Programação 2

Coleções genéricas

Prof. Domingo Santos

domingos.santos@upe.br

Resumo

Collection:

- Listas
- Métodos de coleções
- organizações de dados:
  - set
  - list
  - map
  - fila

Conteúdo baseado no capítulo 16 do livro de Deitel

UNIVERSIDADE DE PERNAMBUCO

Visão geral das coleções

- framework collection do Java, que contém muitas outras estruturas de dados genéricos predefinidas

- exemplos de coleções são:
  - suas músicas preferidas armazenadas no celular
  - lista de contatos
  - membros do seu time favorito

- Uma coleção é uma estrutura de dados, que pode armazenar referências a outros objetos

Visão geral das coleções

- Exemplos

Interface	Descrição
Collection	A interface-raiz na hierarquia de coleções a partir da qual as interfaces Set, Queue e List são derivadas
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
Fila	Em geral, uma coleção primeiro a entrar, primeiro a sair que modela uma fila de espera

UNIVERSIDADE DE PERNAMBUCO

Classes empacotadoras de tipo

Boolean, Byte, Character, Double, Float, Integer, Long e Short

Estruturas de dados que serão estudadas não podem manipular variáveis de tipos primitivos, apenas objetos

O Java fornece conversão boxing e unboxing que convertem automáticamente entre valores de tipo primitivo e objetos empacotadores de tipo:

- Uma conversão **boxing** converte um valor de um tipo primitivo em um objeto da classe empacotadora de tipo correspondente.

- Uma conversão **unboxing** converte um objeto de uma classe empacotadora de tipo em um valor do tipo primitivo correspondente.

```
Integer[] integerArray = new Integer[5]; // cria integerArray
integerArray[0] = 10; // atribui Integer 10 a integerArray[0]
int value = integerArray[0]; // obtém valor int de Integer
```

Interface Collection e classe Collections

Operações de volume (isto é, operações realizadas na coleção inteira) para operações como adicionar, limpar e comparar objetos (ou elementos) em uma coleção.

Uma Collection também poder ser convertida em um array

Fornece um método que retorna um objeto Iterator, que permite a um programa percorrer a coleção e remover elementos da coleção durante a iteração

Permitem a um programa determinar o tamanho de uma coleção e se uma coleção está ou não vazia

Fornece métodos static que pesquisam, classificam e realizam outras operações sobre as coleções

Listas

É uma Collection ordenada que pode conter elementos duplicados

Além dos métodos herdados de Collection, List fornece métodos para manipular elementos por meio de seus índices, manipular um intervalo especificado de elementos, procurar elementos e obter um ListIterator para acessar os elementos.

A interface List é implementada por várias classes, inclusive as classes ArrayList e LinkedList

Inserir um elemento entre os elementos existentes de um **ArrayList** é uma **operação ineficiente**, todos os elementos depois do novo devem ser removidos, o que pode ser uma operação cara em uma coleção com um grande número de elementos.

Uma **LinkedList** permite a inserção (ou remoção) **eficiente dos elementos no meio de uma coleção**, mas é muito **menos eficiente** que um ArrayList para pular **para um elemento específico na coleção**.

ArrayList e Iterator

O programa coloca dois arrays Color em ArrayLists e utiliza um Iterator para remover elementos na segunda coleção ArrayList

Exemplo ArrayList e Iterator pt 1

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
public class CollectionTest {
   . . .
   private static void removeColors(Collection<String> collection1, Collection<String> collection2) {
       // obtém o iterador
       Iterator<String> iterator = collection1.iterator();

       while (iterator.hasNext()) {// loop enquanto a coleção tiver itens
           if (collection2.contains(iterator.next())) {
               iterator.remove(); // remove o elemento atual
           }
       }
   }
}
```

Listas

Exemplo ArrayList e Iterator pt 2

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

           String[] colors = {"MAGENTA", "RED", "WHITE", "BLUE", "CYAN"};  // adiciona elementos no array colors a listar
           List<String> list = new ArrayList<String>();
           for (String color : colors) {
               list.add(color); // adiciona color ao final da lista
           }

           String[] removeColors = {"RED", "WHITE", "BLUE"}; // adiciona elementos no array removeColors em removeList
           List<String> removeList = new ArrayList<String>();

           for (String color : removeColors) {
               removeList.add(color);
           }

           System.out.println("ArrayList: ");  // gera saída do conteúdo da lista
           for (int count = 0; count < list.size(); count++) {
               System.out.printf("%s ", list.get(count));
           }

           removeColors(list, removeList);// remove da lista as cores contidas em removeList

           System.out.printf("%n%nArrayList after calling removeColors:%n");// gera saída do conteúdo da lista
           for (String color : list) {
               System.out.printf("%s ", color);
           }
       }
   . . .
}
```

ArrayList:
MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:
MAGENTA CYAN

Listas

Visualizando arrays como Lists e convertendo Lists em arrays

```
import java.util.LinkedList;
import java.util.Arrays;

public class UsingToArray {
   public static void main(String[] args) {
       String[] colors = {"black", "blue", "yellow"};
       LinkedList<String> links = new LinkedList<>(Arrays.asList(colors));

       links.addLast("red"); // adiciona como o último item
       links.add("pink"); // adiciona ao final
       links.add(3, "green"); // adiciona no terceiro índice
       links.addFirst("cyan"); // adiciona como primeiro item

       colors = links.toArray(new String[links.size()]); // obtém elementos LinkedList como um array

       System.out.println("colors: ");
       for (String color : colors) {
           System.out.println(color);
       }
   }
}
```

```
colors:
cyan
black
blue
yellow
green
red
pink
```

UNIVERSIDADE DE PERNAMBUCO

Métodos de coleções

A classe Collections fornece vários algoritmos de alto desempenho para manipular elementos de coleção

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.
frequency	Calcula quantos elementos da coleção são iguais ao elemento especificado.
disjoint	Determina se duas coleções não têm nenhum elemento em comum.

UNIVERSIDADE DE PERNAMBUCO

Métodos de coleções

Método sort

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Sort1 {
   public static void main(String[] args) {
       String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};

       List<String> list = Arrays.asList(suits);

       System.out.printf("Unsorted array elements: %s%n", list);
       Collections.sort(list); // classifica ArrayList
       System.out.printf("Sorted array elements: %s%n", list);
   }
}
```

Unsorted array elements: [Hearts, Diamonds, Clubs, Spades]
Sorted array elements: [Clubs, Diamonds, Hearts, Spades]

Métodos de coleções

Classificando em ordem decrescente

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Sort2 {
   public static void main(String[] args) {
       String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};
       // Cria e exibe uma lista contendo os elementos do array naipes
       List<String> list = Arrays.asList(suits); // cria List
       System.out.printf("Unsorted array elements: %s%n", list);
       // classifica em ordem decrescente utilizando um comparador
       Collections.sort(list, Collections.reverseOrder());
       System.out.printf("Sorted list elements: %s%n", list);
   }
}
```

Unsorted array elements: [Hearts, Diamonds, Clubs, Spades]
Sorted list elements: [Spades, Hearts, Diamonds, Clubs

Métodos de coleções

Métodos reverse, fill, copy, max e min pt 1

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Algorithms1 {
   . . .
   private static void output(List<Character> listRef) {// envia informações de List para saída

       System.out.print("The list is: ");
       for (Character element : listRef) {
           System.out.printf("%s ", element);
       }
       System.out.printf("%nMax: %s", Collections.max(listRef));
       System.out.printf(" Min: %s%n", Collections.min(listRef));
   }
}
```

Métodos de coleções

Métodos reverse, fill, copy, max e min pt 2

```
public class Algorithms1 {

   public static void main(String[] args) {

      Character[] letters = {'P', 'C', 'M'}; // crie e exibe uma List<Character>
      List<Character> list = Arrays.asList(letters); // obtém List
      System.out.println("list contains: ");
      output(list);

      Collections.reverse(list); // inverte a ordem dos elementos
      System.out.printf("%nAfter calling reverse, list contains:%n");
      output(list);
      // cria CopyList de um array de 3 caracteres
      Character[] lettersCopy = new Character[3];
      List<Character> copyList = Arrays.asList(lettersCopy); // copia o conteúdo da lista para copyList

      Collections.copy(copyList, list);
      System.out.printf("%nAfter copying, copyList contains:%n");
      output(copyList);

      Collections.fill(list, 'R');// preenche a lista com Rs
      System.out.printf("%nAfter calling fill, list contains:%n");
      output(list);

   }
}
```

```
list contains:
The list is: P C M
Max: P Min: C

After calling reverse, list contains:
The list is: M C P
Max: P Min: C

After copying, copyList contains:
The list is: M C P
Max: P Min: C

After calling fill, list contains:
The list is: R R R
Max: R Min: R
```

Métodos de coleções

Método binarySearch pt 1

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.ArrayList;

public class BinarySearchTest {
   . . .
   private static void printSearchResults(List<String> list, String key) {
       int result = 0;
       System.out.printf("%nSearching for: %s%n", key);
       result = Collections.binarySearch(list, key);

       if (result >= 0) {
           System.out.printf("Found at index %d%n", result);
       } else {
           System.out.printf("Not Found (%d)%n", result);
       }
   }
}
```

Métodos de coleções

Método binarySearch pt 2

```
public class BinarySearchTest {

   public static void main(String[] args) {
       String[] colors = {"red", "white", "blue", "black", "yellow",
           "purple", "tan", "pink"};
       List<String> list
               = new ArrayList<>(Arrays.asList(colors));
       Collections.sort(list); // classifica a ArrayList
       System.out.printf("Sorted ArrayList: %s%n", list);

       printSearchResults(list, "black");
       printSearchResults(list, "red");
       printSearchResults(list, "pink");
       printSearchResults(list, "aqua");
       printSearchResults(list, "gray");
       printSearchResults(list, "teal");
   }
  . . .
}
```

Sorted ArrayList: [black, blue, pink, purple, red, tan, white, yellow]

Searching for: black
Found at index 0

Searching for: red
Found at index 4

Searching for: pink
Found at index 2

Searching for: aqua
Not Found (-1)

Searching for: gray
Not Found (-3)

Searching for: teal
Not Found (-7)

Métodos de coleções

Métodos addAll, frequency e disjoint

```
import java.util.ArrayList; import java.util.List; import java.util.Arrays; import java.util.Collections;
public class Algorithms2 {

   public static void main(String[] args) {

      String[] colors = {"red", "white", "yellow", "blue"}; // inicializa list1 e list2
      List<String> list1 = Arrays.asList(colors);
      ArrayList<String> list2 = new ArrayList<>();
      list2.add("black"); // adiciona "black" ao final da list2
      list2.add("red"); // adiciona "red" ao final da list2
      list2.add("green"); // adiciona "green" ao final da list2
      System.out.print("Before addAll, list2 contains: ");

      for (String s : list2) { // exibe os elementos em list2
         System.out.printf("%s ", s);
      }
      Collections.addAll(list2, colors); // adiciona Strings colors à list2
      System.out.printf("%nAfter addAll, list2 contains: ");

      for (String s : list2) {// exibe os elementos em list2
         System.out.printf("%s ", s);
      }

      int frequency = Collections.frequency(list2, "red"); // obtém frequência de "red"
      System.out.printf("%nFrequency of red in list2: %d%n", frequency);

      boolean disjoint = Collections.disjoint(list1, list2);
      System.out.printf("list1 and list2 %s elements in common%n", (disjoint ? "do not have" : "have"));
   }
}
```

Before addAll, list2 contains: black red green

After addAll, list2 contains: black red green red white yellow blue

Frequency of red in list2: 2

list1 and list2 have elements in common

Classe PriorityQueue e interface Queue

Representa uma fila de espera

Normalmente, inserções são feitas na parte de trás de uma fila e exclusões são feitas a partir da frente

operações PriorityQueue comuns:

- **offer**, para inserir um elemento na posição apropriada com base na ordem de prioridade
- **poll,** para remover o elemento de mais alta prioridade da fila de prioridade (isto é, a cabeça da fila)
- **peek,** para obter uma referência ao elemento de mais alta prioridade da fila de prioridade (sem remover esse elemento)
- **clear,** para remover todos os elementos da fila de prioridade
- **size**, para obter o número de elementos da fila de prioridade

Classe PriorityQueue e interface Queue

Exemplo

```
import java.util.PriorityQueue;

public class PriorityQueueTest {

   public static void main(String[] args) {

      PriorityQueue<Double> queue = new PriorityQueue<>();
      queue.offer(3.2);// insere elementos na fila
      queue.offer(9.8);
      queue.offer(11.6);
      queue.offer(5.4);

      System.out.print("Polling from queue: ");

      while (!queue.isEmpty()) { // exibe elementos na fila
          System.out.printf("%.1f ", queue.peek()); // visualiza o elemento superior
          queue.poll(); // remove o elemento superior
      }
   }
}
```

Polling from queue: 3,2 5,4 9,8 11,6

Conjuntos

Set é uma Collection não ordenada de elementos únicos:

- **HashSet** armazena seus elementos em uma tabela de hash
- **TreeSet** armazena seus elementos em uma árvore.
- Interface **SortedSet** (que estende Set) para conjuntos que mantêm seus elementos em ordem classificada. A classe TreeSet implementa SortedSet

Conjuntos

Exemplo :

```
import java.util.List;import java.util.Arrays;import java.util.HashSet;import java.util.Set;import
java.util.Collection;

public class SetTest {

   public static void main(String[] args) {

      String[] colors = {"red", "white", "blue", "green", "gray",
          "orange", "tan", "white", "cyan", "peach", "gray", "orange"};
      List<String> list = Arrays.asList(colors);
      System.out.printf("List: %s%n", list);

      printNonDuplicates(list); // elimina duplicatas, então imprime os valores únicos
   }

   private static void printNonDuplicates(Collection<String> values) {
      Set<String> set = new HashSet<>(values);
      System.out.printf("%nNonduplicates are: " );
      for (String value : set) {
         System.out.printf("%s ", value);
      }
      System.out.println();
   }
}
```

List: [red, white, blue, green, gray, orange, tan, white, cyan, peach, gray, orange]

Nonduplicates are: red orange tan green gray white blue peach cyan

Conjuntos

Exemplo :

```
import java.util.Arrays; import java.util.SortedSet; import java.util.TreeSet;

public class SortedSetTest {

   public static void main(String[] args) {
       String[] colors = {"yellow", "green", "black", "tan", "grey", "white", "orange", "red", "green"};
       SortedSet<String> tree = new TreeSet<>(Arrays.asList(colors));

       System.out.print("sorted set: ");
       printSet(tree);  // obtém headSet com base em "orange"
       System.out.print("headSet (\"orange\"): ");
       printSet(tree.headSet("orange")); // obtém tailSet baseado em "orange"

       System.out.print("tailSet (\"orange\"):");
       printSet(tree.tailSet("orange")); // obtém primeiro e último elementos

       System.out.printf("first: %s%n", tree.first());
       System.out.printf("last : %s%n", tree.last());
   }
   private static void printSet(SortedSet<String> set) {
       for (String s : set) {
           System.out.printf("%s ", s);
       }
       System.out.println();
   }
}
```

sorted set: black green grey orange red tan white yellow

headSet ("orange"): black green grey

tailSet ("orange"):orange red tan white yellow

first: black
last : yellow

Mapas

Maps associam chaves a valores

As chaves em um Map devem ser únicas, mas os valores associados não precisam ser

Maps diferem de Sets pelo fato de que Maps contêm chaves e valores, enquanto Sets contêm somente valores.

Três das várias classes que implementam a interface Map são **Hashtable, HashMap e TreeMap**. Hashtables e HashMaps armazenam elementos em tabelas de hash e TreeMaps armazenam elementos em árvores

A interface **SortedMap** estende Map e mantém suas chaves em ordem classificada

Mapas

Exemplo contar caracteres em uma palavra pt 1

```
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;

public class WordTypeCount {

   public static void main(String[] args) { // cria HashMap para armazenar chaves de Strings e valores Integer
       Map<String, Integer> myMap = new HashMap<>();

       createMap(myMap); // cria mapa com base na entrada do usuário
       displayMap(myMap); // exibe o conteúdo do mapa
   }
 . . .
}
```

Mapas

Exemplo contar caracteres em uma palavra pt 2

```
public class WordTypeCount {
   . . .
   private static void createMap(Map<String, Integer> map) {
       Scanner scanner = new Scanner(System.in); // cria o scanner
       System.out.println("Enter a string:"); // solicita a entrada do usuário
       String input = scanner.nextLine(); // tokeniza a entrada

       String[] tokens = input.split(" "); // processamento de texto de entrada

       for (String token : tokens) {
           String word = token.toLowerCase(); // obtém a palavra em letras minúsculas

           if (map.containsKey(word)) // a palavra está no mapa
           {
               int count = map.get(word); // obtém a contagem atual
               map.put(word, count + 1); // incrementa a contagem
           } else {
               map.put(word, 1); // adiciona nova palavra com uma contagem de 1 para mapa
           }
       }
   }
 . . .
}
```

Mapas

Exemplo contar caracteres em uma palavra pt 2

```
public class WordTypeCount {
   . . .
   private static void displayMap(Map<String, Integer> map) {
       Set<String> keys = map.keySet(); // obtém chaves

       TreeSet<String> sortedKeys = new TreeSet<>(keys); // classifica as chaves
       System.out.printf("%nMap contains:%nKey \t\tValue%n");

       for (String key : sortedKeys) { // gera saída de cada chave no mapa
           System.out.printf("%-10s%10s%n", key, map.get(key));
       }
       System.out.printf("%nsize: %d%nisEmpty: %b%n", map.size(), map.isEmpty());
   }
}
```

Mapas

Exemplo contar caracteres em uma palavra, resultado

Enter a string:
utilizando java com maps , maps é uma estrutura de dados que relaciona chave-valor. No java existem muitas classes baseada em collection

Map contains:

Key	Value
,	1
baseada	1
chave-valor.	1
classes	1
collection	1
com	1
dados	1
de	1
em	1
estrutura	1
existem	1
java	2
maps	2
muitas	1
no	1
que	1
relaciona	1
uma	1
utilizando	1
é	1

size: 20
isEmpty: false

Atividade

Atividade 1: Uso de ArrayList

Crie um programa em Java que utiliza um ArrayList genérico para armazenar uma lista de nomes de estudantes. Adicione cinco nomes ao ArrayList e, em seguida, imprima todos os nomes usando um loop for-each.

Dica use o “add”

Atividade

Atividade 2: Uso de HashMap

Crie um programa em Java que utiliza um HashMap genérico para armazenar as notas de alunos. A chave deve ser o nome do aluno (String) e o valor deve ser a nota (Double). Adicione cinco pares de nomes e notas ao HashMap e, em seguida, imprima todos os pares.

Dica, será necessário usar os métodos “put” e get