Coleções em Java

Vitor Brandi Junior

Baseado, entre outros, no mini curso “Introduction to the Collections Framework” do Magelan Institute

Roteiro

- O que é uma coleção ?
- O que existia antes do JDK 1.2 para a manipulação de coleções
  - detalhes sobre Vector
  - detalhes sobre Stack
  - detalhes sobre Hashtable
- A Java Collection Framework
  - definições
  - vantagens
  - componentes
  - quem faz o que ?
  - quando usar o que ?
  - exemplos

O que é uma coleção ?

- Uma coleção (também denominada *container*) é simplesmente um objeto que agrupa múltiplos elementos dentro de uma única unidade
- São utilizadas para armazenar, recuperar e manipular dados, além de transmitir dados de um método para outro.
- Tipicamente representam itens de dados que naturalmente formam um grupo. Exemplo:
  - uma mão de baralho (coleção de cartas)
  - um *mail folder* (uma coleção de e-mails)
  - uma agenda telefônica (coleção de nomes e telefones)

O que existia antes da versão 1.2. do JDK

- O suporte às estruturas de dados era feita pelas seguintes classes:
  - `Vector`
  - `Stack`
  - `Hashtable`
  - array
  - `BitSet`

Características de `Vector`

- Um objeto `Vector` é um array ajustável que pode aumentar ou diminuir seu tamanho, dependendo do número de elementos que necessitar armazenar
- O ajuste do tamanho é feito de forma automática
- Provê (*um monte de*) métodos que permitem a inserção, remoção e busca de elementos, a saber:
  - `add, add, addAll, addAll, addElement, capacity, clear, clone, contains, containsAll, copyInto, elementAt, elements, ensureCapacity, equals, firstElement, get, hashCode, indexOf, indexOf, insertElementAt, isEmpty, lastElement, lastIndexOf, lastIndexOf, remove, remove, removeAll, removeAllElements, removeElement, removeElementAt, removeRange, retainAll, set, setElementAt, setSize, size, subList, toArray, toArray, toString, trimToSize`

Exemplo de `Vector`

```
import java.util.*;
class Empregado {
  String nome;
  String fone;
  int prontuario;
  Empregado(String nome, String fone, int prontuario) {
    this.nome = new String(nome);
    this.fone = new String(fone);
    this.prontuario = prontuario;
  }
  public String toString() {
    return("Nome: " + nome + "\nfone: " + fone + "\nprontuario: " + prontuario);
  }
}
public class ExemploVector1 {
  public static void main(String args[]) {
    Vector ListaEmpregados = new Vector();
    ListaEmpregados.addElement(new Empregado("Jose Luis Zem","9821234",4444));
    ListaEmpregados.addElement(new Empregado("Bianca Pedrosa","4220000",1234));
    ListaEmpregados.addElement(new Empregado("Vitor Brandi Junior","9861234",4321));
    for(int i = 0; i < ListaEmpregados.size(); i++) {
      Empregado atual =(Empregado) ListaEmpregados.elementAt(i);
      System.out.println("Empregado " + i + ":");
      System.out.println(atual);
    }
  }
}
```

Mais um exemplo de `Vector`

```
import java.util.*;
public class ExemploVector2
{
  public static void main(String args[])
  {
    int arrayDeInteiros[] = { 1, 3, 2, 5, 7, 0 };
    Vector listaDeInteiros = new Vector();
    for(int i = 0; i < arrayDeInteiros.length; i++)
      listaDeInteiros.addElement(new Integer(arrayDeInteiros[i]));
    for (int i = 0; i < listaDeInteiros.size(); i++)
    {
      int valor =((Integer) listaDeInteiros.elementAt(i)).intValue();
      System.out.println("Elemento " + i + ": " + valor);
    }
  }
}
```

Objetos da classe Vector somente podem armazenar quaisquer outros tipos de objetos. No entanto, não podem armazenar tipos primitivos.

Quando isto for necessário, deve-se utilizar classes denominadas *Wrappers* (para o tipo `int`, o correspondente é a classe `Integer`.

Pequeno exercício com `Vector`

- Sabendo para que servem os métodos abaixo (todos da classe `Vector`), solicita-se a complementação do programa Java denominado ***ExemploVector3.java***
  - `void insertElementAt (Object obj, int indice)`: insere um novo objeto em uma posição específica dentro do `Vector`
  - `boolean removeElement (Object obj)`: remove o objeto especificado do `Vector` (retorna `true` se foi bem sucedido)
  - `void removeElementAt (int indice)`: remove um elemento da posição especificada por `indice`
  - `void removeAllElements()`: auto explicativo
  - `boolean contains (Object obj)`: retorna `true` se o objeto está contido no `Vector`
  - `int indexOf (Object obj)`: retorna o índice onde se encontra `obj` dentro do `Vector` ou **-1** caso ele não exista

ExemploVector3.java

```
public class ExemploVector3
{
  public static void main(String args[])
  {
    Vector ListaEmpregados = new Vector();
    ListaEmpregados.addElement(new Empregado("Jose Luis Zem","9821234",4444));
    ListaEmpregados.addElement(new Empregado("Bianca Pedrosa","4220000",1234));
    ListaEmpregados.addElement(new Empregado("Vitor Brandi Junior","9861234",4321));
/*  pseudocódigo
    criar novo objeto empregado da classe Empregado
    se (empregado nao existe) então
      inserir empregado no final de ListaEmpregados
    senão
      imprimir o indice de ListaEmpregados onde se encontra o empregado já armazenado
      remover o empregado que já se encontra armazenado no vetor
      inserir o novo empregado exatamente no mesmo índice do empregado removido
      imprimir todos os componentes do vetor
    fim do pseudocódigo
*/
  }
}
```

Características de `Stack`

- é subclasse da classe `Vector`
- implementa cinco novos métodos que permitem utilizar o `Vector` para armazenar uma pilha:
  - `push`: insere elementos no topo da pilha
  - `pop`: remove elementos do topo da pilha
  - `peek`: retorna o objeto que se encontra no topo, sem retirá-lo da pilha
  - `empty`: retorna true se a pilha está vazia
  - `search`: retorna o índice onde se encontra um objeto dentro da pilha ou **-1** caso ele não esteja presente

Exemplo do uso de `Stack`

```
import java.util.*;
public class ExemploStack1 {
  public static void main(String args[]) {
    String s;
    Stack pilha = new Stack();
    pilha.push(new String("Um"));
    System.out.println("Quem estah no topo da pilha = "+ (String) pilha.peek());
    pilha.push(new String("Dois"));
    System.out.println("Quem estah no topo da pilha = "+ (String) pilha.peek());
    pilha.push(new String("Tres"));
    System.out.println("Quem estah no topo da pilha = "+ (String) pilha.peek());
    int existe = pilha.search("Dois");
    if (existe != -1)
      System.out.println("Dois esta na posicao " + existe + " da pilha");
    s =(String) pilha.pop();
    System.out.println("Elemento retirado da pilha = " + s);
    s =(String) pilha.pop();
    System.out.println("Elemento retirado da pilha = " + s);
    pilha.push(new String("Quatro"));
    System.out.println("Quem estah no topo da pilha = "+ (String) pilha.peek());
    s =(String) pilha.pop();
    System.out.println("Elemento retirado da pilha = " + s);
    s =(String) pilha.pop();
    System.out.println("Elemento retirado da pilha = " + s);
    s =(String) pilha.pop();
    // favor consertar o erro de runtime que acontece depois da linha anterior
  }
}
```

Características de `Hashtable`

- Como você se lembra (:)), uma tabela Hash é uma estrutura de dados que permite procurar itens armazenados utilizando uma chave associada
- A chave pode ser de qualquer tipo
- Obrigatoriamente a chave tem de ser única (não podem haver repetições da chave dentro do mesmo objeto `Hashtable`)

Exemplo de Hashtable

```
import java.util.*;
class Estudante {
  String nome;
  String RA;
  float media;
  Estudante(String nome, String RA, float media) { // construtora
    this.nome = new String(nome);
    this.RA = new String(RA);
    this.media = media;
  }
  public String toString(){
    return("Nome: " + nome + "\nRA: " + RA +"\nMedia.: " + media);
  }
}
public class TesteHash {
  public static void main(String args[]) {
    Hashtable TabelaEstudantes = new Hashtable();
    TabelaEstudantes.put("9716093",new Estudante("Andre Barnabe", "9716093", 8.0f));
    TabelaEstudantes.put("9717000",new Estudante("Beatriz Amaro", "9717000", 7.5f));
    TabelaEstudantes.put("9716127",new Estudante("Tales Raduan","9716127", 6.5f));
    TabelaEstudantes.put("8711970",new Estudante("Vitor Brandi","8711970", 3.5f));
    Estudante estudante = (Estudante) TabelaEstudantes.get("8711970");
    System.out.println("Aluno: ");
    System.out.println(estudante);
    Estudante estudante1 = (Estudante) TabelaEstudantes.get("8611970");
    System.out.print("Aluno: ");
    System.out.println(estudante1);
  }
}
```

Mais um exemplo de Hashtable

```
import java.util.*;
class Estudante {
  String nome;
  String RA;
  float media;
  Estudante(String nome, String RA, float media) {
    this.nome = new String(nome);
    this.RA = new String(RA);
    this.media = media;
  }
  public String toString(){
    return("Nome: " + nome + "\nRA: " + RA +"\nMedia.: " + media);
  }
}
public class TesteHash1 {
  public static void main(String args[]) {
    Hashtable TabelaEstudantes = new Hashtable();
    TabelaEstudantes.put("9716093",new Estudante("Andre Barnabe", "9716093", 8.0f));
    TabelaEstudantes.put("9717000",new Estudante("Beatriz Amaro", "9717000", 7.5f));
    TabelaEstudantes.put("9716127",new Estudante("Tales Raduan","9716127", 6.5f));
    TabelaEstudantes.put("8711970",new Estudante("Vitor Brandi","8711970", 3.5f));
    Estudante estudante;
    Enumeration enum = TabelaEstudantes.elements();
    while (enum.hasMoreElements()) {
      estudante = (Estudante) enum.nextElement();
      System.out.println(estudante.nome);
    }
  }
}
```

Pequeno exercício sobre Hashtable

- Construa com o auxílio do JBuilder um aplicativo que simule o funcionamento de um *Draft Folder,* executando as seguintes tarefas:
  - **permita ler uma séria de dados (destinatário, assunto, texto) relacionados a uma série de e-mails**
  - **armazene cada um destes e-mails em uma Hashtable**
  - **permita a listagem de todos os e-mails armazenados na Hashtable**
  - **permita a recuperação, apresentação (em uma TextArea) e posterior exclusão de cada um dos e-mails armazenados na Hashtable**
  - **permita o completo esvaziamento da Hashtable**

Java Collection Framework

- Com o objetivo de permitir a manipulação mais eficiente de estruturas de dados, a linguagem Java traz implementada, a partir da versão 1.2, a ***Java Collection Framework*** (ou uma coleção de interfaces e classes que implementam alguns dos algoritmos e estruturas de dados mais comuns)
- ela se assemelha à biblioteca para manipulação de coleções do C++ (*STL - Standard Template Library*)

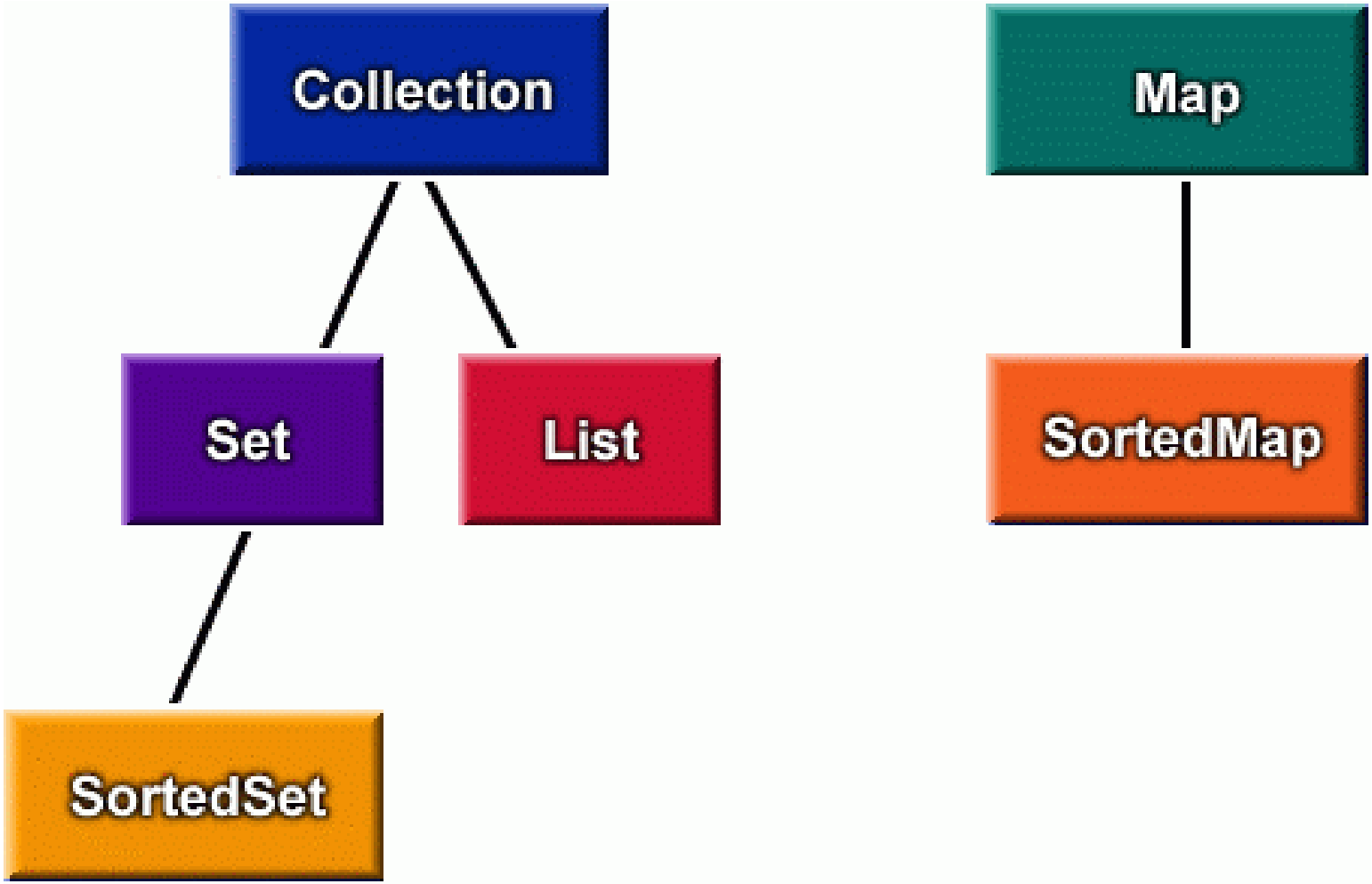
O que é a Java Collection Framework ?

- É uma arquitetura unificada para a representação e manipulação de coleções. Contém três diferentes coisas:
  - **Interfaces**: tipos abstratos de dados (ADTs) representando coleções e que permitem que elas sejam manipuladas independentemente dos detalhes de suas representações.
  - **Implementações**: classes concretas que implementam as interfaces.
  - **Algoritmos**: métodos que executam a manipulação (pesquisa e ordenação, p. ex.) em objetos instanciados das implementações.

Quais as vantagens da Java Collection Framework ? (segundo a Sun)

- Reduzem o esforço de programação
- aumentam a velocidade e a qualidade do programa
- permitem a interoperabilidade entre APIs não relacionadas
- reduzem o esforço para aprender e utilizar novas APIs:
- reduzem o esforço para se projetar novas APIs
- estimulam o reuso de software

Hierarquia das Interfaces da *Java Collection Framework*



A *Java Collection Framework*

Área Geral	Interfaces / Classes Abstratas	Classes Concretas
Coleções	`Collection`	acesso básico e funções de atualização
	`Set`	• `HashSet` (conjunto de valores armazenados em uma Hash Table) • `TreeSet` (conjunto ordenado de valores)
	`List`	• `ArrayList` (pode substituir a classe `Vector` a partir do JDK 1.2) • `LinkedList`
	`Map`	• `HashMap` (pode substituir a classe `Hashtable` a partir do JDK 1.2) • `TreeMap` (um *map* ordenado) • `WeakHashMap` (tabela cujas entradas são eliminadas sempre que a coisa a qual ela se refere é eliminada da memória (através da coleta automática de lixo)
	Outros	• `Stack` • `Array` • `BitSet` • `Iterator` (substitui a classe `Enumerator`)

Quem faz o que ?

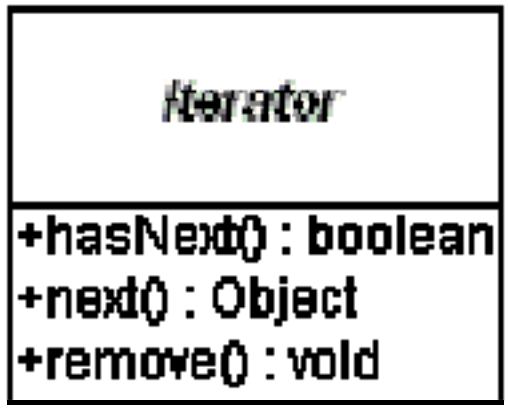
- **Collection**:
  - representa um grupo de objetos denominados elementos
  - é apenas uma interface que define métodos para se adicionar, remover e pesquisar em uma estrutura de dados
  - o JDK não provê implementação direta desta Interface.
- **Interface Set**:
  - coleção que não permite elementos duplicados
  - um set não pode conter um par e1, e2, tal que e1.equals(e2)
- **Hashset extends AbstractSet implements Set:**
  - mantém uma coleção de objetos nos quais se pode aplicar operações de intersecção, diferença e iteração
- **TreeSet:**
  - apresenta a característica de armazenar os elementos em ordem crescente

Quem faz o que ? **(cont)**

- **List**
  - é uma coleção que traz uma ordem associada aos seus elementos
  - os elementos podem ser acessados pelos seus respectivos índices dentro da lista
  - permitem elementos duplicados
- **ArrayList extends AsbtractList implements List**
  - provê métodos para se manipular o tamanho do array utilizado para armazenar a lista
  - cada instância de ArrayList possui uma capacidade (10 elementos por default)
  - permite todos os elementos (inclusive elementos *null*)
- **LinkedList implements List**
  - provê métodos para se manipular elementos nas extremidades da lista

Quem faz o que ? **(cont)**

- **Map**
  - permite armazenar pares chave/valor (similar a uma tabela de 2 colunas)
  - dada a chave, permite recuperar o valor associado
  - não permite chaves duplicadas
- **HashMap extends AbstractMap implements Map**
  - permite o armazenamento de *null* para valores e chaves
- **Stack**
  - é a mesma implementação do JDK 1.1
- **Iterator**
  - permite a recuperação de todos os elementos da estrutura de dados sem se preocupar em como estes elementos estão armazenados



Quando usar o que ?

- Se a coleção não apresenta elementos duplicados e se deseja que os elementos estejam em ordem: `TreeSet`
- se existem entradas duplicadas: qualquer implementação de `List`
- se a coleção é composta por pares chave/valor: qualquer implementação de `Map`

Exemplo de `Set` e `HashSet`

```
import java.util.*;
public class Duplicadas
{
  public static void main(String args[])
  {
    Set s = new HashSet();
    for (int i=0; i<args.length; i++)
      if (!s.add(args[i]))
        System.out.println("Palavra duplicada: "+args[i]);
    System.out.println(s.size()+" palavras distintas
                        encontradas: "+s);
  }
}
```

Outro exemplo de `Set` e `HashSet`

```
import java.util.*;
public class Duplicadas2
{
  public static void main(String args[])
  {
      Set unicas = new HashSet();
    Set duplicadas = new HashSet();
    for (int i=0; i<args.length; i++)
      if (!unicas.add(args[i]))
        duplicadas.add(args[i]);
    unicas.removeAll(duplicadas);
    System.out.println("Palavras unicas.....: " + unicas);
    System.out.println("Palavras duplicadas : " + duplicadas);
  }
}
```

Exemplo de `List`

```
import java.util.*;
public class Mistura
{
  public static void main(String args[])
  {
    List l = new ArrayList();
    for (int i=0; i<args.length; i++)
      l.add(args[i]);
    Collections.shuffle(l, new Random());
    System.out.println(l);
  }
}
```

Baralho.java

```
import java.util.*;
class Baralho
{
  public static void main(String args[])
  {
    int jogadores = Integer.parseInt(args[0]);
    int cartasPorJogador = Integer.parseInt(args[1]);
    String[] naipes = new String[] {"espadas", "copas", "ouros", "paus"};
    String[] cartas = new String[]
            {"as","2","3","4","5","6","7","8","9","10","valete","dama","rei"};
    List monte = new ArrayList();
    for (int i=0; i<naipes.length; i++)
      for (int j=0; j<cartas.length; j++)
        monte.add(cartas[j] + " de " + naipes[i]);
    Collections.shuffle(monte);
    for (int i=0; i<jogadores; i++)
      System.out.println(darCartas(monte, cartasPorJogador));
  }
  public static List darCartas(List monte, int n)
  {
    int tamanhoMonte = monte.size();
    List topoDoMonte = monte.subList(tamanhoMonte-n, tamanhoMonte);
    List mao = new ArrayList(topoDoMonte);
    topoDoMonte.clear();
    return mao;
  }
}
```

Exemplo de Tree/HashMap

```
import java.util.*;
public class Freq
{
  private static final Integer UM = new Integer(1);
  public static void main(String args[])
  {
    Map m = new TreeMap();
    for (int i=0; i<args.length; i++)
    {
      Integer freq = (Integer) m.get(args[i]);
      m.put(args[i],(freq==null ? UM : new Integer(freq.intValue()+1)));
    }
    System.out.println(m.size() + " palavras distintas encontradas:");
    System.out.println(m);
  }
}
```

Exemplo: HashSet e TreeSet

```
import java.util.*;
public class ExemploDeSet
{
  public static void main(String args[])
  {
    Set set = new HashSet();
    set.add("Dida");
    set.add("Kleber");
    set.add("Adilson");
    set.add("Fabio Luciano");
    set.add("Daniel");
    Set setOrdenado = new TreeSet(set);
    System.out.println(setOrdenado);
  }
}
```

Exemplo: ArrayList e LinkedList

```
import java.util.*;
public class ListExample {
  public static void main(String args[]) {
    List list = new ArrayList();
    list.add("Bernadine");
    list.add("Elizabeth");
    list.add("Gene");
    list.add("Elizabeth");
    list.add("Clara");
    System.out.println(list);
    System.out.println("2: " + list.get(2));
    System.out.println("0: " + list.get(0));
    LinkedList queue = new LinkedList();
    queue.addFirst("Bernadine");
    queue.addFirst("Elizabeth");
    queue.addFirst("Gene");
    queue.addFirst("Elizabeth");
    queue.addFirst("Clara");
    System.out.println(queue);
    queue.removeLast();
    queue.removeLast();
    System.out.println(queue);
  }
}
```