Estruturas de Dados

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Vetor

- Também conhecidos como arrays unidimensionais os vetores em Java são alocados dinamicamente.

- Declaração:

<tipo_do_dado> nome_vetor[] = new <tipo_do_dado>[quantidade]

- Ex:

```
int n[] = new int[100];
```

Obs1. Em C faríamos: int *n = (int*)malloc(100*sizeof(int));

Obs2. Será que Java não tem ponteiro ?

Vetor

- Uso de vetor:

```
n[0] = 100;
```

- Inicialização de um vetor:

<tipo_do_dado> nome_vetor[] = { valores separados por vírgulas }

Ex: char caracterArray[] = { ‘a’, ‘b’, ‘c’ }

Matrizes

- Também conhecidas como arrays multidimensionais as matrizes em Java são alocadas dinamicamente.

- Declaração:

<tipo_do_dado> nome_matriz[][] = new <tipo_do_dado>[numLinhas] [numColunas]

- Ex:

float notas[][] = new float[3][2];

- Uso da Matriz:

notas[0][1] = 23.5;

Passagem de Vetores (Arrays) em métodos

- Para passar um vetor como parâmetro deve-se utilizar a seguinte assinatura (suponhamos um método public static para facilitar a leitura da assinatura):

public static <tipo_do_retorno><nome_metodo>(<tipo_do_vetor> <nome_vetor>[])

Ex: public static int meuMetodo(int arr[])

Retornando Vetores

- Para retornar um vetor deve-se utilizar a seguinte assinatura (suponhamos um método public static para facilitar a leitura da assinatura):

public static <tipo_do_dado>[] <nome_metodo> (<parâmetros>)

Ex: public static int[] vetorDados(int x)

Exemplo de Passagem de vetor como parâmetro e retorno vetor

```
package arrays;

public class VetorDinamico {

    public static int[] insereValorArray(int arr[], int
      valor)

    {

        int novo_array[] = new int[arr.length + 1];

        for(int i = 0; i < arr.length; i++)

        {

            novo_array[i] = arr[i];

        }

        novo_array[arr.length] = valor;

        return novo_array;

    }
```

```
    public static void main(String[] args) {

        // TODO code application logic here

        int n1[] = { 1, 2, 3 };

        int n2[] = insereValorArray(n1, 4);

        for(int i = 0; i < n2.length; i++)

        {

            System.out.println(n2[i]);

        }

        System.out.println("fim");

    }

}
```

Vetor de Objetos

- Para declarar um vetor de objetos deve-se trabalhar da mesma forma que se trabalha com tipos primitivos, ou seja:

<tipo_do_dado> nome_vetor[] = new <tipo_do_dado>[quantidade]

Onde <tipo_do_dado> é uma classe

Ex: Pessoa pessoa[] = new Pessoa[100];

Problemas com o uso de vetor

- O uso de vetores (arrays) na programação Java tem uma característica que é a imutabilidade dos objetos array, assim como C, ou seja suas dimensões uma vez definidas não se modificam, a não ser que seja criado um novo array e os dados copiados para ele. Ex:

```
Classe[] original = new Classe[<tamanho_original>];
...
Classe objetoNovo = new Classe();

Classe[] novo = new Classe[original.length + 1];

for (int i = 0; i < original.length; i++)
{
        novo[i] = original[i]
}
novo[novo.length - 1] = objetoNovo;
original = novo;
```

Listas em Java

- Uma lista é um conjunto de elementos que possuem alguma ordem e onde é possível inserir ou excluir elementos sem a necessidade de reconstrução da lista.

- Em Java, analisaremos 2 tipos de listas:
  - ArrayList: é bastante semelhante ao vetor (array) que conhecemos, na realidade internamente o ArrayList é implementado com vetor, de forma que, quando adicionamos um novo elemento ao ArrayList, internamente ele aloca um novo vetor com uma posição a mais, em seguida promove a cópia dos elementos antigos e finalmente coloca o novo elemento no vetor.
  - LinkedList: é implementada de forma semelhante a uma lista duplamente encadeada.

Iterator

- Iterator: é uma interface genérica para navegação em estruturas de dados, facilitando o uso das mesmas.

- Para seu uso as estruturas de dados precisam implementar os seguintes métodos:
  - boolean hasNext(): verifica se existe um próximo elemento.
  - next(): retorna o próximo elemento da coleção.
  - void remove(): remove o elemento sob o iterator. Deve ser chamado somente após a chamada de next().

- As duas listas que estamos estudando implementam esses métodos visto que implementam essa interface.

Comparando as classes ArrayList e LinkedList

- ArrayList
  - É mais rápida para acesso utilizando índices.
  - Para remoção é necessário corrigir todos os índices
- LinkedList
  - É mais rápida para inserção e iteração (utilizando iterator).
  - Para remoção do primeiro ou do último elementos da lista existem métodos específicos de ótimo desempenho.
  - Para remoção de um elemento no interior da lista basta acertar as referências posterior e anterior ao nó, entretanto é necessário iterar até encontrar esse nó.

ArrayList - Métodos principais

- Construtor:
  - public ArrayList(): constroi o vetor interno com tamanho 10 por padrão.
  - public ArrayList(int initialCapacity): constroi o vetor interno com tamanho initialCapacity.
  - public ArrayList(<coleção>): constroi uma lista a partir da coleção fornecida como parâmetro.

- public boolean add(<elemento>): adiciona <elemento> no final da lista (<elemento> pode ser um objeto ou um valor de tipo primitivo – int, float, etc).
- public void add(index, <elemento>): adiciona o <elemento> na posição index (deslocando os elementos posteriores para frente). Se index maior que o tamanho interno do vetor dispara uma exceção do tipo IndexOutOfBoundsException.
- public <elemento> get(int index): obtem o elemento de índice index. Se o índice não existir é acionada uma exceção do IndexOutOfBoundsException.

ArrayList - Métodos principais

- public <elemento> remove(int index): retorna o elemento de índice index e o elimina da lista.
- public boolean isEmpty() : determina se a lista está vazia.
- public <elemento_antigo> set (int index, <elemento_novo): altera o valor no índice index de <elemento_antigo> para <elemento_novo>.
- public boolean addAll(<coleção>): adiciona no final da lista todos os elementos de coleção.

Exemplo - ArrayList

```
package arrays;

import java.util.ArrayList;

public class UsaArrayList {

    public static void main(String[] args) {

        ArrayList als1 = new ArrayList();

        ArrayList als2 = new ArrayList(80);

        als1.add("1");

        als1.add("2");

        als1.add(1, "3");

        System.out.println(als1.get(1));

        System.out.println(als1.remove(0));

        System.out.println(als1.get(1));

        System.out.println(als1.set(0, "4"));

        als2.add("5");

        als2.addAll(als1);

        while ( !als1.isEmpty() )

        {

            System.out.println(als1.remove(0));

        }

        ArrayList als3 = new ArrayList(als2);

    }

}
```

LinkedList – Métodos Principais

- Construtor:
  - public LinkedList(): construtor padrão
  - public LinkedList(<coleção>): constroi uma lista a partir da coleção fornecida como parâmetro.
- public boolean add(<elemento>): adiciona <elemento> no final da lista (<elemento> pode ser um objeto ou um valor de tipo primitivo – int, float, etc).
- public void addFirst(<elemento>): adiciona <elemento> no inicio da lista.
- public void addLast(<elemento>): equivale a add.
- public <elemento> get(int index): obtem o elemento de índice index.
- Os outros métodos discutidos para ArrayList também existem em LinkedList, mas suas implementações são diferentes.

Exemplo – LinkedList com iterator

```
import java.util.LinkedList;
import java.util.Iterator;
public class UsaLinkedList {
    public static void main(String[] args) {
        LinkedList ls1 = new LinkedList();
        ls1.add(1);
        LinkedList ls2 = new LinkedList(ls1);
        ls1.addFirst(2);
        ls1.addLast(3);
        System.out.println(ls1.get(1));
        Iterator it = ls1.iterator();
        // it.remove();
        while (it.hasNext())
        {
            System.out.println(it.next());
            it.remove();
        }
    }
}
```

Experimente descomentar a linha: // it.remove();

removeIf

- Esse método é utilizado para remover todos os elementos que satisfazem a um dado predicado (função de verificação de condição).
- Aqui temos o uso de funções lambda. Esse assunto será melhor estudado posteriormente.

```
public class Principal {

    public static void main(String[] args) {

        LinkedList<String> ls1 = new LinkedList<String>();

        ls1.add("Pedro");

        ls1.add("Ana");

        ls1.add("Jaci");

        ls1.add("Paulo");

        ls1.add("Analto");



        ls1.removeIf(x -> x.charAt(0) == 'P');

        System.out.println(ls1);

    }

}
```

Condição

```
compile:
run:
[Ana, Jaci, Analto]
BUILD SUCCESSFUL (total time: 2 seconds)
```

for-each

- É uma forma alternativa a navegação por índices em Java.
- Seu formato é:

```
for(<tipo> <variável>: <lista>){
    …
}
```

Onde:

- tipo é a classe dos objetos que compõe a lista
- variável a cada “rodada” do for-each será o elemento corrente da lista.
- lista é a coleção que queremos iterar (“andar sobre ela”)

```
public class Principal {

    public static void main(String[] args) {

        LinkedList<String> ls1 = new LinkedList<>();

        ls1.add("Pedro");

        ls1.add("Ana");

        ls1.add("Jaci");

        ls1.add("Paulo");

        ls1.add("Analto");


        LinkedList<String> ls2 = new LinkedList<>();

        for(String nome: ls1){

            if(nome.charAt(0) != 'P'){

                ls2.add(nome);

            }

        }

        System.out.println(ls2);

    }
}
```

```
compile:
run:
[Ana, Jaci, Analto]
BUILD SUCCESSFUL (total time: 1 second)
```

Streams

- Permite ao desenvolvedor trabalhar com coleções de forma mais simples e com menor quantidade de linhas de código.
- Pode ser utilizado para substituir de forma simples o for-each.

```
ls1.stream().forEach(nome -> System.out.println(nome));
```

- Podemos também fazer filtragens nos elementos da lista usando o método **filter**.
- A Stream API possui uma série de funcionalidades para se trabalhar com listas. Não vamos tratá-las a fundo aqui pois exigem diversos conceitos, como por exemplo, funções lambda, entretanto é bastante recomendável que futuramente sejam fonte de estudo para otimização de desempenho e quantidade de linhas de código no contexto do uso de listas (coleções).

```
public class Principal {

    public static void main(String[] args) {

        LinkedList<String> ls1 = new LinkedList<>();

        ls1.add("Pedro");

        ls1.add("Ana");

        ls1.add("Jaci");

        ls1.add("Paulo");

        ls1.add("Analto");

        ls1.stream().forEach(nome -> System.out.println(nome));

        LinkedList<String> ls2 = new LinkedList<>();

        ls1.stream().filter((nome) -> (nome.charAt(0) !=
          'P')).forEachOrdered((nome) -> {

              ls2.add(nome);

        });

        System.out.println(ls2);

    }
}
```

```
Pedro
Ana
Jaci
Paulo
Analto
```

```
[Ana, Jaci, Analto]
BUILD SUCCESSFUL (total time: 2 seconds)
```

Dúvidas?