Programação de computadores II

Conteúdo:

- Introdução à Linguagem de programação Java

Histórico (1/3)

- Início em 1991, com um 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), que será renomeada para Java devido a problemas de direitos autorais.

Histórico (2/3)

- 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"*, através de *applets, junto com o conteúdo* **(*HTML por si só não faz isso*)**.
- Em 1994 Jonathan Payne e Patrick Naughton desenvolveram o programa navegador *WebRunner*.

Histórico (3/3)

- 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).
- Para Sun Solaris e Microsoft Windows 95/NT.
- Progressivamente surgiram *kits* para outras plataformas como Linux e Applet Macintosh.

Características da linguagem Java

- simples,
- orientada a objeto,
- distribuída,
- alta performance,
- robusta,
- segura,
- interpretada,
- neutra,
- portável,
- dinâmica e
- *multithread*.

Simples e orientada a objetos

- É uma *linguagem simples* de fácil aprendizado.

- É uma linguagem *puramente orientada a objetos*.

- A abordagem de OO permite o desenvolvimento de sistemas de uma forma mais natural.

Distribuída

- Java foi projetada para trabalhar em um ambiente de redes

- Na realidade, Java não é uma linguagem para programação distribuída; apenas *oferece bibliotecas para facilitar o processo de comunicação*.

Alta performance

- Java é uma *linguagem interpretada*, logo ela nunca será tão rápida quanto as linguagens compiladas.

- Java chega a ser 20 vezes mais lento que C.

- Compiladores *just in time* (JIT), *que interpretam os bytecodes para um código nativo* durante a execução.

Robusta e segura

- Java possui as seguintes características que contribuem para torná-la 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***.

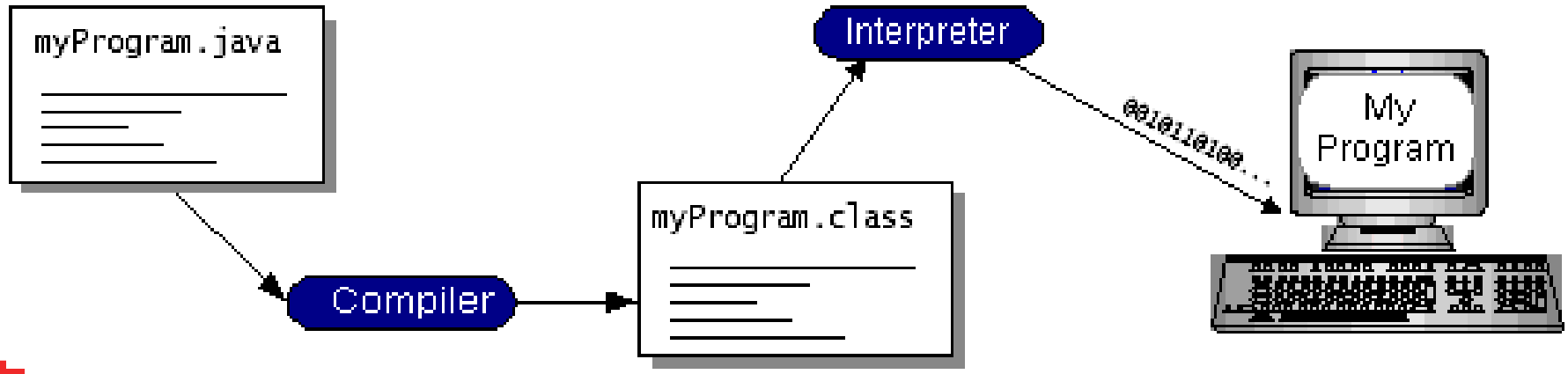
- **Segurança**: Java possui mecanismos de segurança que podem no caso de *applets*, evitar qualquer operação no sistema de arquivos da máquina alvo, minimizando problemas.

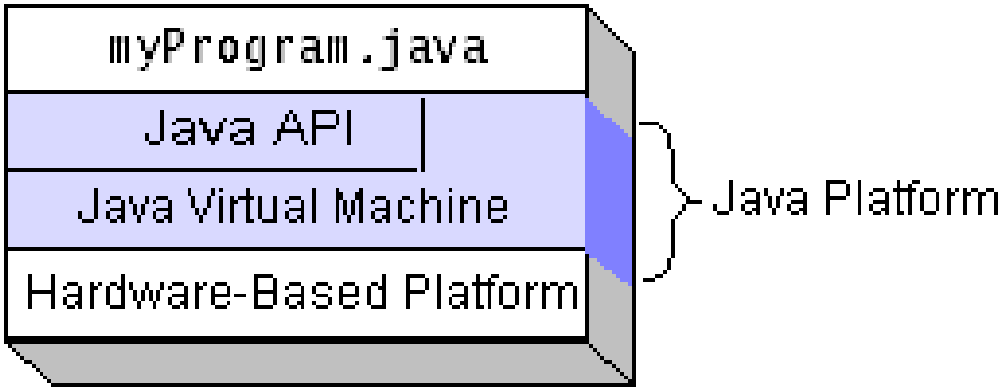
Interpretada, Neutra, Portável (1/3)

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

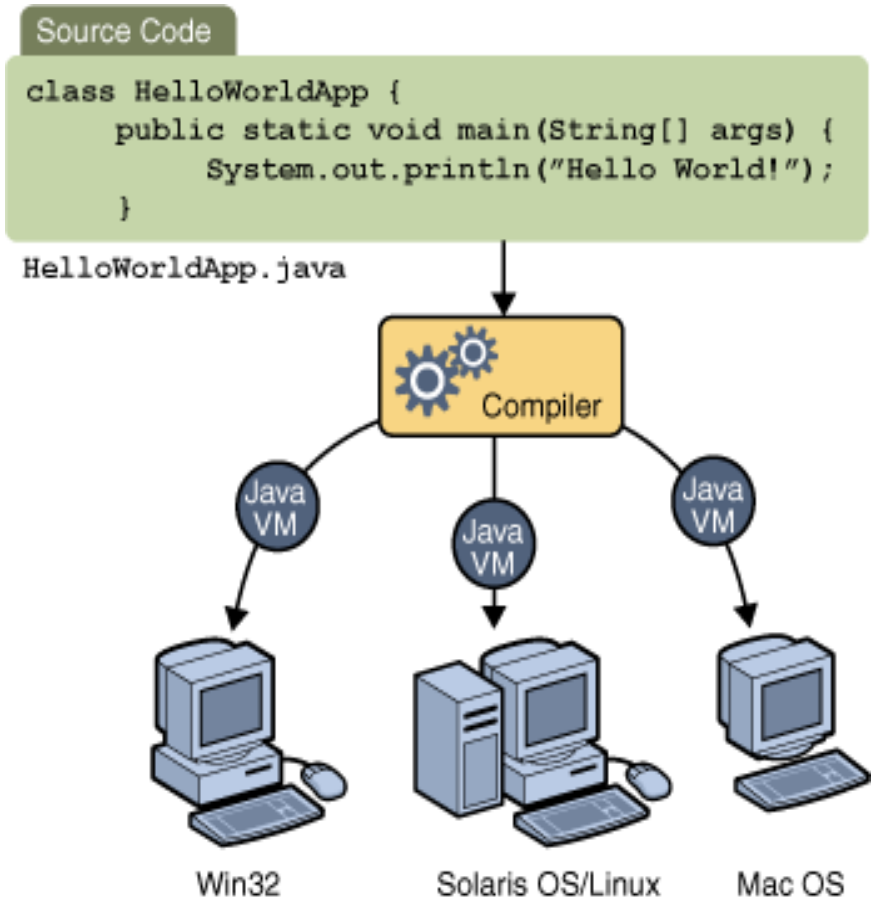
- A característica de ser *neutra em relação à arquitetura* permite uma grande *portabilidade*.

Interpretada, Neutra, Portável (2/3)





Interpretada, Neutra, Portável (3/3)



Dinâmica e Multithread

- Java possui mecanismos para a resolução de referências em tempo de execução, permitindo flexibilidade nas aplicações, sobre o custo da performance.

- Java provê *suporte para múltiplas threads de execução* (processos leves), que podem *tratar diferentes tarefas concorrentemente*.

O Ambiente Java (1/2)

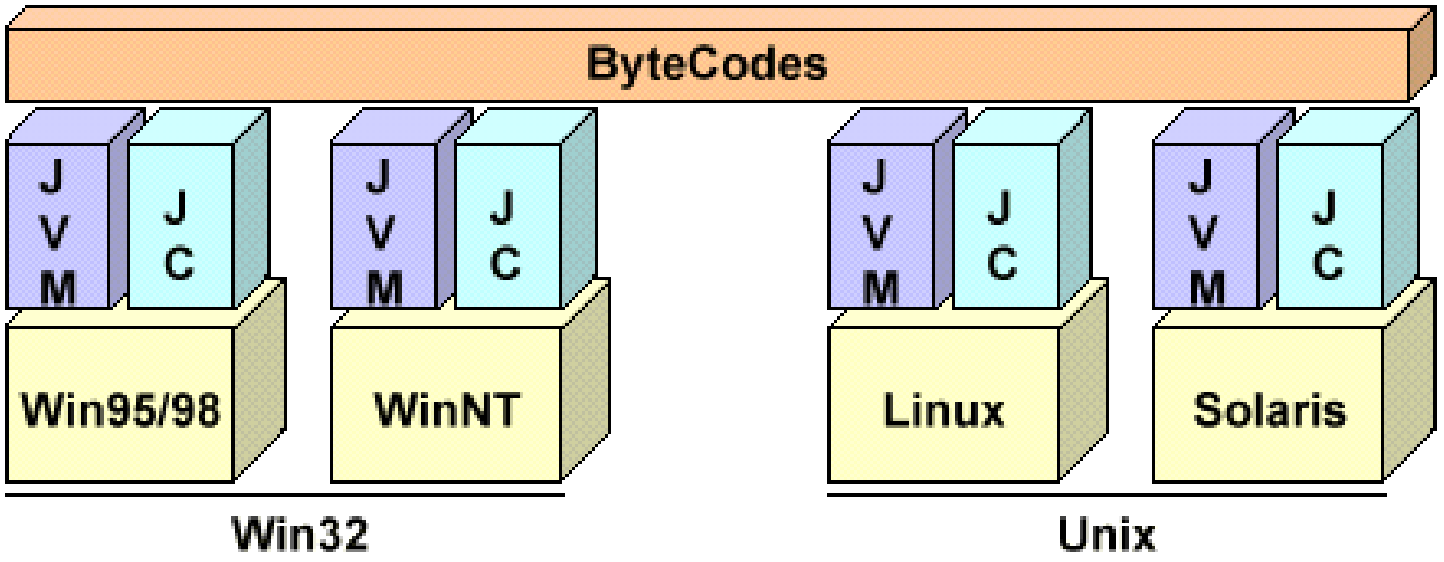
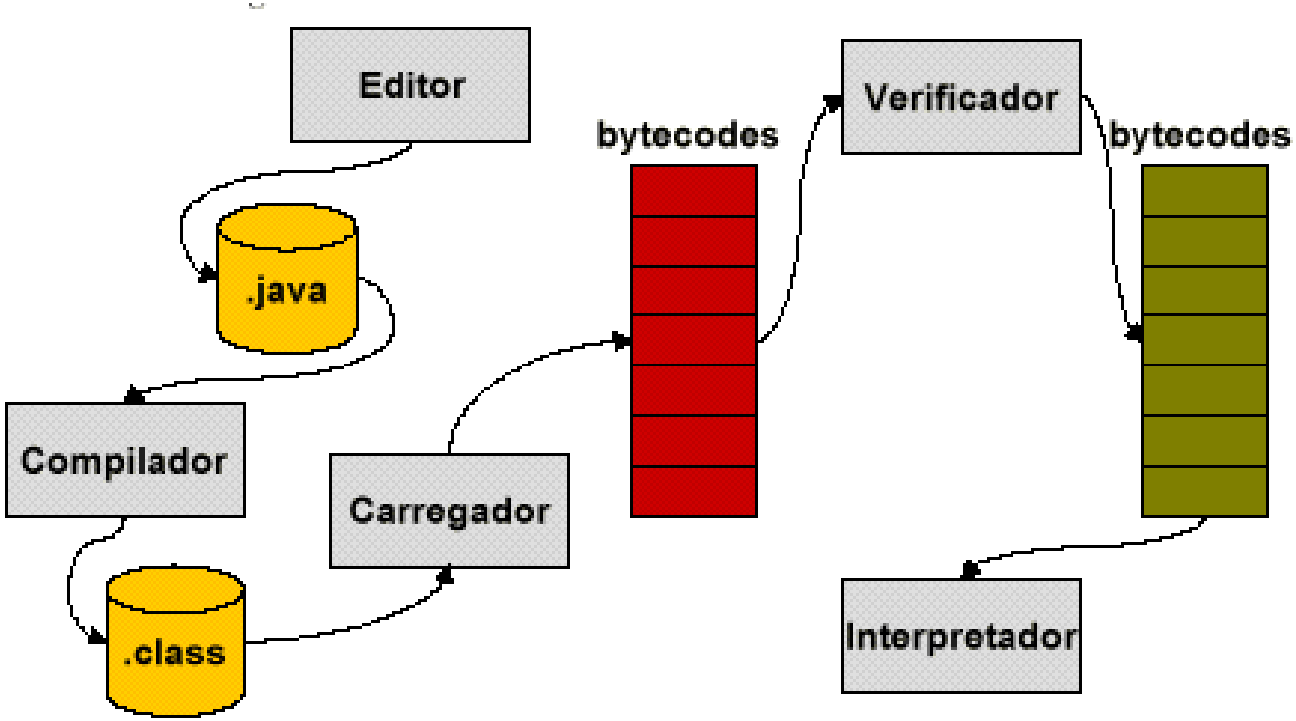


Figura 3 Ambiente Java e os *Bytecodes*

O Ambiente Java (2/2)



Ambiente de Desenvolvimento (1/2)

- 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*).

Ambiente de Desenvolvimento (2/2)

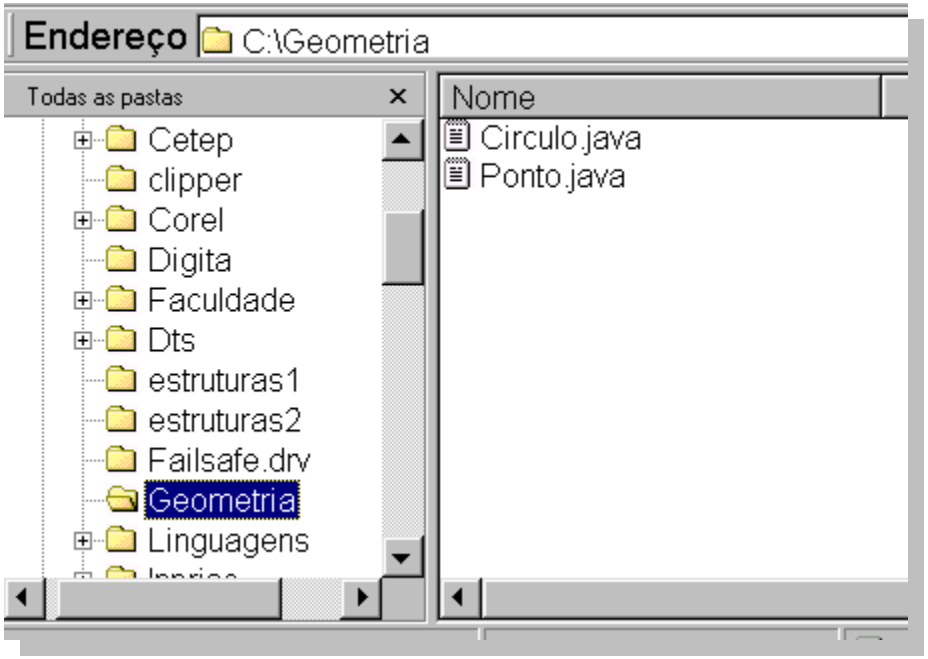
- **Algumas ferramentas do Java SDK**:

  - o ***compilador*** Java (javac)
  - o ***interpretador*** de aplicações Java (java)
  - o ***interpretador de applets*** Java (appletsviewer)
  - e ainda:

  - javadoc (**um *gerador de documentação*** para programas Java)
  - jar (o ***manipulador de arquivos comprimidos*** no formato Java Archive)
  - jdb (um ***depurador de programas*** Java), entre outras ferramentas.

Packages (1/3)

- **Os arquivos Java serão armazenados fisicamente em uma pasta.**
- **No nosso exemplo ao lado estes arquivos estão no diretório Geometria.**
- **Com o uso de *packages* podemos organizar de forma física algo lógico (um grupo de classes em comum);**



Packages (2/3)

- 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.

Packages (3/3)

- 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.

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 no caminho de busca de classes definido por CLASSPATH.

Tipos Primitivos (1/6)

- **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 Primitivos - Inteiros (2/6)

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

Tipos Primitivos – Ponto Flutuante _(3/6)

Tipos de Dados em Ponto Flutuante	Faixas
Float	$\pm$ 1.40282347 x 10^{-45} a $\pm$ 3.40282347 x 10^{+38}
Double	$\pm$ 4.9406564841246544 x 10^{-324} a $\pm$ 1.7976931348623157 x 10^{+308}

- Exemplos:
  - 1.44E6 é equivalente a 1.44 x 10^{6} = 1.440.000.
  - 3.4254e-2 representa 3.4254 x 10^{-2} =0.034254.

Tipos Primitivos - Caractere (4/6)

- O tipo **char** permite a representação de caracteres individuais.

- Ocupa 16 bits interno permitindo até 32.768 caracteres diferentes.

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

Tipos Primitivos - Caractere (5/6)

\b	backspace
\t	tabulação horizontal
\n	newline
\f	form feed
\r	carriage return
\"	aspas
\'	aspas simples
\\	contrabarra
\xxx	o caráter com código de valor octal xxx, que pode assumir valores entre 000 e 377 na representação octal
\uxxxx	o caráter Unicode com código de valor hexadecimal xxxx, onde xxxx pode assumir valores entre 0000 e ffff na representação hexadecimal.

Tipos Primitivos - Booleano (6/6)

- 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.

Palavras reservadas

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	

Além dessas existem outras que embora reservadas não são usadas pela linguagem

const	future	generic	goto	inner	operator
outer	rest	var	volatile		

Declaração de Variáveis (1/2)

- 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 (2/2)

- 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 ProgrammingLanguage.*

Comentários (1/2)

- Exemplos:

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

Comentários (2/2)

- /** 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

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

Operadores Relacionais

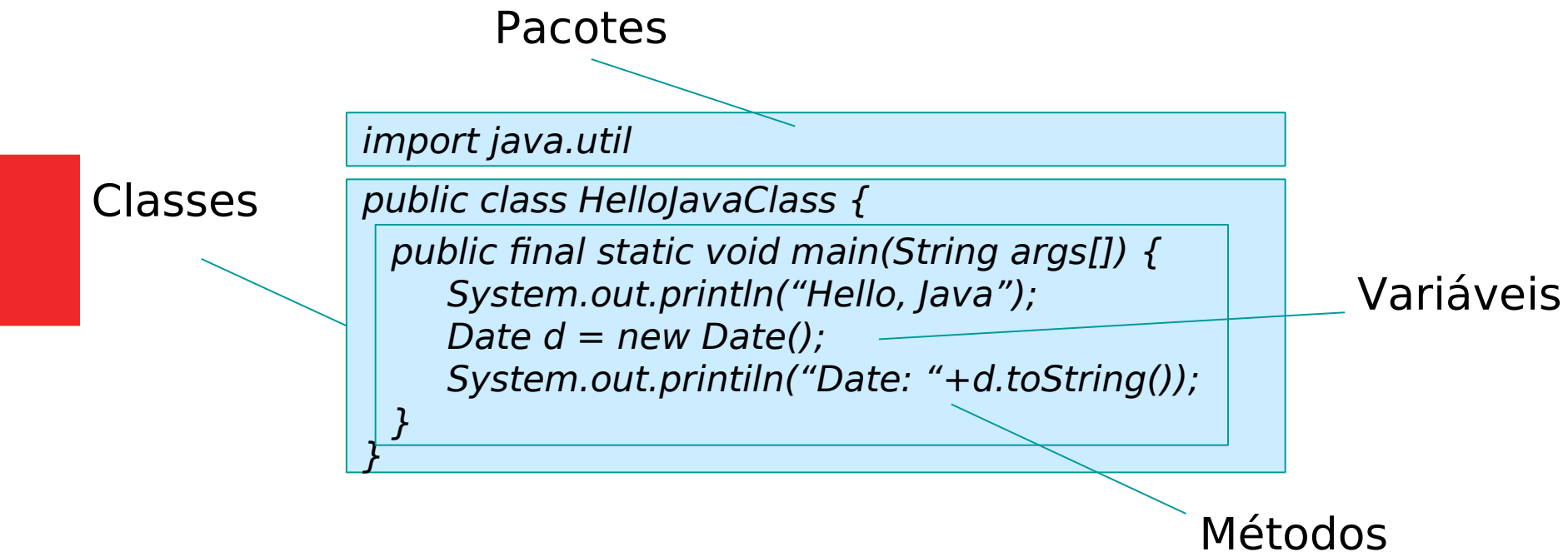
Operador	Significado	Exemplo
==	Igual	a == b
!=	Diferente	a != b
>	Maior que	a > b
>=	Maior ou igual a	a >= b
<	Menor que	a < b
<=	Menor ou igual a	a >= b

Operadores Lógicos

Operador	Significado	Exemplo
&&	E lógico (*and*)	a && b
||	Ou Lógico (*or*)	a || b
!	Negação (*not*)	!a

Programa Java

- Todos os programas em Java possuem quatro elementos básicos:



Controle do fluxo de execução (1/2)

- Normalmente *seqüencial*.

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

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

Controle do fluxo de execução (2/2)

```
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
}
```

Instrução de Desvio de Fluxo (1/2)

- São as duas, o *If* e o *Switch*

- *Exemplo do 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");
  }
}
```

Instrução de Desvio de Fluxo (2/2)

```
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");
    }
  }
}
```

Estrutura de Repetição Simples

```
import java.io.*;

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

Estrutura de Repetição Condicional

```
public class exemploWhile {

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

```
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.");
  }
}
```

Estruturas de Controle de Erro (1/5)

- Diretivas ***Try*** e ***Catch:***

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

Estruturas de Controle de Erro (2/5)

- Com o tratamento de Erros (1 Excecão)

```
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.");
    }
  }
}
```

Estruturas de Controle de Erro (3/5)

- Com o tratamento de Erros (2 Excepções)

```
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.");
    }
  }
}
```

Estruturas de Controle de Erro (4/5)

- Diretivas ***Try*** e ***Catch:***

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

Estruturas de Controle de Erro (5/5)

- A 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.
}
```

Arrays (1/2)

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

- *Exemplos de dados armazenados através de arrays:*
  - *Notas de alunos*
  - *Nucleotídeos em uma cadeia de DNA*
  - *Frequencia de um sinal de audio*

Arrays (2/2)

- *Arrays são especialmente importantes quando é necessário o acesso direto aos elementos de uma representação de uma coleção de dados*

- *Arrays são relacionados ao conceito matemático de função discreta, que mapeia valores em um conjunto finito de índices consecutivos (por exemplo, um subconjunto de inteiros não negativos) em um conjunto qualquer de objetos de mesmo tipo.*

- $F(x) \rightarrow S, x \in U$ *tal que U é um conjunto finito de valores*

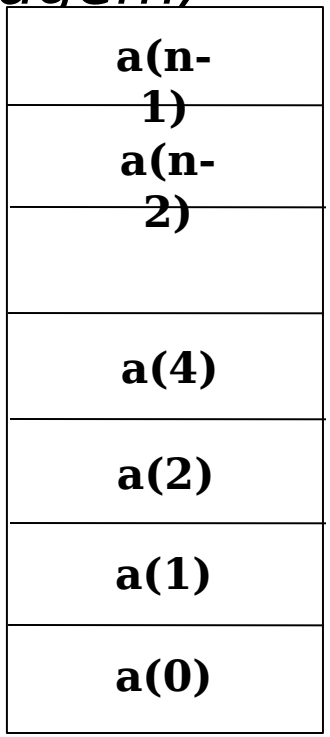
Arrays unidimensionais (1/2)

- *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 (2/2)

- *Um elemento em uma posição indicada por um índice i, em um array A, é acessado através do indentificador do array seguido do índice i (entre chaves ou parênteses, dependendo da linguagem)*



Um array com n elementos

Arrays unidimensionais em Java (1/3)

- *A criação de um array em Java requer 3 passos:*
  - *Declaração do nome do array e seu tipo*
  - *Criação do array*
  - *Inicialização de seus valores*

- *Exemplo: array de 10 elementos de tipo double*

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

Arrays unidimensionais em Java (2/3)

- *O número de elementos de um array em Java pode ser determinado através do nome do array seguido de .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 em Java (3/3)

- *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 em Java

- *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];*

Exercícios: Conversão de cores (1/2)

- Diferentes sistemas são utilizados para representar cores

- Por exemplo, o sistema mais comum para representação de cores em display LCD, câmeras digitais e páginas web conhecido como sistema RGB, especifica os níveis de vermelho(R), verde(G) e azul(B) em uma escala de 0 a 255

- O sistema utilizado na publicação de livros e revistas, conhecido como CMYK, especifica os níveis de ciano,magenta, amarelo e preto em um escala de
0.0 a 1.0

Exercícios: Conversão de cores (1/2)

- Escreva, um programa Java que receba três inteiros r,g e b representando um cor no sistema RGB e imprima os valores das componentes c,m,y,k correspondentes no sistema CMYK
- Se r=g=b=0 então c=m=y=0 e k = 1, caso contrário utilize a fórmula abaixo:

$$w = \max(r/255, g/255, b/255);$$
$$c = (w - (r/255))/w$$
$$m = (w - (g/255))/w$$
$$y = (w - (b/255))/w$$
$$k = 1 - w$$

Exercícios: Padrão de divisores

- Escreva um programa Java que receba um inteiro N e imprima um tabela NxN com um asterístico na linha i e coluna j se ou i divide j ou j divide i.

Exercícios: Padrão de divisores

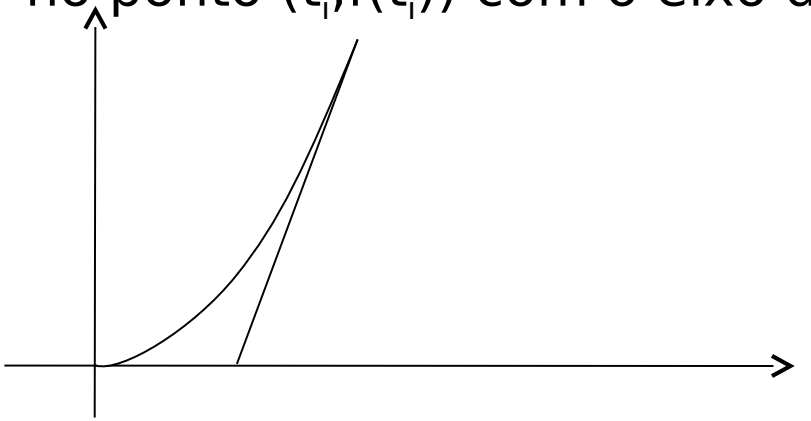
- Escreva um programa Java que receba um inteiro N e imprima um tabela NxN com um asterístico na linha i e coluna j se ou i divide j ou j divide i.

Exercícios: Fatoração de inteiros

- Escreva um programa Java que receba um inteiro N e imprima sua fatorização (sequencia de inteiros primos que multiplicados iguala a N)

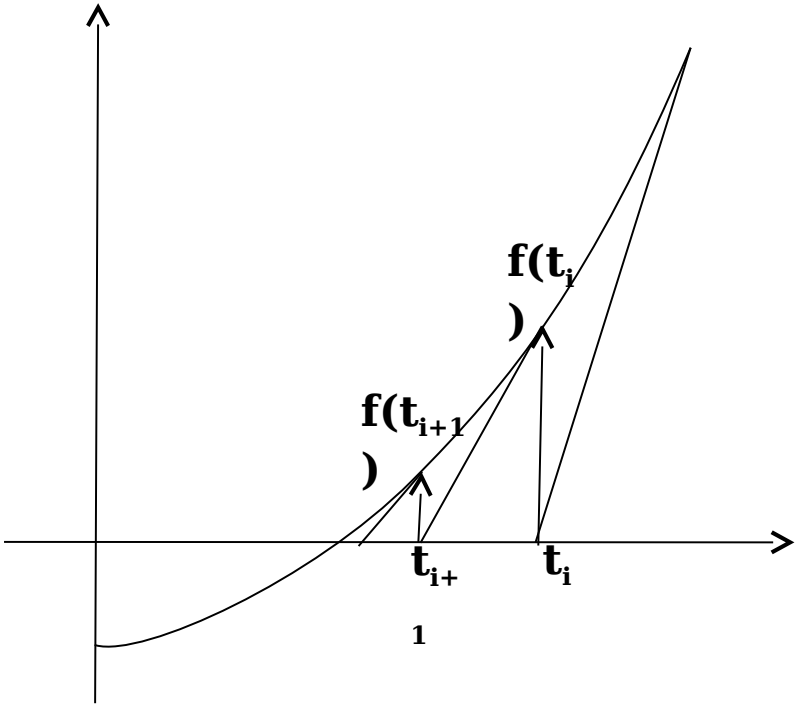
Exercícios: Cálculo de raíz quadrada

- Escreva um programa que ache a raiz quadrada de um número real c utilizando método de Newton-Raphson.
- Sob certas condições, dada uma função f(x), o método de Newton-Raphson é capaz de encontrar as raízes de uma equação f(x) {x|f(x)=0}.
- O método inicia com uma estimativa da raíz t_0
- A partir de uma estimativa ti, compute uma nova estimativa t_{i+1} onde ti é a interseção da linha tangente a curva y no ponto $(t_i, f(t_i))$ com o eixo das abcissas.



Exercícios: Cálculo de raíz quadrada

- O método inicia com uma estimativa da raíz t_0
- A partir de uma estimativa t_i, compute uma nova estimativa t_{i+1} onde t_i é a interseção da linha tangente a curva y no ponto $(t_i, f(t_i))$ com o eixo das abcissas.



$$f'(t_i) = \frac{f(t_i) - 0}{t_i - t_{i+1}}$$

$$f'(t_i)(t_i - t_{i+1}) = f(t_i)$$

$$t_{i+1} = t_i + \frac{f(t_i)}{f'(t_i)}$$

Exercícios: Cálculo de raíz quadrada

- Computar a raiz de um número equivale a achar a raíz da função $f(x)=x^2-c$
- Considere uma estimativa inicial $t_0 = c$
- Se $t_i * t_i - c = epsilon$ então tome t_i como raiz de c

Exercícios: Fatorização

- Escreva um programa que receba um número inteiro N e gere todos os fatores primos de N
- Escreva uma versão mais eficiente do seu algoritmo

Exercícios: Sequencia de símbolos

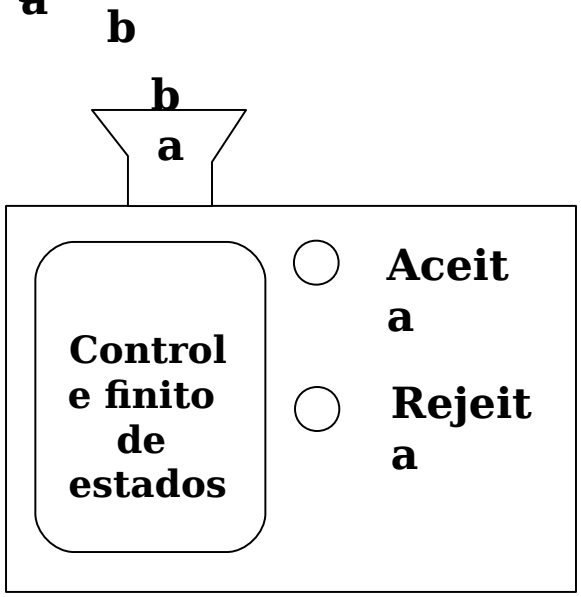
- Escreva um programa em Java que receba uma sequencia de letras da linha de comando formada por um conjunto de símbolos e gere as seguintes mensagens:
  - Para uma letra **a** “Soco”
  - Para uma letra **b** “Chute lateral”
  - Para duas letras **a’s** consecutivas “Chute circular”
  - Para uma letra **a** seguida de uma letra **b** imprime a mensagem (“dragon punch”)

Exercícios: Sequencia de símbolos

- O problema pode ser resolvido através de uma automato finito determinístico, que é um modelo para definição de linguagens regulares composto de cinco elementos: $\langle \Sigma, S, s_o, \delta, F \rangle$, onde:
  - Σ é o alfabeto sobre o qual a linguagem é definida;
  - S é um conjunto finito de estados não vazio;
  - S_o é o estado inicial, $s_o \in S$;
  - $\delta: S \times \Sigma \rightarrow S$ é a função de transição de estados;
  - F é o conjunto de estados finais $F \subseteq S$

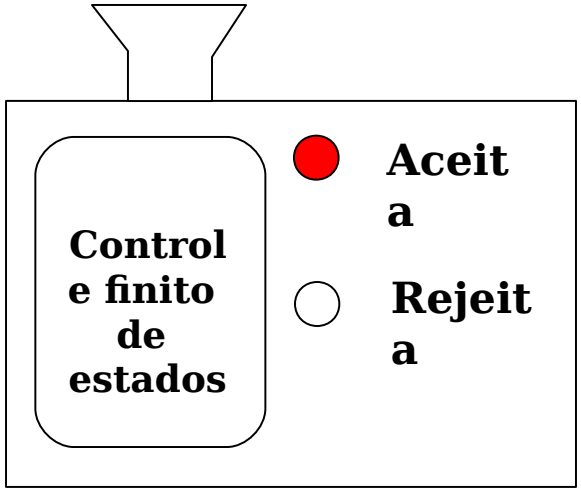
Exercícios: Sequencia de símbolos

- Um AFD é uma máquina reconhecedora de cadeias que pertencem a linguagem
- Ele recebe uma cadeia e diz se ela pertence ou não a linguagem modelada
- Ele possui um controle de estados S
- O CFE sempre coloca a máquina em um estado pertencente a S



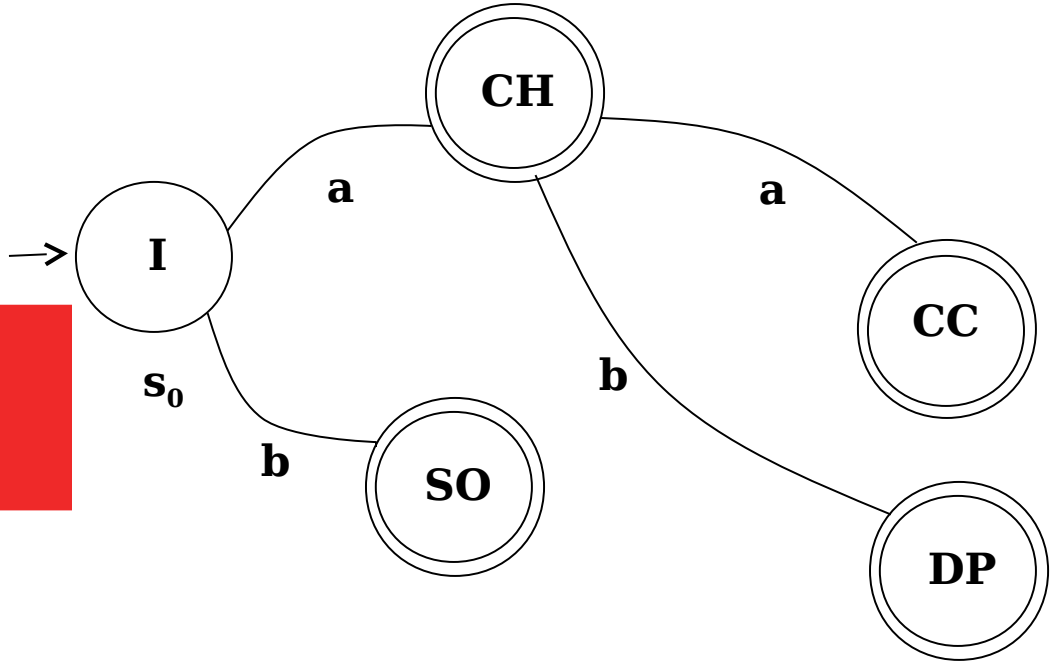
Exercícios: Sequencia de símbolos

- A função F diz como a máquina deve mudar de estado, à medida em que os símbolos da cadeia são analisados
- Após processar todos os símbolos e realizar as mudanças determinadas o AFD aceita ou não a cadeia
- Uma cadeia é rejeitada quando o autômato para em um estado que não é final
- Referência: Como construir um compilador utilizando ferramentas Java – Márcio Delamaro – novatec)



Exercícios: Sequencia de símbolos

- Autômato finito determinístico para o problema da sequência de símbolos



S	a	b	mensagem
I	C	B	
CH	CC	DP	Chute
SO			Soco
CC			Chute circular
DP			Dragon Punch

Exercícios: Caminhos aleatórios sem auto-interseção

- Suponha que você abandone seu cão no meio de uma grande cidade cujas eruas formam uma estrutura de reticulado
- Considera-se que existam N ruas na direção norte-sul e M na direção leste-oeste
- Com o objetivo de escapar da cidade, o cão faz uma escolha aleatória de qual direção ir em cada interseção, mas sabe através do faro como evitar visitar um lugar previamente visitado
- Apesar de tudo é possível que o cão fique perdido em um beco sem saída onde a próxima escolha obrigatóriamente leva a um lugar já percorrido

Exercícios: Caminhos aleatórios sem auto-interseção

- Escreva um programa Java que receba como parâmetros a largura e altura do reticulado e simule o caminho percorrido por um cão T vezes
- O programa deve determinar o número de vezes em que o cão fica sem saída

Referência

Professor elaborador:

Anselmo Montenegro
www.ic.uff.br/~anselmo