Java – Aula 01

Introdução

01/08/2012

Celso Olivete Júnior

olivete@fct.unesp.br

Professor Celso Olivete Júnior

- Bacharelado em Ciência da Computação (Unoeste-2002)
- Mestrado e Doutorado em Engenharia Elétrica – Área: Visão Computacional (USP-SC-2005/2009)
- Áreas de interesse e atuação:

  processamento de imagens médicas

  desenvolvimento Web

  inteligência artificial e lógica *fuzzy*

  teoria da computação

  compiladores

Site do Curso

www.fct.unesp.br/docentes/dmec/olivete/java

- slides, exercícios, notas e demais materiais estarão disponíveis no site

- Envio de trabalhos e dúvidas através do email

olivete@fct.unesp.br

Objetivo

- Prover o aluno de uma visão geral dos recursos oferecidos na plataforma Java e proporcionar condições para o aluno implementar aplicações nesta plataforma.

- Foco na TECNOLOGIA!!!

Programa do Curso

- Introdução ao Java
- Tipos primitivos e seus operadores
- Estruturas de controle
- Introdução à orientação a objetos
- Definição de classes e manipulação de objetos
- Encapsulamento
  - Modificadores de acesso
- Componentes de software
  - Javabeans

- Arrays
- Herança
- Polimorfismo
  - Amarração estática e dinâmica
- Exceções e asserções
- Classes abstratas e interfaces
- Padrões de projeto
- *Frameworks*
- Coleções
- Entrada e saída
- Serialização de objetos e persistência
- Metaclasses
- Interfaces Gráficas

Bibliografia básica

• DEITEL, H.M. & DEITEL, P.J. - Java : como programar, Porto Alegre : Bookman, 2003.

• HORSTMANN, C.S. & CORNELL, G. - Core java 2, volume I, São Paulo : Makron Books, 2001.

• HORSTMANN, C.S. & CORNELL, G. - Core java 2, volume II, São Paulo : Makron Books, 2001.

• CHAN, M.C., GRIFFITH, S.W. & IASI, A.F. - Java 1001 dicas de programação, São Paulo : Makron Books, 1999

Bibliografia básica

• DEITEL, H.M. & DEITEL, P.J. - Java : como programar, Porto Alegre : Bookm

• HORSTMANN, C.S L, G. - Core java 2, volume I, São Paulo : s, 2001.

• HORSTMANN, C.S L, G. - Core java 2, volume II, São Paulo ks, 2001.

• CHAN, M.C., GRIFFITH, S.W. & IASI, A.F. - Java 1001 dicas de programação, São Paulo : Makron Books, 1999

Metodologia

- Aulas expositivas teórico-práticas
- Exercícios práticos
- Projetos individuais

Avaliação

- A cada bimestre
  - Uma prova: NP
  - Trabalhos INDIVIDUAIS: MT
  - MB = (7*NP + 3*MT)/10 **SE E SOMENTE SE** (NP>=5 **E** MT>=5)
    - Caso contrário (MT<5 OU NP<5)
      - MB = Menor Nota (NP ou MT)
  - Onde:
  - NP = Nota da prova
  - MT = Média dos trabalhos
  - MB = Média do Bimestre
- A nota final (NF) do aluno no curso será a média das notas obtidas nos 2 bimestres
- Caso o aluno não obtenha a nota mínima para aprovação, será oferecida uma terceira avaliação (EXAME) e a Média Final passa a ser obtida pela expressão:
  - NFnova = (NFanterior + EXAME)/2

Atividades práticas INDIVIDUAIS

- serão parte da avaliação;
- deverão ser entregues no PRAZO;
- a não entrega no prazo acarretará em diminuição gradativa do valor máximo.

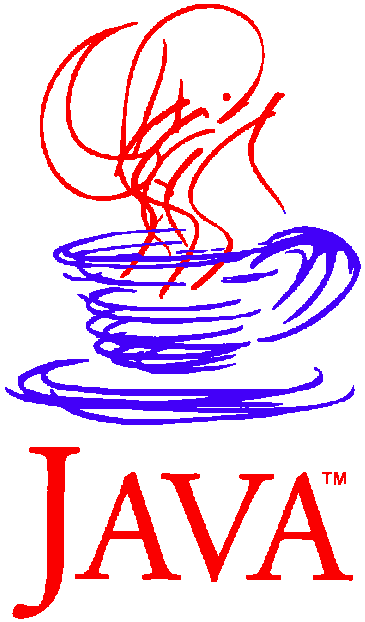
Avaliação

- A "cola" ou plágio em provas, exercícios ou atividades práticas implicará na atribuição de nota zero para todos os envolvidos. Dependendo da gravidade do incidente, o caso será levado ao conhecimento da Coordenação e do Conselho do Departamento, para as providências cabíveis. Na dúvida do que é considerado cópia ou plágio, o aluno deve consultar o professor antes de entregar um trabalho.

Ferramentas

- Netbeans → facilidades:
  - desenvolvimento da interface gráfica
  - modelagem
  - ...

Introdução – Tecnologia JAVA: o que é?



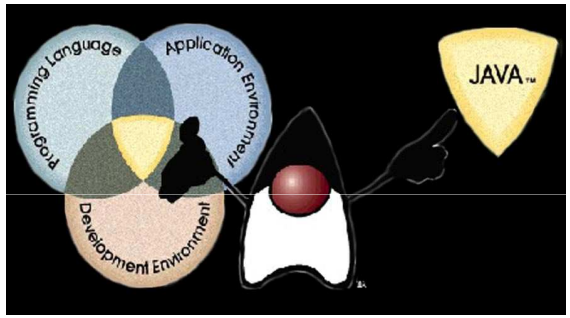
- Baseado na linguagem C++
- Uma linguagem de programação orientada a objetos
- Uma coleção de APIs (classes, componentes, *frameworks*) para o desenvolvimento de aplicações multiplataforma

Introdução – Tecnologia JAVA: o que é?

- Um ambiente de execução presente em browsers, mainframes, SOs, celulares, palmtops, cartões inteligentes, eletrodomésticos

Introdução – Tecnologia JAVA: o que é?

- Fornece:
  - segurança de tipos: linguagem tipada
  - concorrência: suporte a *threads*
  - modularidade: classes e packages
  - robustez: sistema de tratamento de exceções



- Aplicação de princípios de LP + OO
  - classes encapsulam dados e operações (TADs)
  - classes compiladas separadamente

Tecnologia JAVA: importância

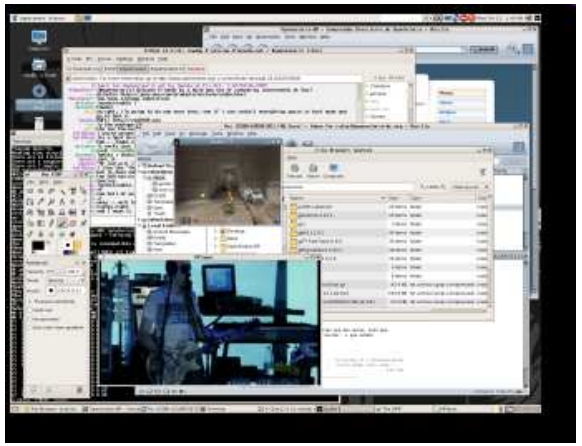
- **Java é essencial:**
  - Acesso remoto a bancos de dados
  - Bancos de dados distribuídos
  - Comércio eletrônico na WWW
  - Interatividade em páginas WWW
  - Interatividade em ambientes de Realidade Virtual distribuídos
  - Gerência de Documentos
  - Integração entre dados e forma de visualização
  - Ensino à distância
  - Jogos e entretenimento

Position Jul 2012	Position Jul 2011	Delta in Position	Programming Language	Ratings Jul 2012	Delta Jul 2011	Status
1	2	↑	C	18.331%	+1.05%	A
2	1	↓	Java	16.087%	-3.16%	A
3	6	↑↑↑	Objective-C	9.335%	+4.15%	A
4	3	↓	C++	9.118%	+0.10%	A
5	4	↓	C#	6.668%	+0.45%	A
6	7	↑	(Visual) Basic	5.695%	+0.59%	A
7	5	↓↓	PHP	5.012%	-1.17%	A
8	8	=	Python	4.000%	+0.42%	A
9	9	=	Perl	2.053%	-0.28%	A
10	12	↑↑	Ruby	1.768%	+0.44%	A
11	10	↓	JavaScript	1.454%	-0.79%	A
12	14	↑↑	Delphi/Object Pascal	1.157%	+0.27%	A
13	13	=	Lisp	0.997%	+0.09%	A
14	15	↑	Transact-SQL	0.954%	+0.15%	A
15	25	↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	0.917%	+0.43%	A
16	16	=	Pascal	0.837%	+0.17%	A
17	19	↑↑	Ada	0.689%	+0.14%	B
18	11	↓↓↓↓↓↓	Lua	0.684%	-0.89%	B
19	21	↑↑	PL/SQL	0.645%	+0.10%	A--
20	26	↑↑↑↑↑↑	MATLAB	0.639%	+0.19%	B

Fonte:http://www.tiobe.com **07/2012**

JAVA: produtos e APIs

- Java SE
  - Ambiente de desenvolvimento Java mais utilizado
  - Composição: ferramentas e APIs essenciais para qualquer aplicação Java (inclusive GUI)
  - Seu uso é voltado a PCs e servidores, onde há bem mais necessidade de aplicações
  - Por ser a plataforma mais abrangente do Java, o J2SE é a mais indicada para quem quer aprender a linguagem



JAVA: produtos e APIs

•Java EE

- •Composição: ferramentas e APIs para o desenvolvimento de aplicações distribuídas
- •Plataforma Java voltada para redes, internet, intranets e afins
- •Contém bibliotecas especialmente desenvolvidas para o acesso a servidores, a sistemas de e-mail, etc
- •A plataforma J2EE contém uma série de especificações, cada uma com funcionalidades distintas. Entre elas, tem-se:

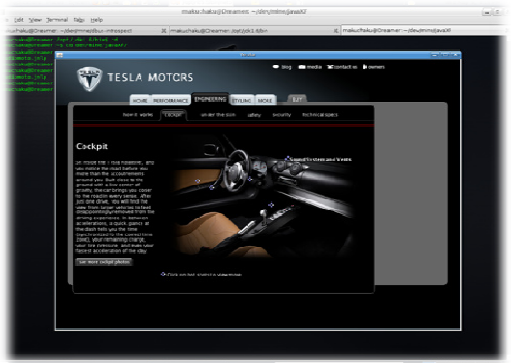
- •JSP (Java Server Pages) e
- •Servlets, para o desenvolvimento de aplicações Web

JAVA: produtos e APIs

- Java ME
  - Composição: ferramentas e APIs para o desenvolvimento de aplicações para aparelhos portáteis: celulares, PDAs, etc.

JAVA: produtos e APIs

- Java Fx
  - Plataforma de software multimídia baseada em Java para a criação e disponibilização de interface que pode ser executada em dispositivos diferentes.
  - A versão atual (JavaFX 2.0, 2011) permite a criação para desktop, browser e telefone celulares. TVs, videogames, Blurays players e outras plataformas estão sendo planejadas para serem adicionadas no futuro.



JAVA: ambiente de execução e desenvolvimento

- Java System Development Kit (JDK)
  - Coleção de ferramentas de linha de comando para, entre outras tarefas, compilar, executar e depurar aplicações Java.
  - Para habilitar o ambiente via linha de comando é preciso colocar o caminho $JAVA_HOME/bin no PATH do sistema.

- Java Runtime Environment (JRE)
  - Tudo o que é necessário para executar aplicações Java.

Relembrando...Implementação de LP

- Compilação:
  - geração de código executável
  - depende da plataforma de execução
  - tradução lenta X execução rápida

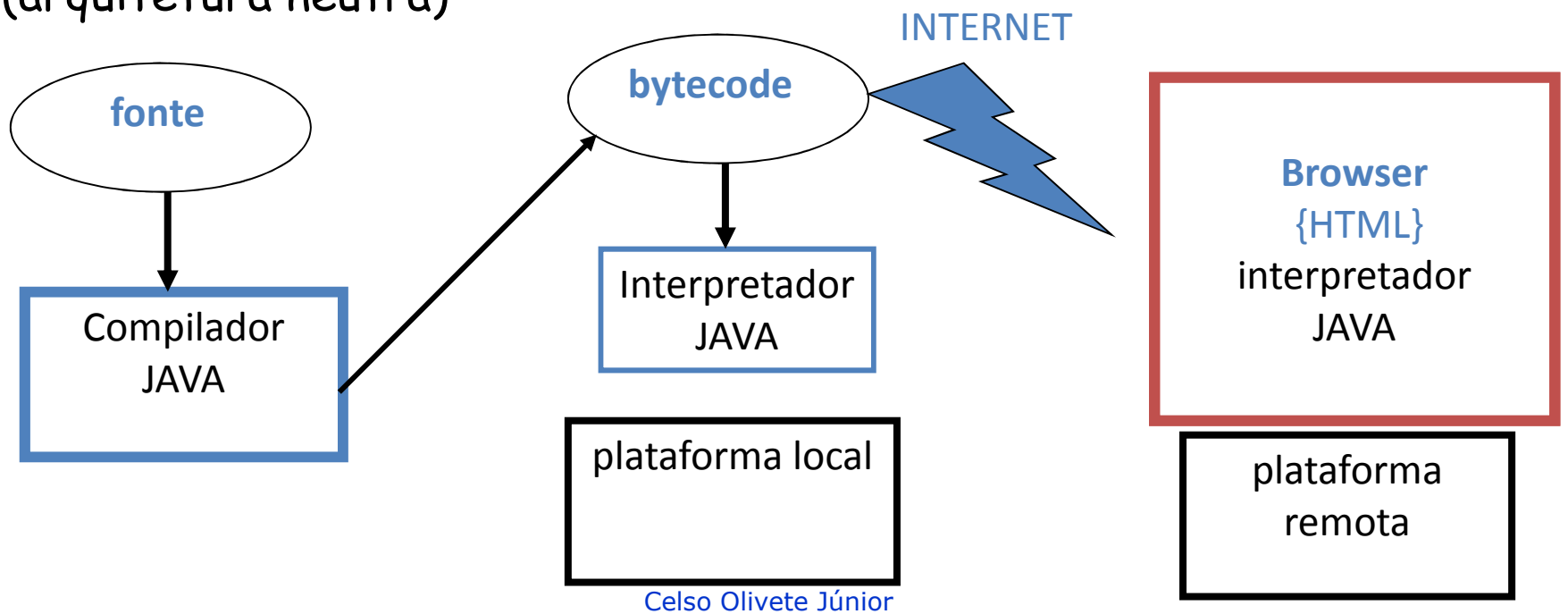
- Interpretação pura
  - sem geração de código
  - execução lenta, independente de plataforma

- Híbrida
  - geração de código intermediário
  - independente de plataforma de execução
  - tradução rápida X execução não muito rápida

Ambiente de compilação e execução (Funcionamento)

• Compilação: Java é compilado para um código intermediário conhecido como ***bytecode***

•Execução: ***Bytecodes*** são interpretados pela JVM (Just-In-Time/Cache) - *Java Virtual Machine* executada no ambiente hospedeiro (arquitetura neutra)



Compilação para *bytecode:* Exemplo

Código Java (texto)

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

HelloWorld.java

Compilação (javac)

Uma "classe" Java

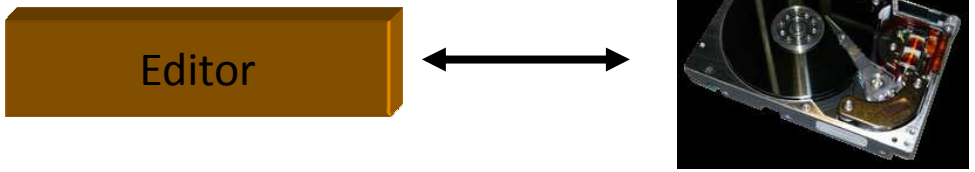
Bytecode Java (código de máquina virtual)

```
F4 D9 00 03 0A B2 FE FF FF 09 02 01 01 2E 2F 30 62 84 3D 29 3A C1
```

HelloWorld.class

Ambiente de desenvolvimento

Fase 1: Edita



O programa é criado em um editor e armazenado no disco em um arquivo .java

Ambiente de desenvolvimento

Fase 1: Edita

Editor

O programa é criado em um editor e armazenado no disco em um arquivo .java

Fase 2: Compila

Compilador

O compilador cria bytecodes e os armazena no disco em um arquivo .class

Ambiente de desenvolvimento

Fase 1: Edita

Editor

O programa é criado em um editor e armazenado no disco em um arquivo .java

Fase 2: Compila

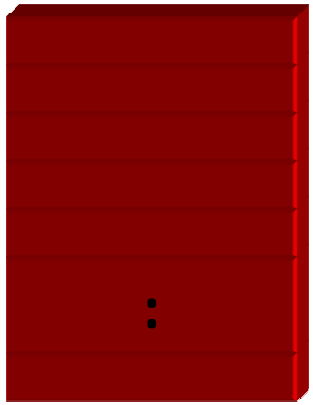
Compilador

O compilador cria bytecodes e os armazena no disco em um arquivo .class

Fase 3: Carrega

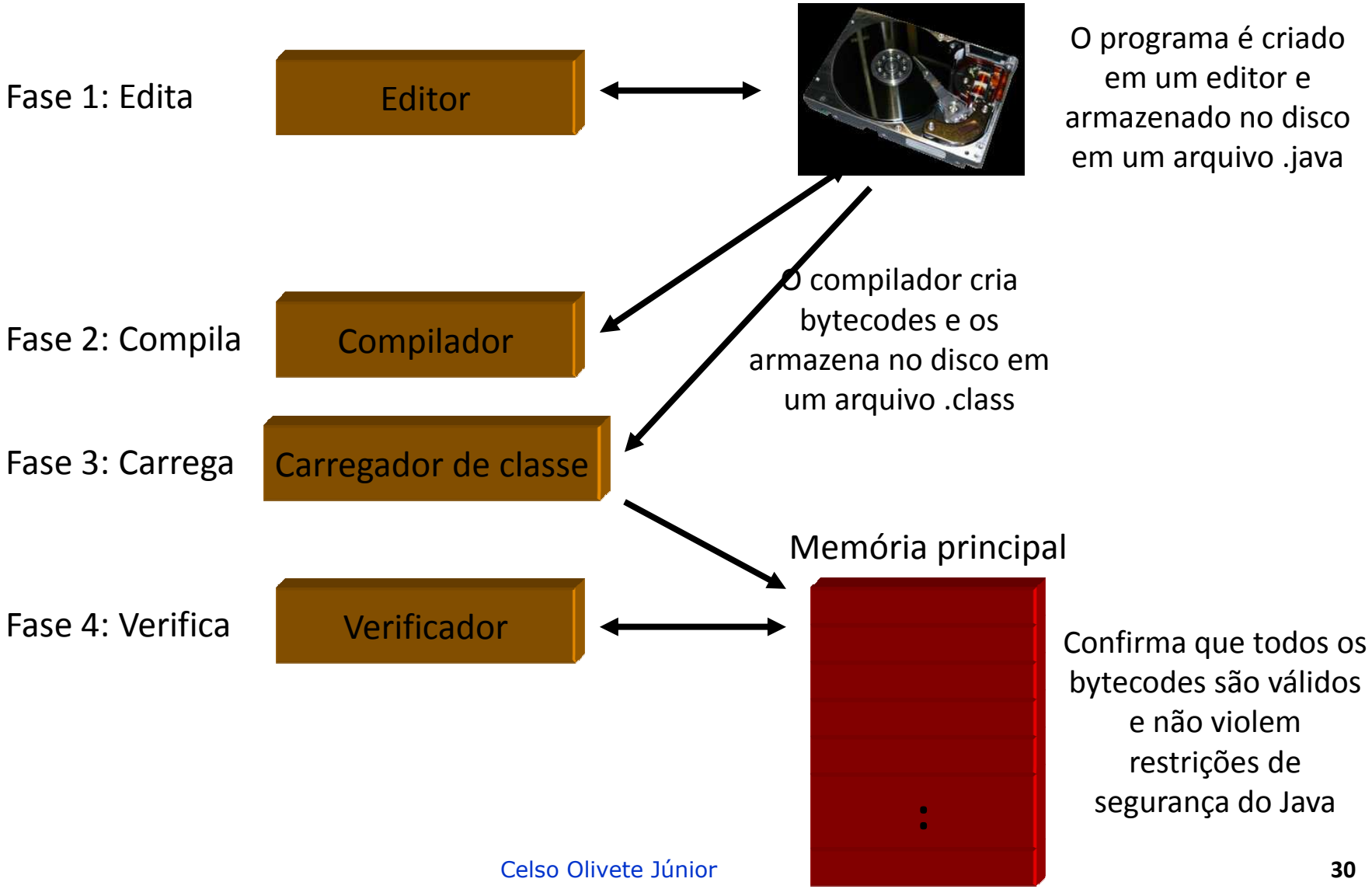
Carregador de classe

Memória principal



Lê arquivos .class que contém bytecodes a partir de um disco e coloca estes bytecodes na memória

Ambiente de desenvolvimento



Ambiente de desenvolvimento

Fase 1: Edita

Editor

O programa é criado em um editor e armazenado no disco em um arquivo .java

Fase 2: Compila

Compilador

O compilador cria bytecodes e os armazena no disco em um arquivo .class

Fase 3: Carrega

Carregador de classe

Memória principal

Fase 4: Verifica

Verificador

Fase 5: Executa

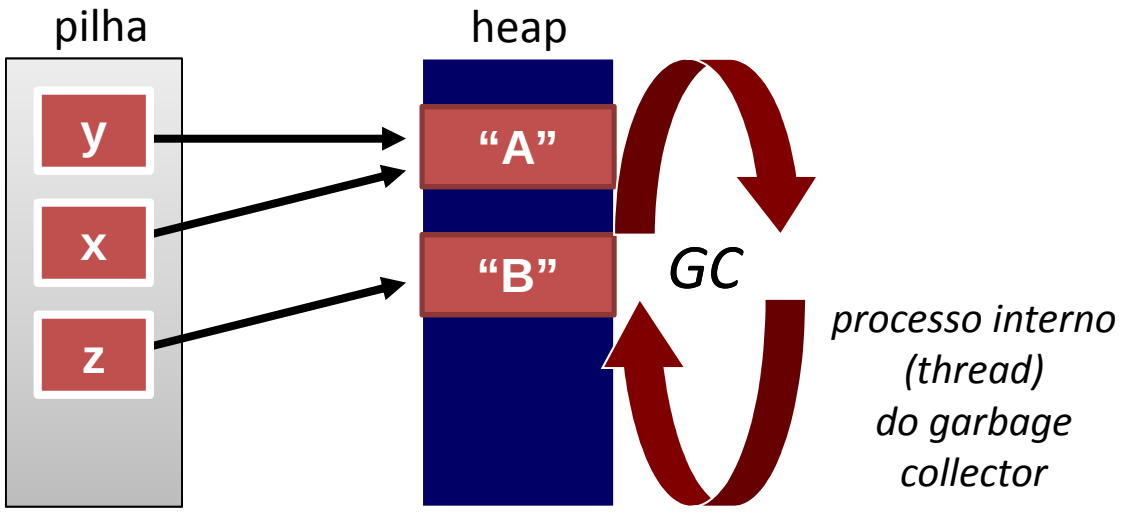
Java Virtual Machine

Para executar o programa, o JVM lê os bytecodes e os traduz para uma linguagem que o computador possa entender

Coleta de lixo

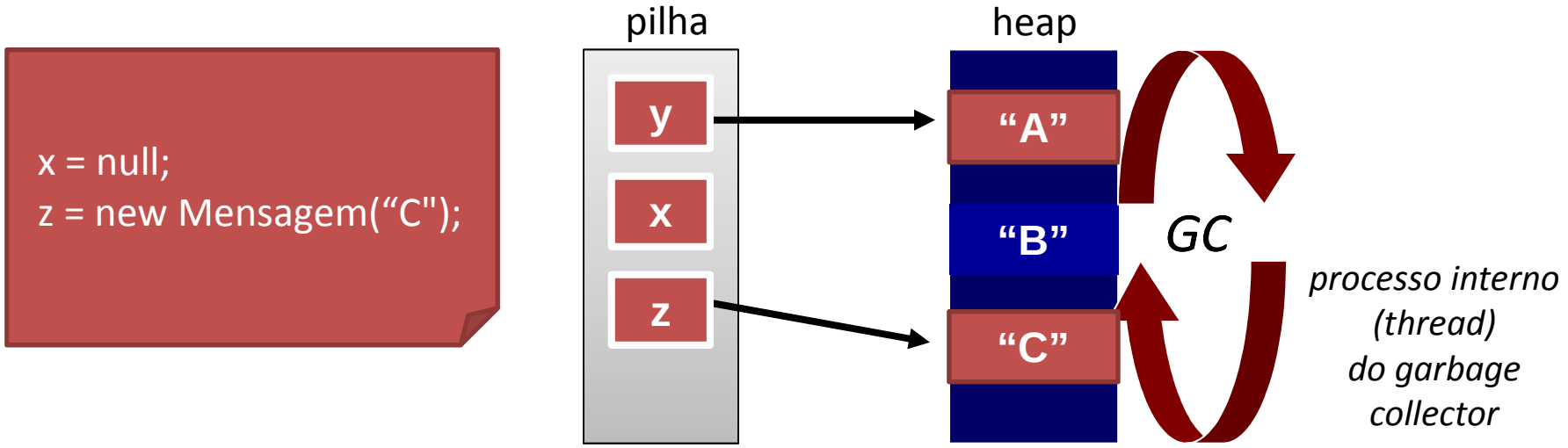
- Memória alocada em Java não é liberada pelo programador
  - Ou seja, objetos criados não são destruídos pelo programador
- A criação de objeto objetos em Java consiste de
  1. Declarar o objeto (pilha)
  2. Inicializar o objeto: alocar memória no heap para armazenar os dados do objeto
  3. Atribuir o endereço de memória alocada ao objeto (referência)
- Mais de uma referência pode apontar para o mesmo

```
Mensagem x, y, z;
y = new Mensagem("A");
x = y;
z = new Mensagem("B");
```



Coleta de lixo

- Quando um objeto não tem mais referências apontando para ele, seus dados não mais podem ser usados, e a memória deve ser liberada.
- O coletor de lixo irá liberar a memória na primeira oportunidade



Principais elementos Java

- Identificadores: nomes de variáveis, de unidades, palavras-chave, palavras reservadas
- Expressões: compostas por operandos e operadores aritméticos, lógicos, strings
- Comandos: compostos por palavras-chave ou símbolos e expressões
- Unidades: agrupamento de comandos
  - comandos compostos e blocos
  - procedimentos, funções, classes, pacotes, etc.
- Programa: contém uma unidade principal onde inicia o processo de execução

Principais elementos Java

- Identificadores: nomes de variáveis, de unidades, palavras-chave, palavras reservadas
- Expressões: compostas por operandos e operadores aritméticos, lógicos, strings
- Comandos: compostos por palavras-chave ou símbolos e expressões
- Unidades: agrupamento de comandos
  - comandos compostos e blocos
  - procedimentos, funções, classes, pacotes, etc.
- Programa: contém uma unidade principal onde inicia o processo de execução

Uma aplicação básica: escrevendo uma linha de texto

// Comentário de uma linha
/* comentário de N linhas */

O arquivo fonte deve ter o mesmo nome da classe e terminar com a extensão .java
Welcome1.java

```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

```
Olá Mundo!
```

Mesmo exemplo na Linguagem C

```
#include <stdio.h>
int main ()
{
    printf("Olá Mundo");
    return 0;
}
```

Uma aplicação básica: escrevendo uma linha de texto

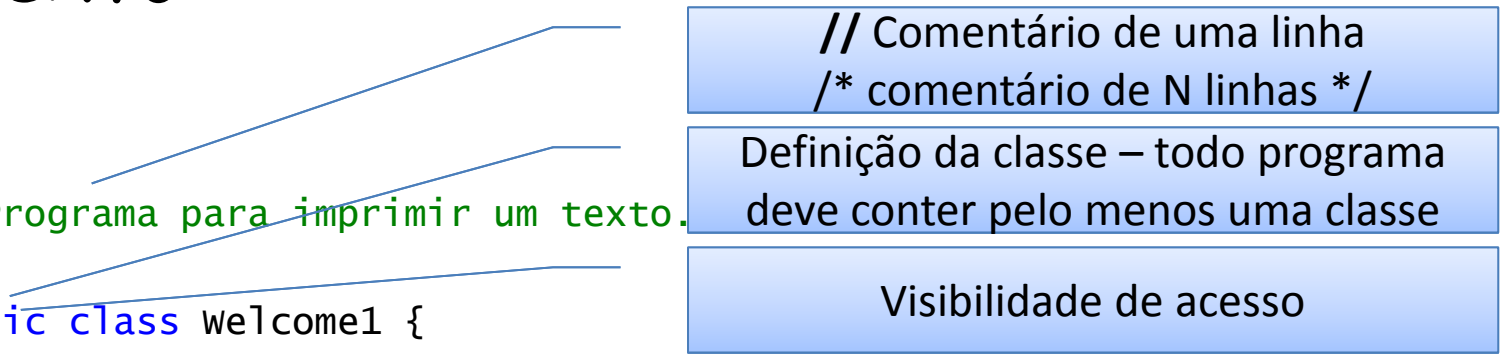
// Comentário de uma linha
/* comentário de N linhas */

Definição da classe – todo programa deve conter pelo menos uma classe

```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

```
Olá Mundo!
```

Uma aplicação básica: escrevendo uma linha de texto




```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

```
Olá Mundo!
```

Uma aplicação básica: escrevendo uma linha de texto

```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

// Comentário de uma linha
/* comentário de N linhas */

Definição da classe – todo programa deve conter pelo menos uma classe

Visibilidade de acesso

Palavra reservada e, em seguida, nome da classe (inicia-se com letra maiúscula por convenção)

```
Olá Mundo!
```

Uma aplicação básica: escrevendo uma linha de texto

```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

```
Olá Mundo!
```

// Comentário de uma linha
/* comentário de N linhas */

Definição da classe – todo programa deve conter pelo menos uma classe

Visibilidade de acesso

Palavra reservada e, em seguida, nome da classe (inicia-se com letra maiúscula por convenção)

Ponto de partida (execução) da aplicação. Método é obrigatório ser **public, static** e **void** (não retorna valor)

Uma aplicação básica: escrevendo uma linha de texto

```
1
2   // Programa para imprimir um texto.
3
4   public class Welcome1 {
5
6    // metodo principal inicia execucao de uma aplicacao java
7    public static void main( String args[] )
8    {
9       System.out.println( "Olá Mundo!" );
10
11   } // final do metodo main
12
13  } // final da classe Welcome1
```

```
Olá Mundo!
```

// Comentário de uma linha
/* comentário de N linhas */

Definição da classe – todo programa deve conter pelo menos uma classe

Visibilidade de acesso

Palavra reservada e, em seguida, nome da classe (inicia-se com letra maiúscula por convenção)

Ponto de partida (execução) da aplicação. Método é obrigatório ser **public, static** e **void** (não retorna valor)

Objeto e método para saída padrão

resultado

Comando de Saída: System.out.println

- caracteres de escape

Sequência de excape	Descrição
\n	Nova linha. Posiciona o cursor de tela no início da próxima linha
\t	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
\r	Retorno. Posiciona o cursor de tela no início da linha atual Não avança para a próxima linha. Qualquer saída de caracteres depois do retorno sobrescreve a saída anterior.
\\	Imprime a Barra invertida
\"	Aspas Duplas. Usado para imprimir o caractere de aspas duplas `System.out.println( "\“teste \"" );` mostrará “teste"

Console: Entrada de dados

- Sequencia de instruções
  - Criação do objeto de entrada de dados

    Scanner <entrada> = new Scanner(System.in);

  - Para cada leitura teclado

    teclado.nextLine()

    - Função nextLine() retorna String

Console: Entrada de dados

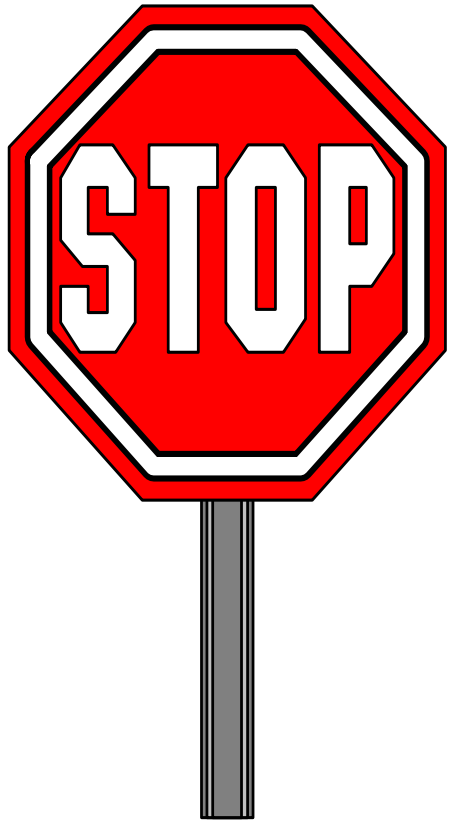
- Exemplo: lê uma idade (string), converte e exibe

```
import java.util.Scanner; //necessário
Scanner teclado = new Scanner(System.in);
System.out.print("Idade:");
int idade = Integer.parseInt(teclado.nextLine());
System.out.println(idade);
```

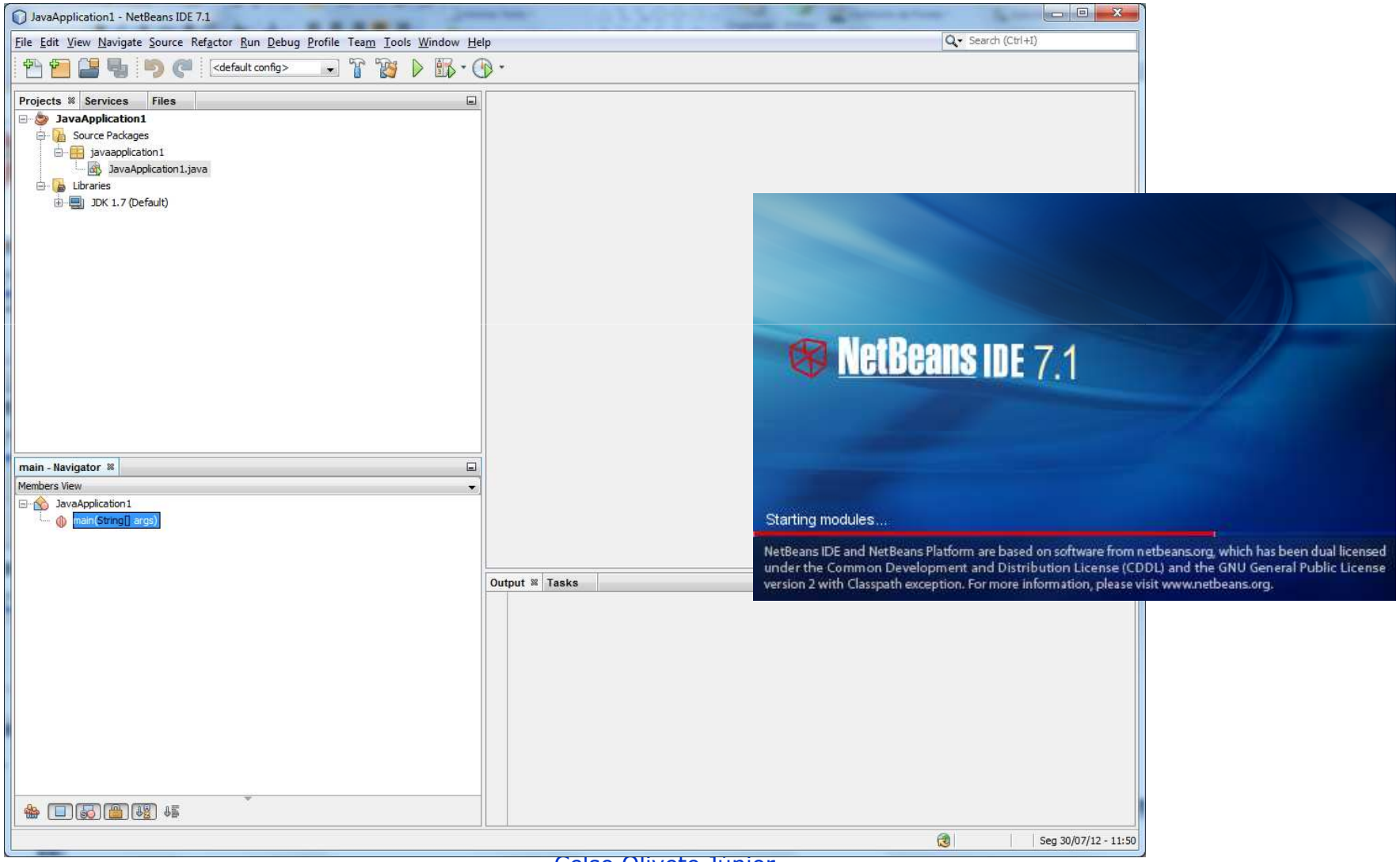
Retorna uma String, necessário realizar a conversão para o tipo apropriado

Exercícios

- Criar um diretório para seus programas
- Localizar o ambiente
- Fazer exemplos (slides anteriores)

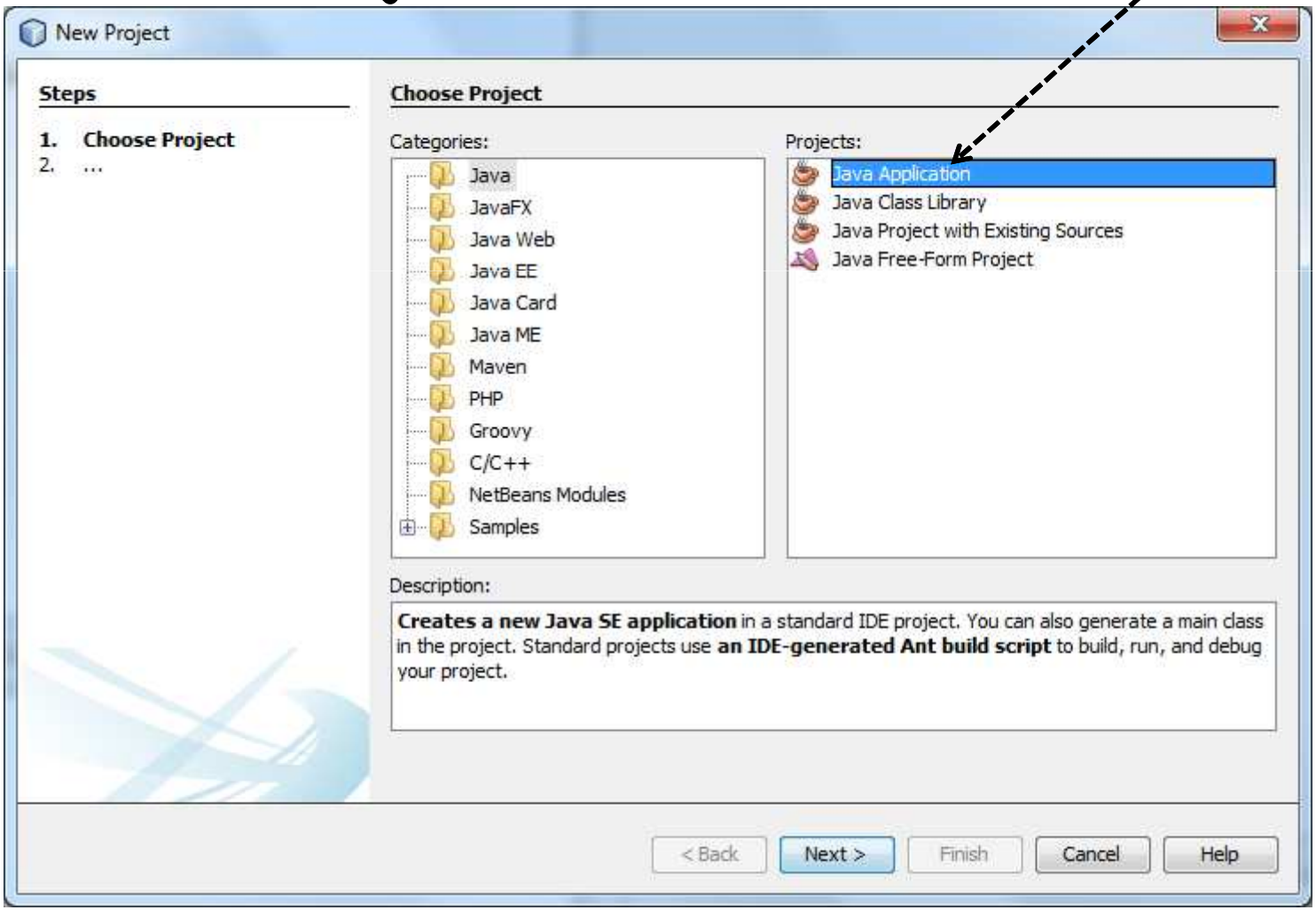


O ambiente NetBeans

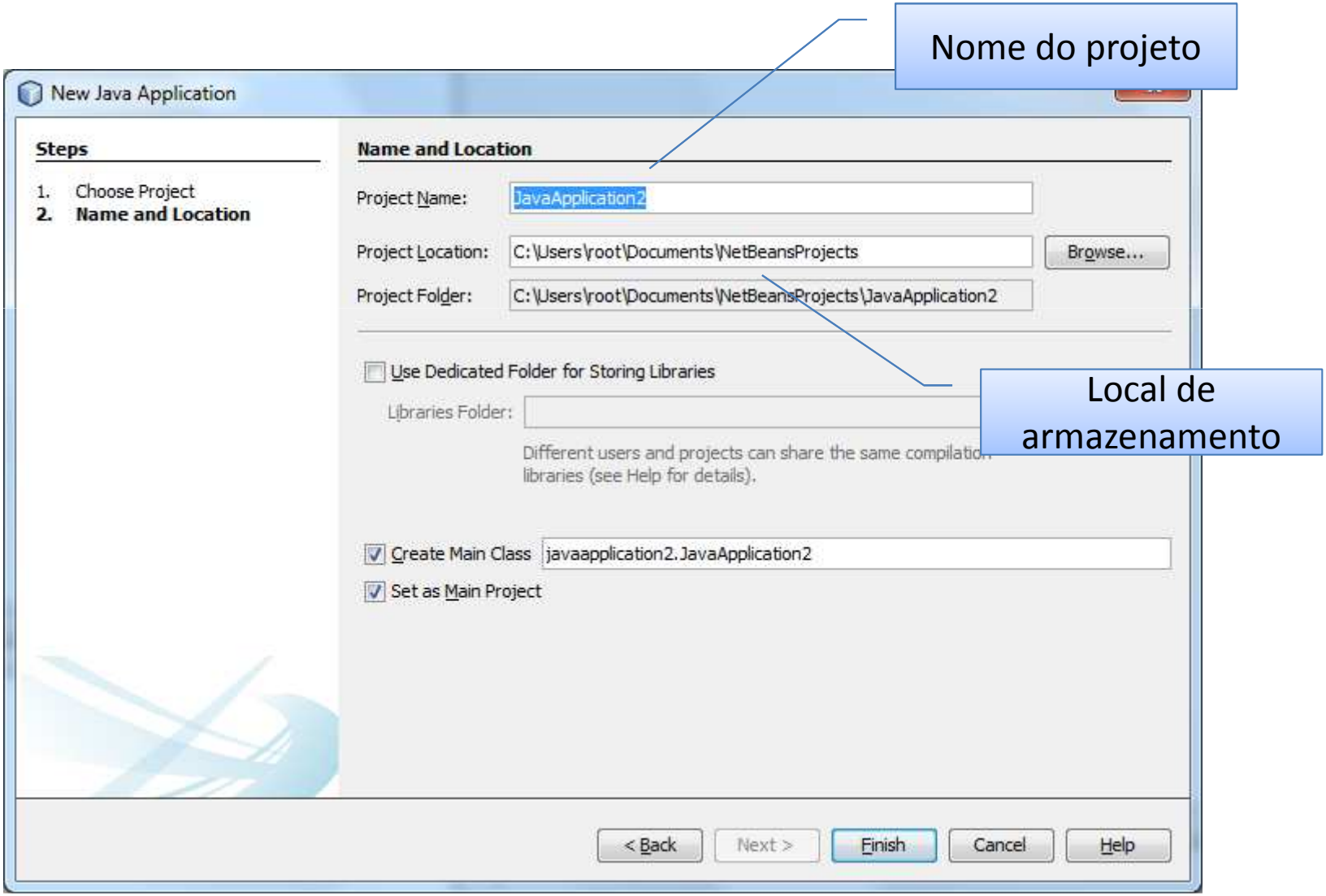


Passo a passo: criando um novo projeto

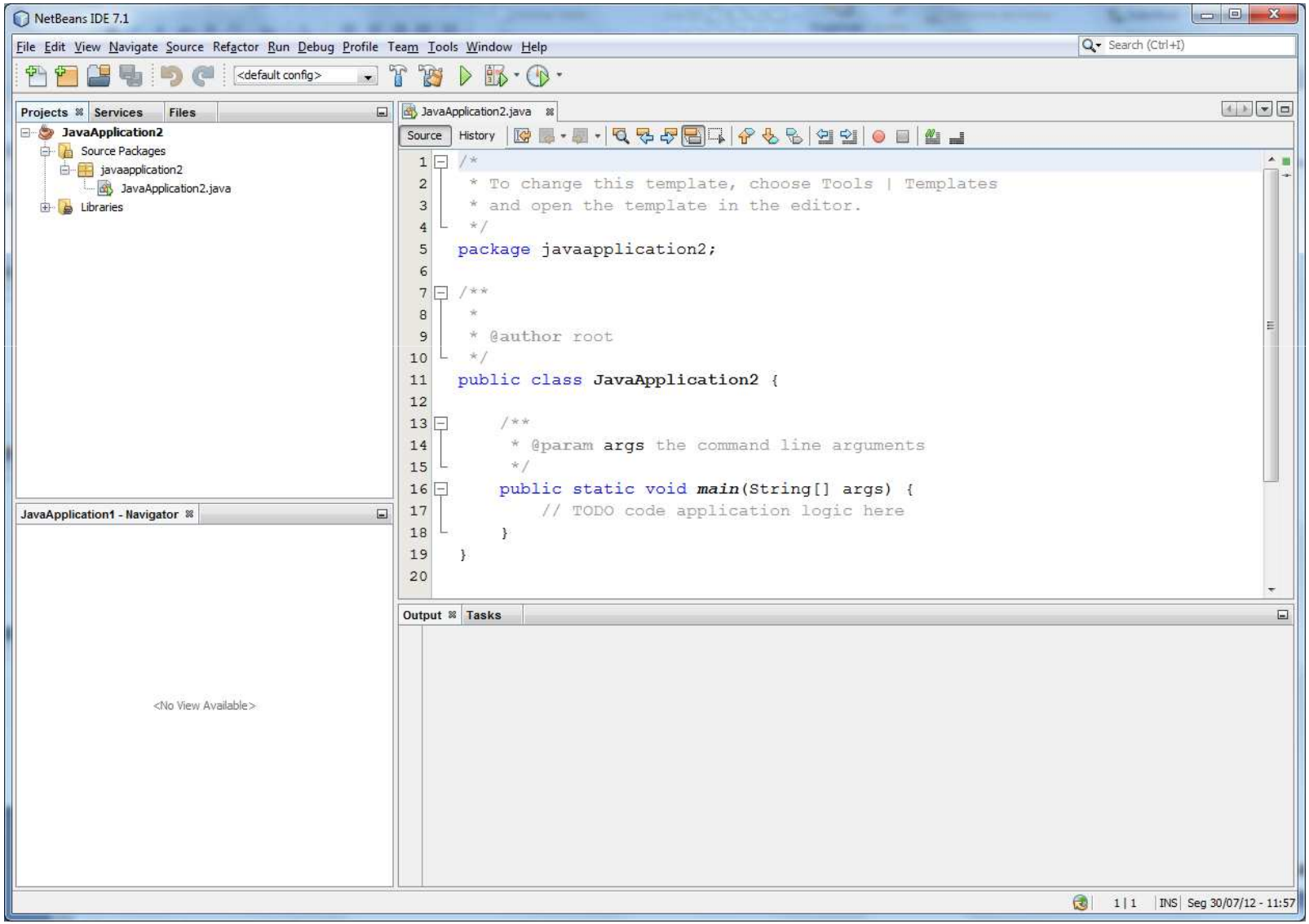
- File → New Project...



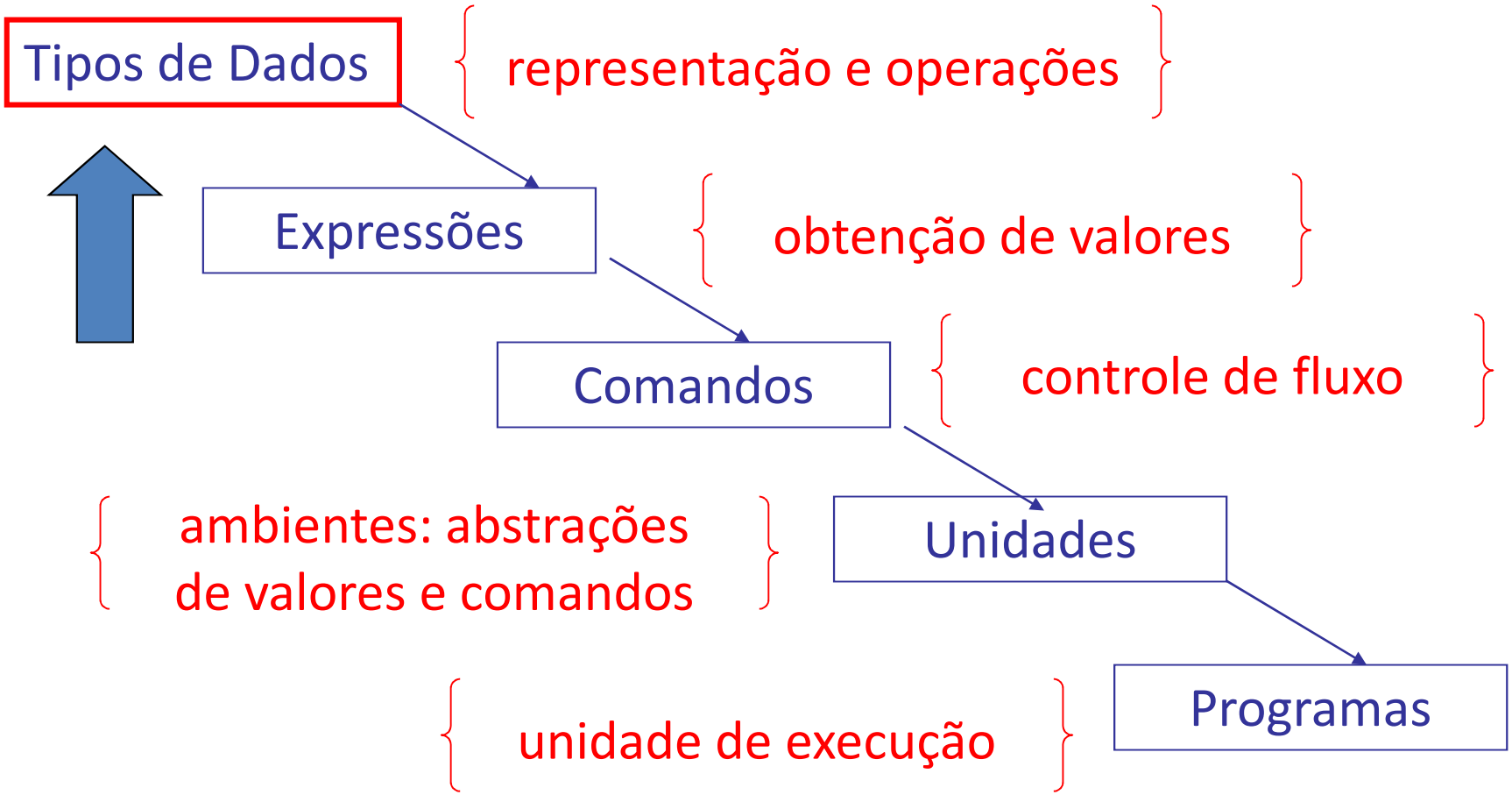
Passo a passo: criando um novo projeto



Passo a passo: criando um novo projeto



LP: hierarquia de componentes



Tipos de dados

- Tipos primitivos
- Tipos estruturados
- Classes x Tipos
- Variáveis X instâncias
- Composição de tipos
- Extensão de tipos
- Conversão de tipos
- Tipos como parâmetros

Tipos primitivos

Tipo	Tamanho	Mínimo	Máximo	Default
boolean	------	false	true	false
char	16 bits	Unicode 0	Unicode $2^{16}-1$	\u0000
byte	8 bits	-128	127	0
short	16 bits	-32768	32767	0
int	32 bits	-2.147.483.648	2.147.483.647	0
long	64 bits	-2^{63}	$2^{63}-1$	0
float	32 bits	$-3,4^{-38}$	$3,4^{+38}$	
double	64 bits	$-1,7^{-308}$	$1,7^{+308}$	
void	-------	-------	-------	-------

Tipos de dados primitivos - funções de conversão

Short.parseShort(argumento)

Integer.parseInt(argumento)

Long.parseLong(argumento)

Float.parseFloat(argumento)

Double.parseDouble(argumento)

Console: Entrada de dados – de acordo com o tipo esperado

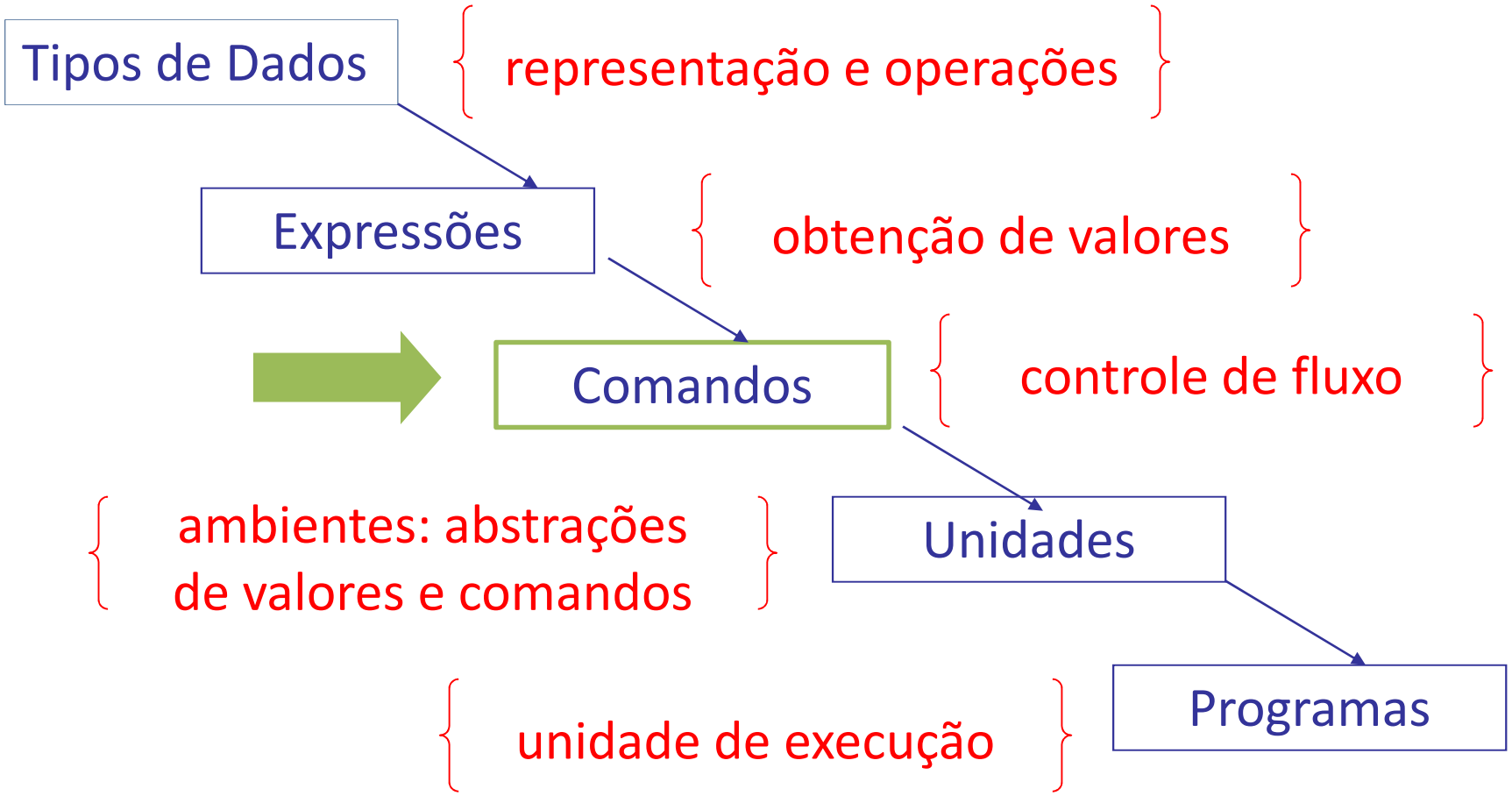
```
Scanner.next( );
Scanner.nextByte( );
Scanner.nextFloat( );
Scanner.nextDouble( );
Scanner.nextInt( );
Scanner.nextLong( );
Scanner.nextShort( );
...
```

Cuidado:
Os métodos aceitam somente valores correspondentes ao tipo e retornam esse tipo!

Ex:

nextDouble( ) → digitar "Java"

Solução
JOptionPane
....mais adiante...


Tipos de Dados
representação e operações
Expressões
obtenção de valores
Comandos
controle de fluxo
ambientes: abstrações
de valores e comandos
Unidades
unidade de execução
Programas

Fluxo de controle

- Objetivo: Determinar o próximo comando a ser executado
- Fluxo de controle puramente **sequencial**
  - Próximo comando na sequência
- Fluxo de controle **condicional**
  - Comandos de seleção
- Fluxo de controle **iterativo**
  - Comandos de repetição

Comando de seleção simples e dupla

- Seleção simples: if (teste) comando

```
if ((arg1 == 20) && (arg2 != 10))
      System.out.println(" algo ");
```

- Seleção dupla: if (teste) comando else comando

```
if (teste == true){comandos}
        else{comandos}
```

- Importante: a seleção também pode ser usada como expressão condicional

```
String ms = i==1 ? "um objeto" : "objetos";
```

Operadores

Padrões algébricos de equivalência ou Operadores relacionais	Operadores relacionais em Java	Example of Java condition	Significado da condição em Java
Equivalência			
=	==	x == y	x é igual a y
	!=	x != y	x é diferente de y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior ou igual a y
≤	<=	x <= y	x é menor ou igual a y

Seleção múltipla

- ***switch*** (seletor){casos}

```
switch(variavel){
        case 1: //trecho de código
                    break;
        case 2: //trecho de código
                    return 0;
        default: // nenhuma opção anterior
                return -1;
    }
```

Comandos de iteração

- **for**: valor inicial, teste de fim, passo (+, -)

```
for(k=0; k<100; k++)
{
  //faz algo
}
```

- **while**: executa o teste no início do laço

```
while(k<100)
{
  //faz algo
}
```

- **do-while**: executa o teste no final do laço

```
do{ //faz algo
}while(k<100);
```

Comando CONTINUE

- Continue: usado para passar para a próxima iteração de um laço

```
meses: for(int m=1; m<=12; m++){
          //fazer algo
          for(int d=1; d<=31; d++){
            //fazer algo
            if((m==2)&&(d==28))
              continue meses;
          }
       }
```

Comando *BREAK*

- usado para sair de um laço
- **meses**:

```
for(int m=1; m<=12; m++){
        //fazer algo
    for(int d=1; d<=31; d++){
            //fazer algo
        if(v1<v2)
          break meses;
            }
        }
    v1 = 0;
```

Strings

- Sequência de caracteres *imutável*
  - Representa uma cadeia de caracteres Unicode
  - Otimizada para ser lida, mas não alterada
  - Nenhum método de String modifica o objeto armazenado

- Há duas formas de criar Strings
  - String s1 = new String("Texto");
  - String s2 = "Texto";

Strings

- Exemplo de declarações e atribuições em Strings:

```
String nome, nomeCompleto;  // declaração simples
String sobrenome="da Silva"; // decl. + inicialização
nome="Paulo";
nomeCompleto=nome+" "+sobrenome;
if (sobrenome.equals("da Silva"))
        System.out.println("É parente!");
```

- **As strings podem ser concatenadas através do operador + e devem ser comparadas pelo método equals**

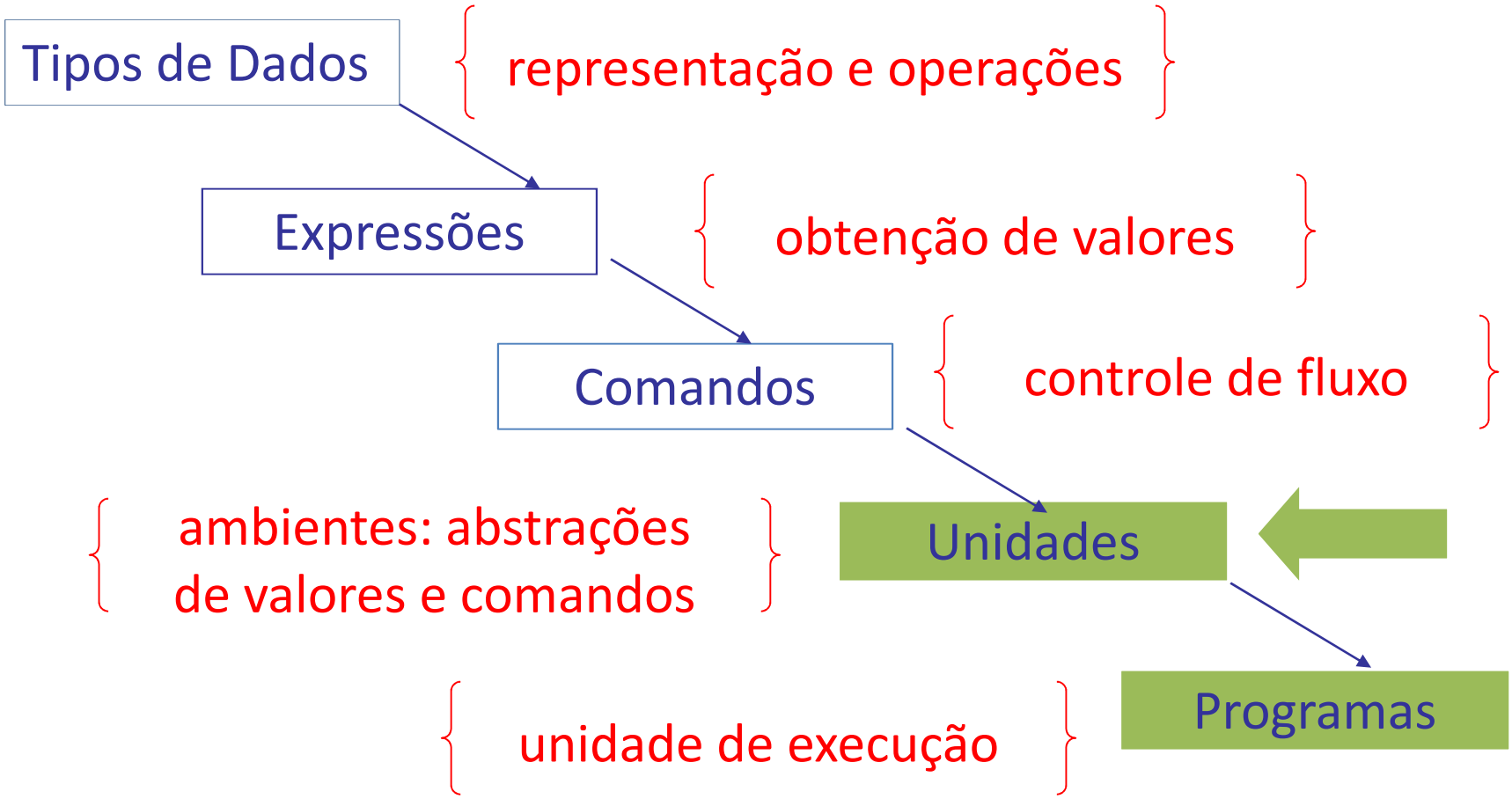
Strings

- Métodos que criam novas Strings
  - String concat(String s): retorna a concatenação da String atual com outro passado como parâmetro
  - String replace(char old, char new): troca todas as ocorrências de um caractere por outro
  - String substring(int start, int end): retorna parte do String incluindo a posição inicial e excluindo a final
  - String toUpperCase() e String toLowerCase(): retorna a String em caixa alta e caixa baixa respectivamente

Strings

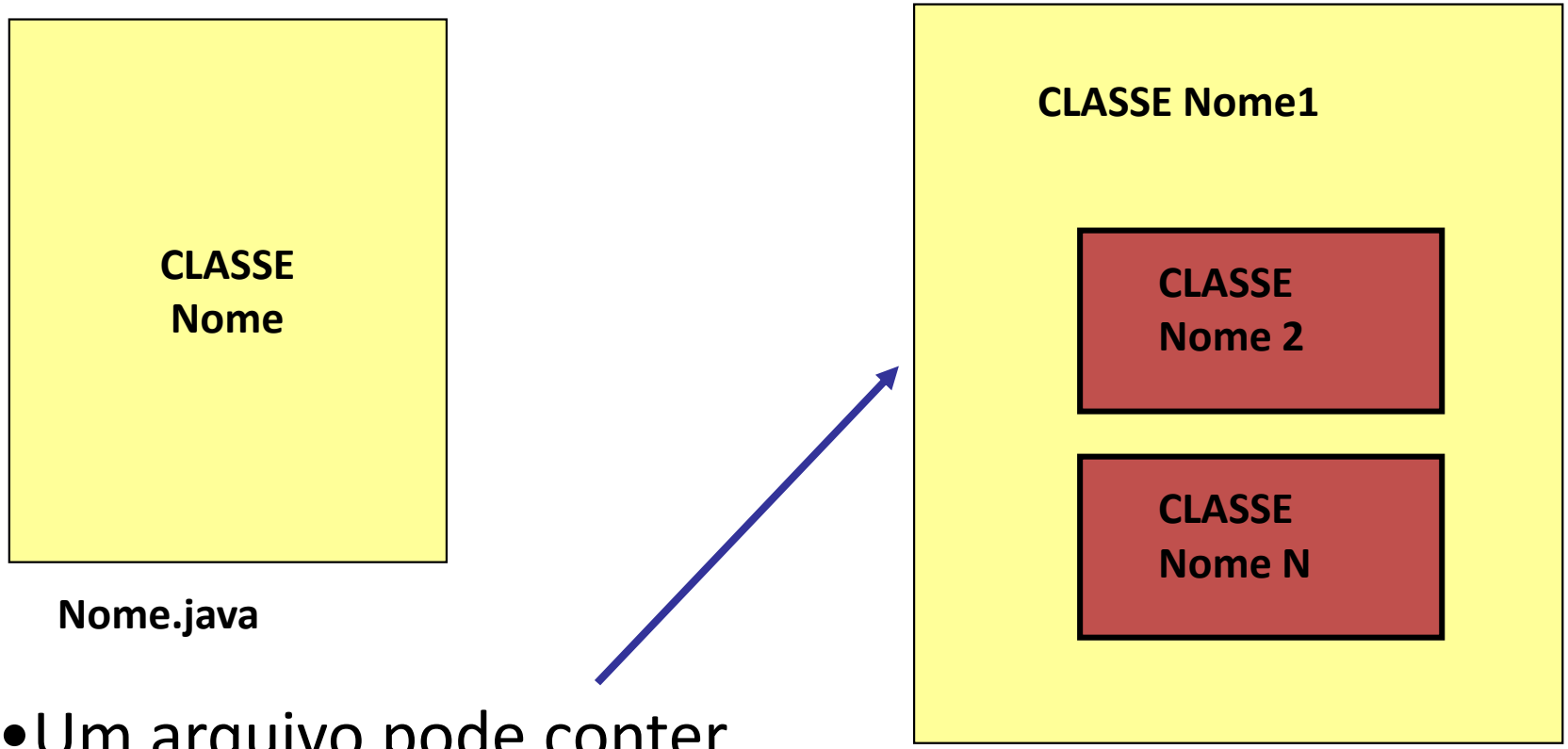
- Métodos para pesquisa
  - boolean endsWith(String) e startsWith(String)
  - int indexOf(String), int indexOf(String, int offset): retorna posição
  - char charAt(int posição): retorna caractere na posição

- Outros métodos
  - char[] toCharArray(): retorna o vetor de char correspondente à String
  - int length(): retorna o comprimento da String.


Tipos de Dados
representação e operações
Expressões
obtenção de valores
Comandos
controle de fluxo
ambientes: abstrações
de valores e comandos
Unidades
Programas
unidade de execução

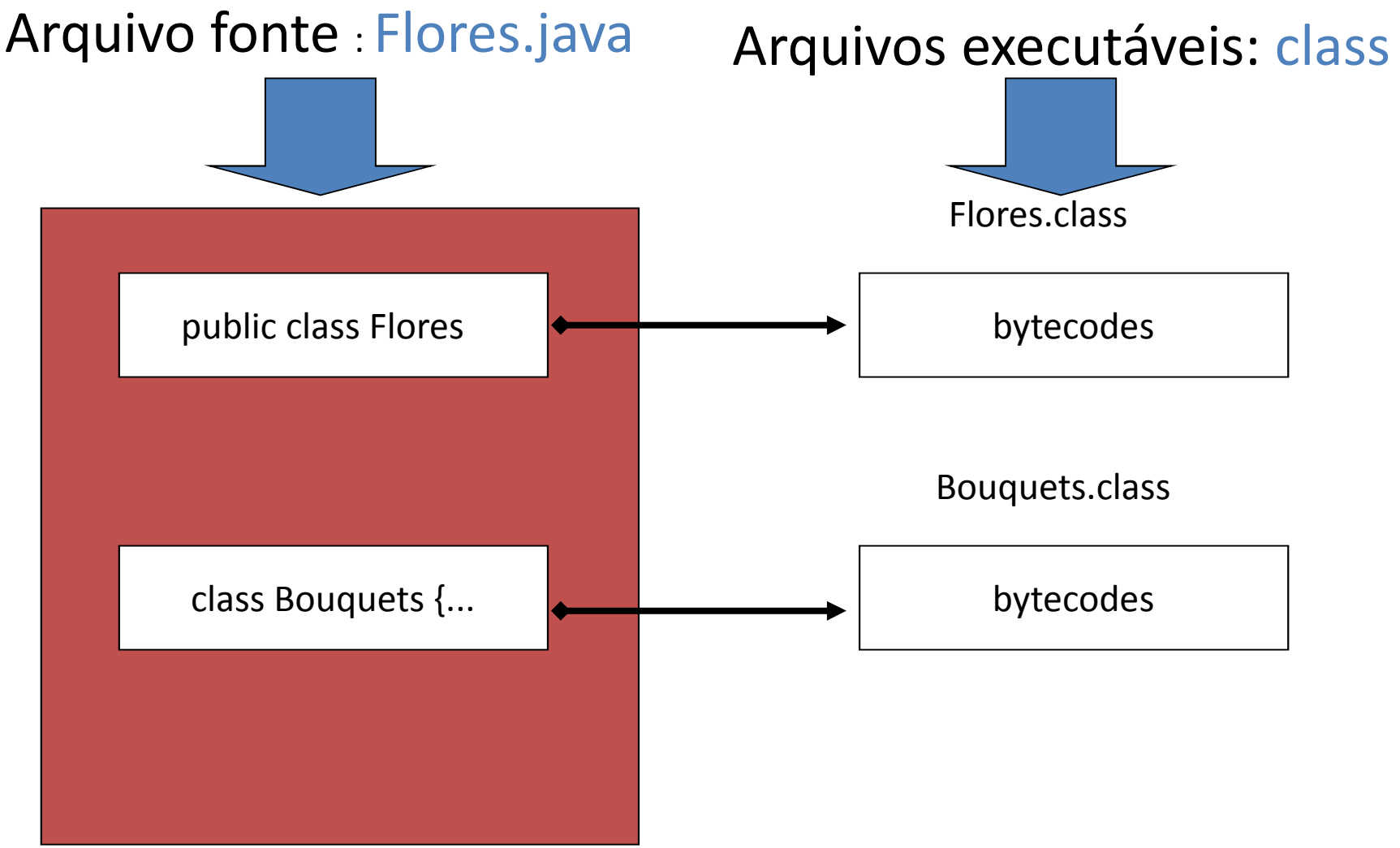
Programa Java: Classe

- Uma classe é um arquivo fonte



- Um arquivo pode conter várias classes

Arquivos fonte e executáveis



Classe Java: apenas o método principal

```
class Bemvindo {
        public static void main
_____(String[] args) {
        System.out.println
          (“Bem-vindo a Java.”);
        }
}
```

Cuidado maiúsculas e minúsculas!
Case Sensitive

Método ***main***: onde inicia a execução

Classe Java: com dois métodos

```
class Bemvindo {
        static void imprime(){
         System.out.println
          (“Bem-vindo a Java.”);
        }
public static void main_(String[] args) {
        imprime();
                }
}
```

Classe Java: definição de método estático

```
static <tipo> <nome> (<param1>,...,<paramn>)
{
<instruções>
return <expressão>
}
```

<tipo>: tipo de retorno do método

"void": (vazio) indica que método não retorna nada

<nome>: nome do método usualmente inicia com minúscula

<param>:

- parâmetro de entrada do método
- Sintaxe: <tipo_parâmetro> <nome_parâmetro>

Convenção para nomes

- Java, como C/C++ distingue entre letras maiúsculas e minúsculas
  - Exemplo: fruta difere de Fruta
- Nomes de classes iniciam com maiúscula
  - Exemplo: class Fruta
- Nomes de variáveis iniciam com minúsculas
  - Exemplo: int calorias
- Nomes de métodos são verbos que iniciam com minúscula e após usam maiúsculas
  - Exemplo: alteraPeso

Continuando...

- Tipos definidos pelo usuário
- Tipos estruturados de dados
  - Tipo array
  - Tipo String
- Classes como tipos de dados
  - Composição de tipos
  - Extensão de tipos

Tipos definidos pelo usuário

- Mnemônicos associados a tipos existentes

- Sinonímia: não criam novos tipos de dados; apenas usam identificadores simbólicos.

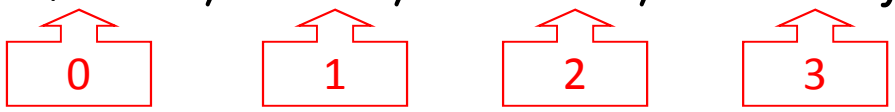
- Composição: podem ser combinados com outros tipos de dados.

  - Permitem restrições / enumerações

- Objetivo principal: legibilidade

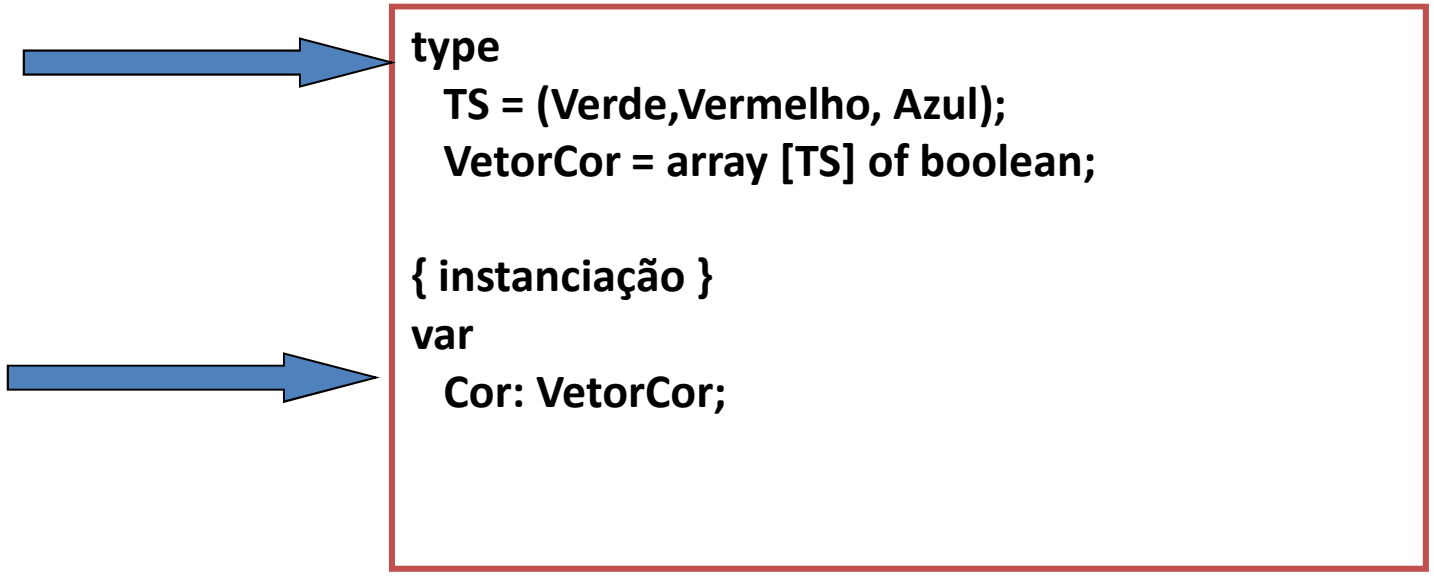
Tipos definidos pelo usuário: ordinais

- Definidos por associação ao um domínio de inteiros positivos
  - Restrição: sub-seqüência de um tipo
    - Exemplo de restrição em Pascal:
      - maiusculas= 'A'.. 'Z'; dias= 1..31;
  - Enumeração: descrição dos elementos
    - Exemplo de enumeração em C/C++:
      - enum estacoes {primavera, verão, outono, inverno}
      - .. chuva[verao] ...



Tipos x Instâncias

- Variável: instância concreta de um tipo
- Tipo: descritor
- Exemplo em Pascal:



```
type
  TS = (Verde,Vermelho, Azul);
  VetorCor = array [TS] of boolean;

{ instanciação }
var
  Cor: VetorCor;
```

Tipos ordinais em Java

- Em Java não existe definição de tipos como em C++ e Pascal/ObjectPascal(Delphi):
  - C++: typedef
  - Pascal/ObjectPascal(Delphi): seção type
- Enumerações são definidas através de constantes em interfaces. Exemplo:

```
public interface Meses {..
int  janeiro=1,
   fevereiro=2, ....dezembro=12;
}
```

Tipos estruturados

- São tipos compostos a partir de outros tipos de dados
  - homogêneos: todos os componentes pertencem ao mesmo tipo: Ex: *array* e *String*
  - heterogêneos: os componentes podem ser de tipos de dados diferentes. Ex. *struct*, *union*
    - são definidos pelo usuário através de classes

Arrays e Strings: declaração e instanciação

- Array

```
int numeros[ ]; int[ ] vet1, vet2, vet3;

numeros = new int[5];
```

- String

```
String s;

String s = new String ("Teste");
```

- Arrays de Strings

```
String[ ] str = new String[3];
```

Arrays

- Arrays são objetos
  - criados na memória dinâmica (heap) - referências
  - embora tenham tamanho fixo, pode-se alterar sua referência para um array de outro tamanho
- Declaração de arrays

```
int numero[ ];    int[ ] v1,v2;      char mc[ ][ ];
```

- Depois é preciso criar (instanciar) o array, usando o operador new:

```
numbers = new int[5];
str[] = new String[3];
```

- Declaração, instanciação e inicialização

```
int n[ ] = { 10, 20, 30, 40 };
```

Um array bidimensional

- é um array de arrays: cada elemento aponta para um array
- os arrays que formam a segunda dimensão podem ter tamanhos diferentes
- Exemplo:

```
int b[ ] [ ];
b = new int [2] [ ]; // instanciação de linhas
b[0] = new int [5]; // colunas da linha 0
b[1] = new int [3]; // colunas da linha 1
```

Arrays como parâmetros

- Cada array 'conhece' seu tamanho

  Exemplo: int a[ ] = {1,2,3}; tam = a.length;

- Arrays inteiros são passados por referência

  int temperaturas[ ] = new int[31];

  assinatura do método: void calcTemp(int temp[])

  chamada do método: calcTemp(temperaturas);

- Passagem de parâmetros:

  - tipos primitivos: por valor
  - tipos compostos (objetos): por referência

Arrays como valor de retorno

- Um método pode devolver um array como resultado.

Exemplo:

```
class Testarray{ .....
    static  int [ ] devolveArray (int q)  {
    return new int[q]; }
.....
public static void main (Strings args[]){
int vetor 20[] = new int[];          // tamanho indefinido
vetor20=devolveArray(20);            // demais comandos main
}}
```

tipo de resultado

Tipo String

- É um objeto da classe String (java.lang)
  - literais: Strings anônimos

    Exemplo: "azul"
  - variáveis: Referências a Strings

    Exemplo: String cor = " azul";
  - Lembrete: cada classe define um tipo de dado (representação do tipo e operações sobre o tipo)
- Exemplos de declaração, instanciação e inicialização

```
String s1;          s1 = new String();
String s2 = new String("Guga");
```

Strings x Arrays: exemplos

- Uso com parâmetro

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

- Array de caracteres

```
char charArray[ ] = { 'G' , 'u' , 'g' , 'a' };
```

notar tipo char

- Criando um String a partir de um array

```
s3 = new String (charArray);
```

converte int para String

- Concatenação de Strings

```
saida =  s3 + "venceu por " + n + "sets a" + m;
```

- Sugestão: estudar a classe String e ver seus recursos de conversão de dados

Classes como tipos de dados

- Uma definição de classe cria um novo tipo de dado
- Membros de uma classe: variáveis (campos) e métodos (funções)
- Formato simplificado de definição de classe

```
class NomeDaClasse
    { // inicia corpo da definição
            // definição dos campos
            // definição das funções (opcionais)
} // fim da classe NomeDaClasse
```

Classes x Registros

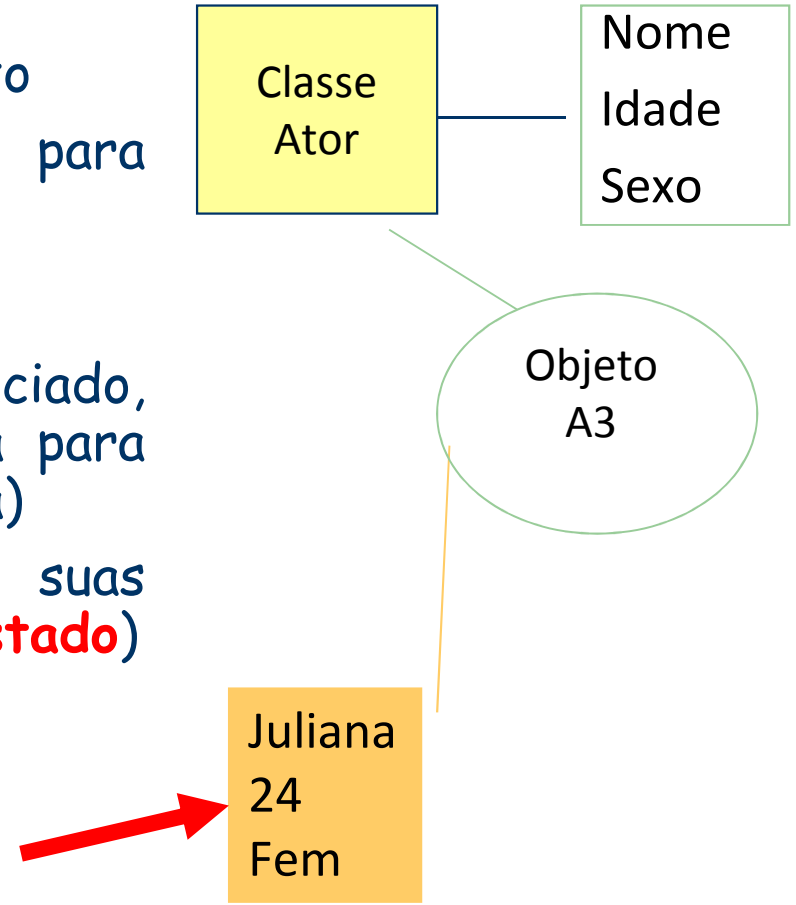
- Uma classe pode conter apenas definição de campos
- Os campos de uma classe formam um registro.

Exemplo:

```
class Ator {
    String nome;
    int idade;
    char sexo;
}
```

Classes x Instâncias

- **Classes**
  - definem o **tipo** de um objeto
  - não reservam **memória** para instâncias
- Instâncias = objetos
  - Cada **objeto**, quando instanciado, ocupa espaço de memória para suas variáveis (de instância)
  - Cada objeto possui suas variáveis e seus valores (**estado**)



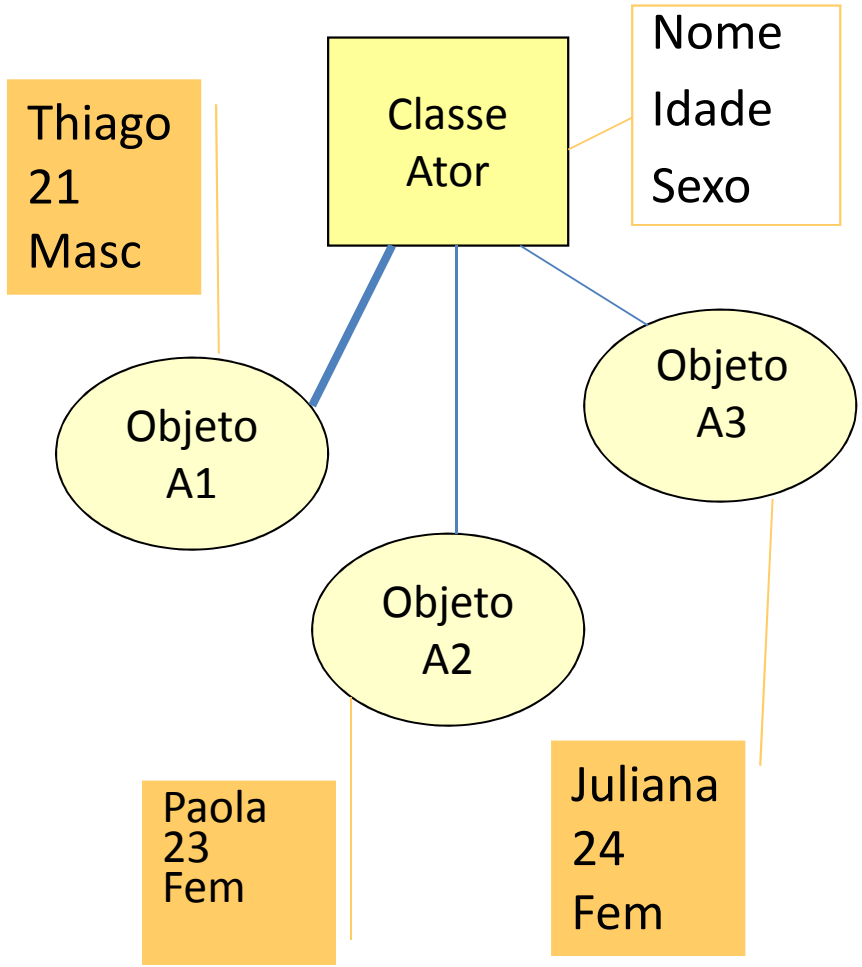
Classes e Objetos

Classes

- são descritores (de objetos)
- mesma classe pode se usada para instanciar inúmeros objetos

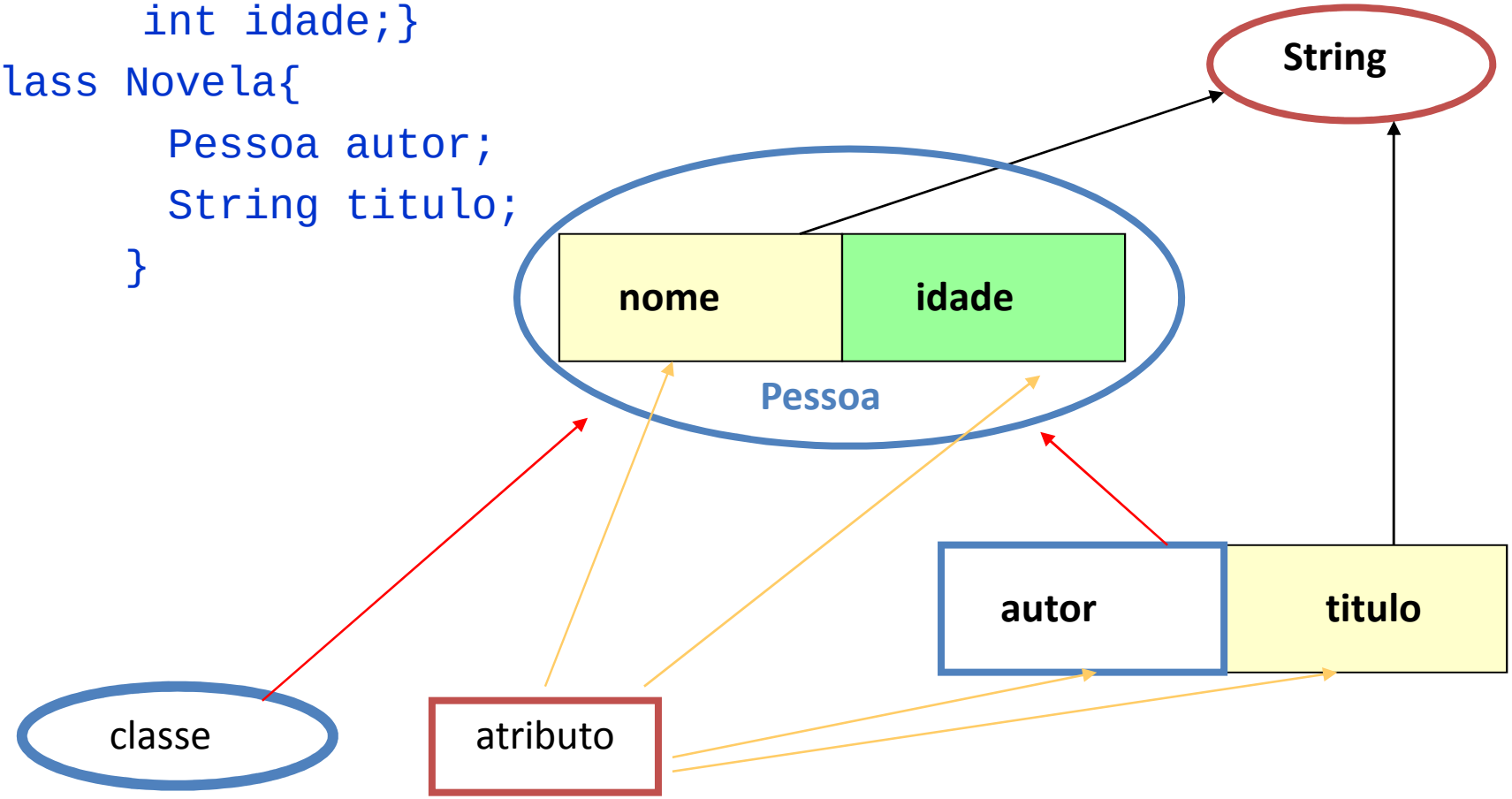
Objetos

- são instâncias (de classe)
- objetos de mesma classe: mesmo **tipo**
- idênticas propriedades, diferentes dados
- podem ser atribuídos e passados como parâmetros (referências!)

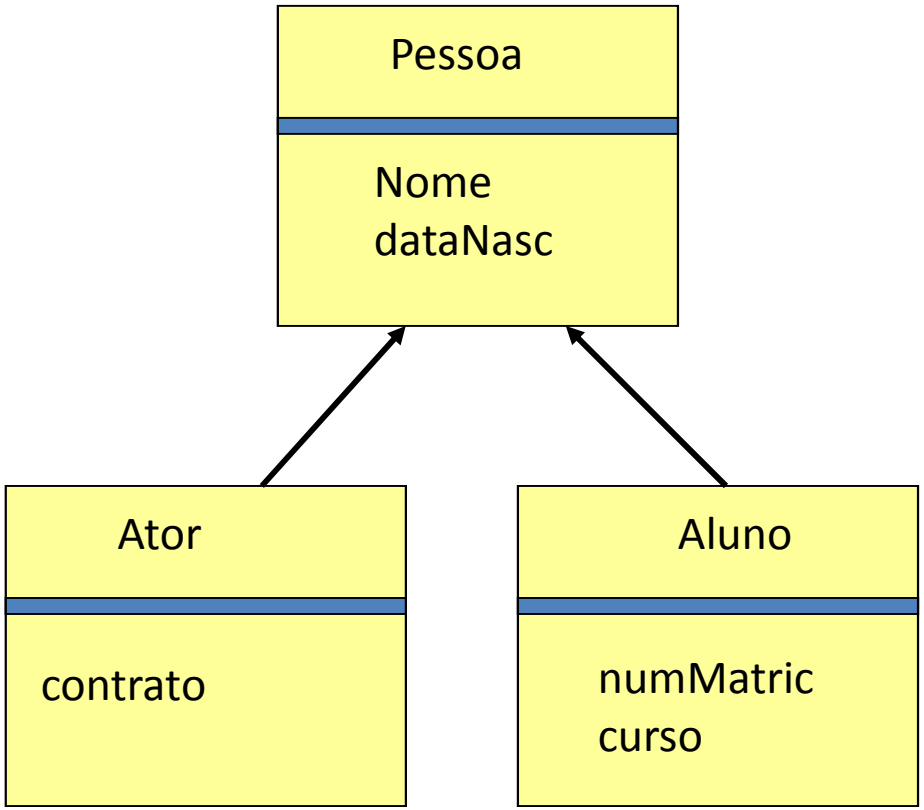


Composição de tipos com classes

```
class Pessoa{
       String nome;
       int idade;}
class Novela{
        Pessoa autor;
        String titulo;
      }
```



Extensão de tipos: herança




```
class Pessoa {
        String nome;
        int dia, mês, ano;
        // demais membros}

class Ator extends Pessoa {
        String contrato ;
        // demais membros}

class Aluno extends Pessoa{
        String num_matric, curso;
        // demais membros}
```

Exemplo de classes e objetos

```
class Ator {
      String nome;
      int idade;
      char sexo;
public static void main (String args[])
{
Ator ana = new Ator(); // cria instância
ana.nome= “Ana Paula”;
ana.idade= 22;
ana.sexo= ‘f’;
} // Ator
```

Objetos: parâmetros e resultado

- Objetos podem ser usados como parâmetros e tipo de resultado. Exemplo:

```
class Fruta {
    int gramas;
    int calorias;
Fruta somaFruta(Fruta par){
Fruta temp=new Fruta();
    temp.gramas = par.gramas +10;
    temp.calorias = par.calorias + 100;
    return temp)
} }
```

tipo de resultado

tipo de parâmetro

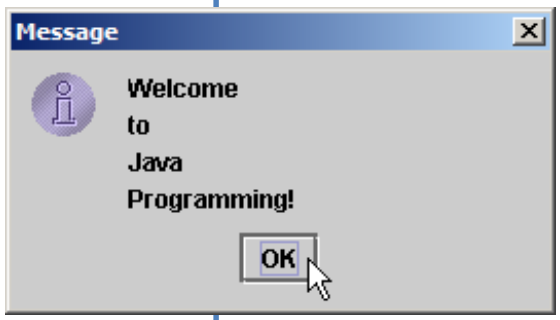
Caixas de diálogo

Exibindo texto em Caixas de diálogo

- Exibição
  - A maioria das aplicações Java utilizam windows ou caixas de dialogo (*dialog box*)
    - utilizamos as command window
  - A classe JOptionPane permite utilizar dialog boxes
- Packages
  - Conjunto pré-definido de classes para utilização
  - Grupos de classes relacionadas são chamadas de packages
  - Grupos de todos os packages são conhecidos como Java class library ou Java applications programming interface (Java API)
- A classe JOptionPane está no package javax.swing
  - Essa classe é utilizada na construção de *Graphical User Interfaces* (GUIs)

Exibindo texto em Caixas de diálogo - exemplo

```
1
2   // Imprimindo múltiplas linhas em uma caixa de diálogo.
3
4   // Pacotes de extensão de Java
5   import javax.swing.JOptionPane;  // importa a classe JOptionPane
6
7   public class Welcome4 {
8
9      // método main começa execução do aplicativo Java
10     public static void main( String args[] )
11     {
12        JOptionPane.showMessageDialog(
13           null, "Welcome\nto\nJava\nProgramming!" );
14
15        System.exit( 0 );  // termina o aplicativo
16
17     } // fim do método main
18
19  } // fim da classe Welcome4
```



```
4   // Pacotes de extensão de Java
```

Existe dois tipos de pacotes na API do Java

1. Pacotes do núcleo

- Iniciam com a palavra java
- Incluídos como parte do Java 2 Software Development Kit

2. Pacotes de extensão

- Começam com javax
- São pacotes novos de Java

```
5   import javax.swing.JOptionPane;  // importa a classe JOptionPane
```

declaração import

- Utilizadas para o compilador localizar as classes externas utilizadas pelo programa java.
- Faz o compilador ler a classes JOptionPane do package javax.swing

```
12          JOptionPane.showMessageDialog(
13             null, "Welcome\nto\nJava\nProgramming!" );
```

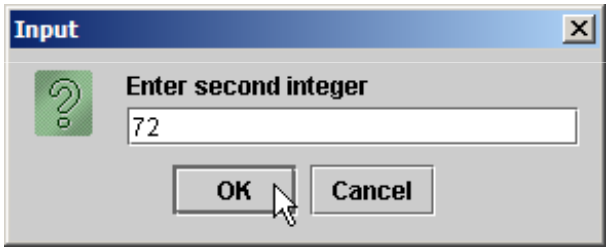
- Chama o método showMessageDialog da classe JOptionPane
  - Requer dois argumentos
  - Múltiplos argumentos são separados por vírgulas (,) como em C
  - Por enquanto, vamos usar o primeiro argumento sempre como **null**
  - O segundo argumento é a string a ser exibida
- showMessageDialog é um método static da classe JOptionPane
  - Métodos static são chamados usando-se o nome da classe, ponto (.) e o nome do método

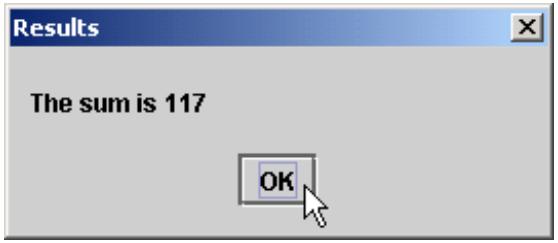
```
15         System.exit( 0 );  // terminate application with window
```

Chama o método **static exit** da classe System

- Termina a aplicação
- Esta linha é necessária em qualquer aplicativo que exibe uma interface gráfica com o usuário (GUI)
- O argumento 0 indica que a aplicação finalizou com sucesso
- Um valor diferente de zero geralmente indica que ocorreu um erro
- A classe System é parte do pacote java.lang
- Não há necessidade da instrução import para java.lang
- java.lang é automaticamente importada em todo programa Java

Mais um exemplo: somando inteiros





```
1    //
2    // Programa para mostrar o resultado da adição de dois números.
3
4    // Pacotes de extensão java
5    import javax.swing.JOptionPane;  // importa a classe JOptionPane
6
7    public class Addition {
8
9       // método main inicia a execução do aplicativo java
10      public static void main( String args[] )
11      {
12         String firstNumber;   // primeiro string inserido pelo usuário
13         String secondNumber;  // primeiro string inserido pelo usuário
14
15         int number1;          // primeiro número a somar
16         int number2;          // segundo número a somar
17         int sum;              // soma de number1 e number2
18
19         // lê o primeiro número do usuário como uma String
20        firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
21
22         // lê o segundo número do usuário como uma String
23         secondNumber =
24            JOptionPane.showInputDialog( "Enter second integer" );
25
26         // converte os números do tipo String para o tipo int
27         number1 = Integer.parseInt( firstNumber );
28         number2 = Integer.parseInt( secondNumber );
29
30         // Adiciona os números
31         sum = number1 + number2;
32
```

```
1   //
2   // Programa para mostrar o resultado da adição de dois números.
3
4   // Pacotes de extensão java
5   import javax.swing.JOptionPane;  // impo
6
7   public class Addition {
8
9      // método main inicia a execução do aplicativo java
10     public static void main( String args[] )
11     {
12        String firstNumber;   // primeiro st
13        String secondNumber;  // primeiro st
14
15        int number1;          // primeiro nu
16        int number2;          // segundo número a somar
17        int sum;              // soma de number1 e number2
18
19        // lê o primeiro número do usuário como uma String
20       firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
21
22        // lê o segundo número do usuário como uma String
23        secondNumber =
24           JOptionPane.showInputDialog( "Enter second integer" );
25
26        // converte os números do tipo String para o tipo int
27        number1 = Integer.parseInt( firstNumber );
28        number2 = Integer.parseInt( secondNumber );
29
30        // Adiciona os números
31        sum = number1 + number2;
32
```

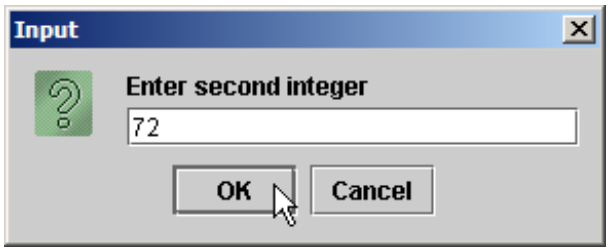
Declaração de variáveis: tipo e nome

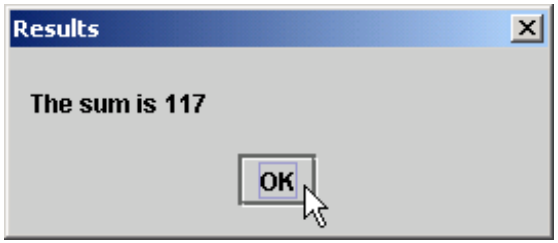
Lê o primeiro inteiro como um String, e atribui para firstNumber.

Converte strings para inteiros.

Soma, e coloca o resultado em sum.

```
33          // exibe os resultados
34          JOptionPane.showMessageDialog( null, "The sum is " + sum,
35             "Results", JOptionPane.PLAIN_MESSAGE );
36
37          System.exit( 0 );   // termina o aplicativo
38
39       } // fim do método main
40
41    } // fim da classe Addition
```

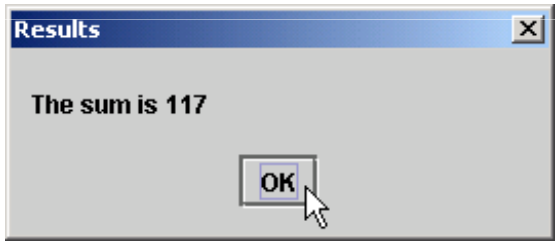




```
34          JOptionPane.showMessageDialog( null, "The sum is " + sum,
35             "Results", JOptionPane.PLAIN_MESSAGE );
```

Versão diferente de showMessageDialog

- Requer quatro argumentos (ao invés de dois como antes)
- Primeiro argumento: null por enquanto, como antes
- Segundo: string a ser exibida
- Terceiro: string na barra de títulos



- Quarto: tipo de diálogo de mensagem com ícone
- Linha 35 nenhum ícone: JOptionPane.PLAIN_MESSAGE

Outros tipos de caixas de diálogos...

Tipo de diálogo de mensagem	Ícone	Descrição
`JOptionPane.ERROR_MESSAGE`		Exibe um diálogo que indica um erro para o usuário.
`JOptionPane.INFORMATION_MESSAGE`		Exibe um diálogo com uma mensagem de informação para o usuário. O usuário poderá simplesmente dispensar o diálogo
`JOptionPane.WARNING_MESSAGE`		Exibe um diálogo que adverte o usuário sobre um problema em potencial.
`JOptionPane.QUESTION_MESSAGE`		Exibe um diálogo que impõe uma pergunta para o usuário. Este diálogo normalmente exige uma resposta, Tal como clicar em um botão Yes ou No
`JOptionPane.PLAIN_MESSAGE`	Sem ícone	Exibe um diálogo que simplesmente contém uma mensagem, sem nenhum ícone

Observações e Práticas de Programação

Observação de engenharia de software

- Utilize uma abordagem de blocos de construção para criar programas.
- Evite reinventar a roda — utilize partes existentes onde for possível.
- Chamada de reutilização de software, essa prática é fundamental para a programação orientada a objetos.

Observação de engenharia de software

- Ao programar em Java, você geralmente utilizará os seguintes blocos de construção:
  - classes e métodos de bibliotecas de classe;
  - classes e métodos que você mesmo cria; e
  - classes e métodos que outros criam e se tornam disponíveis para você

Dica de desempenho

- Utilizar as classes e os métodos da API do Java – em vez de escrever suas próprias versões – pode melhorar o desempenho de programas, porque eles são cuidadosamente escritos para executar de modo eficiente. Essa técnica também diminui o tempo de desenvolvimento dos programas.

Dica de portabilidade

- Utilizar as classes e os métodos da API do Java – em vez de escrever suas próprias versões – melhora a portabilidade de programa, porque esses são incluídos em cada implementação Java.

Erro comum de programação

- Os erros como divisão por zero ocorrem enquanto um programa executa; portanto, eles são chamados de erros de tempo de execução ou erros de *runtime*.
- Erros de tempo de execução fatais fazem com que os programas sejam imediatamente encerrados sem terem realizado seus trabalhos com sucesso.
- Erros de tempo de execução não-fatais permitem que os programas executem até sua conclusão, produzindo, frequentemente, resultados incorretos.

Dica de portabilidade

- Embora seja mais fácil escrever programas portáveis em Java do que em outras linguagens de programação, diferenças entre compiladores, JVMs e computadores podem tornar a portabilidade difícil de alcançar.

- Simplesmente escrever programas em Java não garante portabilidade.

Dica de prevenção de erro

- Sempre teste os programas Java em todos os sistemas nos quais você quiser executá-los, para, assim, assegurar que eles funcionem corretamente para o público-alvo.

Boa prática de programação

- Leia a documentação da versão do Java que você está utilizando. Consulte-a com frequência para certificar-se de que você está ciente da rica coleção de recursos Java e de que está utilizando esses recursos corretamente.

Boa prática de programação

- Seu computador e compilador são bons professores. Caso, depois de ler cuidadosamente o manual de documentação do Java, você não esteja seguro sobre como um recurso do Java funciona, experimente-o para ver o que acontece. Estude cada erro ou mensagem de advertência/aviso que surge durante a compilação de um programa (chamados erros de tempo de compilação ou erros de compilação), e corrija os programas para eliminar essas mensagens.