Programação Java
Collections

Profª. Heloisa Moura

Programação Java
Collections

Collections Framework é um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade, que pode ser chamada coleção, ou collection.

O pacote java.util oferece algumas classes definidas na API do Java que implementam funcionalidades associadas a estruturas de dados.
Essas classes são conhecidas como “collections
Algumas das classes que representam coleções providas pela API do Java são:
ArrayList, LinkedList
HashSet, TreeSet, HashMap, entre outras

Programação Java
Collections

A Collections Framework contém os seguintes elementos:
Interfaces: tipos abstratos que representam as coleções. Permitem que coleções sejam manipuladas tendo como base o conceito "Programar para interfaces e não para implementações", desde que o acesso aos objetos se restrinja apenas ao uso de métodos definidos nas interfaces;

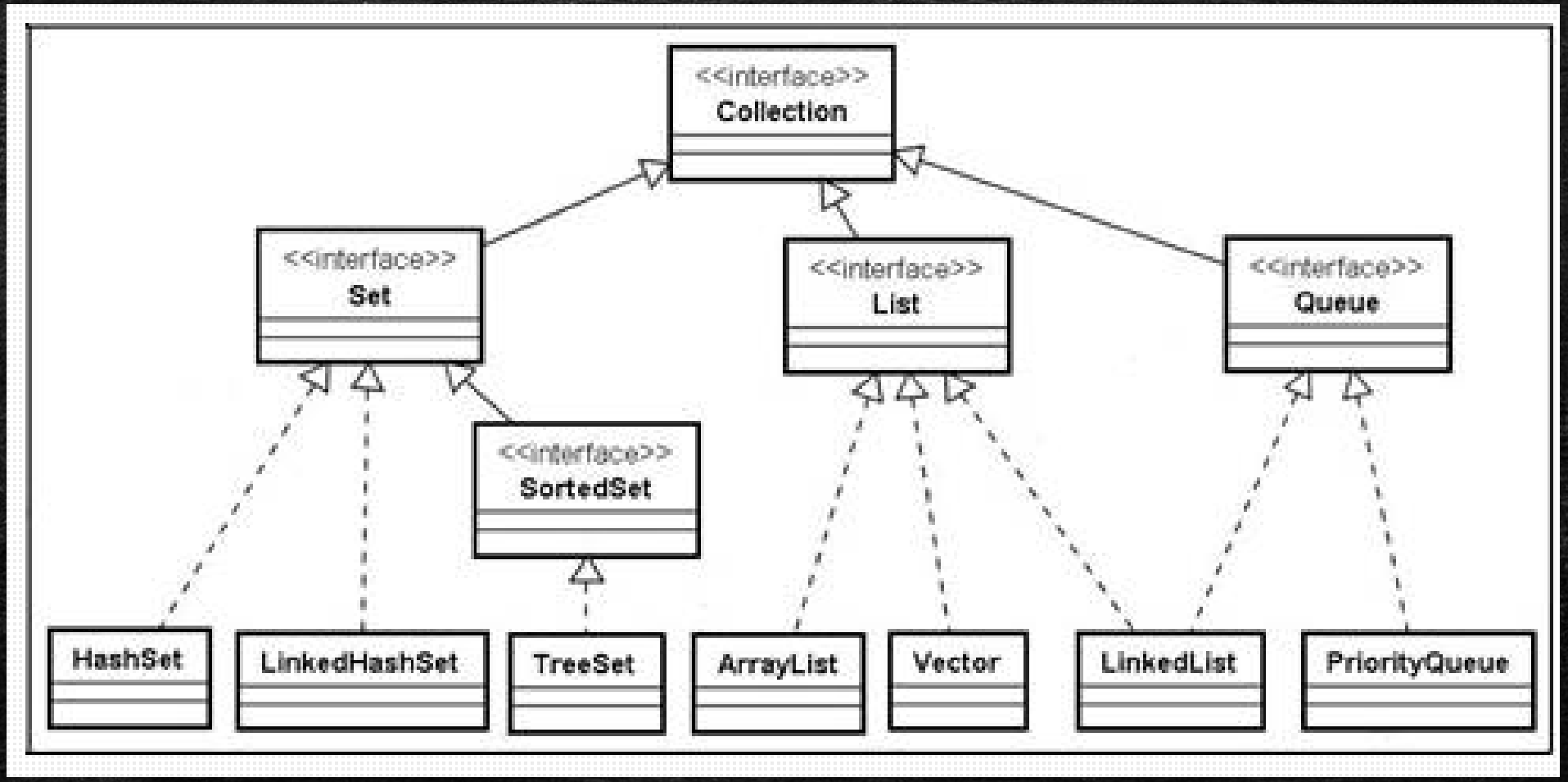
Implementações: são as implementações concretas das interfaces;

Algoritmos: são os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.

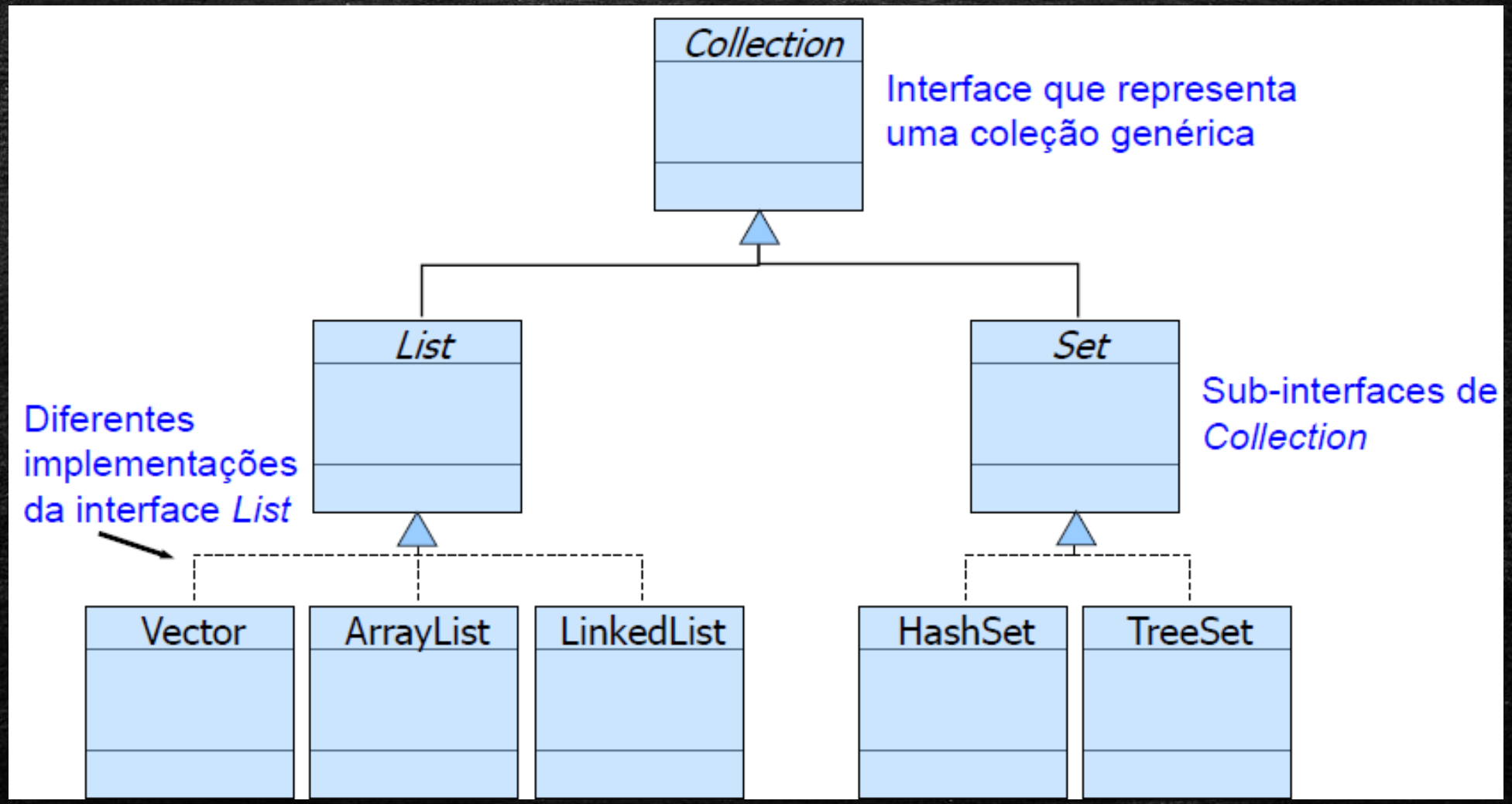
Programação Java
Collections

A seguir a árvore da hierarquia de interfaces e classes da Java Collections Framework que são derivadas da interface Collection. O diagrama usa a notação da UML, onde as linhas cheias representam extends e as linhas pontilhadas representam implements.

Programação Java
Collections



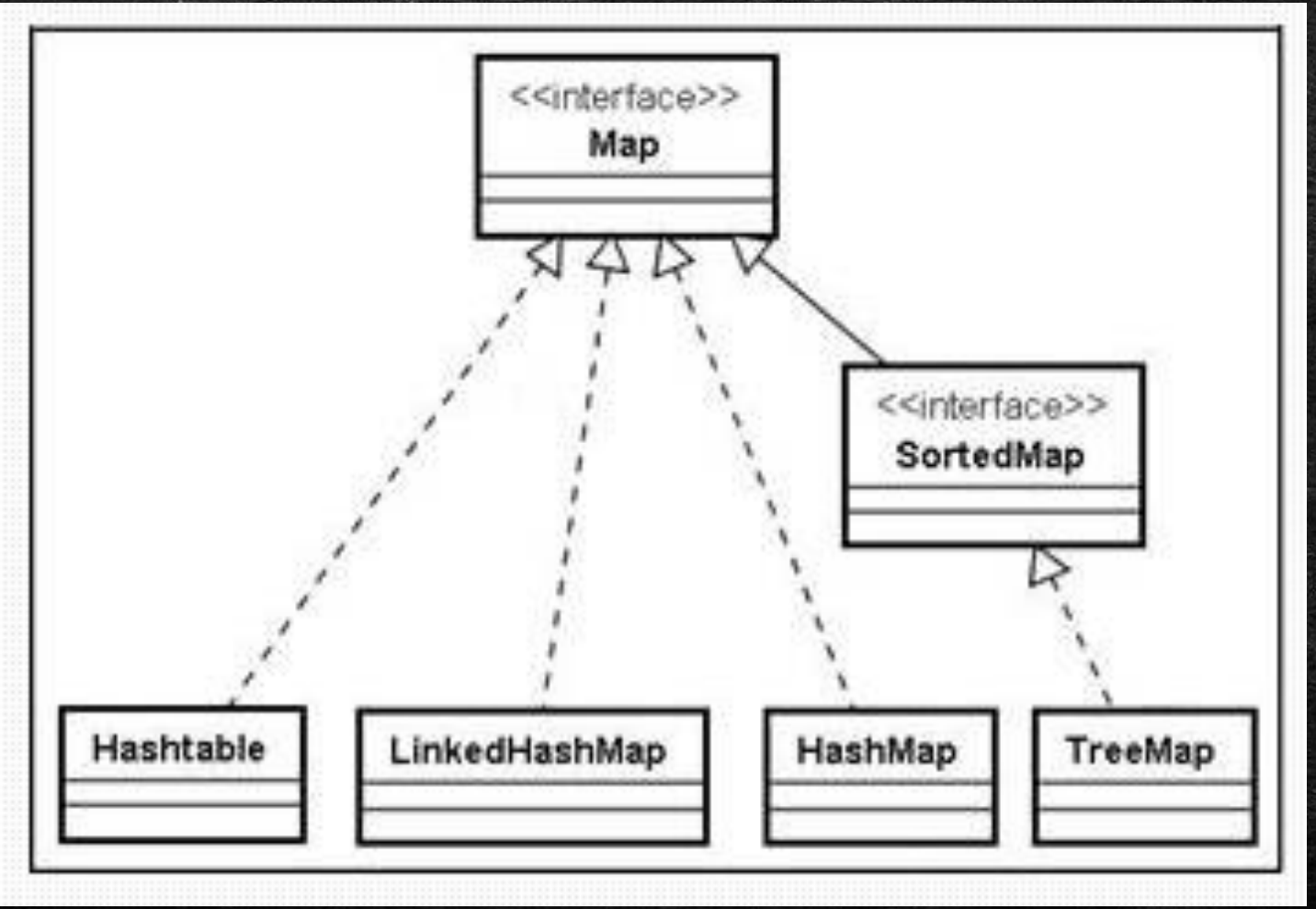
Programação Java
Collections



Programação Java
Collections

A hierarquia da Collections Framework tem uma segunda árvore. São as classes e interfaces relacionadas a mapas, que não são derivadas de Collection. Essas interfaces, mesmo não sendo consideradas coleções, podem ser manipuladas como tal.
A seguir a árvore de hierarquia de mapas.

Programação Java
Collections



Programação Java
Collections

As Interfaces

Collection – Está no topo da hierarquia. Não existe implementação direta dessa interface, mas ela define as operações básicas para as coleções, como adicionar, remover, esvaziar, etc.

- List – Define uma coleção ordenada, podendo conter elementos duplicados. Em geral, o desenvolvedor tem controle total sobre a posição onde cada elemento é inserido e pode recuperá-los através de seus índices. Prefira esta interface quando precisar de acesso aleatório, através do índice do elemento;
- Set - Interface que define uma coleção que não permite elementos duplicados. A interface SortedSet, que estende Set, possibilita a classificação natural dos elementos, tal como a ordem alfabética. Set representa os conjuntos.

Programação Java
Collections

As Interfaces

Map – Mapeia chaves para valores. Cada elemento tem na verdade dois objetos: uma chave e um valor. Valores podem ser duplicados, mas chaves não. SortedMap é uma interface que estende Map, e permite classificação ascendente das chaves. Uma aplicação dessa interface é a classe Properties, que é usada para persistir propriedades/configurações de um sistema, por exemplo.

A API oferece também interfaces que permitem percorrer uma coleção derivada de Collection.

Iterator – possibilita percorrer uma coleção e remover seus elementos;

ListIterator – estende Iterator e suporta acesso bidirecional em uma lista, modificando e/ou removendo elementos.

Programação Java

Collections

- Implementações

  Vamos ver algumas características das implementações que são mais utilizadas e podem ajudar a decidir qual delas utilizar em uma aplicação:

  ArrayList – É como um array cujo tamanho pode crescer. A busca de um elemento é rápida, mas inserções e exclusões não são. Pode ser criado com um tamanho inicial e, se esse tamanho se tornar insuficiente, automaticamente o “array” será aumentado de modo transparente para o usuário da classe. Esta implementação é preferível quando se deseja acesso mais rápido aos elementos. Por exemplo, se você quiser criar um catálogo com os livros de sua biblioteca pessoal e cada obra inserida receber um número sequencial (que será usado para acesso) a partir de zero;

Programação Java
Collections

- Implementações
  ArrayList – Se nenhum valor é passado no construtor, um objeto com capacidade para 10 elementos é criado.

  Exemplo:

- List lista = new ArrayList(15);

Programação Java

Collections

- Implementações
  LinkedList – Implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Pode inserir elementos no início e deletar elementos no interior da lista, porém o acesso aleatório é lento e necessita de um objeto nó para cada elemento, que é composto pelo dado propriamente dito e uma referência para o próximo nó, ou seja, consome mais memória. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões. Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos.

Programação Java
Collections

- Implementações
  LinkedList – Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões. Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos.
  Exemplo:
- List lista = new LinkedList ();
  Para apoiar na decisão de usar ArrayList ou LinkedList na implementação é melhor fazer testes de desempenho.

Programação Java
Collections

Exemplo de um teste simples de desempenho usando ArrayList ou LinkedList.

Execute o programa a seguir e anote o tempo. Substitua ArrayList por LinkedList e repita o teste. Ao final escolha a implementação mais eficiente, para aplicação que possa usar uma ou outra.

```
public class TesteDesempenhoUsandoList
```

Programação Java
Collections

- ArrayList

Exemplo pratico usando ArrayList e LinkedList

ExemploArrayListListaAluno

ExemploLinkedListListaAluno

Programação Java
Collections

ArrayList

Colocando a lista em Ordem crescente e descendente

Na implementação da classe ArrayList , não existe um método de ordenação. Para solucionar este requisito, uma opção seria mudar nossa aplicação para utilizar a interface Set, onde os elementos estariam classificados pela ordem natural, no entanto a inserção de novos elementos seria mais lenta. Sendo assim, vamos utilizar a classe utilitária Collections. Esta classe dispõe do método sort(), que pode classificar uma interface List em ordem natural ou classificar de acordo com a implementação da interface Comparator.

```
Collections.sort(lista);
```

Programação Java
Collections

ArrayList

Adicionando novos dados. Objeto Aluno ao invés de String.

```
public class ListaAluno {
  public static void main(String[] args) {
    List<Aluno> lista = new ArrayList<Aluno>();
    Aluno a = new Aluno("João da Silva", "Linux básico", 0);
    Aluno b = new Aluno("Antonio Sousa", "OpenOffice", 0);
    Aluno c = new Aluno("Lúcia Ferreira", "Internet", 0);
    lista.add(a);
    lista.add(b);
    lista.add(c);
    System.out.println(lista);
```

Programação Java
Collections

ArrayList

No exemplo anterior o método sort() não funciona, pois trocamos a classe String pela classe Aluno.
A documentação da classe Collections nos informa que o método sort() aceita apenas listas cujos elementos sejam de tipos que implementem a interface Comparable, e Aluno não implementa Comparable. Já String é uma classe comparável, isto é, já implementa Comparable (método compareTo()) único método dessa interface. Para que a classificação funcione, a classe Aluno deve implementar a interface Comparable e sobrescrever o método compareTo(), conforme a necessidade do desenvolvedor ou implementar a interface Comparator, para uma classificação mais especifica.

Programação Java
Collections

ArrayList

A interface Comparable tem apenas um método a ser implementado, compareTo(). Sua implementação deve ser feita de forma a retornar um inteiro negativo, zero ou um inteiro positivo caso o objeto que execute o método seja menor, igual ou maior que o objeto passado como parâmetro. Cabe ao desenvolvedor decidir o critério que será adotado para comparar dois objetos.

Programação Java
Collections

For-each em Collections

O For-each é um ciclo for, mas que é adaptado para utilização em Collections e outras listas. Ele serve para percorrer todos os elementos de qualquer Collection contida na API Collections.

Sintaxe

for(tipo elemento:tipo)

```
List<Integer> minhaLista = new ArrayList<Integer>();
minhaLista.add(1);
minhaLista.add(2);
for (Integer listaElementos : minhaLista) {
    System.out.println(listaElementos);
}
```

Programação Java
Collections

A interface Iterator

O iterator é uma interface disponível no pacote java.util que permite percorrer coleções da API Collection, desde que tenham implementado a Collection, fornecendo métodos como o next(), hasnext() e remove().

```
List<Integer> minhaLista = new ArrayList<Integer>()
 minhaLista.add(1);
 minhaLista.add(2);
 Iterator iMinhaLista = minhaLista.iterator();

 for(Integer listaElementos: minhaLista){
     System.out.println(iMinhaLista.next());
 }
```

Programação Java
Collections

Para finalizar ArrayList

```
 // alguns métodos muito utilizados no dia a dia.

//Adiciona elemento no Array
lista.add(a);

// recupera um objeto do ArrayList
lista.get(0);

//adiciona outras listas na já existente
 lista.addAll(lista);

// Verifica se um determinado item está na lista.
lista.contains("João da Silva");

//Remove um  elemento da lista
 lista.remove(0);

//Remove todos os elementos da lista
lista.clear();
```

Programação Java
Collections

- Implementação
  - HashSet: É uma classe que tem implementação concreta da interface Set não organizada, ou seja, os elementos são percorridos aleatoriamente, e também não é ordenada, não há regras de ordenação. Não aceita itens duplicados. O acesso aos dados é mais rápido que em um TreeSet, mas nada garante que os dados estarão ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante. Poderíamos usar esta implementação para criar um catálogo pessoal das canções da nossa discografia;

Programação Java
Collections

- HashSet

Note que forçamos a inserção de um objeto duplicado, mas quando executamos a aplicação constatamos que o objeto foi inserido. Se um Set não permite elementos duplicados, onde está o erro? Como HashSet determina que dois objetos estão duplicados?

ExemploHashSetListaAluno

Programação Java

Collections

- HashSet

HashSet usa o código hash do objeto – dado pelo método hashCode() para saber onde deve por e onde buscar o mesmo no conjunto (Set). Antes ele verifica se não existe outro objeto no Set com o mesmo código hash. Se não há código hash igual, então ele sabe que o objeto a ser inserido não está duplicado. Dessa forma, classes cujas instâncias são elementos de HashSet devem implementar o método hashCode(). Como consequência disso, a classe Aluno, no nosso exemplo, deve sobrescrever o método hashCode().

ExemploHashSetListaAluno

Programação Java

Collections

- HashSet

ExemploHashSetListaAluno

Conforme o contrato geral de hashCode(), que consta na especificação da classe Object, se dois objetos são diferentes de acordo com equals() então não é obrigatório que seus códigos hash sejam diferentes.

Portanto, objetos que retornam o mesmo código hash não são necessariamente iguais. Assim, quando encontra no conjunto um objeto com o mesmo código hash do objeto a ser inserido, HashSet faz uma chamada ao método equals() para verificar se os dois objetos são iguais. Dessa forma, a classe Aluno deve sobrescrever o método equals() também.

Programação Java
Collections

HashSet

Criar código para equals() e hashCode() não é trivial, pois existem contratos definidos pela API de Java que devem ser rigorosamente seguidos. Por exemplo: se dois objetos são iguais, eles devem permanecer iguais durante toda a aplicação e devem resultar no mesmo hashCode(). Para facilitar essa tarefa, Eclipse e NetBeans têm opções para gerar esses métodos para as classes.

ExemploHashSetListaAluno

Programação Java
Collections

Implementação

TreeSet – Os dados são classificados, mas o acesso é mais lento que em um HashSet. Se a necessidade for um conjunto com elementos não duplicados e acesso em ordem natural, prefira o TreeSet. É recomendado utilizar esta coleção para as mesmas aplicações de HashSet, com a vantagem dos objetos estarem em ordem natural.

ExemploTreeSetListaAluno

Programação Java

Collections

- Implementação HashMap

  Baseada em tabela de espalhamento, permite chaves e valores null. Não existe garantia que os dados ficarão ordenados. Escolha esta implementação se a ordenação não for importante e desejar uma estrutura onde seja necessário um ID (identificador). Um exemplo de aplicação é o catálogo da biblioteca pessoal, onde a chave poderia ser o ISBN (International Standard Book Number);

Programação Java
Collections

HashMap

Vamos supor que agora queremos uma estrutura onde possamos recuperar os dados de um aluno passando apenas o seu nome como argumento de um método. Ou seja, informamos o nome do aluno e o objeto correspondente a esse nome é devolvido. Para isso vamos usar a interface Map, que não estende Collection. Isso causa uma mudança profunda na aplicação, visto que os métodos usados anteriormente não poderão ser usados. Map tem seus próprios métodos para inserir/buscar/remover elementos na estrutura.

ExemploHashMapMapaAluno

Programação Java
Collections

- HashMap

  Esta interface mapeia chaves para valores. Considerando a nova proposta do problema, a chave será o nome do aluno e o valor será o objeto aluno.

  Para usar uma classe que implementa Map, quaisquer classes que forem utilizadas como chave devem sobrescrever os métodos hashCode() e equals(). Isso é necessário porque em um Map as chaves não podem ser duplicadas, apesar dos valores poderem ser.

  Vamos utilizar as classes HashMap e TreeMap e vamos ver a diferença entre elas

  ExemploHashMapMapaAluno

Programação Java
Collections

Conclusão

Com tudo o que foi apresentado, podemos constatar que não existe a melhor implementação que resolve todos os problemas de estruturas de dados. Cada tipo de problema requer uma implementação diferente dependendo das características do mesmo. Escolher a implementação certa envolve saber o que sua interface oferece, quais as suas características e como ela será usada.

- Documentação da API
  https://docs.oracle.com/javase/tutorial/collections/index.html

Programação Java
Collections

EXERCÍCIO