Collections

Introdução

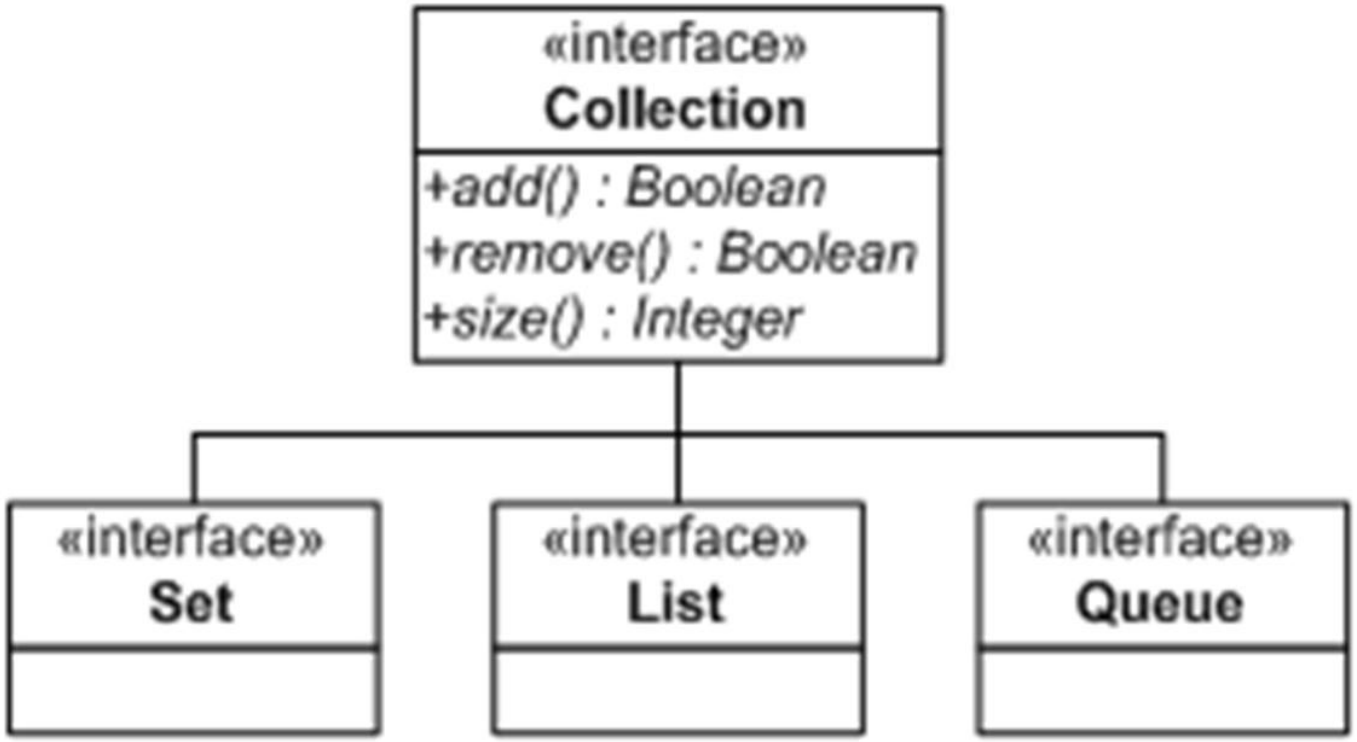
Collections

Collection é uma estrutura de dados, que pode guardar referências para outros objetos. (DEITEL, 2012)

Collection, é uma interface java.util.Collection que representa um grupo de objetos do mesmo tipo.

No mesmo pacote java.util existem 3 interfaces que herdam de Collection.

Hierarquia de Collections



Set

Set, é uma coleção de objetos, onde não são aceitos elementos duplicados. Neste caso, ao tentar adicionar um novo elemento que já exista a operação **add()** irá retornar false e objeto não será adicionado no conjunto.
A estrutura **Set** permite que os objetos sejam armazenados de forma não sequencial assim como uma árvore, portanto os objetos não podem ser acessados através de um índice.

Set

Classes que implementam a interface **Set**

- **AbstractSet**
- **ConcurrentSkipListSet**
- **CopyOnWriteArraySet**
- **EnumSet**
- **HashSet**
- **JobStateReasons**
- **LinkedHashSet**
- **TreeSet**

Set

HashSet, é uma implementação de uma tabela **Hash** da interface **Set**. Permite a inclusão de qualquer elemento inclusive **null**, desdeque não haja elementos duplicados.

Exemplo de uso da classe **HashSet**

```
Set hashList = new HashSet();
Object o = new Object();
hashList.add(o);
```

List

List, é uma coleção onde os objetos são armazenados em sequência, portanto podem ser acessados através de um índice numérico.

A estrutura **List** permite a armazenagem de objetos duplicados, ela mantém a ordem a qual os objetos foram inseridos .

List

Classes que implementam a interface **List**

- **AbstractList**
- **AbstractSequentialList**
- **ArrayList**
- **AttributeList**
- **CopyOnWriteArrayList**
- **LinkedList**
- **RoleList**
- **RoleUnresolvedList**
- **Stack**
- **Vector**

List

ArrayList, é uma implementação de array-redimensionável da interface **List**. Permite todos os elementos inclusive **null**.

- Os objetos são armazenados internamente em um array redimensionável, esta classe é semelhante a classe **Vector** porém ela não é *sincronizável* (ver uso de **Threads**).
- Devido ao armazenamento ser feito em um array redimensionável o processo de adição de objetos demora mais para executar, e os demais processos gastam um tempo linear na pesquisa.
- Exemplo de uso da classe **ArrayList**

```
List lista = new ArrayList();
```

List

LinkedList, é uma implementação de lista duplamente ligada da interface **List**. Permite todos os elementos inclusive **null**.

- Os objetos são armazenados internamente em uma lista duplamente ligada. Esta classe não é *sincronizável* (ver uso de **Threads**).
- Devido ao armazenamento ser feito em lista duplamente ligada o processo de adição de objetos é mais rápido, porém a pesquisa e outros acessos que dependem de pesquisa são mais morosos.
- Exemplo de uso da classe **LinkedList**

```
List lista = new LinkedList();
```

Queue

Queue, é uma coleção de objetos armazenados em sequência, onde são aceitos elementos duplicados, contudo não há métodos especiais para resgatar e inserir elementos, apenas os elementos no topo da fila podem ser acessados, e os objetos são inseridos no final da fila.

Queue

Classes que implementam a interface **Queue**

- **AbstractQueue**
- **ArrayBlockingQueue**
- **ArrayDeque**
- **ConcurrentLinkedDeque**
- **ConcurrentLinkedQueue**
- **DelayQueue**
- **LinkedBlockingDeque**
- **LinkedBlockingQueue**
- **LinkedList**
- **LinkedTransferQueue**
- **PriorityBlockingQueue**
- **PriorityQueue**
- **SynchronousQueue**

Queue

PriorityQueue, é uma implementação de **Queue** baseado em uma prioridade que pode ser conforme a ordem natural de inserção na fila, ou através da implementação de um **Comparator.**

- Os objetos são armazenados internamente em uma fila. Esta classe não é *sincronizável* (ver uso de **Threads**).
- Exemplo de uso da classe **PriorityQueue**

```
Queue q = new PriorityQueue();
```

Map

Map, é um objeto que mapeia **valores** para as **chaves**, estruturando os objetos e pesquisando os através de **chaves**. A estrutura **Map** não pode conter **chaves** duplicadas, embora valores duplicados são permitidos.

Map

Classes que implementam a interface **Map**

- **AbstractMap**
- **Attributes**
- **AuthProvider**
- **ConcurrentHashMap**
- **ConcurrentSkipListMap**
- **EnumMap**
- **HashMap**
- **Hashtable**
- **IdentityHashMap**
- **WeakHashMap**
- **LinkedHashMap**
- **PrinterStateReasons**
- **Properties**
- **Provider**
- **RenderingHints**
- **SimpleBindings**
- **TabularDataSupport**
- **TreeMap**
- **UIDefaults**

Map

HashMap, é uma implementação de uma tabela **Hash** da interface **Set** e da interface **Map** que permite o armazenamento e busca através de chaves. Permite todos os elementos inclusive **null**, porém não pode haver chaves duplicadas.

- Exemplo de uso da classe **HashMap**

```
Map<String, Object> hashMap = new HashMap<String, Object>();
Object o1 = new Object();
Object o2 = new Object();
hashMap.put("Objeto 1", o1);
hashMap.put("Objeto 2", o2);
```

Bibliografia

JAVADOC ArrayList acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html

JAVADOC LinkedList acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/LinkedList.html

JAVADOC HashSet acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashSet.html

JAVADOC TreeSet acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/TreeSet.html

JAVADOC HashMap acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html

JAVADOC Hashtable acessed in july 2009 - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Hashtable.html

DEITEL, Java - Como Programar - 6ª Edição, Pearson Education

SIERRA, KATHY, Use a Cabeça Java, Alta Books - Capitulo 16 – páginas 369 à 402