Ciências da Computação

FP

Fundamentos de Programação

03-Novembro-2014

Docente: Amândio de Jesus Almada, Lic

CAP V – Arrays (Vectores e Matrizes)

Motivação

Número de variáveis que declaramos reflecte de certa forma a quantidade de elementos que desejamos manipular. Imagine que deseja armazenar três notas de um estudante?

Neste caso o mais comum é declarar uma variável do tipo String para o Nome e três variáveis do tipo Float para as notas. Agora imagine o que seria para armazenar dados de 50 estudantes cada um com três notas? Declararias 150 variáveis?

Ou ainda como resolveria a leitura de 20 números inteiros e que se deseja obter o maior e o menor destes? Declararias 20 variáveis?

Claro que não 150 ou 20 variáveis normais. Mas sim vectores ou matrizes.

CAP V – Arrays (Vectores)

VECTORES

São considerados como variáveis compostas homogéneas unidimensional (uma dimensão), ou seja conjunto de linhas com apenas uma coluna ou o conjunto de colunas com apenas uma linha, capazes de armazenar dados de apenas um tipo.

Ex:

Posição	0	1	2	3	4
	Maria	Valdino	Sónia	Eliana	Sandra

Posição	
0	1
1	2
2	4
3	6
4	3
5	8

CAP V – Arrays (Vectores)

Declaração

Em Java, um vector é declarado de várias formas, obedecendo a seguinte sintaxe:

tipo_dado [ ] nome_vector = new **tipo_dado** [dimensão];

Ex: int [ ] num = new int [5];

Implica que a variável **num** tem a capacidade para armazenar 5 elementos; a sua primeira posição é 0 e a última é 4. Com esta opção, todas as posições de num são inicializadas com Zero (0) tendo em conta o seu tipo (int).

Pode ser igualmente declarado e inicializado da seguinte maneira:

tipo_dado [ ] nome_vector;

nome_vector = new **tipo_dado** [dimensão];

CAP V – Arrays (Vectores)

A diferença existente entre os dois modos de declaração consiste no seguinte:

No primeiro caso:

tipo_dado [ ] nome_vector = new **tipo_dado** [dimensão];

Nesta instrução ocorre a declaração e alocação de espaço (uso do **new**).

No segundo caso:

tipo_dado [ ] nome_vector;

nome_vector = new **tipo_dado** [dimensão];

Na linha mais acima ocorre apenas a declaração, não ocorrendo a alocação de espaço. Na linha mais abaixo, coloca-se o **nome do vector**, seguido do operador **new** fazendo menção da dimensão (número de elementos) que o vector possui.

CAP V – Arrays (Vectores)

Inicialização explícita:

tipo_dado [ ] nome_vector= { a0,a1,a2,...an };

Para este caso, após a declaração, a inicialização é feita de forma explícita: ou seja a quantidade de elementos entre as chavetas determina a dimensão do vector. E cada elemento entre a chaveta ocupa uma posição isto é:

Ex1: int vect [ ] = {2,6,8,3,5};

O vector acima possui a dimensão 5, e na sua primeira posição vect[0] encontramos o valor 2, vect[1]=6, vect [2]=8,... vect[4]=5.

```
Ex2: double[] m = {};
```

Para o exemplo 2, não existe alocação de espaço; isto implica que não podemos aceder a nenhuma posição do vector.

CAP V – Arrays (Vectores)

Manipulação de vectores:

Atribuição

Ex: String nome[]=new String [2];

nome[0]="Eliana";

nome[1]="Francisco";

Leitura de dados:

```
int num = new int [dim];
for (int i=0;i<dim;i++){
num[i]= teclado.nextInt();
}
                    }
```

Escrita

```
System.ou.print(a[0]);
ou
for (int i=0;i<dim;i++){
System.ou.print(num[i]);
}
```

CAP V – Arrays (Vectores)

Exercícios

1.Faça um programa que recebe uma quantidade de números determinados pelo utilizador e imprime-os de forma inversa.

2. Faça um programa em Java que lê 20 números inteiros e mostra o maior e o menor.

3. Faça um programa que lê 30 números e mostra o vector ordenado em forma decrescente.

CAP V – Arrays (Matrizes)

As matrizes, são nada mais do que o conjunto de elementos dispostos em linhas e colunas; o acesso a cada elemento de uma matriz, implica especificar a referência (nome) da mesma seguido da sua posição (linha, coluna).

Sintaxe e declaração

Tipo_dado [][] referência = new tipo_dado[dimmlinha] [dimcoluna];

Ex: String [][] alunos=new String[2][2];

Ou ainda:

Tipo_dado [][] referência; // Apenas declaração, sem alocação

referência= new tipo_dado[dimmlinha] [dimcoluna]; // Alocação de espaço

CAP V – Arrays (Matrizes)

Declaração explícita

Tipo_dado [ ][ ] referência ={{a00,a01,a02,amn},{a10,a11,a12,amn}};

Ex: int [ ][ ] ex ={{2,6,7,8},{1,4,6,3}};

Nota: é possível declarar explicitamente numa matriz quantos elementos quiser. Saiba simplesmente que o número de sub-pares de chavetas existentes determina a dimensão da matriz.

No exemplo acima, a matriz **ex** possui a dimensão (2x4).

CAP V – Arrays (Matrizes)

Para a leitura e a escrita é aconselhável a utilização de estruturas de repetição, num total de 2 no mínimo para manipular os elementos da linha e da coluna.

Ex:

Leitura de dados

```
int [ ][ ] elem =new int[3][2];
for(int i=0;i<elem.length;i++){

for(int j=0;j<elem[i].length;j++){
    elem[i][j]=teclado.nextInt();
}
}
```

Escrita de dados

```
int [][] elem =new int[3][2];
for(int i=0;i<elem.length;i++){

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

Nota: A função length pode ser utilizada para obter o número de linhas e o número total de colunas por cada linha de uma matriz. Verifique o exemplo acima.

CAP V – Arrays (Matrizes)

Exercícios

1. Implemente um programa que recebe o nome de N estudantes com K disciplinas e mostre o estudante com a maior e menor média.

2. Implemente um programa para ler uma matriz NxM de inteiros e imprime a diferença do somatório dos elementos da diagonal principal pelos elementos da diagonal secundária.

3. Implemente um programa para ler um inteiro K e uma matriz NxM inteiros e imprime a posição de K na matriz e a quantidade de múltiplos deste K.

CAP VI – Strings

Motivação

Tal como já estudado na aula sobre tipos de dados primitivos, os caracteres são suportados pelo tipo de dado Char (apenas um caractere em cada variável). Neste caso, se desejássemos armazenar mais do que um caractere (seja um nome, frase, conjunto de símbolos) criaríamos um vector de caracteres para solucionar o devido problema.

Para o caso do Java existe o recurso do tipo String para representar o conjunto de caracteres. Não como um tipo de dado primitivo mas como uma referência (objecto).

Definição: String é um tipo de dado que permite armazenar um conjunto de caracteres.

CAP VI – Strings

Declaração

Por ser uma classe, para podermos utiliza-la, carecemos da criação de um objecto do tipo String. Sendo assim vejamos:

Declaração -1: String nome_da_string;

Método construtor

Declaração -2: String nome_da_string = new String(); onde: nome_da_string indica a referência por onde o objecto é encontrado.

Inicialização

String nome_string= new String (); - Inicializa com espaço em branco

String nome_string= new String ("Ola colegas"); - Inicializa com a String Ola colegas.

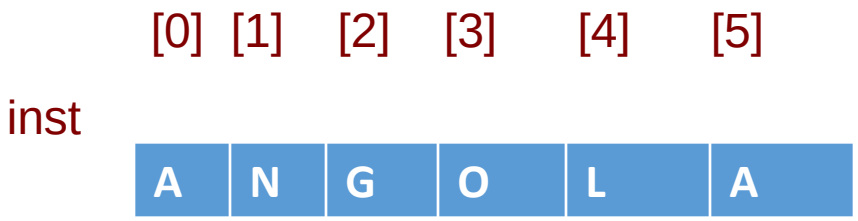
String nome_string= null; - Inicializa com o valor default para os objectos.

CAP VI – Strings

Inicialização

A representação de uma String obedece a regra dos vectores de char. Ou seja cada caractere está localizado em determinada posição contando de 0 à n-1.

Ex: String inst= “ANGOLA”



Utilizando o tipo char faz-se: char [] a ={‘A’,’N’,’G’,’O’,’L’,’A’}

String inst= new String (a);

System.out.print(inst); // mostra ANGOLA

CAP VI – Strings

Uma das principais características que a String possui é a imutabilidade. Isto é, o conteúdo de uma String não altera por qualquer que seja a operação em que esteja envolvida. Exemplo:

a) int x; int y;

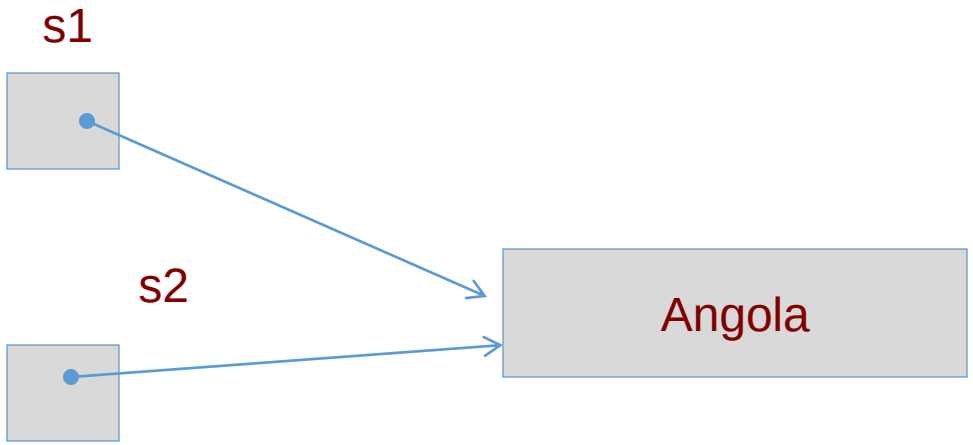
x = 5;

y = x;

b) String s1; String s2;

s1 = new String (”Angola");

s2 = s1;



CAP VI – Strings

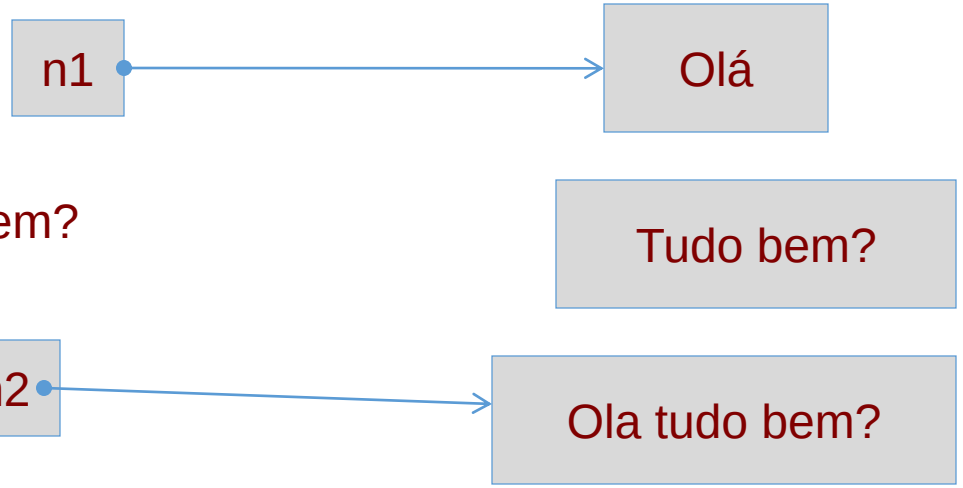
No exemplo b), a atribuição efectuada não corresponde ao conteúdo da String s1, mas sim a s1 e s2 possuirão a mesma referência para o objecto.

Exemplo 2:

String n1="Ola";

String n2= "tudo bem?"

n2=n1+n2



System.out.print(n2) // Olá tudo bem?

CAP VI – Manipulação de Strings

Como toda a classe, a String possui um conjunto de métodos que facilitam a sua manipulação. Dentre os quais refere-se:

length() – método que retorna o comprimento de uma String.

Ex: String disciplina =“programação II”;

System.out.print(disciplina.length()); // mostrará 14

charAt()- método que retorna um caractere em determinada posição.

Ex: considere a String disciplina: System.out.print(disciplina.charAt(6)); // m

concat()- método que retorna a concatenação (união) de duas Strings.

Ex: System.out.print(disciplina.concat(“ é divertida”));

// Mostra: programação II é divertida

CAP VI – Manipulação de Strings

indexOf() – método que retorna a posição de um dado caractere dentro da String.

Ex: String disciplina =“programação II”;

System.out.print(disciplina.indexOf(‘ç’)); // mostrará 8

System.out.print(disciplina.indexOf(‘a’,6)); // mostrará 7

compareTo() e compareToIgnoreCase() – são dois métodos que permitem comparar duas Strings. compareTo (considera maiúsculas e minúsculas), compareToIgnoreCase(ignora o facto de maiúsculas ou minúsculas)

Ex:

if(s1.compareTo(s2) < 0)

- s1< s2 – retorna um valor negativo
- s1= s2 – retorna zero pois são iguais
- s1>s2 – retorna um valor positivo

CAP VI – Manipulação de Strings

equals() e equalsIgnoreCase() – compara duas Strings e retorna true se as duas Strings forem iguais ou false caso as Strings sejam diferentes.

Ex: String disciplina =“programação II”;

System.out.println(disciplina.equals(‘Programação II’)); // False

System.out.print(disciplina. equalsIgnoreCase(ProGramação II); // True

toUpperCase()- retorna uma nova String com caracteres em maiúsculo.

Ex: System.out.println(disciplina.toUpperCase()); // PROGRAMAÇÃO II

toLowerCase()- retorna uma nova String com caracteres em minúsculo.

Ex: System.out.print(disciplina.toLowerCase()); // programação ii

Substring(): retorna parte de uma String dependendo do intervalo.

Ex: System.out.println(disciplina.substring(3,8)) // retorna grama

System.out.println(disciplina.substring(4)) // retorna ramação II

CAP VI – Strings

Exercícios

1. Implemente um programa em java para ler uma String e diga se a mesma é capicua. Uma String é capicua se a String original for igual a sua inversa.
Ex: ana – inverso: ana; ovo – inverso ovo.

2. Implemente um programa que lê duas Strings e imprime a quantidade de vezes que cada caractere da primeira ocorre na segunda.

3. Implemente um programa que lê uma String, e imprime a String resultante das seguintes operações e a quantidade de substituições caso ocorra:
'a' substitui por 't'
'e' substitui por 'h'
'i' substitui por 'a'

CAP VII – Modularização

Modularização consiste em repartir um problema em pequenos subprogramas, onde a solução de todos os subprogramas depois de associados, resolvem o problema geral.

Em programação, a modularização é aplicada utilizando procedimentos ou funções. Cada módulo (procedimento ou função) é também denominado subrotina ou método e pode ser desenvolvido de forma independente.

CAP VII – Modularização

Desvantagens ao não modularizar

- São construídos programas bastante extensos que podem dificultar a compreensão do código.
- A alteração das variáveis são feitas directamente
- Um único programador é sacrificado para a implementação da solução
- Não há reutilização do código
- Dificulta a localização e correcção de erros (depurar)
- Programas pouco eficientes

CAP VII – Modularização

Vantagens ao aplicar a modularização

- Facilidade na solução de problemas complexos.
- Reutilização de código (codificar uma única vez e utilizar quantas vezes desejar)
- Facilidade na compreensão do código
- Abertura da possibilidade do trabalho em equipa (solução de pequenos problemas separadamente por cada membro da equipa)
- Facilidade na depuração do código
- Geralmente são utilizados parâmetros para permitir trabalhar com a cópia das variáveis
- Para o utilizador do módulo, interessa apenas o COMO utilizar e não o COMO foi implementado (caso das API)

CAP VII – Modularização

Funções (Definição e Declaração)

É sequência de instruções bem definidas que efectuam determinado cálculo ou operação e no final retorna explicitamente um valor ao exterior. Este valor deve ser **obrigatoriamente** do tipo declarado na função. Ex: Cálculo do factorial, média de um estudante, soma de números, etc.

Sintaxe:

```
public static tipo nome(parâmetros) {
   instruções;
   return variável;
}
```

Declaração / cabeçalho

CAP VII – Modularização

Exemplo de uma função (método com retorno)

```
import java.util.Scanner;
public class P_programa {
public static int factorial(int num){
int fact=1;
if(num==0)
return 1;
for(int i=1;i<=num;i++){
fact*=I; }
return fact; }
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
System.out.println("Digite um inteiro não negativo");
int n=teclado.nextInt();
if(n>=0){
System.out.print(factorial(n));
}else
System.out.print("Não existe factorial de números negativos"); } }
```

Parâmetro

Este bloco representa a criação de uma função. Os métodos (função ou procedimento)devem ser declarados após a classe, e podem ou não ter parâmetros.

Chamada da função

Argumento

Programa principal

CAP VII – Modularização

Procedimentos (Definição e Declaração)

Define-se como uma sequência de instruções bem definidas que executam instruções sem o retorno de um valor para o exterior. Ex: menu de opções, leitura de dados, impressão de um resultado, etc.

Sintaxe:

```
public static void nome(parâmetros) {

    Instruções;

}
```

Declaração / cabeçalho

Corpo

CAP VII – Modularização

Exemplo de um procedimento (método sem retorno)

```
import java.util.Scanner;
public class P_programa {
public static void impvector(int num[]){
for(int i=0;i<num.length;i++){
System.out.println(num[i]);
}
}
public static void main(String []args){
Scanner teclado=new Scanner(System.in);
int a[]=new int[3];
for(int j=0;j<a.length;j++){
System.out.println("Digite o "+(j+1)+" elemento");
a[j]=teclado.nextInt();
}
impvector( a); }}
```

Parâmetro

Este bloco representa a declaração e implementação de um procedimento.

Programa principal

Chamada do procedimento

Argumento

CAP VII – Modularização

Parâmetro e argumento

Parâmetros e argumentos, definem o meio na qual os dados podem ser introduzidos dentro de um método. Importa não confundir com as variáveis locais ou globais (abordado mais adiante).

Parâmetro: é definido no instante em que declaramos ou construímos o protótipo do método. Este possui um tipo e um identificador.

Ex:

Parâmetros

```
public static int contNumero(int x, int y){
        ... }
```

Estes podem ser simples variáveis, vectores, matrizes, objectos, etc.

CAP VII – Modularização

Argumento: define o valor concreto do parâmetro declarado no método. Diferente do parâmetro que é definido na criação do método, este (argumento) existe no momento em que invocamos o método no programa principal ou no interior de outro método e deve ser do mesmo tipo que o parâmetro.

Um argumento pode ser uma constante, variável ou endereço.

Ex:

```
public static void main(String []args){
        Scanner teclado=new Scanner(System.in);
        int a[]=new int[3];
        for(int j=0;j<a.length;j++){
           System.out.println("Digite o "+(j+1)+" elemento");
           a[j]=teclado.nextInt();
        }
        impvector( a);
    }}
```

Chamada do método

Argumento

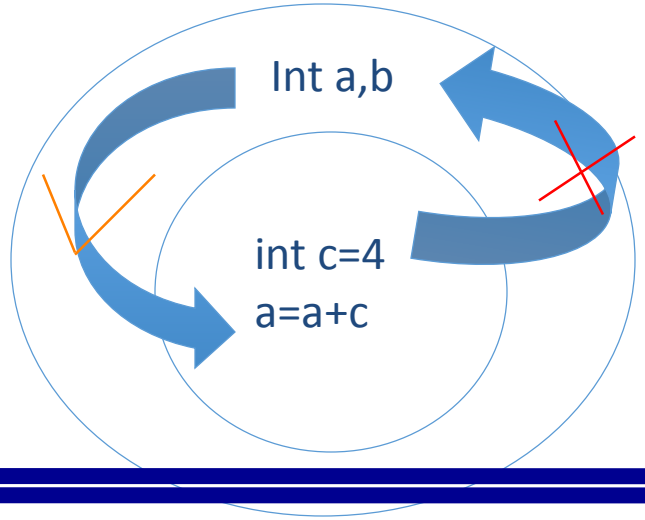
CAP VII – Modularização

Escopo de uma Variável

Representa a visibilidade ou o raio de acção que uma variável ocupa dentro do programa ou método. Uma variável pode ser Local ou Global.

Variável Local: possui visibilidade limitada, são declaradas normalmente no interior de uma estrutura de selecção, repetição, método, ou ainda como parâmetro de um método. Isto implica que após o bloco em que ela foi declarada, esta variável não será reconhecida.

Ex:



A variável C não é visível na instrução acima

CAP VII – Modularização

Escopo de uma Variável

Variável Global: são declaradas após a criação da classe, e podem ser acedidas em toda a parte do programa.

Elas devem estar associadas ao modificador static (será abordado mais adiante) para serem utilizadas directamente no método principal.

Ex: Suponha o programa abaixo

```
public class Exercicio2 {
(1*) static int k;
public static void main(String[]args){
k=0;
(2*) for(int i=1;i<=10;i++){
if(i%2==0){
k=k+1;}}
System.out.print(k);}}
```

(1*) Declaração de uma variável global.

(2*) A variável que controla o ciclo for, tem visibilidade apenas dentro do ciclo. Isto implica dizer que ela não é permitida fora do ciclo sem que seja novamente declarada.

CAP VII – Modularização Exercícios

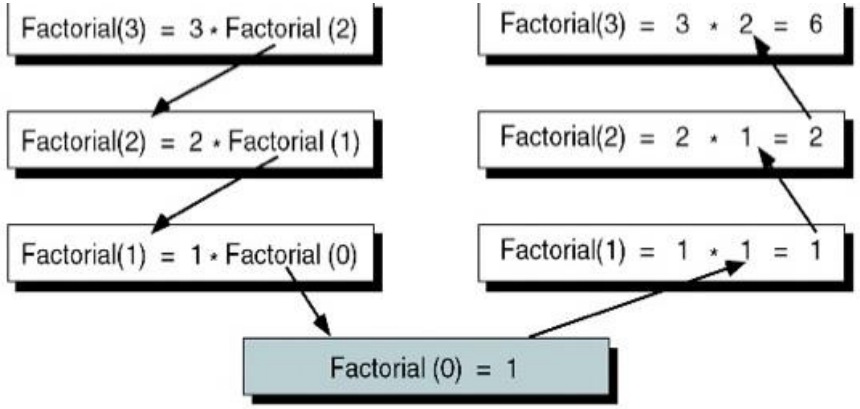
1. Implemente uma função que recebe uma String e retorna o número de vogais existentes nesta String.
2. Implemente uma função que retorna a quantidade de números pares na diagonal principal de uma matriz quadrada MxN.
3. Implemente uma função que retorna a potência de M^N.
4. Implemente uma função que retorna o MDC de dois inteiros.

CAP VII – Modularização – Funções recursivas

Uma função é considerada recursiva quando esta é capaz de invocar a si mesma afim de solucionar um determinado problema. A solução parte da decomposição de um problema por vários do mesmo tipo, e no final constrói-se a solução geral por intermédio das pequenas soluções.

Isto é, a decomposição do problema é feita do topo para a base e a solução obtém-se da base para o topo.

Um dos grandes exemplos do uso da recursão é o cálculo do factorial. Vejamos:



CAP VII – Modularização – Funções recursivas

Regras para a implementação de uma função recursiva

1. Primeiramente determine o caso Base (caso por meio do qual a solução é construída)
2. Determina o caso genérico (o modo como o problema deve ser resolvido)
3. Combine o 1º e o 2º ponto num só algoritmo

Cada chamada recursiva reduz o tamanho do problema e tender ao caso base.
O caso base deve terminar o algoritmo, retornando no final a solução do problema.

CAP VII – Modularização – Funções recursivas

Exemplo: Crie uma função em Java que retorna a soma dos números de 1 à n.

Função sem recursividade

```
public static int recfuncao(int num){
int soma=0;
if(num==1)
return 1;
for(int i=1;i<=num;i++){
soma=soma+i;
}
return soma;
}
```

Função com recursividade

```
public static int recfuncao (int num){
if(num==1)
return 1;
return num+recfuncao(num-1);
}
```

CAP VII – Modularização – Funções recursivas

Exemplo2: Crie uma função em Java que imprime todos os números no intervalo de A à B inclusive.

Função sem recursividade

```
public static void imprime(int a,int b){
for(int i=a;i<=b;i++){
System.out.println(i);
}
}
```

Função com recursividade

```
public static void imprime(int a,int b){
if(a<=b){
System.out.println(a);
a++;
imprime(a,b);
}
}
```