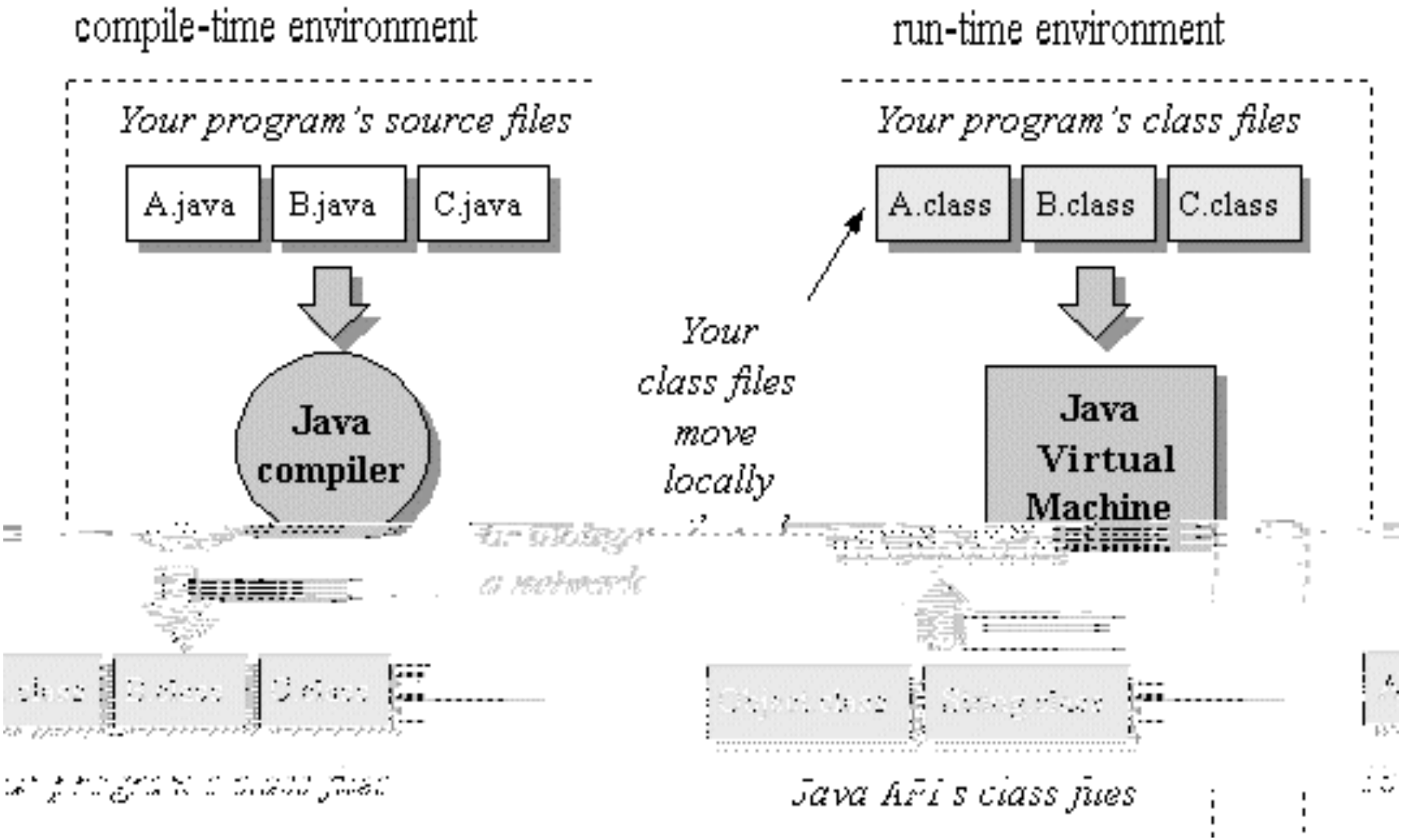
Introdução à Linguagem Java

- Introdução à Arquitetura Java
- Vantagens da Linguagem Java
- Ambiente de programação Java.

Arquitetura Java

- A arquitetura Java é composta de quatro itens principais:
  - A linguagem de programação Java
  - Formato de arquivos de classe Java
  - Java API (Application Programming Interface)
  - Máquina Virtual Java (Java Virtual Machine)
- Quando se escreve e executa programas em Java, estes quatro itens estão sendo utilizados.
- Programas são escritos usando a linguagem de programação Java, compilados para arquivos de formato de classe Java, sendo estes executados pela Máquina Virtual Java.
- Ao se escrever um programa, são usados recursos do sistema (I/O, por exemplo) através da chamada de métodos de classes que implementam Java API.
- Ao ser executado, ocorrem chamadas para Java API através do uso de métodos nos arquivos de classe que implementam Java API.

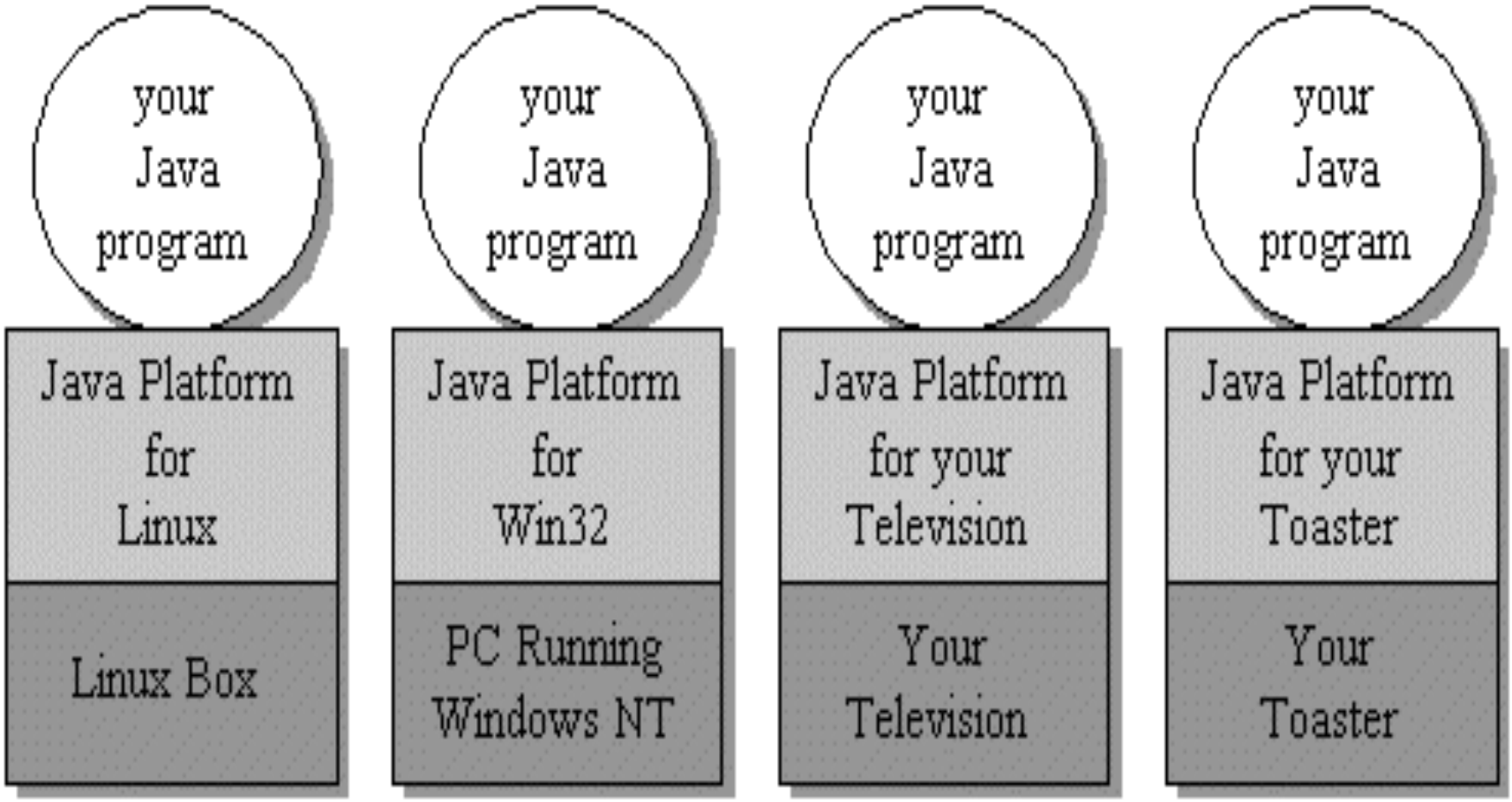
Ambiente de Programação



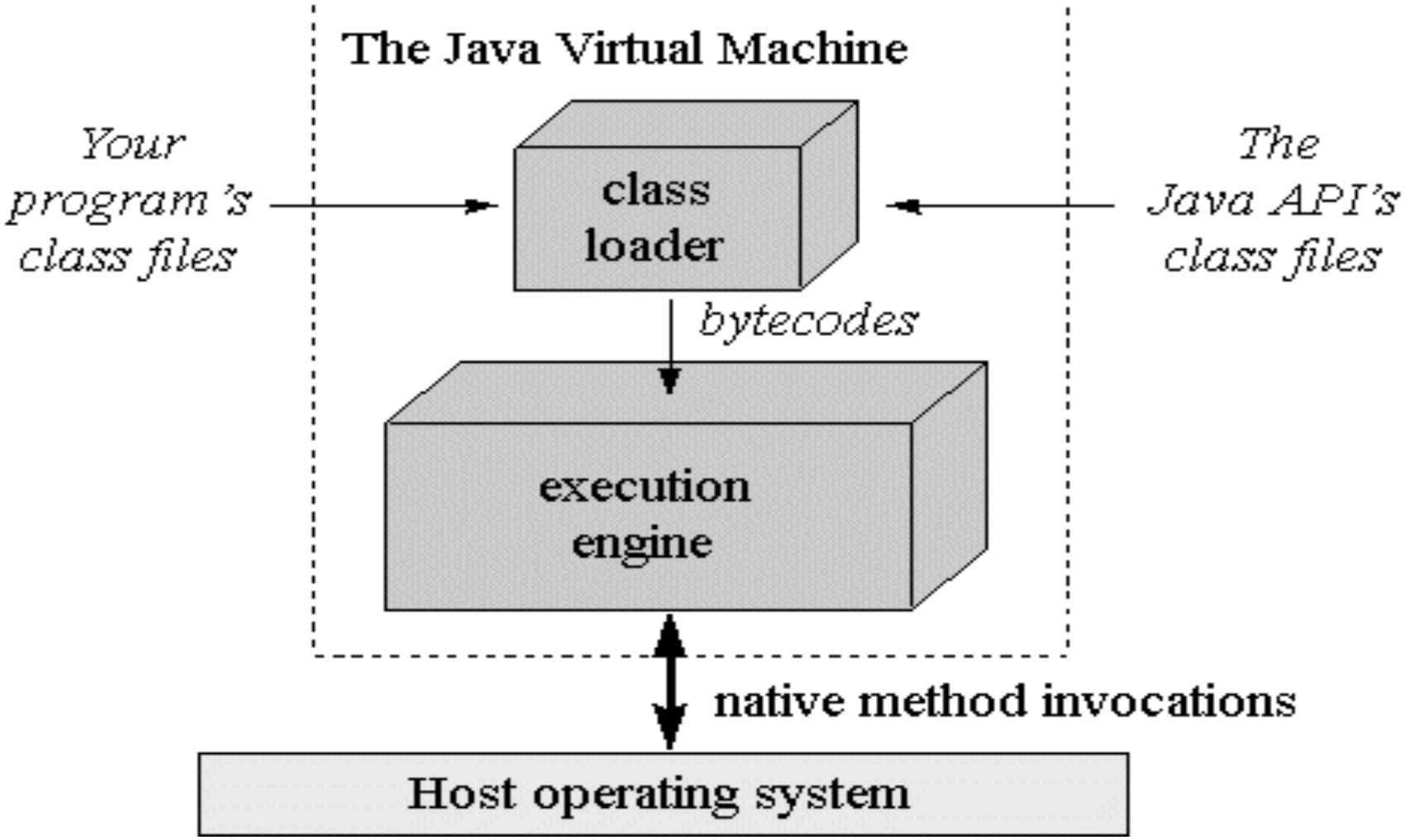
Arquitetura Java

- Juntos, a Máquina Virtual Java e Java API formam a plataforma na qual todos os programas Java são compilados.
- Além de ser chamado de Java runtime System, também é chamado de Java Plataform ou a partir da versão 1.2, Java 2 Plataform.
- Programas em Java podem ser executados em diferentes tipos de plataformas computacionais, pelo fato da Java Plataform poder ser implementado por software.

Plataforma Java



Máquina Virtual Java



Máquina Virtual Java (Cont.)

- O execution engine é uma parte da máquina virtual Java que pode apresentar diferentes implementações.
- A implementação mais simples da JVM é a interpretação do bytecode um de cada vez.
- Uma outra forma é o compilador just-in-time, mais rápido, porém requer mais memória. Os bytecodes de um método são compilados para o código nativo da máquina na primeira vez que o método for chamado e, em seguida, armazenados em cache para uso posterior.
- Um terceiro tipo de execution engine é o adaptive optimzer. Os bytecodes são inicialmente interpretados, mas monitora-se a atividade dos programas em execução e são identificadas as áreas mais intensamente usadas pelo código. Quando o programa é executado, a JVM compila para código nativo e otimiza aquelas áreas mais usadas. O restante do código não usado intensamente é usado como bytecode a ser interpretado pela JVM.
- Uma quarta opção é o execution engine implementado em um chip que executa bytecodes Java.

Máquina Virtual Java (Cont.)

- Na maioria das vezes, uma JVM é chamada de interpretador Java. Entretanto, devido às várias formas através da qual os bytecodes podem ser executados este termo pode inadequado. Enquanto Interpretador Java pode ser usado para JVM que interpreta bytecodes, JVM´s também usam outras tecnologias para executar bytecodes.
- Assim, apesar de todo Interpretador Java ser um JVM, nem todo JVM pode ser considerado um Interpretador Java.

Java é case-sensitive

- Java é case-sensitive.
- Isto faz diferença ao se nomear um arquivo.
- Um arquivo de nome “ ” terá uma classe associada de mesmo nome “ ”.
- Ao compilar o arquivo através do comando: o compilador solicitará ao DOS para abrir o arquivo. Será dada uma mensagem de erro informando que o arquivo não foi encontrado.

Compilando e Executando Programas em Java

- Para compilar um programa em Java utilize o comando:

- O programa é o compilador Java.
  é compilado para o arquivo
- Para executá-lo:
- O programa java é o interpretador de ***bytecodes*** que o compilador colocou em

Parte 1 - Declarações, Variáveis, e Expressões

Declarações

```
                .    .*;

      =1;

S       .    .          (
   E                        +          );

 .       F                  =        ;
```

Bloco de Declarações

```
  ( >0)
//
          ;
   =   /   + 2* ;
  --;
//
```

Comentários

```
    ( >0)
/*
  C             :

*/
  //
            ;
     =   /   + 2* ;
    --;
  //
```

Declarando Variáveis

```
            C       ; //

      , , ; //
S           ,           ;

      =4; //
S          N     = M       ;
          I     ,     P   = 38;

/*
                      */
      =4,   =5,   =6;
```

Tipos de Variáveis (1)

```
// O

      ;    // 8       ,   -128    127
       ;   // 16       , -32.768    32767
     ;     // 32       , -2.14    10^9    2.14    10^9
      ;    // 62       , -9.22    10^18    9.22    10^18

        ; // 32        IEEE 754,
        ; // 64        ,

      ;    // 16           UNICODE (65.536)

          ; //
```

Tipos de Variáveis (2)

```
// N

                .      .D    ; //
                .     .F    ;   //                    F

S               ;

F           F      ;

D             D    ;
```

Tipos de Variáveis (3)

```
// A          (       )

                    ; //
              V     ; //
                T       ; //

//
              U V       =           10   ;
S                   =       S       100   ;

//
                    =   9, 13, 15, 16, 20, 23   ;
```

Atribuindo Valores Variáveis

```
//     -     =

                    ; //
          I       ; //
S         N    ; //

//                                  10
            =           100 ;
            5   = 10;

     I        =   M ;
   N     = (    C        (     I      )).  S     ()
             + "     ";
```

Literais (1)

```
//

     = 4;          //
     = 4.333;      //                    (        )
     = 4.33 45;    //
     = 4.333F;     //

//
   =       ;
   =         ;
```

Literais (2)

```
//
   =      ; //
   =     ; //              (                  )
   =     ; //
   =     ; //                 (       )
   =     ; //                          (                    )
   =     ; //                (                 )
   =     ; //
   =     ; //                        (                   )
   =     ; //                        (                  )
   =        ; //
   =        ; //
   =           ; //                 UNICODE
   =    2122; //                    UNICODE ™
```

Literais (3)

```
// E

 1 =    ; //

 2 =   S                                ;

 3 =   S                    S                    ;

 4 =   U                UNICODE     J      2122 ;
```

Declarando Constantes

```
// E

            PI = 3.141592;

              DEBUG =       ;

          LEFT   = 0;
          RIGHT  = 1;
          CENTER = 2;
```

Expressões Aritméticas

```
// A
   = 2 + 4; //
   = 4 - 2; //
   = 3 * 4; //
   = 5 / 2; //
   = 5 % 2; //

// A
   +=   ; //    =    +   ;
   -=   ; //    =    -   ;
   *=   ; //    =    *   ;
   /=   ; //    =    /   ;
```

Incrementando e Decrementando

```
// I                                    -

  = 1; //
  =  ++; //          2;          1
  =  --; //          1;          2

// I

  = 1; //
  = ++ ; //          2;          2
  = -- ; //          1;          1
```

Expressões Lógicas (1)

```
// C
   == 3; //          (                    )
   != 3; //               (                    )
   > 3; //
   < 3; //
   <= 3; //
   >= 3; //

// O
   =   1  &&   2; // AND
   =   1       2; // OR
   =  !  1;       // NOT
```

Expressões Lógicas (2)

```
// O                          (          )
   =    & 0; // AND              (0)
   =       ; // OR             (-32,768)
   =       ; //  OR             (0)
   =    << 4;//        4              S
   =    >> 3;//        3              S
   =    >>> 2;//
   =    ;      //
   <<=   ;     //     =    <<   ;
   >>=   ;     //     =    >>   ;
   >>>=   ;    //     =    >>> ;
   &=   ;      //     =    &   ;
    =   ;      //     =        ;
    =   ;      //     =        ;
```

Operadores de Precedência

```
//

.      ()     ++ --  !

      (type)exp     * / %     + -

<< >> >>>     < > <= >=     == !=

&                &&              (      )?       :

= += -= *= /= %=   = &=   = <<= >>= >>>=
```

Parte 2.1 - Tipos Estruturados de Dados

Tipos de Dados Estruturados

Tipos construídos como agregados de outros tipos, normalmente chamados de . Componentes podem ser tipos primitivos ou outro tipo estruturado.

Especificação: o que é o tipo e quais operações podem ser feitas sobre ele. Ex., uma fila com Enqueue e Dequeue.

Implementação: como o tipo será representado e codificado. Uma fila representada sobre um arranjo unidimencional circular.

Registros

Já vimos arranjos. Vamos ver registros. Um registro é um tipo estruturado composto de um número fixo de componentes.

Ex. em C

```
struct E        T
             ID;
                  ;
                      ;
                          ;
        E         ;
```

Registros Variáveis

Registro variável é um tipo estruturado que representam tipos de dados similares mas com alguma(s) variante.

Ex. em Pascal

```
type      D P          = (        ,            );

var            : record
          ID:          ;
              :      ;
                :          ;
          case    :     D P          of
                     : (        M      :      ;
                                I      :         );
                        : (      H    :      ;
                                   :         ;
                                 E      :         );
     end;
```

Em Java

Em Java, assim como em outras Linguagens OO, o conceito de registro é sobrescrito pelo conceito de Objeto.

Parte 2.2 - Trabalhando com Objetos em Java

Criando Objetos

```
//  D                O

S                =       S       ();

C                =       C       ();
```

O quê o faz

- A classe chamada é instanciada
- Memória é alocada
- O **c** (constructor) da classe é chamado para o objeto

C é um método especial para criar e inicializar novas instâncias da classe. Construtores podem ser sobrecarregados.

Acessando Valores de Variáveis de Instância

```
// L

        =     .        ; // O     meu        C

      C       =     .      .                ;

// E

   .          = 1;

   .      .                  =        ;
```

Exemplo: acessando variáveis

```
        .   .P    ;

   T    P


                          (S             )
P         P      =      P     (10,10);

S      .    .          (        +     P     . );
S      .    .          (        +     P     . );

    P     .   = 5;
    P     .   = 15;
S      .    .          (        +     P     . );
S      .    .          (        +     P     . );
```

Variáveis de Classe

```
      M      D F

         S                  ; //

  S                N    ; //
           ;
  ...


...
//
M      D F                =      M      D F        ();
       .      N     =   M          ;
M      D F              =      M      D F        ();
S      .    .          (   .          );
```

Chamando Métodos(1)

```
//

//
   O      .       U ( 1, 2, 3);
//
   O      .       D   ();


//
   O      .       T   ().        ();

//     1
   O      .   1.        ();
```

Chamando Métodos(2)

```
    T    S

                         (S             )
  S          =  B     T        A       ;

  S     .   .       (     =   +    );
  S     .   .       ( T       =   +    .      ());
  S     .   .       ( S           6   10 =    +
                            .         (6,10));
  S     .   .       (                    =   +
                            .  U    C   ());



//
```

str = Bahia Terra da Alegria
Tamanho = 22
Substring de 6 a 10 = Terra
str em maiusculas = BAHIA TERRA DA ALEGRIA

Métodos de Classe

```
//
S        1, 2;

 2 =                    ;

 2 =  1.      O (5);
 2 = S      .      O (5);

/*
          O                                        S
```

Explicação do método valueOf usado na página anterior

blic a ic S i g al eOf(i i)

Método de classe (da classe String) que retorna uma string representando o argumento int. A representação é exatamente a retornada pelo método Integer.toString

Métodos similares: valueOf(boolean); valueOf(char); valueOf(char[]); valueOf (char[],int,int); valueOf(float); valueOf(double); valueOf(long); valueOf(Object)

Referências para Objetos

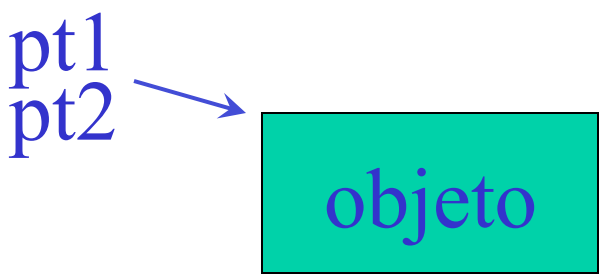
```
// A
      T    R
                              (S              )
    P        1,    2;

      1 =      P     (100,100);
    pt2 = new Point(100,100);

      2 =    1;

      1.   = 200;
      1.   = 200;
    S      .    .          ( P1:    +    1.   +   ,    +    1. );
    S      .    .          ( P2:    +    2.   +   ,    +    2. );


//
P1: 200, 200
P2: 200, 200
```



Conversão (Casting) de Tipos

```
// C
        , ;
     ;
...
  = (    ) ( / );

// C                  O
J    M       1;
J       2;
...
 1 =       J    M    ();
 2 = (J    ) 1;
```

j1 e j2 têm que estar na mesma árvore hierárquica

Comparando Objetos

```
J       1, 2;
              1,      2;

 1 =        J    ();
 2 =   1;

     1 = ( 1 ==   2); //
     2 = ( 1 !=   2); //
```

1 é verdadeiro 2 é falso

Comparando Strings

```
S           1,    2;
              1,      2;


   1 =   J     ;
   2 =   J     ;

    1 = (    1 ==     2); //
    2 =     1.       (    2); //
```

1 é falso 2 é verdadeiro

Determinando a Classe de Um Objeto

```
/* O              C
       O
C    . O            N    ()
       C                                  . */

S          1 =     .    C     ().    N    () ;

/* O                    O

                   . */

              =    1          O   S      ;
```

Principais Bibliotecas de Classes (1)

- . : Classes que contêm a linguagem propriamente dita. Inclui a classe O , a classe S , a classe S , e classes especiais para os tipos primitivos - I , C , F , etc.

- . : Classes de utilidade. Inclui classes como: D , V , H .

- . : Classes de entrada e saída de dados que escrevem e lêem fluxo de dados em arquivos, ou entrada e saída padrão, entre outras..

Principais Bibliotecas de Classes (2)

- . : Classes para suporte de comunicação de dados (networking), inclui as classes `S` e `URL` (suporta referências a documentos na WWW).

- . : Classes para implementar interfaces gráficas. Inclui classes como: `W` , `M` , `B` , `F` , etc.

- . : Classes para implementar applets Java. Inclui a classe `A` entre outras.

Parte 3 - Fluxo de Controle

Condicional

```
//
     ( > )
   //
    ;

//
     ( > )
   //

   //
    ;
```

Operador Condicional

```
//
//          ?              T     :              F

// E        :
//
        =   <   ?     :   ;
```

Condicional

```
        (    D        )
              1:
     //         D          ==        1

        ;
              2:
     //         D          ==        2

        ;
  ...
            :
     //

        ;
;
```

D pode ser dos tipos , , ,

Ciclos

```
// E
    (inicialização; teste; incremento)

   bloco de execução


// E
    ( =0;  <100;  ++)

   bloco de execução


// E
    ( =0;  <100;  ++);
   ++;
```

Ciclos

```
// E
        (condição)

  bloco de execução


// E
  = 0;
        ( <          1.        )

   bloco de execução
    ++;
```

Ciclos

```
// E

  bloco de execução
        (condição)

// E
  = 0;

  bloco de execução
   ++;
        ( <10)
```

Saindo de ciclos usando

```
// E
  = 0;
       ( <          1.        )

   bloco de execução
       (          1     == 0)
            ;

            2     =           1   ++ ;
```

Saindo de ciclos aninhados

```
// E
   :
      (      =1; <=5; ++)    //          1
         (      =1; <=5; ++)    //          2
          (  +    > 5)
                  ;



   //                  (pulando dois ciclos)
```

Parte 4 - Criando Classes e Aplicações em Java

Definindo Classes

```
// E
       M     C
  bloco de execução


// C
       M     C                    M     S     C
  bloco de execução


// C
       M     C                          U   I
  bloco de execução
```

discutida mais tarde

Definindo Métodos

```
// E
tipoRetornado     M       (tipo    1,
                            tipo    2,
                            ...,
                            tipo    N)
  bloco de execução
```

tipoRetornado é um tipo primitivo ou uma classe. Os argumentos `(  1,  2, ...,  N)` são passados por valor se são primitivos e por referência se são arrays ou objetos. Os argumentos serão variáveis locais ao método.

Criando Uma Aplicação

```
// E
       M     A

  bloco de execução

                                  (S                )
    bloco de execução
```

C m ila d :

```
       M     A           .
```

R da d :

```
      M     A             C
      M     A           C
```

Exemplo de Aplicação

```
// E
      E   A
                          (S            )
       (       =0;  <     .      ;  ++)
    S     .   .        ( A           +   +  :  +
                              );
```

Rodando a Aplicação Exemplo

C m ila d :

```
       E    A    .
```

R da d :

```
      E    A       E
```

```
Argument 0: Ecoa um argumento
```

```
      E    A     E
```

```
Argument 0: Ecoa
Argument 1: tres
Argument 2: argumentos
```

Overloading Métodos (1)

Um exemplo

```
            .   .P    ;

// C       M  R
      M  R
       1=0,  1=0,  2 = 0,  2 = 0;

  M  R       R  (P         E  , P          D  )
     1 =     E  . ;
     1 =     E  . ;
     2 =      D  . ;
     2 =      D  . ;
              ;


  M  R       R  (     1,      1,      2,      2)
        . 1 =  1;
        . 1 =  1;
        . 2 =  2;
        . 2 =  2;
              ;
```

Overloading Métodos (2)

Um exemplo

```
M  R        R  (P          E  ,      ,      )
   1 =      E  . ;
   1 =      E  . ;
   2 =  1 +  ;
   2 =  1 +  ;
             ;


          R  ()
   S      .   .      ( R  : <  +  1 +  ,  +  1);
   S      .   .        ( ,  +  2 +  ,  +  2 +  > );
```

Overloading Métodos (3)

```
                                   (S                 )
     M   R          =       M   R     ();

     S       .    .         ( C               R   (25,25,50,50) );
         .     R   (25,25,50,50);
         .         R   ();
     S       .    .         ( -------------------------- );

     S       .    .         ( C               R   (P(10,10),P(20,20)) );
         .     R   (     P     (10,10),     P     (20,20));
         .         R   ();
     S       .    .         ( -------------------------- );

     S       .    .         ( C               R   (P(10,10),50,50) );
         .     R   (     P     (10,10),50,50);
         .         R   ();
     S       .    .         ( -------------------------- );


  //                                    M   R
```

Rodando o Exemplo

```
C m ila d :
       M   R   .

R da d :
      M   R

Chamando consRet(25,25,50,50)
Ret: <25,25,50,50>
-------------------------
Chamando consRet(P(10,10),P(20,20))
Ret: <10,10,20,20>
-------------------------
Chamando consRet(P(10,10),50,50)
Ret: <10,10,60,60>
-------------------------
```

Métodos Construtores

M d c (c c) é um método especial que determina como um objeto é inicializado quando a classe é instanciada. Métodos construtores têm o mesmo nome da classe e não retornam nenhum valor. Métodos construtores podem ser sobrecarregados (overloaded).

Exemplo Anterior com Construtores

(1)

```
            .    .P     ;

/* C        M   R    2

       M   R   2
         1=0,   1=0,   2 = 0,   2 = 0;

  M   R   2(P           E   , P           D   )
      1 =       E    . ;
      1 =       E    . ;
      2 =         D    . ;
      2 =         D    . ;


  M   R   2(      1,        1,      2,        2)
          . 1 =   1;
          . 1 =   1;
          . 2 =   2;
          . 2 =   2;
```

Exemplo Anterior com Construtores

```
M  R  2(P          E  ,      ,      )
   1 =     E  . ;
   1 =     E  . ;
   2 =  1 +  ;
   2 =  1 +  ;


            R  ()
  S     .    .      ( R  : <  +  1 +  ,  +  1);
  S     .    .        ( ,  +  2 +  ,  +  2 +  > );
```

(2)

Exemplo Anterior com Construtores

```
                              (S              )
   M   R   2     ;

   S      .    .          ( C        M   R   2(25,25,50,50) );
       =       M   R   2(25,25,50,50);
      .          R   ();
   S      .    .          ( ------------------------- );

   S      .    .          ( C        M   R   2(P(10,10),P(20,20)) );
       =       M   R   2(     P     (10,10),     P     (20,20));
      .          R   ();
   S      .    .          ( ------------------------- );

   S      .    .          ( C        M   R   2(P(10,10),50,50) );
       =       M   R   2(     P     (10,10),50,50);
      .          R   ();
   S      .    .          ( ------------------------- );


//                                 M   R
```

(3)

Rodando o Exemplo

```
C m ila d :
        M   R   .

R da d :
       M   R

Criando MeuRet2(25,25,50,50)
Ret: <25,25,50,50>
-------------------------
Criando MeuRet2(P(10,10),P(20,20))
Ret: <10,10,20,20>
-------------------------
Criando MeuRet2(P(10,10),50,50)
Ret: <10,10,60,60>
-------------------------
```

Sobrescrevendo Métodos (1)

Um exemplo

```
   C      I
    =0,   =1;

      I        ()
S       .    .          (          +    +  ,          +  );
S       .    .          ( S                                  +
                              .    C     ().    N    ());



   S  C      I                    C      I
    =3;

                           (S                )
S  C      I                =      S  C      I        ();
    .  I        ();
```

Rodando o Exemplo

C m ila d :

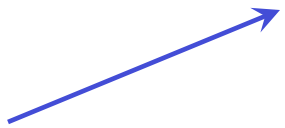
```
C      I
S  C      I
```

R da d :

```
S  C      I
```

X é 0, Y é 1
Sou uma instância da classe SubClasseImprima

isso compilaria
C I
de qualquer forma



Sobrescrevendo Métodos (2)

```
   S  C     I      2          C     I
    =3;

      I       ()
S      .   .        (       +   +  ,       +   +
                        ,       +  );
S      .   .        ( S                          +
                          .   C    ().   N   ());


                        (S             )
S  C     I       2      =     S  C     I      2();
   .  I       ();
```

Rodando o Novo Exemplo

C m ila d :

```
S  C     I      2.
```

R da d :

```
S  C     I      2
```

X é 0, Y é 1, Z é 3
Sou uma instância da classe SubClasseImprima2

Chamando o Método Original

```
// E

        M       (S        , S        )
  bloco de comandos
       .   M      ( , );
  bloco de comandos
```

Sobrescrevendo um Construtor

(N c e de e dade i el)

```
          .   .P    ;

    N    P              P

S          ;

N    P    (     ,      , S          )
  // I          P
       ( , );
  // I
       .     =     ;
```

Método Finalizador

M d fi ali ad é o oposto do método construtor, ele é chamado logo antes do objeto ser destruído pela coleta de lixo. A classe Object define um método finalizador vazio. Você pode sobrescrevê-lo fazendo:

```
()
  bloco de comandos para limpar ações do objeto
```

Você pode também usar o método . () para pedir a super classe para limpar as suas ações também.