LCP - Aula 03

Prof. Lucas Guerreiro

Aula anterior

- ➔ Orientação a Objeto
- ➔ Práticas em OO
- ➔ Modificadores de Acesso
- ➔ Pacotes em Java

Aula 03

Objetivo: Vetores e Coleções

Classes Wrapper

Objetivo: Instanciar objetos que armazenam valores primitivos.

Primitivo	**Wrapper**
int	Integer
double	Double
char	Character
boolean	Boolean

Vetores (arrays)

- Estruturas de armazenamento de uma quantidade fixa de dados.
- Deve-se instanciar um objeto para se referir a vetores.
- Têm tamanho fixo e não são redimensionáveis.

```
int vetor[] = new int[10];

vetor[0] = 1000;

System.out.println(vetor[0]); // 1000
```

Exemplo

Receber um número no console e alimentar um vetor com a "tabuada" de 1 a 10.

Depois do vetor completo, exibir os valores do vetor.

Exemplo

```
Scanner scan = new Scanner(System.in);
int num = scan.nextInt();
int[] vetor = new int[10];
for (int i = 0; i < 10; i++) {
    vetor[i] = (i+1) * num;
}
for (int i = 0; i < 10; i++) {
    System.out.println(vetor[i]);
}
scan.close();
```

Exemplo01.java

Vetores

Formas de se percorrer vetores:

```
for (int i = 0; i < 10; i++){ … }  // vetor[i]
for (int i = 0; i < vetor.length; i++) { … }
for (int i: vetor) { … } // i
for (int i: listaobjetos) { … } // i
```

No exemplo anterior, poderíamos fazer:

```
for (int i : vetor) {
    System.out.println(i);
}
```

Exercício

Criar uma classe FuncionariosArray. Nesta classe teremos atributos de nome e salário, alimentados pelo construtor. No método principal, instanciar um vetor de 5 posições da classe. Receber do usuário os nomes e salários de cada um dos funcionários via console. Por fim, exibir os nomes e salário (implementados no toString) de cada um percorrendo o array.

Exercício

```
public class FuncionariosArray{
	private String nome;
	private double salario;
	public FuncionariosArray(String nome, double salario) {
		this.nome = nome;
		this.salario = salario;
	}
	@Override
	public String toString() {
		return "Funcionário: " + nome + " recebe R$" + salario;
	}
	public static void main(String[] args) {
		FuncionariosArray[] funcionarios = new FuncionariosArray[5];
		Scanner scan = new Scanner(System.in);
		for (int i = 0; i < funcionarios.length; i++) {
			System.out.print("Digite o nome do funcionário: ");
			String nome = scan.nextLine();
			System.out.printf("Digite o salário do funcionário %s: ", nome);
			double salario = scan.nextDouble();
			funcionarios[i] = new FuncionariosArray(nome, salario);
			scan.nextLine();
		}
		scan.close();
		for (FuncionariosArray func : funcionarios) {
			System.out.println(func);
		}
	}
}
```

FuncionariosArray.java

Matriz

Matriz bidimensional:

```
int[][] matriz = new int[][];

matriz[linha][coluna];

int matriz[][] = new int[5][];

matriz[0] = new int[10];
```

Acessos a matriz de maneira análoga a como fizemos com vetores.

Exemplo - Printar * de *1* a *num* com matriz

```
Scanner scan = new Scanner(System.in);
int num = scan.nextInt();
String[][] matriz = new String[num][num];

for (int i = 0; i < num; i++) {
    for (int j = 0; j < i + 1 ; j++) {
        matriz[i][j] = "*";
    }
}

for (int i = 0; i < num; i++) {
    for (int j = 0; j < num; j++) {
        if (matriz[i][j] != null) {
            System.out.print(matriz[i][j]);
        }
    }
    System.out.println();
}
```

```
for (int i = 0; i < num; i++) {
    for (int j = 0; j <= i  ; j++) {
        matriz[i][j] = "*";
    }
}
```

Matriz.java

Vetores

Quando instanciamos um array estamos instanciando um espaço de memória. Ainda, com um espaço limitado reservado.

O que acontece neste caso?

```
int[] x = new int[3];
x= {1, 2, 3};
System.out.println("x= " + (x[0] + " " + x[1] + " " + x[2]);
y = x;
y[0] = 4;
y[1] = 5;
y[2] = 6;
System.out.println("y= " + y[0] + " " + y[1] + " " + y[2]);
System.out.println("x= " + x[0] + " " + x[1] + " " + x[2]);
```

Vetores

```
int[] x = new int[3];
x= {1, 2, 3};
System.out.println("x= " + (x[0] + " " + x[1] + " " + x[2]);
y = x.clone();
y[0] = 4;
y[1] = 5;
y[2] = 6;
System.out.println("y= " + y[0] + " " + y[1] + " " + y[2]);
System.out.println("x= " + x[0] + " " + x[1] + " " + x[2]);
```

Vetores

Se tivermos uma nova entrada para adicionar em um array:

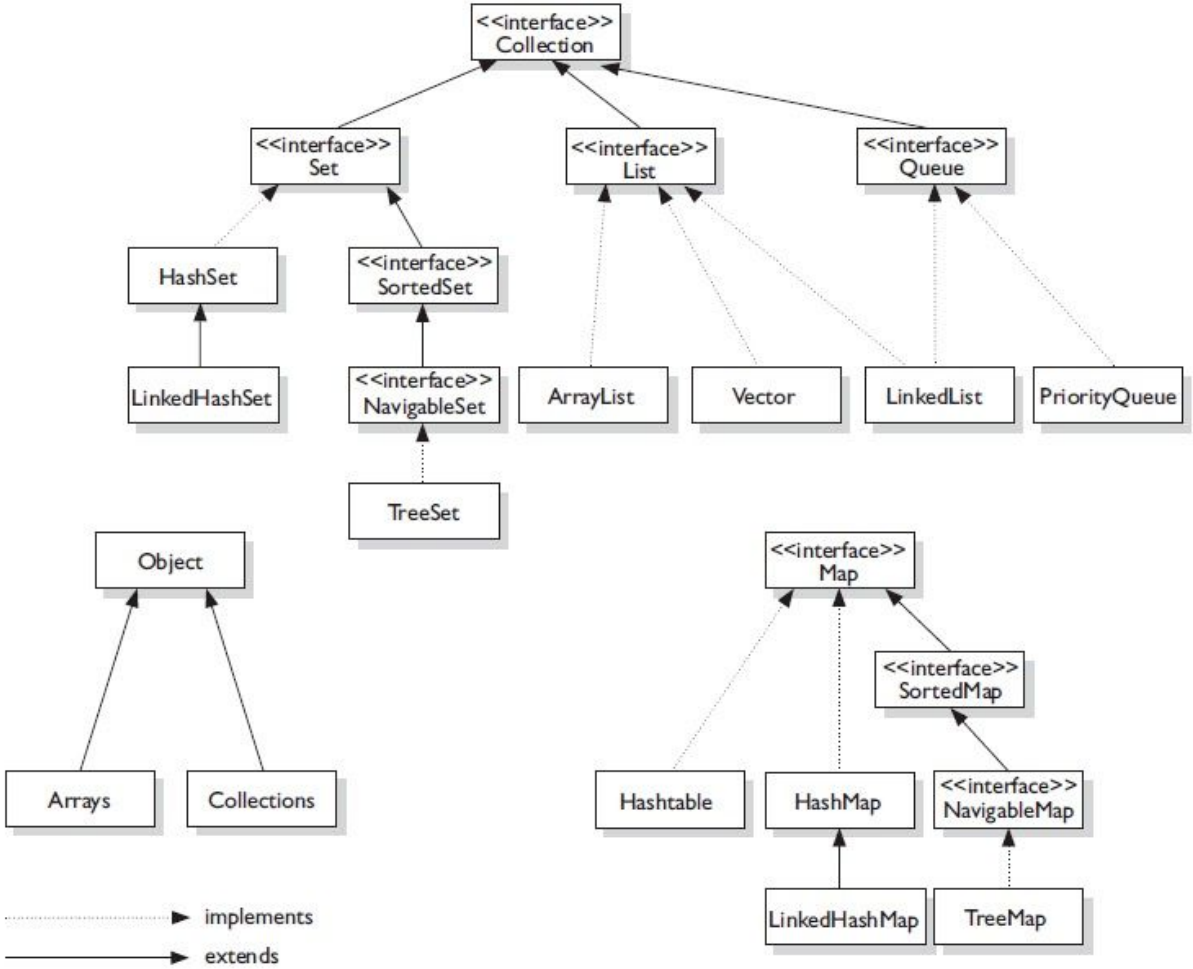
```
int[] x = new int[3];
x= {1, 2, 3};
for (int i: x) System.out.println(i);
int[] y = new int[x.length+1];
for (int i=0; i < x; i++) y[i] = x[i];
y[y.length] = 1000;
```

Solução otimizada: uso de coleções

Coleções

Uma coleção é uma estrutura de dados — na realidade, um objeto — que pode armazenar referências a outros objetos.

Interface	Descrição
Collection	A interface-raiz na hierarquia de coleções a partir da qual as interfaces Set, Queue e List são derivadas. Sem implementação direta
Set	Uma coleção que não contém duplicatas.
List	Uma coleção ordenada que pode conter elementos duplicados.
Map	Uma coleção que associa chaves a valores e que não pode conter chaves duplicadas. Map não deriva de Collection.
Queue	Em geral, uma coleção primeiro a entrar, primeiro a sair que modela uma fila de espera; outras ordens podem ser especificadas.


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
HashSet
<<interface>>
SortedSet
LinkedHashSet
<<interface>>
NavigableSet
ArrayList
Vector
LinkedList
PriorityQueue
TreeSet
Object
<<interface>>
Map
<<interface>>
SortedMap
Arrays
Collections
Hashtable
HashMap
<<interface>>
NavigableMap
LinkedHashMap
TreeMap
implements
extends

Iterator e ListIterator

Iterator: interface que pode utilizada para percorrer implementações de Collection

ListIterator: permite percorrer Collection no sentido bidirecional. Estende Iterator.

Ambas interfaces permitem diversas operações nas Collections

List

- **Interface** de uso geral para trabalhar com elementos ordenados
- Implementa métodos de acesso a manipulação da lista de acordo com a posição dos elementos
- Tipo de elementos da lista é especificado em List<**E**>
- Principais implementações de List: **ArrayList**, **Vector** e **LinkedList**

ArrayList

```
ArrayList<Integer> numeros = new ArrayList<Integer>();
numeros.add(1);
numeros.add(2);
numeros.add(3);
System.out.println("tamanho da lista: " + numeros.size());
System.out.println("valor posicao 1: " + numeros.get(1));

for (int i = 0; i < numeros.size(); i++) {
    System.out.println("valor em " + i + " = " + numeros.get(i));
}
numeros.remove(1);
for (int i = 0; i < numeros.size(); i++) {
    System.out.println("valor em " + i + " = " + numeros.get(i));
}
```

ArrayList1.java

ArrayList

```
ArrayList<String> nomes_funcionarios = new ArrayList<String>();
nomes_funcionarios.add("Paulo");
nomes_funcionarios.add("José");
nomes_funcionarios.add("Carlos");
for (String nome : nomes_funcionarios) {
      System.out.println(nome);
}
Collections.sort(nomes_funcionarios);
for (String nome : nomes_funcionarios) {
      System.out.println(nome);
}
```

ArrayList2.java

Polimorfismo

```
List<String> lista1 = new ArrayList<String>();

lista1 = new LinkedList<String>();

ArrayList<String> lista2 = new ArrayList<String>();

lista2 = new LinkedList<String>();
```

Polimorfismo

```
ArrayList<String> lista1 = new ArrayList<String>();

ArrayList<String> lista2 = (ArrayList<String>) lista1.clone();
```

~~`List<String> lista1 = new ArrayList<String>();`~~

~~`List<String> lista2 = (ArrayList<String>) lista1.clone();`~~

Exemplo

Vamos ter uma lista de cores iniciais, uma lista de cores a remover da lista original. Tarefa: implementar a remoção da lista de cores a partir da lista de remoção através do uso de lista, collections e iterators.

Exemplo - Parte 1

```
public static void main(String[] args) {
	String[] cores = {"AZUL", "VERMELHO", "VERDE", "AMARELO", "CINZA", "PRETO", "BRANCO"};
	List<String> lista = new ArrayList<String>();
	for (String cor: cores) {
		lista.add(cor);
	}
	String[] coresRemover = {"BRANCO", "VERDE"};
	List<String> listaRemover = new ArrayList<String>();
	for (String cor: coresRemover) {
		listaRemover.add(cor);
	}

	exibeCores(lista, "lista inicial");
	exibeCores(listaRemover, "lista de cores a remover");
	removerCores(lista, listaRemover);
	exibeCores(lista, "lista final");
}
```

Cores.java

Exemplo - Parte 2

```
public static void exibeCores(Collection<String> imprimirLista, String texto) {
        System.out.print("imprimindo " + texto + ": ");
        for (String c : imprimirLista) {
                System.out.print(c + " ");
        }
        System.out.println();
}

public static void removerCores(Collection<String> col1, Collection<String> col2) {
        Iterator<String> iterator = col1.iterator();

        while (iterator.hasNext()) {
                if (col2.contains(iterator.next())){
                        iterator.remove();
                }
        }
}
```

Cores.java

Vector

- ➔ Foi criado desde o Java 1.0 (antes da Collections Framework).
- ➔ Implementação e comportamentos similares à ArrayList.
- ➔ Diferenças:
  - ◆ Vector é sincronizado, ou seja, pode ter performance pior que ArrayList, por ser acessado apenas por uma thread de cada vez.
  - ◆ Ao necessitar e utilizar mais espaço, Vector dobra seu tamanho, enquanto ArrayList aumenta em apenas 50%.
  - ◆ Vector permite uso de Enumeração e Iterator. ArrayList utiliza apenas Iterator.

Exemplo Vector

```
Vector<String> vec = new Vector<String>();
vec.addElement("Pessoa 1");
vec.addElement("Pessoa 2");
vec.addElement("Pessoa 3");
Enumeration<String> e = vec.elements();
while (e.hasMoreElements()){
     System.out.println(e.nextElement());
}
```

ExemploVector.java

LinkedList

- ➔ Lista duplamente ligada
- ➔ Inserção e Remoção são mais rápidas do que em ArrayList, enquanto busca é mais demorada.
- ➔ Pode implementar List e Queue

LinkedList

Construir 2 listas de cores e realizar as seguintes operações:

- concatenar lista2 em lista1
- limpar lista2
- converter a lista em caixa alta
- remover 3 elementos da lista
- exibir a lista em ordem inversa

LinkedList

```
private static void exibeLista(List<String> lista, String texto){
     System.out.printf("imprimindo " + texto + ": ");
     for (String cor : lista)
           System.out.printf("%s ", cor);
          System.out.println();
}

private static void converteCaixaAlta(List<String> lista) {
     ListIterator<String> iterator = lista.listIterator();
     while (iterator.hasNext()){
          String cor = iterator.next();
          iterator.set(cor.toUpperCase());
     }
}

private static void removerCores(List<String> lista, int inicio, int fim){
     lista.subList(inicio, fim).clear();
}

private static void exibeListaInvertida(List<String> lista){
     ListIterator<String> iterator = lista.listIterator(lista.size());
     System.out.printf("Lista Invertida: ");
     while (iterator.hasPrevious())
          System.out.printf("%s ", iterator.previous());
}
```

ListasLigadas.java

LinkedList

```
public static void main(String[] args) {
     String[] colors = {"azul", "vermelho", "verde", "amarelo", "cinza", "preto",
"branco"};
     List<String> lista1 = new LinkedList<>();
     for (String cor : colors)
          lista1.add(cor);
     String[] cores2 = {"marrom", "laranja", "roxo", "rosa"};
     List<String> lista2 = new LinkedList<>();

     for (String cor : cores2)
          lista2.add(cor);

     lista1.addAll(lista2);
     lista2 = null;
     exibeLista(lista1, "lista inicial");

     converteCaixaAlta(lista1);
     exibeLista(lista1, "lista em caixa alta");

     System.out.printf("%nRemovendo elementos de 4 a 6…%n");
     removerCores(lista1, 4, 7);
     exibeLista(lista1, "lista com elementos removidos");
     exibeListaInvertida(lista1);
}
```

ListasLigadas.java

Map

- ➔ Maps fazem a associação de chaves/valores
- ➔ As chaves devem ser únicas dentro de um map
- ➔ Valores podem ser repetidos, mas caso tenhamos uma associação única de chaves e valores, podemos dizer que temos um mapeamento um para um, caso contrário temos um mapeamento um para muitos
- ➔ Principais implementações de Map: Hashtable (tabela hash), HashMap (tabela hash, não sincronizado, permite chaves null) e TreeMap (árvores)

Map

```
public static void main(String[] args) {
    Map<String, Integer> notas = new HashMap<>();
    notas.put("Paulo", 5);
    notas.put("José", 8);
    notas.put("Carlos", 7);
    for (String n : notas.keySet()) {
        System.out.print("A nota de " + n + " foi " + notas.get(n)
+ "\n");
    }
    if(notas.containsKey("Paulo")) {
        System.out.print(notas.get("Paulo"));
    };
}
```

Map1.java

Métodos de Coleções

Método	Descrição
sort	Classifica os elementos de uma List.
binarySearch	Localiza um objeto em uma List usando o algoritmo de pesquisa binária.
reverse	Inverte os elementos de uma List.
shuffle	Ordena aleatoriamente os elementos de uma List.
fill	Configura todo elemento List para referir-se a um objeto especificado.
copy	Copia referências de uma List em outra.
min	Retorna o menor elemento em uma Collection.
max	Retorna o maior elemento em uma Collection.
addAll	Acrescenta todos os elementos em um array a uma Collection.
frequency	Calcula quantos elementos da coleção são iguais ao elemento especificado.
disjoint	Determina se duas coleções não têm nenhum elemento em comum.

Exercício

Criar um programa que irá receber 5 números entre 1 e 15 em qualquer ordem e operar as seguintes operações:

- remover números duplicados, se houver;
- escolher os 3 maiores números;
- se eles formarem uma sequência, exibir uma mensagem.