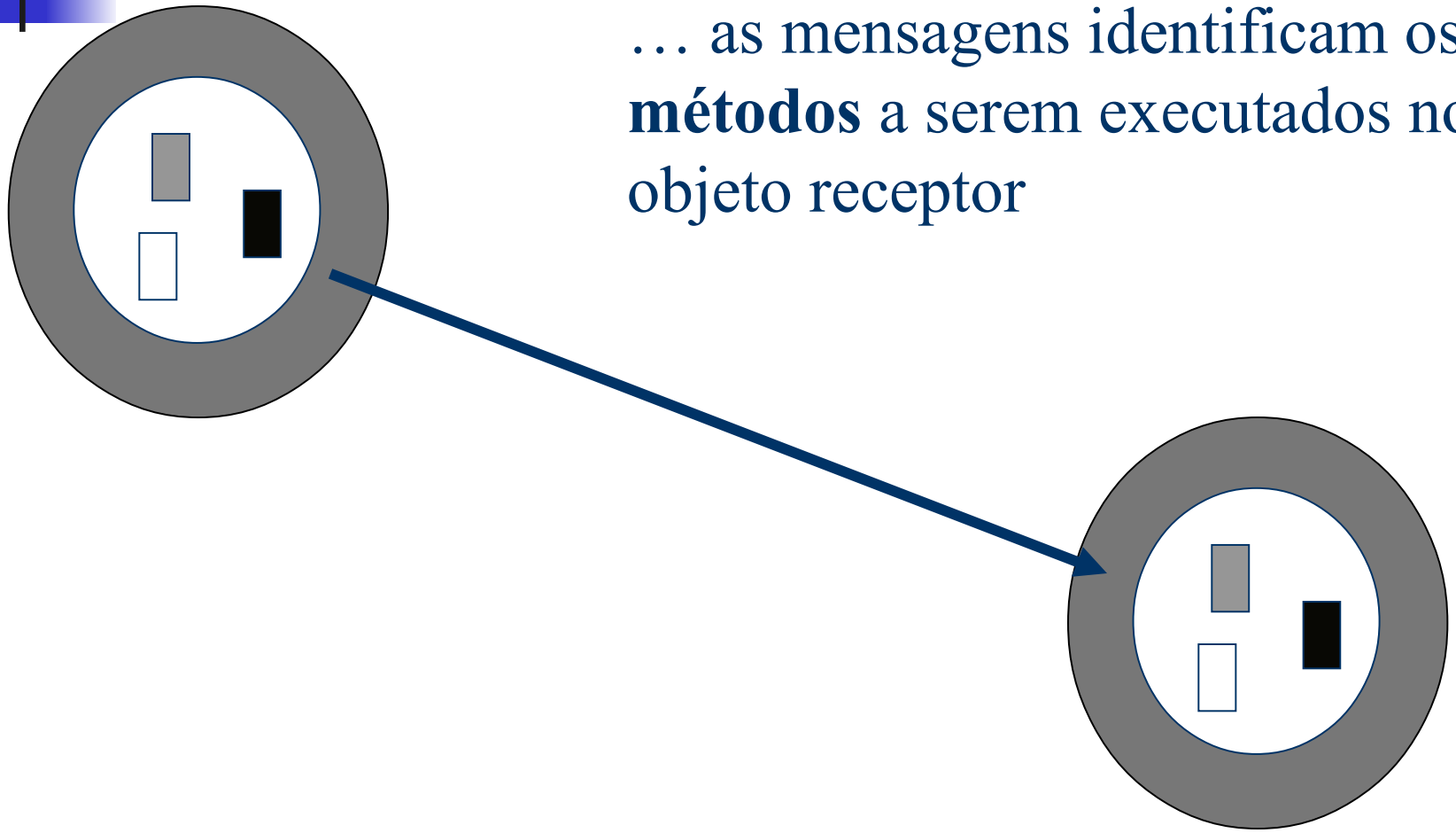

Objetos interagem e se comunicam
através de **mensagens…**
Receptor (servidor)
Emissor (cliente)

# Mensagens - Exemplo



# Métodos

… as mensagens identificam os **métodos** a serem executados no objeto receptor



# Métodos

Um método é uma implementação de uma operação.

É uma operação sobre um objeto, definido como parte da declaração de uma classe.

A classe implementa o comportamento desejado para o objeto através de métodos.

# Métodos - Exemplo

Através dos métodos, a instância tem a capacidade de modificar seus atributos.

# Método –Exemplo de código

No código Java vemos a especificação e a implementação de métodos:

```
interface Conta{
  void deposita(double quantia);
}

    classe ContaCorrente implements Conta{
      double saldo;
      void deposita(double quantia) {
            saldo += quantia;}
      void saca(double quantia) {
            saldo -= quantia;}
    }
```

# Exemplo Geral

| Nome | RG | Salário |
|---|---|---|
| Antonio da Silva | 5.000.421 | R$ 800,00 |
| Manoel Antonio Mello | 19.000.842 | R$ 122,00 |
| Silva Mello | 7.010.421 | R$ 8.000,00 |

# Exemplo Geral (cont.)

Neste exemplo, cada um dos funcionários:

Pode ser *identificado* por um número, para manipulação pelo sistema.

Possui um *estado* interno, definido pela atribuição de valores a Nome, RG e Salário.

# Exemplo Geral (cont.)

Os funcionários poderiam estar sujeitos a:

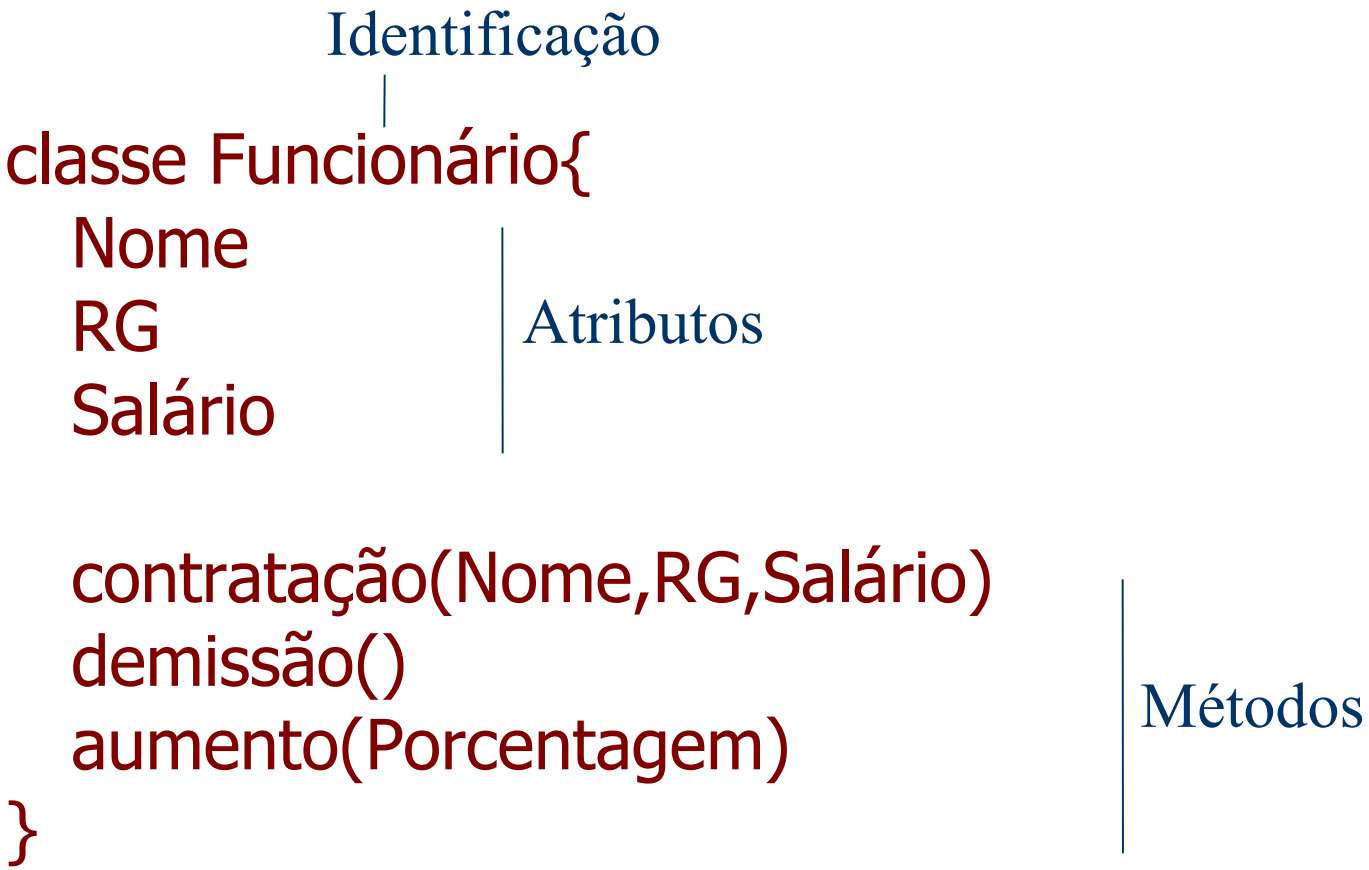
Contratação

Demissão

Aumento de salário

Em função destas operações, cada um deles poderia comportar-se de alguma maneira, inclusive com possibilidade de alteração de estado interno.

# Exemplo Geral (cont.)



# Exercício

Crie a classe Retângulo.

Essa classe deve possuir os seguintes atributos: comprimento e largura, que possuem valores iniciais 1.

Essa classe deve possuir métodos que calculam o perímetro e a área do retângulo.