Programação Orientada à Objetos (POO)

CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. João Paulo Aramuni

UNIVERSIDADE FUMEC

Sumário

* **Vetores e Matrizes**
  * Vetores
  * Matrizes
  * Cópia de Matrizes

Vetores e Matrizes

* No mercado de trabalho, a classe mais utilizada para se trabalhar com vetores é a:
  * java.util.ArrayList;
  * Veremos também a classe **Arrays**.
  * A classe 'Vector' está deixando de ser utilizada pelo mercado devido a questões de performance de busca e ordenação.

Vetores e Matrizes

* Para matrizes, iremos utilizar vetores bidimensionais.

  * Segundo DEITEL (2005, pág. 223), os arrays com duas dimensões costumam ser utilizados para representar tabelas de valores que consistem nas informações dispostas em **linhas** e **colunas**.

Vetores e Matrizes

* Exemplo de array unidimensional de 10 elementos:
  * int x[] = new int[10];
* Exemplo de array bidimensional (matriz) 2x4 (8 elementos):
  * int y[][] = new int[2][4];
* Exemplo de array da classe ArrayList:
  * ArrayList<String> teste = new ArrayList();
* Exemplo de array da classe Vector:
  * Vector<String> teste2 = new Vector<String>();

Vetores e Matrizes

* Manipulando vetores utilizando a classe Vector:
  * Os vetores estão deixando de ser utilizados no mercado devido ao fato de que, se passar do limite, duplica a capacidade automaticamente.
  * Ou seja, para grandes quantidades de dados, o vector crescerá de forma exponencial.
  * Para evitar isso, utilize a classe **ArrayList** nas aplicações que trabalham com grande volume de dados.

Vetores e Matrizes

* Manipulando vetores utilizando a classe Vector:
  * Métodos importantes:

add(objeto)

Inclui objeto no final (pode ter que aumentar a capacidade)

add(posição, objeto)

Insere objeto na posição e desloca os objetos subseqüentes

set(posição, objeto)

Substitui o objeto que esta na posição

get(posição)

Retorna o objeto armazenado na posição

Vetores e Matrizes

* Manipulando vetores utilizando a classe Vector:
  * Métodos importantes:

size()

Retorna o tamanho atual do vetor

remove(objeto)

Procura e remove o primeiro objeto encontrado

remove (posição)

removeAllElements() , **clear**()

firstElement(), **lastElement**()

isEmpty() - Retorna true se alista estiver vazia

contains(objeto)

Retorna true se encontrar o objeto

indexOf(objeto)

Procura o objeto e retorna a posição do objeto ou -1 se não achar

indexOf(objeto, posição)

Pesquisa na lista o objeto a partir a posição

Vetores e Matrizes

* Manipulando vetores utilizando a classe Arrays:
  * A classe **Arrays**, disponível no pacote **java.util**, fornece uma grande quantidade de **métodos utilitários**, como por exemplo métodos para **ordenação**, **procura**, **comparação** e etc..
  * Estes métodos são muito úteis quando **manipulamos arrays**. A seguir serão apresentados os principais métodos e a as respectivas funcionalidades oferecidas.

Vetores e Matrizes

* Manipulando vetores utilizando a classe Arrays:
  * Ordenação: Realizada utilizando-se o método “sort” cujo parâmetro é o vetor a ser ordenado;
  * Pesquisa: A localização de um determinado elemento em um array é realizada utilizando-se o método “binarySearch” que retorna a posição que o elemento foi encontrado no array. Caso o elemento não seja encontrado retorna um valor negativo;
  * Preenchimento: Utilizando-se o método “fill” da classe utilitária Arrays é possível preencher um determinado array com o elemento desejado;
  * Comparação: Dados dois arrays o método “equals” compara valor a valor e retorna true se os vetores são idênticos em valores e índices.

Vetores e Matrizes

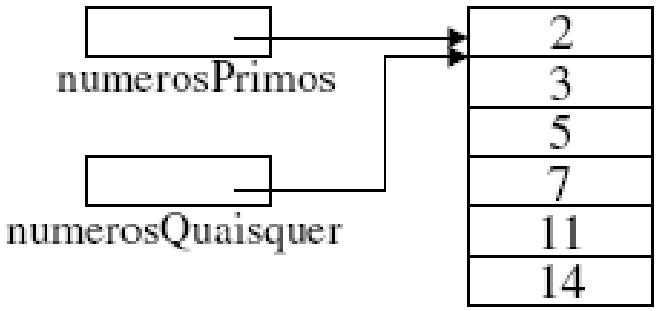
* Manipulando vetores utilizando a classe Arrays:
  * Vejamos um exemplo:

```
import java.util.*;
Import utilitarios.*;
public class Loteria { //  EXEMPLO  LOTERIA
  public static void main(String[ ] args)    {
    int k = Console.readInt("Quantos números você quer jogar? ");
    int n = Console.readInt("Qual é o maior número que você pode jogar? “);
    // Preencher o vetor “números” com números 1 2 3 . . . n
    int [ ] numeros = new int[ n ];  //  Vetor para armazenar todos os números possíveis
    for (int i = 0; i < numeros.length; i++)
    numeros [ i ] = i + 1;
    // Jogar k números e armazená-los no segundo vetor “jogo”
    int [ ] jogo = new int [ k ];  // Vetor para armazenar o jogo
    for (int i = 0; i < jogo.length; i++)   {
      // Criar um index aleatório entre 0 a n - 1
      int r = ( int ) (Math.random() * n); // O método random retorna um número real aleatório entre
      //  0 (inclusive) e 1 (exclusive)
      // Buscar um número localizado na posição aleatória selecionada
      jogo [ i ] = numeros [ r ];
      numeros [ r ] = numeros [n - 1];  // mover o último elemento para a posição aleatória selecionada
      n--; // ajustar n
    }
    // Imprimir o vetor ordenado através do método sort da classe java.util.Arrays e uso do for each
    Arrays.sort (jogo);
    System.out.println("Jogo selecionado. Boa sorte!");
    for (int  elementos : jogo)   //
        System.out.print(elementos + “ “);
  }
}
```

Vetores e Matrizes

* Cópia de arranjos:
  * Uma variável do tipo arranjo é um apontador para uma estrutura contendo os elementos do arranjo.
  * Copiar uma variável do tipo arranjo para outra significa apontar as duas variáveis para a mesma estrutura.
    * Exemplo:

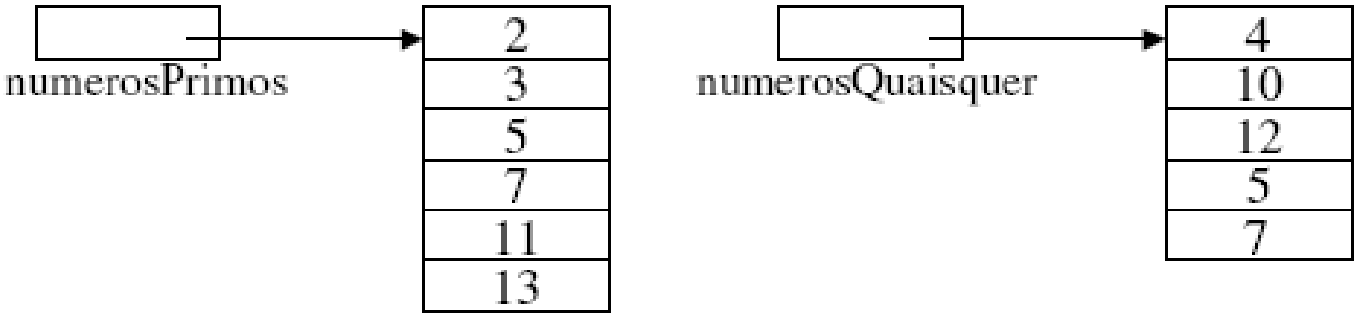
```
int[ ] numerosPrimos = {2, 3, 5, 7, 11, 13};
int[ ] numerosQuaisquer = numerosPrimos;
numerosQuaisquer[5] = 14;
```



Vetores e Matrizes

* Cópia de arranjos:
  * Para realmente copiar valores de um arranjo para outro, pode-se usar o método arraycopy da classe System.
    * static void arraycopy(Object fonte, int posicaofonte, Object destino, int posicaodestino, int tamanho)
    * Exemplo:

```
int[ ] numerosPrimos = {2, 3, 5, 7, 11, 13};
int[ ] numerosQuaisquer = {4, 10, 12, 14, 20};
System.arraycopy(numerosPrimos,2,numerosQuaisquer,3,2);
```



Vetores e Matrizes

* Arranjos Multidimensionais:
  * Declaração:
    * double[ ][ ] matriz= new double[3][2]; // arranjo bi-dimensional
    * double[][] matriz2 = new double[5][4]; // arranjo bi-dimensional // 5x4
    * int[][] vendas = {{1999, 100000}, {2000, 120000}};//inicialização
    * String[][][] triStr = new String[5][4][3]; // arranjo // tri-dimensional

  * Preenchimento:
    * matriz[1][2] = 12;
    * triStr[3][3][2] = “Teste”;

  * Arranjos multidimensionais são arranjos de arranjos.
    * Um arranjo contém elementos e cada um dos elementos pode ser um outro arranjo.
    * Pode-se criar arranjos “irregulares”, ou seja, arranjos em que linhas diferentes têm comprimentos diferentes.

Vetores e Matrizes

* Arranjos Multidimensionais:

	0	1	2
	nome	nascimento	telefone
0	Katia	30/04/1980	2222-3322
1	Antonio	05/09/1985	4444-3333
2	Jaime	15/04/1987	3333-4444

```
nomes [1] [0] = “Antonio”;
```

Vetores e Matrizes

* Arranjos Multidimensionais:
  * Exemplo 1 - Array de Objetos:

```
public static void main(String[] args) {
        int numNomes = Console.readInt("Quantas Pessoas? ");
        Object [ ][ ] nomes = new Object [numNomes] [3]; // Arranjo de Objetos de 2 dimensões
        // Entrada de Dados
        for ( int i=0; i<numNomes; i++ ) {
                nomes[ i ] [0]= Console.readLine(i+1 + "-Digite o nome: “ );
                String [ ] datStr = Console.readLine(i+1 + "-Digite o nascimento: “ ).split("/");
                nomes[ i ] [1] = new GregorianCalendar(Integer.parseInt(datStr[2]),
                Integer.parseInt(datStr[1]) - 1, Integer.parseInt(datStr[0]));
                nomes[ i ] [2] = Console.readLine(i+1 + "-Digite o telefone: “ );
        }
        // Saida de Dados
        for ( int i=0; i<numNomes; i++ ) {
           System.out.println("Nome : " + (String) nomes[i][0] + "; " + "Nascimento : " + new
           SimpleDateFormat("dd/MM/yyyy").format(((GregorianCalendar)nomes[ i ][1]).getTime()) + "; "
           + "Telefone : " + (String) nomes[ i ][2]);
        }
}
```

Vetores e Matrizes

* Arranjos Multidimensionais:
  * Exemplo 2 - Arranjos:

```
public static void main(String[ ] args) {
        int [ ][ ] vendas = {{1999, 100000}, {2000, 120000},
        {2001, 150000}, {2002, 180000},
        {2003, 210000}, {2004, 250000},
        {2005, 320000}};  // Inicialização do Arranjo
        double total = 0; // Variável para totalizar as vendas
        for (int i = 0;  i < vendas.length;  i++) {
        // Linhas diferentes podem ter comprimentos diferentes, neste caso  é sempre 2
        for (int j=0;  j < vendas[ i ].length; j++) {
                System.out.print((j==0 ? "\nAno : " + vendas[ i ][ j ] :
                "\nTotal de Vendas : " + vendas[ i ][ j ]));
                total += (j==1 ? vendas[ i ][ j ] : 0 ); // Acumulador
        }
        }
        System.out.println("\n\nTotal Geral.....: " + total);
}
```

Vetores e Matrizes

* Arranjos Multidimensionais:
  * Exemplo 3 - Arranjos:

```
public static void main(String[] args) {
        // Arranjo de acumuladores – totais – 2 dimensões
        double [ ] [ ] totais = new double [2] [12];
        for (int i=0; i<totais.length; i++) {
                for (int j=0; j<totais[ i ].length; j++) {
                    totais [ i ] [ j ] = 0;
                    System.out.println("Total " + i + "," + j + " : " + totais [ i ] [ j ]);
                }
        }
}
```

* Arranjos Multidimensionais:
  * Exemplo 4 - Aplicação Java explorando os métodos da classe ArrayList:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class Exemplo {

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

  // [ A ] declarando e instanciando uma agenda de contatos
  ArrayList<String> agenda = new ArrayList();

  // [ B ] usando o método add() para gravar 4 contatos na agenda
  agenda.add("Juca Bala;11 1111-1111");
  agenda.add("Marcos Paqueta;22 2222-2222");
  agenda.add("Maria Antonieta;33 3333-3333");
  agenda.add("Antônio Conselheiro;44 4444-4444");

  int i;

  // [ C ] mostrando os "n" contatos da agenda (usando o índice)
  // número de elementos da agenda: método size()
  System.out.printf("Percorrendo o ArrayList (usando o índice)\n");
  int n = agenda.size();
  for (i=0; i<n; i++) {
   System.out.printf("Posição %d- %s\n", i, agenda.get(i));
  }
```

Vetores e Matrizes

* Arranjos Multidimensionais:
  * Exemplo 4 - Continuação:

```
System.out.printf("\nInforme a posição a ser excluída:\n");
   i = ler.nextInt();
   try {
    // [ D ] remove o i-ésimo contato da agenda
    agenda.remove(i);
   } catch (IndexOutOfBoundsException e) {
      // exceção lançada para indicar que um índice (i)
      // está fora do intervalo válido (de 0 até agenda.size()-1)
      System.out.printf("\nErro: posição inválida (%s).",
       e.getMessage());
   }
   // [ E ] mostrando os "n" contatos da agenda (usando for-each)
   System.out.printf("\nPercorrendo o ArrayList (usando for-each)\n");
   i = 0;
   for (String contato: agenda) {
    System.out.printf("Posição %d- %s\n", i, contato);
    i++;}
   // [ F ] mostrando os "n" contatos da agenda (com iterator)
   System.out.printf("\nPercorrendo o ArrayList (usando iterator)\n");
   i = 0;
   Iterator<String> iterator = agenda.iterator();
   while (iterator.hasNext()) {
    System.out.printf("Posição %d- %s\n", i, iterator.next());
    i++;
   }
  }
}
```

Vetores e Matrizes

* Arranjos Multidimensionais:
  * Exemplo 4 - Resultado:

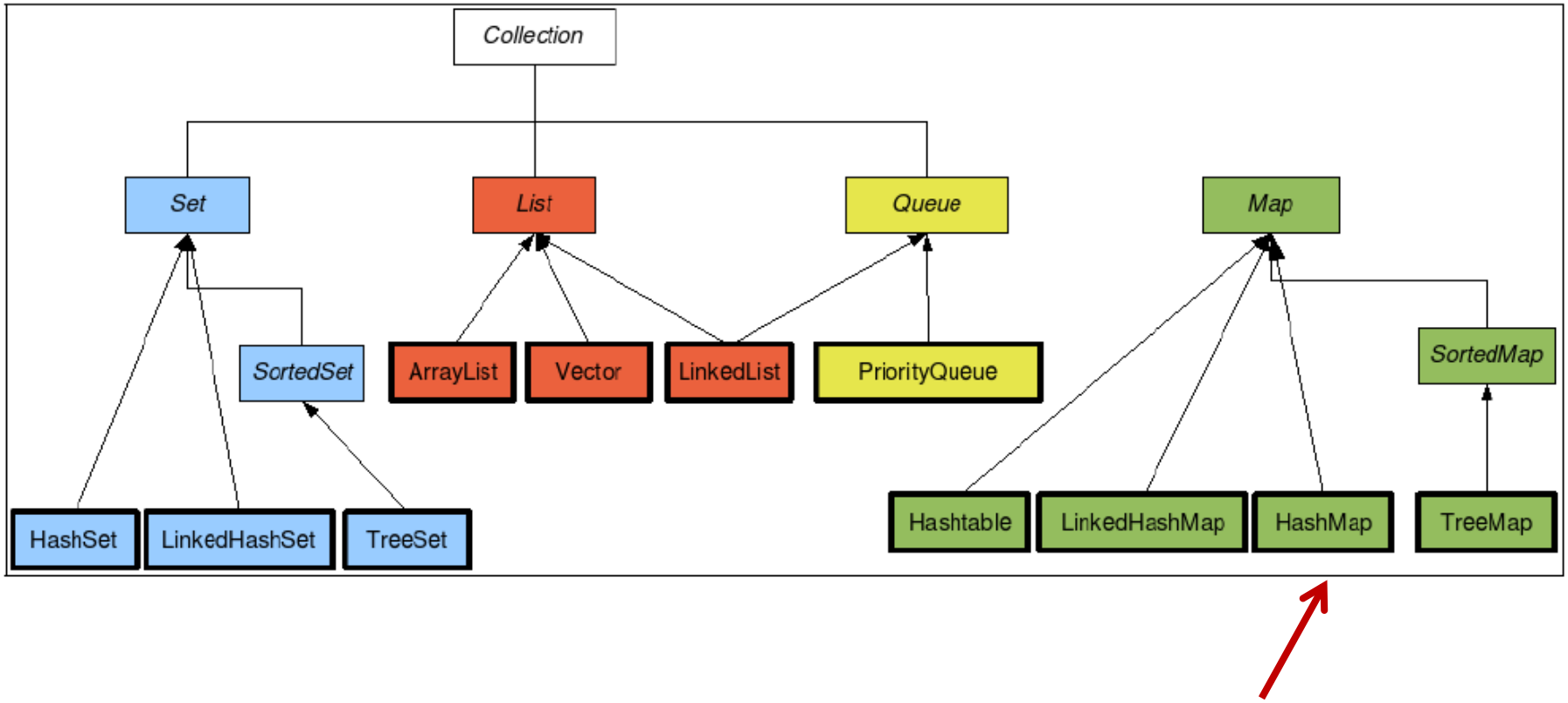
```
Percorrendo o ArrayList (usando o índice)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Marcos Paqueta;22 2222-2222
Posição 2- Maria Antonieta;33 3333-3333
Posição 3- Antônio Conselheiro;44 4444-4444

Informe a posição a ser excluída:
1

Percorrendo o ArrayList (usando for-each)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Maria Antonieta;33 3333-3333
Posição 2- Antônio Conselheiro;44 4444-4444

Percorrendo o ArrayList (usando iterator)
Posição 0- Juca Bala;11 1111-1111
Posição 1- Maria Antonieta;33 3333-3333
Posição 2- Antônio Conselheiro;44 4444-4444
```

Vetores e Matrizes

* Precisamos, também, entender e manipular listas INDEXADAS.
  * Para tal, utilizaremos a classe **HashMap**.

Vetores e Matrizes

* Coleções:



Vetores e Matrizes

* Vantagem: Rapidez no acesso a um objeto da lista
* pré-requisitos: Chave única
  * Exemplos: CPF, CNPJ, Matrícula,...
* Construtor :
  * HashMap <Chave,Classe> ( )
* Exemplo:
  * HashMap<String,Cliente> listaCli = new HashMap<String,Cliente>( );

Vetores e Matrizes

* Métodos principais:
  * incluir : put
    * listaCli.put(objCli.getCpf , objCli);
  * pesquisa rápida : containsKey
    * if (listaCli.containsKey(cpf)) throw new Exception...
  * buscar objeto : get(chave);
    * Cliente objCli = listaCli.get(cpf);
  * remover objeto : remove(chave);

Vetores e Matrizes

* Métodos principais:
  * Acesso aos objetos : values()

  * HashMap <String,Contato> telefones = new HashMap<String,Contato>();

  * // Criação do Vector<Contato> a partir de HashMap <String,Contato>
  * Vector<Contato> listaContatos = new Vector<Contato>();

  * for (Contato objContato : telefones.values())
    listaContatos.add(objContato);

Obrigado.

joaopauloaramuni@gmail.com
joaopauloaramuni@fumec.br

UNIVERSIDADE FUMEC
50 Anos