Programação Orientada a Objetos

Carlos Henrique Bughi, MSc

UNIVALI
Escola Politécnica

Onde estamos?
(e para onde vamos)

- Avaliação Prática M2
  - Manipulando coleções de objetos
- Tratamento de exceções

UNIVALI Escola Politécnica

Aula 13

Manipulando Coleções de Objetos

UNIVALI

Escola Politécnica

Coleções de objetos

- Uma coleção é um objeto que representa um grupo de objetos;
- Desde a versão 1.2 do JDK, a plataforma J2SE inclui um framework de coleções, a Collections Framework;
- Um framework de coleções é uma arquitetura unificada para representação e manipulação de coleções;
- permite que elas sejam manipuladas independentemente dos detalhes de sua implementação;

Coleções de objetos

- Principais vantagens:
  - Redução do esforço de programação, fornecendo estruturas de dados e algoritmos úteis, para que não seja necessário reescrevê-los.
  - Aumento da performance, fornecendo implementações de alta performance. Como as várias implementações de cada interface são substituíveis, os programas podem ser facilmente refinados trocando-se as implementações.
  - Interoperabilidade entre APIs não relacionadas, estabelecendo uma linguagem comum para passagem de coleções.
  - Redução do esforço de aprendizado de APIs, eliminando a necessidade de se aprender várias APIs de coleções diferentes.
  - Redução do esforço de projetar e implementar APIs, eliminando a necessidade de se criar APIs de coleções próprias.
  - Promover o reúso de software, fornecendo uma interface padrão para coleções e algoritmos para manipulá-los.

Coleções de objetos

- Uma coleção consiste em:
  - Interfaces de coleções - representam diferentes tipos de coleção, como conjuntos, listas e arrays associativos. Estas interfaces formam a base do framework.
  - Implementações abstratas - implementações parciais das interfaces de coleções, para facilitar a criação de implementações personalizadas.
  - Algoritmos - métodos estáticos que realizam funções úteis sobre coleções, como a ordenação de uma lista.

Coleções de objetos

- Collection
  - Interface base para todos os tipos de coleção.
  - Define as operações mais básicas para coleções de objetos, como adição (add) e remoção (remove) abstratos (sem informações quanto à ordenação dos elementos), esvaziamento (clear), tamanho (size), conversão para array (toArray), objeto de iteração (iterator), e verificações de existência (contains e isEmpty).

As interfaces de coleções

- ***List***
  - Interface que extende Collection, e que define coleções ordenadas (sequências);
  - Tem-se o controle total sobre a posição de cada elemento, identificado por um índice numérico.
  - Na maioria dos casos, pode ser encarado como um "array de tamanho variável" pois, como os arrays primitivos, é acessível por índices, mas além disso possui métodos de inserção e remoção.

As interfaces de coleções

- ***Set***
  - Interface que define uma coleção que não contém duplicatas de objetos.
  - Isto é, são ignoradas as adições caso o objeto já exista na coleção.
  - Não é garantida a ordenação dos objetos, isto é, a ordem de iteração dos objetos não necessariamente tem qualquer relação com a ordem de inserção dos objetos. Por isso, não é possível indexar os elementos por índices numéricos, como em uma List.

As interfaces de coleções

- ***SortedSet***
  - Interface que estende Set, adicionando a semântica de ordenação natural dos elementos.
  - A posição dos elementos na iteração da coleção é determinado pelo retorno do método compareTo(o), caso os elementos implementem a interface Comparable, ou do método compare(o1, o2) de um objeto auxiliar que implemente a interface Comparator.

As interfaces de coleções

- ***Map***
  - Interface que define um array associativo, isto é, ao invés de números, objetos são usados como chaves para se recuperar os elementos.
  - As chaves não podem se repetir (seguindo o mesmo princípio da interface Set), mas os valores podem ser repetidos para chaves diferentes.
  - Um Map também não possui necessariamente uma ordem definida para a iteração.

As interfaces de coleções

- ***SortedMap***
  - Interface que estende Map, adicionando a semântica de ordenação natural dos elementos, análogo à SortedSet.
  - Também adiciona operações de partição da coleção, com os métodos:
    - headMap(k) - que retorna um SortedMap com os elementos de chaves anteriores a k;
    - subMap(k1,k2) - que retorna um SortedMap com os elementos de chaves compreendidas entre k1 e k2; e
    - tailMap(k) - que retorna um SortedMap com os elementos de chaves posteriores a k.

Principais métodos da interface List

- add(int, object)
  - Adiciona um elemento no indice informado;
- add(object);
  - Adiciona um elemento no final da lista;
- addAll(collection);
  - Adiciona uma coleção no final da lista;
- clear();
  - Remove todos os elementos da lista;
- contains(object);
  - Retorna true se a lista contém um elemento igual ao objeto passado;
- get(index);
  - Retorna o elemento pelo índice;

Principais métodos da interface List

- isEmpty();
  - Retorna true se a lista está vazia;
- remove(int);
  - Remove um elemento pelo índice;
- removeRange(int,int);
  - Remove elementos dentro dos limites especificados;
- set(int, object);
  - Substitui um elemento no índice informado;
- size();
  - Retorna a quantidade de elementos da lista;
- toArray();
  - Retorna um array com os elementos da lista;

Principais métodos da Interface Map

- clear();
  - Remove os elementos do mapa;
- get(object);
  - Retorna um elemento pela chave informada;
- isEmpty();
  - Retorna true se o mapa está vazio;
- size();
  - Retorna o total de elementos do mapa;
- values();
  - Retorna uma Collection com os valores contidos no mapa;

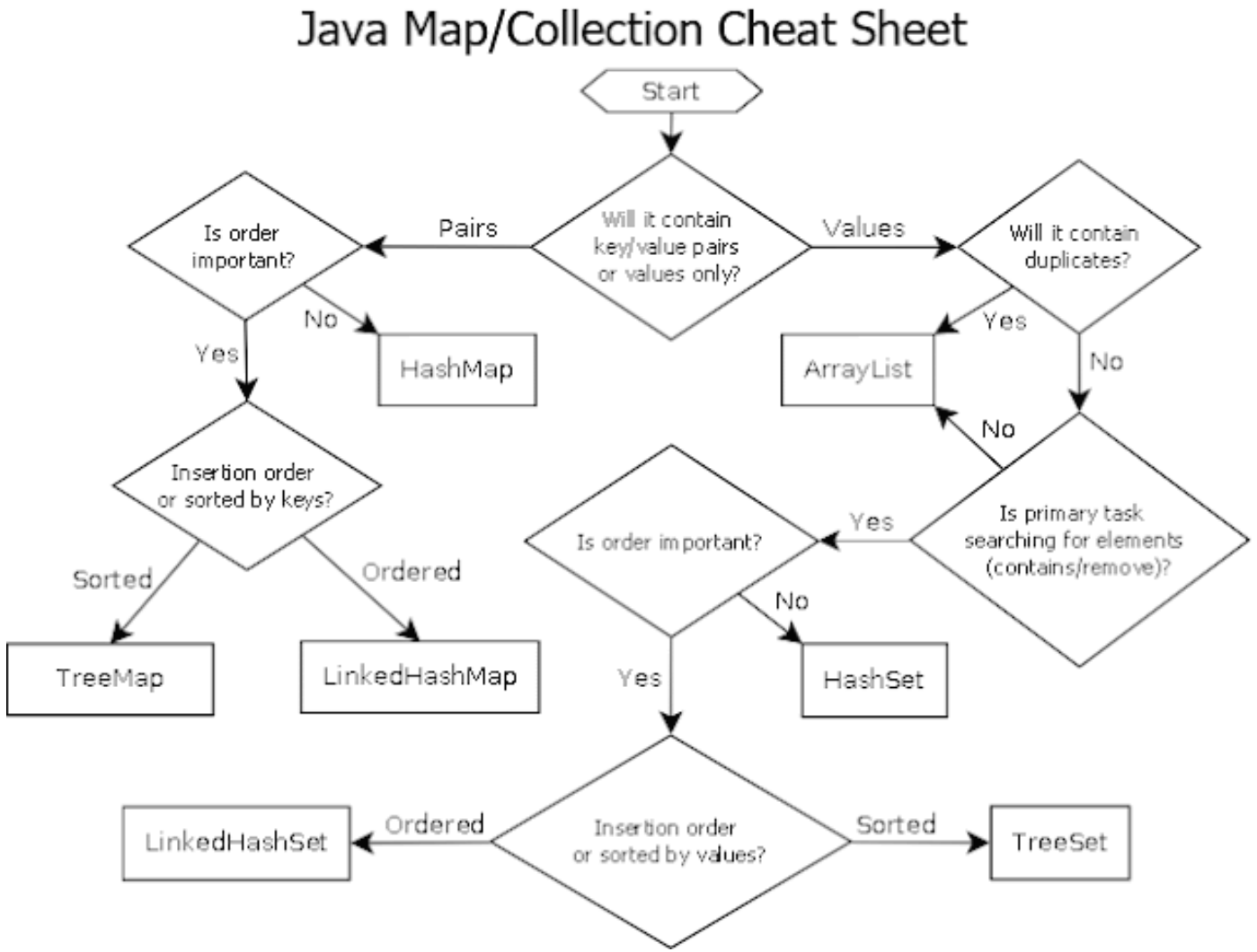
Principais métodos da Interface Map

- keySet();
  - Retorna uma coleção (Set) com as chaves existentes;
- put(object chave, object valor);
  - Insere um novo elemento no mapa;
- remove(object);
  - Remove um elemento do mapa;
- containsKey(object);
  - Retorna true se o mapa contém a chave especificada;
- containsValue(object);
  - Retorna true se o mapa contém chaves para o valor informado;
- entrySet();
  - Retorna uma coleção (Set) com os mapeamentos existentes;

Programando coleções

```
1.  // Versão normal:
2.  List listaCores = new ArrayList();
3.  listaCores.add("vermelho");
4.  listaCores.add("verde");
5.  listaCores.add("amarelo");
6.  listaCores.add("branco");
7.
8.  // Versão abreviada utilizando um array
9.  List listaCores = new ArrayList(Arrays.asList(new String[] {
10.     "vermelho", "verde", "amarelo", "branco" }));
```

Como escolher?



https://www.javaguides.net/2018/06/java-collection-framework-best-practices.html

UNIVALI Escola Politécnica

Melhores práticas

- Foco na Interface e não na implementação
- Usar interface na declaração, no retorno de método e no parâmetro de funções.
- Uso de generics
- Retorne coleções vazias e não null

- https://www.javaguides.net/2018/06/java-collection-framework-best-practices.html
- https://www.codejava.net/java-core/collections/18-java-collections-and-generics-best-practices

Links úteis

overview
https://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html

referência
https://docs.oracle.com/javase/7/docs/technotes/guides/collections/reference.html

tutoriais
https://docs.oracle.com/javase/tutorial/collections/index.html

Como escolher
https://www.javaguides.net/2018/06/java-collection-framework-best-practices.html

Exercício 1

- Criar uma tabela comparativa entre as diferentes implementações

Interface	Implementação	Marque se possui ou não as características abaixo				
		Ordenação	Sincronização	Ordem de iteração	Associação	Permite elementos duplicados

UNIVALI Escola Politécnica