PREPARAÇÃO PARA MARATONA DE PROGRAMAÇÃO
CETEC CPS 2016

2ª Parte

Na segunda parte do nosso treinamento, veremos alguns conceitos básicos de manipulação de arrays e da Classe Collection do Java.

Os códigos fontes aqui apresentados, encontram-se disponíveis na plataforma. Para um melhor entendimento, sugere-se a execução dos mesmos.

Manipulação de Arrays

Vetores (Unidimensionais)

Vetor é uma estrutura que necessita apenas de um índice para a identificação de um elemento nele contido. Sendo assim, para manipular um valor em um vetor é necessário fornecer o nome (identificador) do vetor e o índice do elemento desejado. O índice determina a posição na estrutura onde o elemento está inserido. Cada posição de um vetor contém exatamente um valor que pode ser manipulado individualmente.

Declaração: A declaração de um vetor deverá ser escrita da seguinte forma:

```
String difficult[ ];
double média[ ];
int temp[ ];
```

Outra alternativa de declaração:

```
String[] difficult;
double[] média;
int[]  temp;
```

Criando Objetos Arrays:

Um dos caminhos é usar o operador *new* para criar uma nova instância de um array, por exemplo:

```
int temps[] = new int[9];
```

Quando um objeto array é criado usando o operador *new,* todos os índices são inicializados (0 para arrays numéricos, falso para boolean, '\0' para caracteres, e NULL para objetos). É possível criar e inicializar um array ao mesmo tempo.

```
String nomes[] = new String { "Paulo",  "Fernando",  "Leticia" ,  "Carlos",
"André"};
```

Cada um dos elementos internos deve ser do mesmo tipo e deve ser também do mesmo tipo que a variável que armazena o array. O exemplo acima cria um array de Strings chamado **nomes** que contém 5 elementos.

Acessando os Elementos do Array:

Uma vez que você tem um array com valores iniciais, você pode testar e mudar os valores em cada índice de cada array.

Os arrays em Java sempre iniciam na posição 0. Por exemplo:

```
String[]  arr= new String[10];
arr[10]="fora do limite";
```

Isto provoca um erro de compilação pois o índice 10 não existe. Como inicia na posição 0, 10 está fora do limite do array.

```
arr[9] = "dentro do limite";
```

Esta operação de atribuição é válida e insere na posição 9 do array, a String "dentro do limite".

Obtendo o tamanho do array:

```
char [ ] alfabeto = new char [26];
int tamAlfabeto = alfabeto.length;                    // tamAlfabeto == 26

String [ ] mosqueteiros = { "ioumle", "iodoisle", "iotrêsle" };
int num = mosqueteiros.length;                  // num == 3
```

A forma de acesso a um determinado elemento do array é feita simplesmente fazendo referência ao índice do mesmo.

O acesso a um elemento do array é feito colocando uma expressão de valor inteiro entre colchetes após o nome do array.

nota	6.0	7.0	9.0	5.5	9.1	10.0	4.7	7.5	8.6	8.0
	0	1	2	3	4	5	6	7	8	9

nota[3] faz referência ao elemento do vetor, cujo conteúdo é 5.5.

Desta forma, para se ter acesso a qualquer uma das notas armazenadas basta utilizar uma variável inteira qualquer (a título de exemplo a variável **i**) como sendo o índice. Supondo i=5, uma referência a **nota[i], i** seria substituído pelo seu conteúdo no dado instante e neste caso, o elemento referenciado é = 10.0.

O exemplo a seguir cria um array de inteiros chamado tecladoNum e depois preenche o array com inteiros de 0 a 9:

```
int tecladoNum[ ] = new int [10];
for ( int i = 0; i < tecladoNum.length; i++)
   tecladoNum [i] = i;
```

Tentar acessar um elemento fora do intervalo do array gera uma ArrayIndexOutOfBoundsException. Esse e um tipo de RuntimeException, e por isso você pode apanhá-la e tratá-la, ou então ignorá-la:

```
String [ ] estados = new String [10];
try {
      estados[0] = “Pará”;
      estados[1] = “Amazonas”;
      estados[25]=“Amapá”; // Erro: array fora dos limites
     }
catch (ArrayIndex0utOfBoundsException erro )
JOptionPane.showMessageDialog(null, “Erro tratado:
”+erro.getMessage( ));
```

Exemplo 1:

Dado um conjunto com os seguintes elementos: 1, 2, 4, 8, 6, 7, 15, 9, 10, 18. Elabore um aplicativo para calcular a soma desses elementos.

```
// Calcula a soma dos elementos de um vetor
import javax.swing.*;

public class SomaVetor {
  public static void main( String args[] )
  {
    int a[] = { 1, 2, 4, 8, 6, 7, 15, 9, 10, 18 };
    int total = 0;
    String saida="Elementos do vetor\n";
    for ( int i = 0; i < a.length; i++ )
      { saida+=a[i]+" ";
       total += a[ i ];}
```

```
      JOptionPane.showMessageDialog( null, saida +"\nSoma do elementos do
vetor " + total, "Soma os elementos do vetor",
JOptionPane.INFORMATION_MESSAGE );
      System.exit( 0 );
   }
}
```

Exemplo 2: Entrada de dados pelo usuário.

```
//Arquivo ExemploVetor.java
//Cria e permite o preenchimento  pelo usuário de um vetor com
//10 elementos do tipo int e ao final mostra seus elementos

import javax.swing.*;
public class ExemploVetor{
 public static void main (String args[])
   { int v[]=new int[10];

     for (int i=0; i<v.length; i++)
       {  v[i]=Integer.parseInt(  JOptionPane.showInputDialog(  "Digite  o  valor  do
"+(i+1)+
          "º elemento (posição "+i+")"));       }

     String resposta="Posição\tValor";
     for (int i=0; i<v.length; i++)
       { resposta+="\nv["+i+"]\t"+v[i]; }

     JTextArea saida = new JTextArea(11,10);
     saida.setText(resposta);

     JOptionPane.showMessageDialog(null,saida);

      System.exit(0);
 } }
```

Exemplo 3 – Consulta

```
/*Cria e permite o preenchimento  pelo usuário de um vetor com
 *10 elementos do tipo int e ao final mostra seus elementos
 *e permite a consulta dos valores pela posição */

import javax.swing.*;
public class ExemploVetorConsulta{
  public static void main (String args[])
    { int v[]=new int[10];
```

```
    for (int i=0; i<v.length; i++)
      { v[i]=Integer.parseInt( JOptionPane.showInputDialog( "Digite o valor do
"+(i+1)+ "º elemento (posição "+i+")"));  }

    String resposta="Posição\tValor";
    for (int i=0; i<v.length; i++)
     { resposta+="\nv["+i+"]\t"+v[i]; }

    JTextArea saida = new JTextArea(11,10);
    saida.setText(resposta);

    JOptionPane.showMessageDialog(null,saida);

    int consulta = Integer.parseInt( JOptionPane.showInputDialog( "Digite uma
posição válida para consulta (0-9)"));

    while ((consulta>=0) && (consulta<=9))
     { JOptionPane.showMessageDialog( null,"Posição: "+consulta+" ---> Valor:
"+v[consulta]);
       JOptionPane.showMessageDialog(null,saida);
       consulta = Integer.parseInt( JOptionPane.showInputDialog( "Digite uma
posição válida para consulta (0-9)"));
     }
   System.exit(0);
}
 }
```

Arrays Multidimensionais

Estruturas indexadas que necessitam de mais que um índice para identificar um de seus elementos são chamadas de matrizes de dimensão n, onde n representa o número de índices requeridos.

Uma matriz de duas dimensões (dois subscritos) pode ser compreendida como uma tabela de valores consistindo em informações organizadas em linhas e colunas. Por convenção, o primeiro subscrito identifica a linha do elemento e o segundo identifica a coluna do elemento. Os arrays que necessitam de dois subscritos para identificar um elemento específico são chamados de arrays bidimensionais.

Exemplo:

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]
Linha 1	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]
Linha 2	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

Declaração: A declaração de uma matriz deverá ser escrita da seguinte forma:

```
int vendas [][];
double notasProvas[][];
```

Arrays multidimensionais podem podem ser iniciados em declarações da mesma forma que um array unidimendional, isto é:

```
int vendas [][]={{1,2},{3,4}};
```

Essa matriz poderia ser graficamente representada por:

	Coluna 0	Coluna 1
Linha 0	1	2
Linha 1	3	4

Assim como ocorre com arrays unidimensionais, os elementos de um array multidimenional são automaticamente iniciados quando new cria o objeto array.

Exemplo:

```
int b[][];
b=new int [3] []; //aloca linhas
b[0] = new int [5]; //aloca colunas para a linha 0
b[1] = new int [3]; //aloca colunas para a linha 1
```

Exemplo: Dada uma matriz quadrada de ordem M, separar os elementos da diagonal principal em um vetor.

```
import javax.swing.*;

public class DiagonalPrincipal{

        public static void main (String args[])
                { int mat[][], diag[],lin,col;

                lin=Integer.parseInt(JOptionPane.showInputDialog("Quantas
linhas tem a matriz"));

                col=lin;
                mat=new int[lin][col];
```

```
                diag=new int[lin];

                for (int linha=0; linha<mat.length; linha++)
                        { for (int coluna=0; coluna< mat[linha].length; coluna++)
                                { mat[linha][coluna]=(int)(Math.random()*10); }
                }

                String resposta="Colunas\t0\t1\t2\t3";
                resposta+="\nLinhas";

                for (int linha=0; linha<mat.length; linha++)
                        { resposta+="\n"+linha;
                        for (int coluna=0; coluna<mat[linha].length; coluna++)
                                { resposta+="\t"+mat[linha][coluna]; }
                }

                resposta+="\n\nDIAGONAL PRINCIPAL";

                for (int linha=0; linha<mat.length; linha++)
                        {diag[linha]=mat[linha][linha];
                        resposta+="\t"+diag[linha];}

                JTextArea saida = new JTextArea(resposta);
                JOptionPane.showMessageDialog(null,saida);
                System.exit(0);
        }
}
```

Collection

Como visto no tópico anterior, utilizar arrays é um tanto quanto trabalhoso, a saber:

- um array em Java não pode ser redimensionado em tempo de execução;
- não é possível encontrar diretamente um elemento do qual não saibamos o índice;
- não é possível saber as posições do array que já foram preenchidas, sem criar métodos auxiliares para isso;

- como saber quantas posições já foram usadas no array?

Para resolver essas pendências foi implementado no Java, a partir da versão 2.1.2 a Collections Framework que é um conjunto de diversas classes que implementam estruturas de dados avançadas.

PREPARAÇÃO PARA MARATONA DE PROGRAMAÇÃO
CETEC CPS 2016

Listas

Uma lista é uma coleção que permite elementos duplicados e mantém uma ordenação específica entre esses elementos. Isso permite ações como busca, remoção, tamanho “infinito”, entre outras, de maneira mais fácil e ágil.

A implementação mais utilizada da interface List é a ArrayList, que trabalha com um array interno para gerar uma lista.

Neste treinamento abordaremos apenas a ArrayList, que é mais do que suficiente para a resolução dos problemas apresentados na Maratona de Programação.

Vale frisar que existe muita confusão em relação da ArrayList ser um array. Não é. Internamente, ela lança mão de um array como estrutura para armazenar os dados, porém, este atributo está propriamente encapsulado e não há como acessá-lo. Outra dica é que não se pode usar [] com uma ArrayList e nem acessar o atributo length.

Para criar um ArrayList, chama-se o construtor:

```
ArrayList lista = new ArrayList();
```

Pode-se também abstrair a lista da interface List:

```
List lista = new ArrayList();
```

Para criar uma lista de cidades (String), usamos o seguinte código:

```
List cidades = new ArrayList();
lista.add(“São Paulo”);
lista.add(“Ubatuba”);
lista.add(“Campinas”);
lista.add(“Caraguatatuba”);
```

A interface List possui dois métodos add: um que recebe o objeto a ser inserido e o coloca no final da lista, e um segundo que permite adicionar o elemento em qualquer posição. Perceba que em nenhum momento se faz necessário informar o tamanho da lista. Assim, podemos acrescentar quantos elementos quisermos.

As Listas trabalham de forma genérica. Isto é, não existe ArrayList específica para Strings, outra para Inteiros, etc. Todos os métodos trabalham com Objects.

Podemos então, criar, por exemplo, uma lista de Carros:

```
Carro car1 = new Carro();
car1.marca = “Fiat”;
```

```
Carro car2 = new Carro();
car2.marca = “Ford”;
Carro car3 = new Carro();
car3.marca = “BMW”;

Lista carros = new ArrayList();
carros.add(car1);
carros.add(car2);
carros.add(car3);
```

Para saber quantos elementos há na lista, usamos o método size();

```
System.out.println(carros.size());
```

O método get(int) recebe o argumento do índice do elemento que gostaríamos de recuperar da Lista. Assim, podem fazer um for para iterar com a lista de carros:

```
for(int x=0;x<carros.size();x++{
        System.out.println(carros.get(x).getMarca());
}
```

Uma outra maneira, seria recriar um objeto Carro para cada item da Lista:

```
for(int x=0;x<carros.size();x++{
        Carro car = (Carro) carros.get(x);
        System.out.println(car.getMarca());
}
```

Existem outros métodos como o remove(), que recebe um objeto para ser removido da lista; e contains(), que recebe um objeto como argumento e devolve true ou fale, indicando se o elemento está ou não na lista.

A partir do Java 5.0 podemos usar o recurso Generics para restringir as listas a um determinado tipo de objetos:

```
List<Carro> carros = new ArrayList<Carro>();
carros.add(car1);
carros.add(car2);
carros.add(car3);
```

O uso de Generics elimina a necessidade de casting, já que, seguramente, todos os objetos inseridos na lista serão do tipo Carro.

Nesse caso, podemos utilizar um for-each para percorrer a ArrayList Carro:

```
for(Carro car: carros){
        System.out.println(car.getMarca());
}
```

Ao incluir registros em uma lista, os dados ficam disponíveis na ordem em que foram inseridos. Porém, algumas vezes precisaremos percorrer essas listas de forma ordenada.

A classe Collections fornece um método estático sort que recebe uma List como argumento e o ordena por ordem crescete:

```
List<String> cidades = new ArrayList<>();
cidades.add("São Paulo");
cidades.add("Ubatuba");
cidades.add("Caraguatatuba");

System.out.println(cidades);

Collections.sort(cidades);

System.out.println(cidades);
```

Perceba que na primeira saída a lista é impressa na ordem de entrada, e já na segunda saída ela estará em ordem alfabética.

No entanto, toda lista em Java pode ser de qualquer objeto. Por exemplo, se na classe Carro, tivermos, além da marca, o ano de fabricação, a cor, entre outros dados. E se quisermos ordenar pelo ano de fabricação, como seria ordenada a lista abaixo:

```
Collections.sort(carros);
```

Sempre que formos trabalhar com ordenação de objetos, precisaremos definir um critério de ordenação. Assim, será necessário dizer ao método sort sobre como comparar algum dado do objeto, afim de determinar a ordem da lista.

Assim para ordenar a classe Carro pelo ano de fabricação, basta implementar a interface Comparable, com o método compareTo:

```
public class Carro implements Comparable<Carro>{

        String marca;
        int ano;

        //... o restante do código anterior ficaria aqui

        public int compareTo(Carro outrocarro){
```

```
            if(this.ano < outrocarro.ano){
                return -1;
            }
            if(this.ano > outrocarro.ano){
                return 1;
            }
            return 0;
        }
}
```

Nesse caso, toda vez que chamarmos o método sort de Collections, ele saberá como fazer a ordenação da lista. Utilizará o critério que definirmos no método compareTo.