Instituto de Computação uff

Técnicas de Programação Avançada

TCC-00175

Profs.: Anselmo Montenegro

www.ic.uff.br/~anselmo

Conteúdo: Revisão de Java

Histórico

1991

Início em 1991:
Pequeno grupo de projeto da *Sun Microsystems*, denominado *Green*.

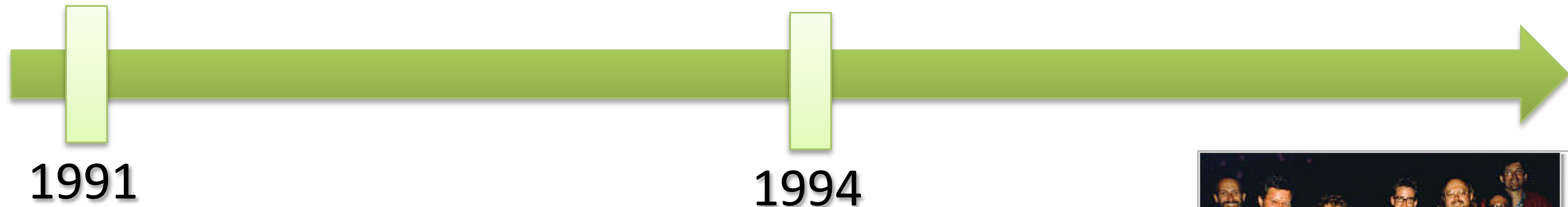
O projeto visava o desenvolvimento de software para uma ampla variedade de dispositivos de rede e sistemas embutidos.

James Gosling, decide pela criação de uma nova linguagem de programação que fosse simples, portátil e fácil de ser programada.

Surge a linguagem interpretada *Oak* (carvalho em inglês), mais tarde rebatizada como *Java* devido a problemas de direitos autorais.

Instituto de computação uff

Histórico



Mudança de foco para aplicação na Internet.
(**visão**: um meio popular de transmissão de texto, som, vídeo).

Projetada para transferência de conteúdo de mídia em redes com dispositivos heterogêneos.

Também possui capacidade de transferir “comportamentos”, junto com o conteúdo.
(**HTML por si só não faz isso**)

Em 1994*:*
Jonathan Payne e *Patrick Naughton* desenvolveram o programa navegador *WebRunner*.

Instituto de computação uff

Histórico

1991 1994 1995

No *SunWorld'95* a *Sun* apresenta formalmente o navegador *HotJava* e a linguagem *Java*.

Poucos meses depois a *Netscape Corp.* lança o seu navegador capaz de fazer download e executar pequenos códigos *Java* chamados de Applets.

Imediatamente a *Sun* decide disponibilizar o *Java* gratuitamente para a comunidade de desenvolvimento de softwares e assim surge o *Java Developer´s Kit 1.0 (JDK 1.0)*.

Inicialmente: *Sun Solaris* e *Microsoft Windows 95/NT*.
Progressivamente surgiram kits para outras plataformas como *Linux* e *Apple Macintosh*.

Instituto de Computação UFF

Características da Linguagem *Java*

simples

neutra

orientada a objeto

distribuída

robusta

interpretada

alta performance

segura

dinâmica

portável

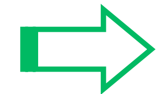
multithread

Características da Linguagem *Java*

Simples e orientada a objetos

É de fácil aprendizado.

Puramente orientada a objetos:



Permite o desenvolvimento de sistemas de uma forma mais natural.

Características da Linguagem *Java*

Distribuída

Projetada para trabalhar em ambiente de redes.

Não é uma linguagem para programação distribuída:

Oferece bibliotecas para facilitar o processo de comunicação.

Instituto de computação

Características da Linguagem *Java*

Alta Performance

É uma linguagem interpretada e existe uma grande discussão quanto a sua performance.

Fato: As melhorias na tecnologia de compilação, tem aproximado o desempenho ao de linguagens como C e C++.

Ex.: Benchmarks numéricos.

Referência

Características da Linguagem *Java*

Robusta e Segura

As seguintes características contribuem para tornar a linguagem mais robusta e segura:

- ✓ É fortemente tipada;
- ✓ Não possui aritmética de ponteiros;
- ✓ Possui mecanismo de coleta de lixo;
- ✓ Possui verificação rigorosa em tempo de compilação;
- ✓ Possui mecanismos para verificação em tempo de execução;
- ✓ Possui gerenciador de segurança.

Segurança: possui mecanismos de segurança que evitam operações no sistema de arquivos da máquina alvo.

Instituto de computação uff

Características da Linguagem *Java*

Interpretada, Neutra, Portável

Bytecodes executam em qualquer máquina que possua uma JVM, permitindo que o código em Java possa ser escrito independente da plataforma.

A característica de ser neutra em relação à arquitetura permite uma grande portabilidade.

Instituto de Computação

Características da Linguagem *Java*

Interpretada, Neutra, Portável

myProgram.java

compilador

myProgram.class

Interpretador

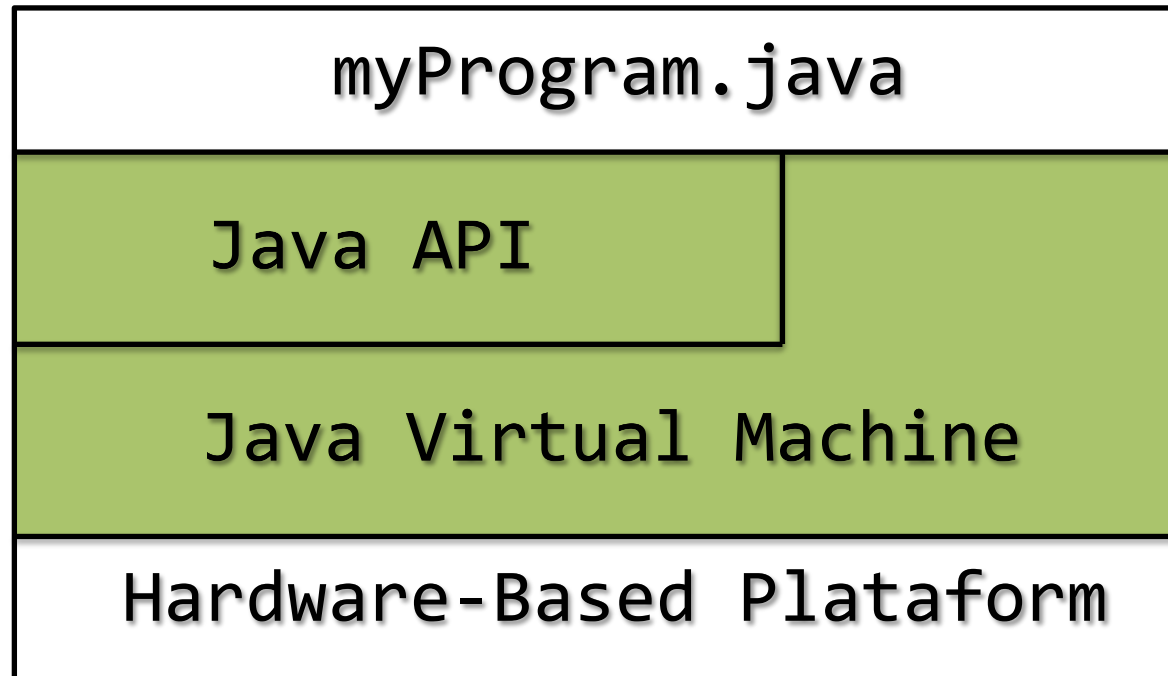
0010110100

My Program

Instituto de Computação

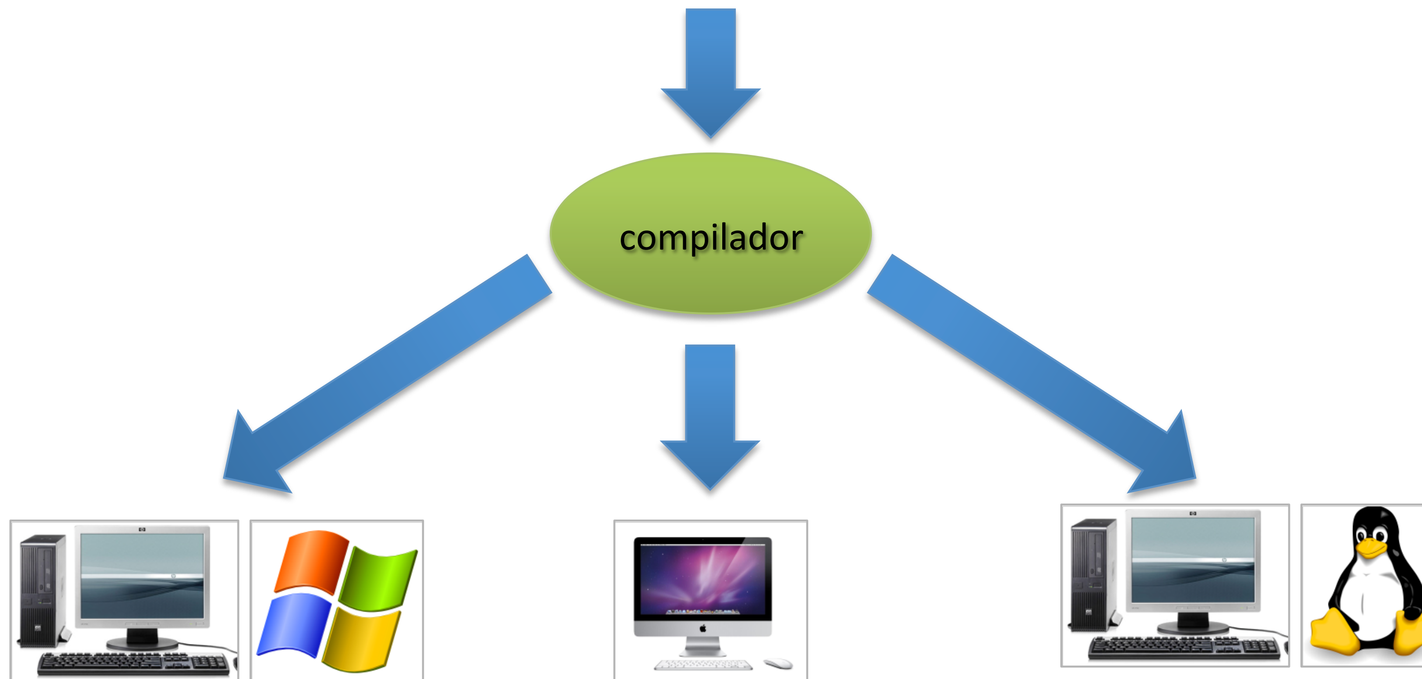
Características da Linguagem *Java*

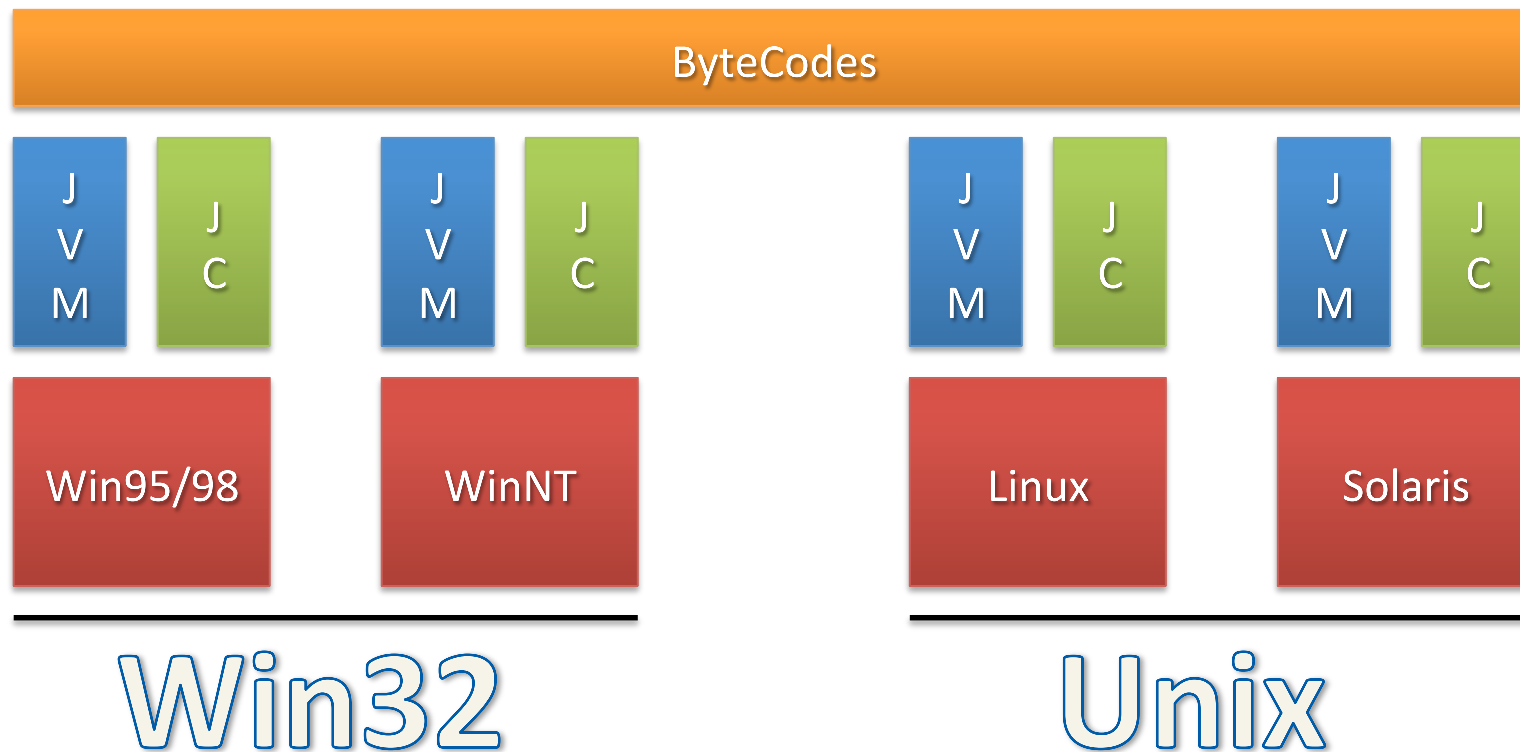
Interpretada, Neutra, Portável

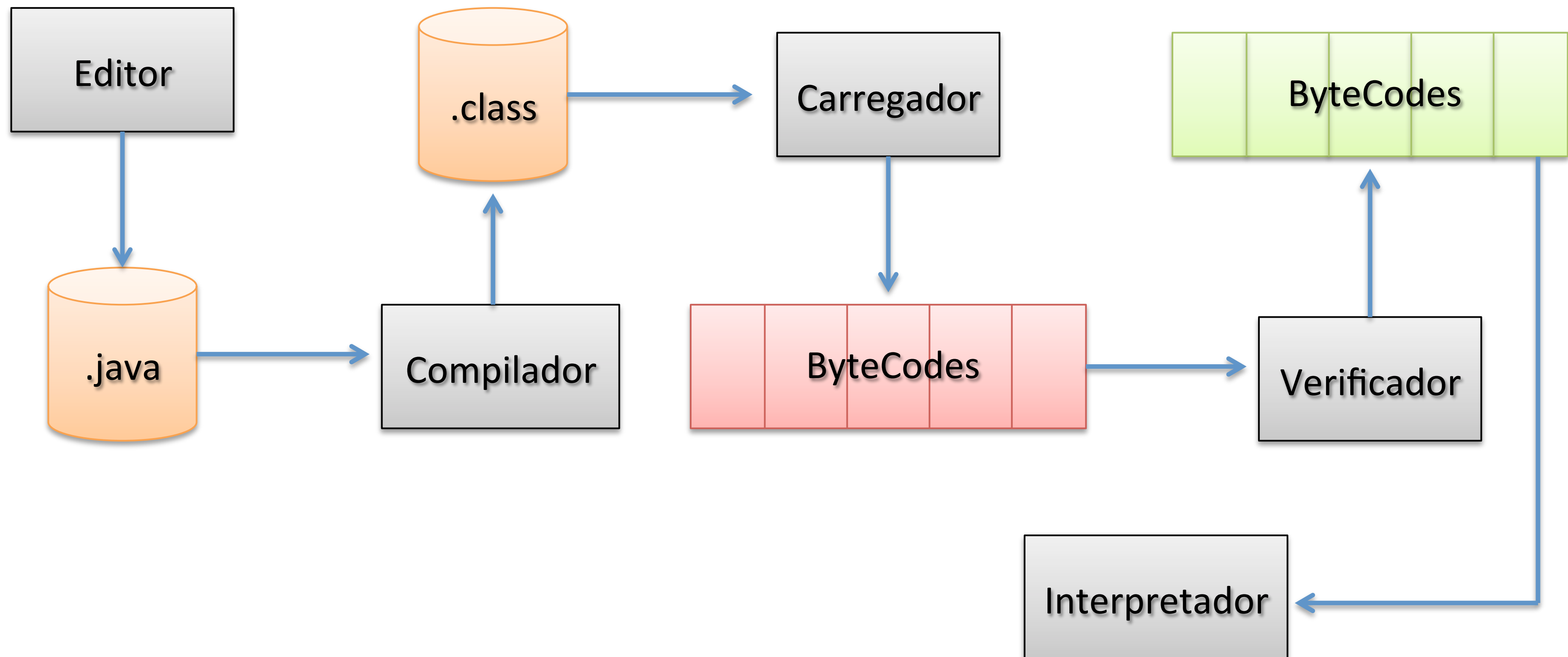


Características da Linguagem *Java*

Interpretada, Neutra, Portável




ByteCodes
J V M
J C
J V M
J C
J V M
J C
J V M
J C
Win95/98
WinNT
Linux
Solaris
Win32
Unix


Editor
.class
Carregador
ByteCodes
.java
Compilador
ByteCodes
Verificador
Interpretador

Resolução de referências em tempo de execução:

flexibilidade VS performance.

suporte para múltiplas threads de execução, que podem tratar diferentes tarefas concorrentemente.

Instituto de computação uff

O Ambiente *de Desenvolvimento*

Java possui um ambiente de desenvolvimento de software denominado Java SDK.

(*Software Development Kit* – antigamente denominado JDK).

Não é um ambiente integrado de desenvolvimento, não oferecendo editores ou ambiente de programação.

O Java SDK *contém um amplo conjunto de APIs*.

(Application Programing Interface).

Instituto de Computação UFF

O Ambiente *de Desenvolvimento*

Algumas ferramentas do Java SDK:

- ✓ o compilador Java (javac)
- ✓ o interpretador de aplicações Java (java)
- ✓ o interpretador de applets Java (appletsviewer)
- ✓ javadoc (um gerador de documentação para programas Java)
- ✓ Jar (o manipulador de arquivos comprimidos no formato Java Archive)
- ✓ jdb (um depurador de programas Java)
- ✓ etc.

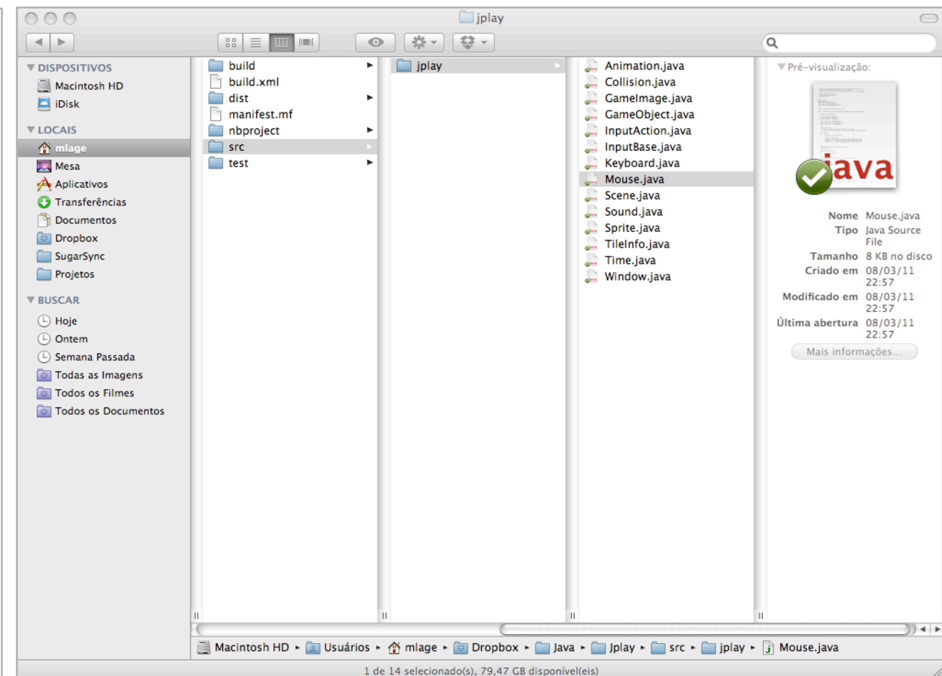
Packages

Os arquivos Java serão armazenados fisicamente em uma pasta.

No nosso exemplo ao lado estes arquivos estão no diretório jplay.

Cada arquivo representa uma classe Java.

Com o uso de packages podemos organizar de forma física algo lógico.
(um grupo de classes em comum)



Packages

Para indicar que as definições de um arquivo fonte Java fazem parte de um determinado pacote, a primeira linha de código deve ser a declaração de pacote:

```
package nome_do_pacote;
```

Caso tal declaração não esteja presente, as classes farão parte do "pacote default", que está mapeado para o diretório corrente.

Instituto de Computação

Packages

Referenciando uma classe de um pacote no código fonte:

```
import nome_do_pacote.Xyz ou simplesmente
import nome_do_pacote.*
```

Com isso a classe Xyz pode ser referenciada sem o prefixo nome_do_pacote no restante do código.

A única exceção refere-se às classes do pacote java.lang.

Instituto de Computação UFF

Classpath

O ambiente Java normalmente utiliza a especificação de uma variável de ambiente CLASSPATH.

CLASSPATH define uma lista de diretórios que contém os arquivos de classes Java.

No exemplo anterior: se o arquivo Xyz.class estiver no diretório */home/java/nome_do_pacote*, então o diretório */home/java* deve estar incluído lista de diretórios definida por CLASSPATH.

Tipos Primitivos

Podem ser agrupados em quatro categorias:

Tipos Inteiros:
Byte, Inteiro Curto, Inteiro e Inteiro Longo.

Tipos Ponto Flutuante:
Ponto Flutuante Simples, Ponto Flutuante Duplo.

Tipo Caractere:
Caractere.

Tipo Lógico:
Booleano.

Tipos de Dados Inteiros	Faixas
byte	-128 a +127
short	-32.768 a +32.767
int	-2.147.483.648 a +2.147.483.647
long	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

O valor default de todos é 0 (zero).

Tipos de Dados em Ponto Flutuante	Faixas
float	de ± 1.40282347 x 10^{-45} até ± 3.40282347 x 10^{+38}
double	de ± 4.94065645841246544 x 10^{-324} até ± 1.79769313486231570 x 10^{+308}

Exemplos:
1.44E6 é equivalente a 1.44 x 106 = 1.440.000.
3.4254e-2 representa 3.4254 x 10-2 =0.034254.

O valor default de ambos é 0 (zero).

Instituto de Computação UFF

Tipos Primitivos

Caractere

O tipo char permite a representação de caracteres individuais.

Ocupa 16 bits internamente permitindo até 32.768 caracteres diferentes.

O valor default é 0 (zero).

Caracteres de controle e outros caracteres cujo uso é reservado pela linguagem devem ser usados precedidos por \ .

Instituto de Computação UFF

Tipos Primitivos

Caractere

\b	backspace
\t	Tabulação horizontal
\n	newline
\f	form feed
\r	carriage return
\”	aspas
\’	aspas simples
\\	contrabarra
\xxx	o caracter com código de valor octal xxx, que pode assumir valores entre 000 e 377.
\uxxxx	o caráter com código de valor hexadecimal xxxx, que pode assumir valores entre 0000 e ffff.

Tipos Primitivos

Booleano

É representado pelo tipo lógico boolean.

Assume os valores *false* (falso) ou *true* (verdadeiro).

O valor default é *false*.

Ocupa 1 bit.

Diferente da linguagem C, onde ocupa 1 byte.

Instituto de computação uff

Palavras reservadas

usadas pela linguagem

abstract	continue	finally	interface	public	throw
boolean	default	float	long	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	synchronized	while
class	final	int	protected	this	

Instituto de Computação UFF

Palavras reservadas

NÃO usadas pela linguagem

const	future	generic	goto	inner	operator
outer	rest	var	volatile		

Declaração de Variáveis

Uma variável não pode utilizar como nome uma palavra reservada da linguagem.

Sintaxe:

```
Tipo nome1 [,  nome2 [,  nome3 [...,  nomeN]]];
```

Exemplos:

```
int i;
float total, preco;
byte mascara;
double valormedio;
```

Declaração de Variáveis

Embora não seja de uso obrigatório, existe a convenção padrão para atribuir nomes em Java, como:

- ✓ Nomes de classes são iniciados por letras maiúsculas;
- ✓ Nomes de métodos, atributos e variáveis são iniciados por letras minúsculas;
- ✓ Em nomes compostos, cada palavra do nome é iniciada por letra maiúscula, as palavras não são separadas por nenhum símbolo.

Documento: Code Conventions for the JavaTM Programming Language.

Comentários

Exemplos:

```
// comentário de uma linha

/* comentário de
            múltiplas linhas */

/**  comentário de documentação
    *  que também pode
    *  possuir múltiplas linhas
    */
```

Comentários

```
/** Classe destinada ao armazenamento
     *  de dados relacionados a arquivos ou
     *  diretórios.
     *  <p> Pode ser usada para armazenar árvores de diretórios.
     *   @author Joao Jr.
     *   @see java.io.File
     */
```

Operadores Aritméticos

+	Adição	a+b
-	Subtração	a-b
*	Multiplicação	a*b
/	Divisão	a/b
%	Resto da divisão inteira	a%b
-	- Unário	-a
+	+ Unário	+a
++	Incremento unitário	++a ou a++
--	Decremento unitário	--a ou a--

Operadores Relacionais

==	Igual	a==b
!=	Diferente	a!=b
>	Maior que	a>b
>=	Maior ou igual a	a>=b
<	Menor que	a<b
<=	Menor ou igual a	a<=b

Instituto de Computação

Operadores Lógicos

&&	E lógico *(and)*	a&&b
||	Ou lógico *(or)*	a||b
!	Negação *(not)*	!a

Programa Java

Todos os programas em Java possuem quatro elementos básicos:

Pacotes

```
import java.util.*;
```

Classes

Métodos

Variáveis

```
public class HelloJavaClass {
  public final static void main(String args[]) {
        System.out.println("Hello, Java");
        Date d = new Date();
        System.out.println("Date: "+d.toString());
  }
}
```

Instituto de Computação

Fluxo de execução

Normalmente sequencial.

Comandos de controle de fluxo permitem modificar essa ordem natural de execução:

```
if (condição)
{
      bloco_comandos
}
```

Fluxo de execução

```
switch (variável)
{
    case valor1:
        bloco_comandos
        break;
    case valor2:
        bloco_comandos
        break;
    ...
    case valorn:
        bloco_comandos
        break;
    default:
        bloco_comandos
}
```

```
while (condição)
{
    bloco_comandos
}


do
{
    bloco_comandos
} while (condição);


for (inicialização; condição; incremento)
{
    bloco_comandos
}
```

Instituto de Computação

Fluxo de execução

Desvio de fluxo

If e *Switch*

Exemplo: 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");
  }
}
```

Instituto de computação uff

Fluxo de execução

Desvio de fluxo

If e *Switch*

Exemplo: Switch

```
public class exemploSwitch {

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

Repetição simples: for

Exemplo:

```
import java.io.*;

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

Fluxo de execução

Estruturas de repetição

Repetição condicional: while e do while

Exemplo: while

```
public class exemploWhile {

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

Repetição condicional: while e do while

Exemplo: 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.");
  }
}
```

Diretiva Try - Catch:

```
try
{
   Fluxo normal do sistema
}
catch (Exceção1)
{
   Diretiva do tratamento do erro 1
}
catch (Exceção2)
{
   Diretiva do tratamento do erro 2
}
```

Repetição condicional: try e catch

Exemplo: 1 exceção

```
public class exemploTryCatch1 {

  public static void main (String args[]) {
    int j = 10;
    try {
      while (j > Integer.parseInt(args[0])) {
        System.out.println(""+j);
        j--;
      }
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Não foi fornecido um argumento.");
    }
  }
}
```

Repetição condicional: try e catch

Exemplo: 2 exceções

```
public class exemploTryCatch2 {

  public static void main (String args[]) {
    int j = 10;
    try {
      while (j > Integer.parseInt(args[0])) {
        System.out.println(""+j);
        j--;
      }
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Não foi fornecido um argumento.");
    } catch (java.lang.NumberFormatException e) {
      System.out.println("Não foi fornecido um inteiro
válido.");
    }
  }
}
```

Diretiva Try - Catch - Finally:

```
try
{
   Fluxo normal do sistema
}
catch (Exceção1)
{
   Diretiva do tratamento do erro 1
}
finally
{
   Fluxo que será sempre executado, independente da
   ocorrência da exceção ou não.
   Liberação de recursos. Ex: Fechamento de arquivos.
}
```

O propósito de um array é permitir o armazenamento e manipulação de uma grande quantidade de dados de mesmo tipo.

Exemplos:

- ✓ Notas de alunos
- ✓ Nucleotídeos em uma cadeia de DNA
- ✓ Frequência de um sinal de áudio

Permitem acesso direto aos elementos de uma representação de uma coleção de dados

Mapeiaum conjunto finito de índices em um conjunto qualquer de objetos de mesmo tipo.

$F(x) \rightarrow S, x \in U$ tal que U é um conjunto finito de índices e S o conjunto dos objetos.

Arrays

Unidimensionais

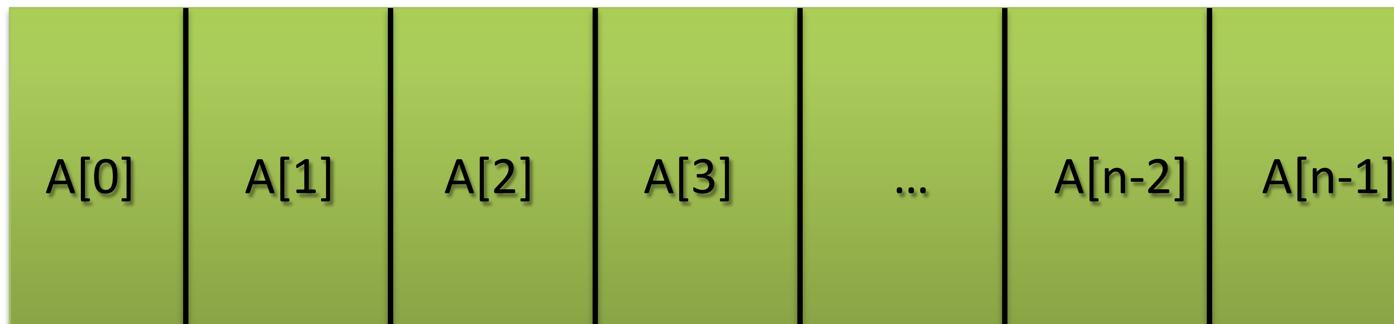
Os elementos de um array são identificados através de índices.

Arrays cujos elementos são indicados por um único índice são denominados arrays unidimensionais.

Arrays

Unidimensionais

Um elemento em uma posição indicada por um índice i, em um array A, é acessado através do identificador do array seguido do índice i entre chaves.



Arrays

Unidimensionais

A criação de um array em Java requer 3 passos:

1. Declaração do nome do array e seu tipo
2. Alocação do array
3. Inicialização de seus valores

```
double[] a;
a = new double[10];
for (int i = 0; i<10;i++)
    a[i] = 0.0;
```

O número de elementos de um array em Java pode ser determinado através do método length()

Exemplo:

a.length()

Arrays em Java são objetos.

(mais detalhes serão vistos posteriormente)

Arrays em Java tem índice base igual a zero.

Arrays

Unidimensionais

Arrays em Java podem ser inicializados em tempo de compilação.

Exemplos:

```
String[ ] naipe = {“copas”,”ouros”, “paus”,”espadas”};
double[ ] temperaturas = {45.0,32.0,21.7,28.2,27.4};
```

Arrays multidimensionais representam agregados homogêneos cujos elementos são especificados por mais de um índice.

Em Java é muito simples especificar um array multidimensional.

Exemplo:
array contendo as notas de 3 provas de 30 alunos

```
int[][] notas = new int[30][3];
```

In / Out

Introdução

Em Java existem diversas formas de tratarmos a entrada e saída de dados:

- ✓ Através da tela (console);
- ✓ Através de janelas gráficas (diálogos);
- ✓ Através de arquivos;
- ✓ Etc ...

Hoje: Console e diálogos

In / Out

Introdução

Nos acostumamos a escrever linhas de código como:

```
System.out.println("Nome do Aluno: " + nome);
System.out.println("Velocidade do Carro: " + 10);
```

Quando desejamos precisamos enviar mensagens ao usuário através do console.

"Console" = "command window" (Windows)
= "terminal" (Linux, Mac)

O Termo *saída padrão (stdout)* se refere a este tipo de saída de dados.

In / Out

Escrevendo na saída padrão

O objeto `System.out` gerencia a tarefa de escrevermos a saída do programa no dispositivo de saída padrão.

Obs:

Este objeto é automaticamente criado pelo Java.

Controlado por 2 métodos:

`print()`: Imprime uma saída no console.
`println()`: Imprime uma saída no console e pula uma linha.

In / Out

Escrevendo na saída padrão

Os métodos `System.out.print[ln]` podem receber como argumento:

- ✓ **Uma String**

```
System.out.print("Entre com o número de tentativas");
```

- ✓ **Um número ou uma variável**

```
int x = 56; System.out.print(x);
System.out.println(18.45);
```

- ✓ **Combinações dos casos anteriores**

```
float media = calculaMedia();
System.out.println("A média da prova foi: " + media );
```

Instituto de computação uff

In / Out

Escrevendo na saída padrão

Obs:

Usamos o operador + para combinarmos ítem na saída.

Ex:

1.

```
float media = calculaMedia();
...
System.out.print("Alunos com nota acima de ");
System.out.print( media );
System.out.print(" estão aprovados\n");
```

2.

```
float media = calculaMedia();
...
System.out.println("Alunos com nota acima de " + media + " estão aprovados");
```

In / Out

Lendo da entrada padrão

O objeto `System.in` gerencia a tarefa de lermos dados a partir do dispositivo de entrada padrão.

Obs:

Este objeto é automaticamente criado pelo Java.

O Termo *entrada padrão (stdin)* se refere a entrada de dados a partir do console.

Lendo da entrada padrão

O uso do objeto `System.in` é mais complexo que o do System.out:

O objeto `System.in` lê um bit por vez.

Porém ...

Tipicamente desejaremos ler mais de um bit por vez.

Instituto de Computação

In / Out

Lendo da entrada padrão

Passo a passo para a leitura de uma linha:

Passo 1:
Criar um objeto do tipo **`InputStreamReader`**
Leitura bit a bit

Passo 2:
Criar um objeto do tipo **`BufferedReader`**
Leitura de uma linha de texto (até um caracter '\n')

Passo 3:
Usar o método *readLine* da classe **BufferedReader**.

In / Out

Lendo da entrada padrão

```
import java.io.*;
...
public static void main(String[] args) {
      InputStreamReader in = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(in);

      String name, ageStr;
      System.out.println(“Qual o seu nome?”);
      name = br.readLine();
      System.out.println(“Qual sua idade?”);
      ageStr = br.readLine();
}
```

Leitura de dados numéricos

No exemplo anterior, a idade é um dado numérico. Entretanto, a leitura trata o valor como uma String:

```
ageStr = br.readLine();
```

Precisamos converter números manualmente:

Inteiros: `Integer.parseInt();`
Float: `Float.parseFloat();`
...

In / Out

Lendo da entrada padrão

```
import java.io.*;
...

public static void main(String[] args) {
     InputStreamReader in = new InputStreamReader(System.in);
     BufferedReader br = new BufferedReader(in);

     String name, ageStr;
     System.out.println("Qual o seu nome?");
     name = br.readLine();
     System.out.println("Qual sua idade?");
     ageStr = br.readLine();
     int age = Integer.parseInt(ageStr);

     ...
}
```

In / Out

Classe Scanner

O Java inclui uma classe para tornar a entrada via teclado mais simples: Scanner

Para usarmos a classe Scanner precisamos do comando: `import java.util.Scanner`

In / Out

Classe Scanner

Para criar um objeto da classe Scanner devemos indicar o objeto System.in:

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

In / Out

Classe Scanner

A classe Scanner contém métodos para a leitura:

De valores inteiros:

```
int idade = keyboard.nextInt();
```

De valores double:

```
float preco= keyboard.nextFloat();
```

De valores String:

```
String word1 = keyboard.next();
```

Da próxima linha:

```
String line = keyboard.nextLine();
```

obs: a leitura da linha acaba em um caracter '\n'

In / Out

Classe Scanner

Entradas multiplas devem ser separadas por *whitespaces* e lidas por multiplas chamandas do método apropriado:

Whitespaces são string de caracteres tais como: espaço, tabulações e quebras de linha.

Ex:

Dado o código:

```
String word1 = keyboard.next();
String word2 = keyboard.next();
```

E a entrada via teclado:

casa carro

O valor de **word1** será **casa**, e o valor de **word2** será **carro.**

Instituto de Computação UFF

In / Out

Classe Scanner

```
import java.io.*;
...

public static void main(String[] args) {
     Scanner in = new Scanner(System.in);

     String name;
     System.out.println("Qual o seu nome?");
     name = in.next();

     int age;
     System.out.println("Qual sua idade?");
     age = in.nextInt();
}
```

Desafio:

Dado o Código:

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();

String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

e a entrada:

2
Carros são mais caros que
1 Casa.

Quais os valores de n, s1 e s2 ?

In / Out

Saída gráfica

Consiste em criar uma janela na tela contendo a mensagem desejada.

Pode ser feita usando:

1. JFrame
2. JDialog

Trabalharemos, por enquanto, com o JDialog.

In / Out

Saída gráfica

Exemplo 01:

```
import javax.swing.*;
...

public static void main(String[] args) {
  JOptionPane.showMessageDialog(null,"Esta é uma janela de
  diálogo criada no centro da tela");
}
```

Exemplo 02:

```
import javax.swing.*;
...

public static void main(String[] args) {
  JFrame win = new JFrame();
  win.setSize(200,200);
  win.setVisible(true);
  JOptionPane.showMessageDialog(win, “Esta é uma janela de
  dialogo criada em uma posição específica”);
}
```

Instituto de Computação UFF

In / Out

Entrada gráfica

Recebe os dados através de um diálogo exibido na tela.

Muito parecido com os diálogos de saida.

Exemplo:

```
import javax.swing.*;
...

public static void main(String[] args) {
  String name, ageStr;
  name = JOptionPane.showInputDialog(null, “Qual o seu nome ?”);
  ageStr = JOptionPane.showInputDialog(null, “Qual a sua idade?”);
  int age = Integer.parseInt(ageStr);
}
```

Arquivos

Introdução

Todo sistema computacional de alguma forma, armazena e manipula informações.

O conceito de informação inclui diversas noções tais como conhecimento, dados, padrões, representação, comunicação, significado, etc.

Uma noção muito importante diz respeito à representação de entidades (concretas ou abstratas) segundo suas propriedades.

Uma entidade é algo que pode ser identificável segundo um conjunto de características.

Exemplo: *Um aluno, um produto, um empregado, etc.*

Um conjunto de entidades com características comuns forma uma classe de entidades.

Exemplo: *Itens de um supermercado, alunos de uma turma, automóveis, etc.*

Uma entidade é caracterizada por um conjunto de atributos que tomam valores pertencentes a um dado domínio

Um conjunto de pares <atributo, valor> determina um registro.

Exemplo:

Atributos:	Nome	matricula	curso	CR
Valores:	Pedro	1123907	comp	8.9

Instituto de computação

Arquivos

Introdução

Um *conjunto de registros* armazenados em memória principal é denominado *tabela*, e quando armazenado em memória secundária é denominado *arquivo*.

Cristiano Prog Comp I A1
Dante Prog Comp I B1
Esteban Prog Comp I C1
Anselmo Prog Comp II A1
Isabel Prog Comp II B1
Dante Prog Comp II C1
Ferraz Estruturas de Dados I A1
Isabel Estruturas de Dados I B1

11013102	8.0
11023506	7.1
12340718	2.1
25314080	3.0
13485710	9.0

É necessário especificar um esquema para que os algoritmos possam acessar registros sem a necessidade de passar todos os valores associados aos atributos.

Isto pode ser feito através de um subconjunto de atributos denominado chave.

Uma chave destaca um único registro (chave primária), um subconjunto de registros (chave secundária) ou todos os registros (chave nula).

Chave primária:

Exemplo 1: (professor, disciplina)

Exemplo 2: matrícula

Chave secundária:

Exemplo 1: disciplina

Exemplo 2: código de turma

Elemento de chave: expressão que especifica um conjunto de valores associados a um atributo.

Exemplo: (nota >= 8)

Seleção: critério que destaca um ou mais registros de um arquivo.

Tipos de arquivos: arquivos podem ser essencialmente classificados em dois tipos: arquivos binários e arquivos texto.

Arquivos

Introdução

Arquivos binários: armazenam as informações utilizando o sistema de numeração binário.

(caracteres são armazenados através de seu código numérico).

Os registros são da forma como estão representados em memória.

A menor unidade de informação capaz de ser lida ou escrita é o registro.

Os registros estão dispostos de forma sequencial no arquivo :

Exemplo: 01000110101011010010 1 ...

Arquivos de texto: os dados são representados sob a forma de caracteres.

A menor unidade de informação é o caractere.

São tipicamente maiores que os arquivos binários.

Os registros podem ser interpretados diretamente.

Os dados são organizados sequencialmente.

Arquivos

Introdução

Arquivos podem ser classificados ainda como sequenciais e arquivos em série.

Arquivos sequenciais são aqueles em que seus os registros são acessados linearmente, isto é, o registro de ordem i só pode ser acessado após o registro de ordem i-1

Este tipo de arquivo é adequado para operações que envolvam o percorrimento de todos os registros.

Exemplos:

geração de folhas de pagamento, listas de presença, transações em arquivos bancários etc.

Instituto de Computação

Nos **arquivos em série**, os registros são organizados seguindo um critério de ordenação com base no valor de uma chave ou subconjunto de chaves.

São utilizados tipicamente em processos nos quais as operações envolvem a seleção ou consulta, de um subconjunto do total de registros.

Arquivos

fluxos

Muitos programas em Java precisam interagir com diferentes fontes de dados.

Estas fontes podem ser arquivos armazenados em um disco rígido ou CD-Rom, páginas Web e até mesmo na memória do computador.

Em Java existe um mecanismo unificado para lidar com estas diferentes fontes.

O mecanismo de comunicação utilizado por Java se baseia no conceito de fluxo (stream).

Instituto de computação uff

Um fluxo ou *stream* é um caminho atravessado por dados em um programa.

Fluxos podem ser de:

Entrada de dados.

envia dados de uma origem para o programa.

Saída de dados.

envia dados do programa para um destino.

Ainda ...

Fluxos de bytes.

Fluxos de caracteres.

Fluxos de bytes: transportam inteiros com valores de 0 a 255.

Uma grande variedade de dados pode ser expressa no formato de byte, incluindo:

Dados numéricos

Programas executáveis

Comunicações pela Internet

Bytecodes

Fluxos de caracteres: tipo especializado de *stream* de bytes que transportam somente dados textuais.

Diferenciam-se do stream de bytes devido ao fato de Java aceitar caracteres Unicode.

Dados que podem ser expressos como fluxos de caracteres:

Arquivos texto

Páginas web

Como usar um fluxo? os passos para manipulação de fluxos de bytes e caracteres são praticamente os mesmos!

Lidando com um fluxo de entrada:

Passo1:

criar um objeto de fluxo, por exemplo FileInputStream, associado à origem dos dados.

Passo2:

utilizar um ou mais métodos, por exemplo read(), para ler informações a partir da origem.

Passo3:

invocar o método close() para indicar o término do uso do fluxo.

Como usar um fluxo? os passos para manipulação de fluxos de bytes e caracteres são praticamente os mesmos!

Lidando com um fluxo de saída:

Passo1:

criar um objeto de fluxo, por exemplo FileOutputStream, associado ao destino dos dados.

Passo2:

utilizar um ou mais métodos, por exemplo write(), para escrever informações no destino.

Passo3:

invocar o método close() para indicar o término do uso do fluxo.

Tratamento de exceções: existem várias exceções no pacote java.io quando se trabalha com fluxos.

Exemplos:

FileNotFound – disparada quando tenta-se criar um objeto de fluxo para uma fonte que não pode ser localizada.

EOFException – final de arquivo atingido inesperadamente, enquanto os dados são lidos de um arquivo através de um fluxo de entrada.

Ambas as exceções são especializações de IOException.

Obs: Uma forma de lidar com tais exceções é delimitar comandos de entrada e saída com blocos try-catch que trata objetos IOException.

Fluxo de bytes: são tratados por subclasses de InputStream e OutputStream. Essas são classes abstratas e portanto não são instanciáveis.

Ao invés de usá-las, utilizamos suas subclasses como, por exemplo:

1) FileInputStream e **FileOutputStream** – usadas para lidar com fluxos de bytes armazenados no disco, CD-Rom ou outros dispositivos de armazenamento.

2) DataInputStream e **DataOutputStream** – usadas para lidar com fluxos de bytes filtrados (que veremos em seguida).

Arquivos

fluxos

FileInputStream:

Um fluxo de entrada de arquivo pode ser criado com o construtor FileInputStream (String), onde String é o nome do arquivo.

Exemplos:

```
FileInputStream f = new FileInputStream(“arq.dat”);
FileInputStream f = new FileInputStream(“c:\\Dados\\arq.dat”); \\ Windows
FileInputStream f = new FileInputStream(“/Dados/arq.dat”); \\ Linux
```

FileInputStream:

após criado o fluxo, bytes podem ser lidos utilizando o método read().

Para ler mais de um byte utiliza-se read(byte[],int,int).

Os argumentos tem o seguinte significado, em ordem:

1) Um array de bytes onde os dados serão armazenados.

2) O elemento dentro do array onde o primeiro byte será armazenado.

3) O número de bytes a serem lidos.

Assim método read(byte[],int,int) retorna o número de bytes lidos ou -1 se nenhum byte tiver sido lido antes de ser atingido o fim do fluxo.

Instituto de Computação UFF

Arquivos

fluxos

FileInputStream (leitura de bytes):

```
import java.io.*;
public class ReadBytes {
    public static void main(String[] arguments) {
        try {
            FileInputStream file = new FileInputStream("class.dat");
            boolean eof = false;
            int count = 0;
            while (!eof) {
                int input = file.read();
                if (input != -1) {
                    System.out.print(input + " "); count++;
                 }
                else  eof = true;
             }
            file.close();
            System.out.println("\nBytes read: " + count);
        } catch (IOException e) {
            System.out.println("Error -- " + e.toString());
        }
      }
  }
```

FileOutputStream:

pode ser criado com o contrutor FileOutputStream (String), onde String é o nome do arquivo.

Exemplo:

```
FileOutputStream f = new FileOutputStream(“arq.dat”);
```

Se o arquivo especificado no argumento do construtor for existente, este será **apagado** quando começar a gravação dos dados no fluxo

É possível criar um fluxo de saída que acrescente dados após o final de um arquivo existente com o construtor FileOutputStream (String, boolean).

FileOutputStream:

após criado o fluxo, bytes podem ser escritos utilizando o método write().

Para escrever mais de um byte utiliza-se write(byte[],int,int).

Os argumentos tem o seguinte significado, em ordem:

1) Um array de bytes onde os dados a serem escritos são armazenados.

2) O elemento dentro do array que contém o primeiro byte a ser escrito.

3) O número de bytes a escritos lidos.

Arquivos

fluxos

FileOutputStream (leitura de bytes):

```
import java.io.*;
public class WriteBytes {
    public static void main(String[] arguments) {
        int[] data = { 71, 73, 70, 56, 57, 97, 13, 0, 12, 0, 145, 0,
            0, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 44, 0,
            0, 0, 0, 13, 0, 12, 0, 0, 2, 38, 132, 45, 121, 11, 25,
            175, 150, 120, 20, 162, 132, 51, 110, 106, 239, 22, 8,
            160, 56, 137, 96, 72, 77, 33, 130, 86, 37, 219, 182, 230,
            137, 89, 82, 181, 50, 220, 103, 20, 0, 59 };
        try {
            FileOutputStream file = new   FileOutputStream("pic.gif");
            for (int i = 0; i < data.length; i++)
                file.write(data[i]);
            file.close();
        } catch (IOException e) {
            System.out.println("Error -- " + e.toString());
        }
    }
}
```

Instituto de computação

Fluxos filtrados:

são fluxos que modificam as informações enviadas por meio de um fluxo existente.

São criados usando subclasses de *FilterInputStream* e *FilterOutputStream* que são abstratas.

Ao invés de usá-las, utilizamos suas subclasses como, por exemplo:

1) BufferedInputStream(InputStream) e BufferedOutputStream(OutputStream) – criam fluxos de buffer de entrada e saída, respectivamente, para os objetos de fluxo de especificados.
2) BufferedInputStream(InputStream,int) e BufferedOutputStream (OutputStream, int) – criam fluxos de buffer de entrada e saída, com buffer de tamanho determinado pelo segundo argumento, respectivamente, para os objetos de fluxo de especificados.

Fluxos filtrados:

Nos fluxos de buffer, os métodos para leitura e escrita são similares aos fluxos não bufferizados.

Quando os dados são direcionados para um fluxo de buffer, eles não saem direto para seu destino até que o buffer seja preenchido ou o método flush() seja invocado.

Arquivos

fluxos

Fluxos filtrados (leitura e escrita):

```
import java.io.*;

public class BufferDemo {
    public static void main(String[] arguments) {
        int start = 0;
        int finish = 255;
        if (arguments.length > 1) {
            start = Integer.parseInt(arguments[0]);
            finish = Integer.parseInt(arguments[1]);
        } else if (arguments.length > 0)
            start = Integer.parseInt(arguments[0]);
        ArgStream as = new ArgStream(start, finish);
        System.out.println("\nWriting: ");
        boolean success = as.writeStream();
        System.out.println("\nReading: ");
        boolean readSuccess = as.readStream();
    }
}
```

Arquivos

fluxos

Fluxos filtrados (leitura e escrita):

```
class ArgStream {
    int start = 0;
    int finish = 255;

    ArgStream(int st, int fin) {
        start = st;
        finish = fin;
    }

    boolean writeStream() {
        try {
            FileOutputStream file = new FileOutputStream("numbers.dat");
            BufferedOutputStream buff = new BufferedOutputStream(file);
            for (int out = start; out <= finish; out++) {
                buff.write(out);
                System.out.print(" " + out);
            }
            buff.close();
            return true;
        } catch (IOException e) {
            System.out.println("Exception: " + e.getMessage());
            return false;
        }
    }
// continua na próxima caixa de texto
```

Instituto de computação uff

Arquivos

fluxos

Fluxos filtrados (leitura e escrita):

```
    // continuação da caixa de texto anterior

     boolean readStream() {
         try {
             FileInputStream file = new FileInputStream("numbers.dat");
             BufferedInputStream buff = new BufferedInputStream(file);
             int in = 0;
             do {
                 in = buff.read();
                 if (in != -1)
                     System.out.print(" " + in);
             } while (in != -1);
             buff.close();
             return true;
         } catch (IOException e) {
             System.out.println("Exception: " + e.getMessage());
             return false;
         }
     }
}
```

Fluxos de caracteres:

são usados para trabalhar com qualquer texto que seja representado por um conjunto de caracteres ASCII ou Unicode.

As classes usadas para ler e escrever em fluxos de caracteres são subclasses de Reader e Writer que são abstratas.

Ao invés de usá-las, utilizamos suas subclasses como:

InputStreamReader(String) e OutputStreamWriter(String) – criam fluxos de caracters de entrada e saída, respectivamente.

É comum o uso da subclasse FileReader de InputStreamReader e a subclasse FileWriter de OutputStreamWriter.

FileReader (leitura de carac):

```
import java.io.*;
public class ReadSource {
    public static void main(String[] arguments) {
        try {
            FileReader file = new  FileReader("ReadSource.java");
            BufferedReader buff = new   BufferedReader(file);
            boolean eof = false;
            while (!eof) {
                String line = buff.readLine();
                if (line == null)   eof = true;
                else   System.out.println(line);
            }
            buff.close();
        } catch (IOException e) {
            System.out.println("Error -- " + e.toString());
        }
    }
}
```

Fluxos de saída (escrita de carac):

```
import java.io.*;

public class AllCapsDemo {
    public static void main(String[] arguments) {
        AllCaps cap = new AllCaps(arguments[0]);
        cap.convert();
    }
}

class AllCaps {
    String sourceName;

    AllCaps(String sourceArg) {
        sourceName = sourceArg;
    }

    void convert() {
        try {
            // Create file objects
            File source = new File(sourceName);
            File temp = new File("cap" + sourceName +
".tmp");

            // Create input stream
            FileReader fr = new
                FileReader(source);
            BufferedReader in = new
                BufferedReader(fr);
// continua na caixa de texto ao lado
```

```
            // Create output stream
            FileWriter fw = new
                FileWriter(temp);
            BufferedWriter out = new
                BufferedWriter(fw);

            boolean eof = false;
            int inChar = 0;
            do {
                inChar = in.read();
                if (inChar != -1) {
                    char outChar = Character.toUpperCase
( (char)inChar );
                    out.write(outChar);
                } else
                    eof = true;
            } while (!eof);
            in.close();
            out.close();

            boolean deleted = source.delete();
            if (deleted)
                temp.renameTo(source);
        } catch (IOException e) {
            System.out.println("Error -- " + e.toString());
        } catch (SecurityException se) {
            System.out.println("Error -- " + se.toString());
        }
    }
}
```

Continua ...

Arquivos de acesso direto (Random Acess Files):

são formas de organização de arquivos que permitem acesso direto a uma dada posição, sem a necessidade de percorrer as posições anteriores em sequencia.

Quando o arquivo é organizado em registros, cada registro ocupa uma posição ou ordem i que pode ser acessada diretamente.

O primeiro registro possui ordem zero.

Instituto de computação uff

Arquivos

fluxos

Os registros tipicamente tem tamanho fixo onde cada campo ocupa um número de bytes pré-determinado.

O registro de ordem 2, por exemplo, está no byte 44.

registro

10 bytes										4 bytes(int)	8 bytes(double)
s	a	b	ã	o						11	6,25
e	s	c	o	v	a					12	7,45
p	a	s	t	a						13	9,99
f	i	o		d	e	n	t	a	l	14	7,25
b	o	l	a	c	h	a				21	4,45
b	o	m	b	o	m					22	12,00
b	i	s	c	o	i	t	o			23	3,45
v	i	n	h	o						31	20,00
c	a	c	h	a	ç	A				32	14,00

Os registros tipicamente tem tamanho fixo onde cada campo ocupa um número de bytes pré-determinado.

O registro de ordem 2, por exemplo, está no byte 44.

seek(TAM_REG*4)

registro

10 bytes										4 bytes(int)	8 bytes(double)
s	a	b	ã	o						11	6,25
e	s	c	o	v	a					12	7,45
p	a	s	t	a						13	9,99
f	i	o		d	e	n	t	a	l	14	7,25
b	o	l	a	c	h	a				21	4,45
b	o	m	b	o	m					22	12,00
b	i	s	c	o	i	t	o			23	3,45
v	i	n	h	o						31	20,00
c	a	c	h	a	ç	A				32	14,00

Arquivos

fluxos

public void seek(long pos) throws IOException.

Posiciona o ponteiro para o arquivo, medido a partir do seu início, na posição em que ocorrerá a próxima leitura ou escrita.

O deslocamento (ponteiro) pode ser posicionado além do fim do arquivo, porém o tamanho do arquivo somente mudará através de uma operação de escrita.

Parâmetros:

pos – a nova posição do ponteiro, medida em bytes a partir do início do arquivo.

Exceções: IOException – se pos for menor que zero ou se um erro de IO ocorrer.

Instituto de computação uff

Arquivos

fluxos

Arquivos de acesso direto:

no exemplo abaixo uma imagem em formato Raw de 64x64 pixels em tons de cinza é lida e corrompida com ruído

```
public class ImageRandomAccessFile {

    public static void main(String[] args) throws IOException {
        ImageRawGrayScale image = new ImageRawGrayScale(64,64);
        image.load("pic");
        image.print();
        image.save("picnoise");

        RandomAccessFileIO randomAccessFile = new RandomAccessFileIO();
        randomAccessFile.open("picnoise", "rw");
        try {
            randomAccessFile.addNoise(0.1);
        } finally {
            randomAccessFile.close();
        }

        ImageRawGrayScale image2 = new ImageRawGrayScale(64,64);
        image2.load("picnoise");
        image2.print();
    }
}
```

Arquivos

fluxos

```
public class ImageRawGrayScale {
  private int width,height;
  private int[] data;

  public ImageRawGrayScale(int width,int height){
      this.width = width;
      this.height = height;
      this.data = new int[width*height];
  }

  public int getWidth() {
      return width;
  }

  public int getHeight() {
      return height;
  }

  public int getGrayLevel(int i,int j){
      return data[i*width+j];
  }
  public void load(String fileName) throws IOException{
      FileInputStream file = new FileInputStream
(fileName);
      try {
          for (int i = 0; i < data.length; i++)
              data[i] = file.read();
      } finally {
          file.close();
      }
  }
```

```
  public void save(String fileName) throws IOException {
      FileOutputStream file = new FileOutputStream(fileName);
        try {
            for (int i = 0; i < data.length; i++)
                file.write(data[i]);
        } finally {
            file.close();
        }
  }

  public void print(){
      for (int i=0;i<height;i++){
          for (int j=0;j<width;j++){
              int grayLevel = getGrayLevel(i,j);
              if (grayLevel==255)
                  System.out.print("  ");
              else if (grayLevel>0)
                  System.out.print("++");
              else
                  System.out.print("oo");
          }
          System.out.println();
      }
  }
}
```

Arquivos

fluxos

Acesso Direto (Registro = 1 Byte):

```
import java.io.*;

public class RandomAccessFileIO {

  private RandomAccessFile raf=null;

  public RandomAccessFileIO(){
  }

  public void open(String fileName, String ioArgs)
    throws FileNotFoundExceptiono {
      if (raf == null)
          raf = new RandomAccessFile(fileName, ioArgs);
  }

  public void close() throws IOException {
      if (raf != null) {
          raf.close();
          raf = null;
      }
  }
```

```
 public void write(int pos, int value) throws IOException {
      raf.seek(pos);
      raf.write(value);
  }

  public int read(int pos, int value) throws IOException {
      raf.seek(pos);
      return raf.read();
  }

  public void addNoise(double percentage) throws IOException{
      for (int i = 0; i < raf.length() * percentage; i++) {
          int pos = (int) (Math.random() * (raf.length()-1));
          raf.seek(pos);
          raf.writeByte(128);
      }
}
```

Serialização:

mecanismo da linguagem Java que permite persistir objetos, isto é, eles passam a existir mesmo quando o programa não está sendo executado

Para que um objeto possa ser persistido, ele precisa ser serializado, isto é, decomposto em cada um de seus elementos que são enviados para memória secundária em série, como numa linha de montagem

Para que um objeto possa ser serializado ele deve implementar a interface Serializable.

Não é necessário implementar nenhum método já que Serializable não especifica nenhum deles.

Serialização:

Quando um objeto é persistido, todos os objetos que o compõem também são desde que implementem Serializable.

Algumas variáveis de instância podem não ser persistidas: para isto devem ser declaradas como transient.

Arquivos

Exercício – Mineração de dados

Considere um conjunto documentos na forma de arquivos textos.

Escreva um programa que receba um conjunto de documentos e determine o grau de similaridade entre cada um deles.

Cada documento deve disponibilizar um modo de medir sua similaridade quando confrontado a outro de mesma natureza

Arquivos

Exercício

Cada documento é caracterizado por um perfil (*profile*).

Os perfis devem ser capazes de distinguir documentos diferentes.

Existem inúmeras formas de se definir perfis.

Uma possibilidade é usar a frequência de elementos no documento

Instituto de computação

Arquivos

Exercício

Um exemplo de perfil é construído com base no conceito de frequência de *k-grams*

Um *k-gram* é uma substring de tamanho *k* na string que compõe o documento

Para construir um histograma de *k-grams* é necessário associar um número a uma string (*hashing*)

Arquivos

Exercício

Como comparar os perfis?

Através de um métrica que meça a distância entre os vetores unitários associados ao histograma

Duas possibilidades: métrica euclidiana e produto escalar

Instituto de Computação UFF

Arquivos

Exercício2

Que mudanças podem ser necessárias caso:

a) for preciso trabalhar com outros Documentos;

b) for permitido usar diferentes formas de descrever os documentos ;

c) for preciso usar métricas diferentes para comparar as similaridades entre os perfis;

d) for permitida diferentes visualizações dos resultados na interface gráfica

e) as configurações puderem ser feitas dinamicamente em tempo de execução