Estrutura de dados e Orientação a Objetos

Diferentes tipos de coleções

Programação 2 – Aulas 19 e 20

Objetivos da seção

- Apresentar tipos diferentes de coleções
- Introduzir o framework Collections de Java

Revisão: o que são coleções?

- Objeto que agrupa vários elementos em uma unidade
- Operações básicas:
  - Adicionar elemento
  - Remover elemento
  - Pesquisar elemento
  - Iterar na coleção
- Representam itens que formam um grupo natural
  - Mao de poker

- Pasta de e-mails
- Lista de telefones

— O que é um framework?

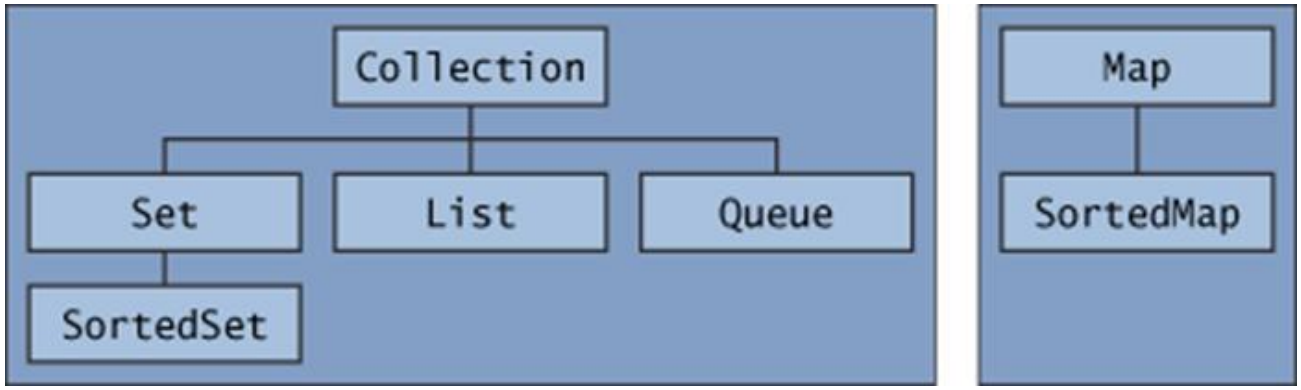
- Um framework captura a funcionalidade comum a várias aplicações
- As aplicações devem ter algo razoavelmente grande em comum: pertencem a um mesmo domínio de problema

— O que é o framework Collections de Java?

- Uma arquitetura unificada para prepresentar e manipular coleções
- Interfaces, implementações e algoritmos

Interfaces

— Duas hierarquias distintas



— Se você entender como usar estas interfaces, você já sabe quase tudo que precisa saber sobre o framework Collections

  ◦ Elas definem o vocabulário relacionado à manipulação de coleções

— Lembre-se: isto não tem a ver apenas com Java. **Estamos a partir falando um pouco de estrutura de dados**

  ◦ Na Ciência da computação, uma estrutura de dados é um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados eficientemente. [veja uma definição aqui]

— Vamos a seguir falar um pouco sobre cada um destes tipos de coleção/mapa

Collection

— Raiz da hierarquia (API)

— Um grupo de objetos conhecidos por elementos

- Um verdadeiro saco de elementos sobre o qual nada sabemos
  - Conseguimos colocar elementos nele, retirar, verificar se um dado elemento existe, se ele está vazio
  - Conseguimos ainda adicionar/retirar coleções completas de dentro dela com uma só operação
- Pode ser transformada em qualquer outro tipo de coleção

Atravessando coleções

- Construção for-each ou iterator
- Veja o exemplo do programa a seguir

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

/**
 * Este programa cria uma Collection, insere elementos nela e
depois caminha
 * nesta coleção usando a construção for-each e um iterador.
 *
 * @author Raquel Lopes
 */
```

```
public class TraversingACollection {

   public static void main(String[] args) {

      // cria uma coleção vazia de funcionários
      Collection<Funcionario> colecao = new
ArrayList<Funcionario>();
      povoaColecao(colecao);
      imprimeTamanhoDaColecao(colecao);

      // Caminhando usando for-each
      for (Funcionario funcionario : colecao) {
         imprimeSalario(funcionario);
      }


System.out.println("=======================================");

      // Pesquisa e remoção
      // Marcus Sampaio vai se aposentar!
      removeFuncionario(colecao, "Marcus Sampaio");

      imprimeTamanhoDaColecao(colecao);

      // Caminhando através de um iterador
      Iterator<Funcionario> it = colecao.iterator();
```

```
        while (it.hasNext()) {
            imprimeSalario(it.next());
        }
    }

    private static void removeFuncionario(Collection<Funcionario>
colecao,
                                          String nomeFuncionario)
{
        for (Funcionario funcionario : colecao) {
            if (funcionario.getNome().equals(nomeFuncionario)) {
                colecao.remove(funcionario);
                break;
            }
        }
    }

    private static void
imprimeTamanhoDaColecao(Collection<Funcionario>
                                                  colecao) {
        System.out.println("Tamanho da colecao e': " +
colecao.size() + "\n");
    }

    private static void imprimeSalario(Funcionario funcionario) {
        System.out.println(funcionario.getNome() + " ganha "
```

```
                        + funcionario.getSalario());
    }

    private static void povoaColecao(Collection<Funcionario>
colecao) {
        try {
            colecao.add(new Funcionario("Raquel Lopes", "3556",
                                        "Dra", 21));
            colecao.add(new Funcionario("Marcus Sampaio", "123",
                                        "Dr", 420));
            colecao.add(new Funcionario("Jacques Sauvé", "1977",
                                        "PhD", 240));
            colecao.add(new Funcionario("Dalton Serey", "2844",
                                        "Dr", 124));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

— A saída deste programa é:

```
Tamanho da colecao e': 4

Raquel Lopes ganha 2408.0
Marcus Sampaio ganha 3240.0
Jacques Sauvé ganha 2880.0
```

```
Dalton Serey ganha 2648.0
==========================================
Tamanho da colecao e': 3

Raquel Lopes ganha 2408.0
Jacques Sauvé ganha 2880.0
Dalton Serey ganha 2648.0
```

Operações com coleções

- Permite adicionar, remover, pesquisar, coleções inteiras e não apenas um único elemento
  - Poderiam ser implementadas usando as operações básicas
  - Chamadas operações *bulk* (em volume?)
- containsAll
- addAll
- removeAll
- retainAll
- clear

Conjuntos (Set)

- Um conjunto é uma coleção que não contém elementos duplicados (API)
  - Modela a abstração matemática de um conjunto
- Tem tudo que a Collection tem, mas adiciona a restrição de que não pode

haver elementos duplicados

- É possível haver várias implementações para esta interface
  - HashSet, TreeSet, LinkedHashSet
  - HashSet tem melhor desempenho, mas não garante a ordem da iteração
  - Cada implementação tem sua própria forma de representar internamente a coleção e sua própria ordem de iteração nos elementos
  - Dois conjuntos são iguais se tiverem os mesmos elementos, mesmo que sua implementação seja diferente. Veja o exemplo a seguir.

```
package p2.exemplos;

import java.util.*;

public class ExemplosDeConjuntos {

   public static void main(String[] args) {

      Set<String> set1 = new HashSet<String>();
      Set<String> set2 = new TreeSet<String>();

      povoaColecao(set1);
      povoaColecao(set2);
```

```
        if (set1.equals(set2)) {
            System.out.println("Os conjuntos são iguais!");
        } else {
            System.out.println("Os conjuntos são diferentes!");
        }

        System.out.println("Quando a colecao eh um conjunto:");
        tentaAdicionarElementoDuplicado(set1);

        Collection<String> col = new ArrayList<String>();
        povoaColecao(col);
        System.out.println("Quando a colecao eh uma lista:");
        tentaAdicionarElementoDuplicado(col);
    }

    private static void
tentaAdicionarElementoDuplicado(Collection<String>
                                                    col) {
        if (!col.isEmpty()) {
            System.out.println("Tamanho do conjunto antes de
adicionar"
                               + "um elemento duplicado: " +
col.size());
            col.add(col.iterator().next());
            System.out.println("Tamanho do conjunto depois de
```

```
adicionar"
                                + "um elemento duplicado: " +
col.size());
        }
    }

    private static void povoaColecao(Collection<String> colecao)
{
        try {
            colecao.add(new String("Raquel Lopes"));
            colecao.add(new String("Jacques Sauvé"));
            colecao.add(new String("Dalton Serey"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

— Vamos rodar este programa?

- Note que os dois conjuntos são iguais, apesar de terem implementações diferentes
- Observe que itens duplicados não são inseridos no conjunto

  i) Ao contrário do que acontece com uma lista

— As operações em volume servem pra realizar operações entre conjuntos

— Veja exemplos:

```
// s1 U s2
Set<Type> union = new HashSet<Type>(s1);
union.addAll(s2);

// s1 ∩ s2
Set<Type> intersection = new HashSet<Type>(s1);
intersection.retainAll(s2);

// s1 - s2
Set<Type> difference = new HashSet<Type>(s1);
difference.removeAll(s2);
```

Listas (List)

— É uma coleção “ordenada” (API)
  - Semelhante a uma implementação de um array
  - Algumas vezes chamada sequencia

— Pode conter elementos duplicados

— Implementações: ArrayList, LinkedList
  - ArrayList: ideal para pesquisa randômica
  - LinkedList: ideal para pesquisa sequencial

— Traz outras operações além das herdadas de Collection. Dentre elas:

- Acesso posicional — é possível manipular os elementos com base em sua posição numérica na lista [**`get(int index)`** e **`set(int index, E element)`**]
- Pesquisa — pesquisa se um determinado elemento está na lista e retorna o índice de sua posição na lista [**`indexOf(Object o)`** e **`lastIndexOf(Object o)`**]
- Oferece visão de sub-lista [**`subList(int from, int to)`**]

ListIterator

— Só existe para listas
— O ListIterator é um iterador mais poderoso porque é bidirecional
— Métodos:
- hasPrevious
- hasNext
- previous
- next
- add (opcional)
- remove (opcional)
- nextIndex
- previousIndex

Filas (Queue)

- Geralmente ordenam os elementos de forma que o primeiro a chegar é o primeiro a sair (API)
- Como uma fila de banco, supermercado, etc.
- LinkedList também implementa a interface Queue
- Alguns métodos:
  - offer e add – adicionam elemento, de acordo com a disciplina da fila, mas podem falhar em caso de fila de tamanho limitado
  - remove e poll – para remover o primeiro elemento da fila (de acordo com sua política de ordenação)
  - element e peek para conhecer o próximo elemento da fila
- **Ver mais detalhes de todos os tipos de coleções sozinhos...**

Mapas (Map)

- Iniciamos o estudo de outra hierarquia de tipos separada de Collections!
- Um mapa armazena pares (chave, valor) chamados itens (API)
  - Chaves e valores podem ser de qualquer tipo, mas devem ser objetos
  - Elemento e Valor são sinônimos
- A chave é utilizada para achar um elemento rapidamente
  - Estruturas especiais são usadas para que a pesquisa seja rápida
- Diz-se portanto que um mapa "mapeia chaves para valores"
  - O mapa pode ser mantido ordenado ou não (com respeito às chaves)
  - Normalmente implementada como "Tabela Hash" ou "Árvore"

- Esses dois tipos de estrutura de dados serão vistos na disciplina Estruturas de Dados

— As operações mais importantes de uma coleção do tipo Mapa são:
  - put – Adicionar um item no mapa (fornecendo chave ou valor)
  - remove – remover um item com chave dada
  - values – acesso aos elementos
  - containsValue – pesquisa de elementos
  - containsKey – descobrir se um elemento com chave dada está na coleção
— Observe que o acesso ao mapa em geral é feito conhecendo a chave
— Você não itera no mapa como um todo, mas no conjunto de chaves ou na coleção de valores
— Em Java a interface Map é implementada por exemplo por HashMap, TreeMap e LinkedHashMap
— Para treinar em casa: escreva o programa Pesquisa2 (da 1ª aula de coleções) usando mapa

Quando você estiver fazendo seus programas...

— Por que é importante conhecer esses diferentes tipos de coleções?
— Dependendo da forma de fazer as 4 operações básicas (adição, remoção, acesso e pesquisa), teremos vários tipos de coleções
  - Certas operações poderão ter um desempenho melhor ou pior dependendo

do tipo de coleção
  - Certas operações poderão ter restrições ou funcionalidade especial dependendo do tipo de coleção

— Para decidir bem, você precisa ter informação adequada

— Quando você for declarar a referência ao objeto, o que você usa? A interface ou a classe que a implementa?
  - Por quê?

Um pouco mais sobre iteradores

— Iteradores são objetos que nos permitem varrer coleções, sem a preocupação com a representação interna da coleção
  - Veja quantas implementações diferentes existem de classes que representam coleções

— É um padrão de projeto que permite ocultação de informação

— Veja exemplo de sua importância no código abaixo:

```
package p2.exemplos;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
```

```
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class CrossContainerIteration {

   private Map<String, String> hmap;
   private List<String> array;
   private Set<String> tset;

   public CrossContainerIteration() {
      hmap = new HashMap<String, String>();
      array = new ArrayList<String>();
      tset = new TreeSet<String>();

      povoaColecoes();
   }

   private void povoaColecoes() {
      String[] conteudo = { "Raquel", "Camila" };
      for (String nome : conteudo) {
         hmap.put(nome, "hmap:" + nome);
         array.add("array:" + nome);
         tset.add("tset:" + nome);
      }
```

```
    }

    public void display(Iterator<String> it) {
        System.out.println("display");
        System.out.print("[ ");
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }
        System.out.println("]");
    }

    public void displayNumLetras(Iterator<String> it) {
        System.out.println("displayNumLetras");
        System.out.print("[ ");
        while (it.hasNext()) {
            System.out.print(it.next().length() + " ");
        }
        System.out.println("]");
    }

    public Map<String, String> getHmap() {
        return hmap;
    }

    public List<String> getArray() {
        return array;
```

```
    }

    public Set<String> getTset() {
        return tset;
    }

    public void imprimeNumLetrasArray() {
        System.out.println("imprimeNumLetrasArray");
        System.out.print("[ ");
        for (int i = 0; i < array.size(); i++) {
            System.out.print(array.get(i).length()+ " ");
        }
        System.out.println("]");
    }

    public void imprimeNumLetrasHmap() {
        System.out.println("imprimeNumLetrasHmap");
        System.out.print("[ ");
        Collection<String> col = hmap.values();
        for (String nome : col) {
            System.out.print(nome.length() + " ");
        }
        System.out.println("]");
    }

    public void imprimeNumLetrasSet() {
```

```
        System.out.println("imprimeNumLetrasSet");
        Object[] col = tset.toArray();
        System.out.print("[ ");
        for (int i = 0; i < col.length; i++) {
           System.out.print(((String)col[i]).length() + " ");
        }
        System.out.println("]");
    }

    public static void main(String[] args) {
        CrossContainerIteration cross = new
CrossContainerIteration();
        cross.display(cross.getArray().iterator());
        cross.display(cross.getHmap().values().iterator());
        cross.display(cross.getTset().iterator());

        cross.imprimeNumLetrasArray();
        cross.imprimeNumLetrasHmap();
        cross.imprimeNumLetrasSet();

        cross.displayNumLetras(cross.getArray().iterator());

cross.displayNumLetras(cross.getHmap().values().iterator());
        cross.displayNumLetras(cross.getTset().iterator());
    }
}
```

Algoritmos

- Algoritmos na classe Collections são usados para manipular objetos do tipo Collection
  - São todos métodos estáticos que recebem ou retornam objetos do tipo Collection. Veja exemplos de algoritmos desta classe:
    i. sort — ordena uma lista usando algoritmo merge sort
    ii. shuffle — muda a posição dos elementos da lista randomicamente
    iii. reverse — inverte a ordem dos elementos em uma lista.
    iv. rotate — rotaciona todos os elementos de uma lista de uma distância especificada
    v. swap — troca um elemento pelo outro na lista
    vi. replaceAll — substitui todas as ocorrências de um elemento por outro
    vii. copy — copia a lista fonte numa lista destino
    viii. binarySearch — pesquisa por um elemento em uma lista ordenadausando o algoritmo de pesquisa binaria