

Object Collections

What Is a Collection?

Collection (sometimes called a container) is simply an object that groups multiple elements into a single unit.

Usage:

- 1) to store and retrieve data
- 2) to manipulate data
- 3) to transmit data from one method to another

Collections typically represent data items that form a natural group like:

- 1) a poker hand (a collection of cards)
- 2) a mail folder (a collection of letters)
- 3) a telephone directory (a collection of name-to-phone-number mappings)

Collection Framework

A collections framework is a unified architecture for representing and manipulating collections.

All collections frameworks contain three things:

1) Interfaces

- abstract data types representing collections.
- Interfaces allow collections to be manipulated independently of the details of their representation.

Collection Framework

2) Implementations

- a. concrete implementations of the collection interfaces.
- b. In essence, these are reusable data structures.

3) Algorithms

- a) methods that perform useful computations like searching and sorting, on objects that implement collection interfaces.
- b) they are polymorphic because the same method can be used on many different implementations of the appropriate collections interface.
- c) In essence, they are reusable functionality.

Benefits

Collection Framework offers the following benefits:

1) It reduces programming effort

Powerful data structures and algorithms

2) It increases program speed and quality

a) High quality implementations

b) Fine tuning by switching implementations

3) It allows interoperability among unrelated APIs

Passing objects around from one API to another

Benefits

4) It reduces the effort to learn and use new APIs

- a) Uniformity of the framework
- b) APIs of applications

5) It reduces effort to design new APIs

6) It fosters software reuse

- a) New data structures and algorithms

Core Collection Interfaces

These are interfaces used to manipulate collections, and pass them from one method to another.

Purpose:

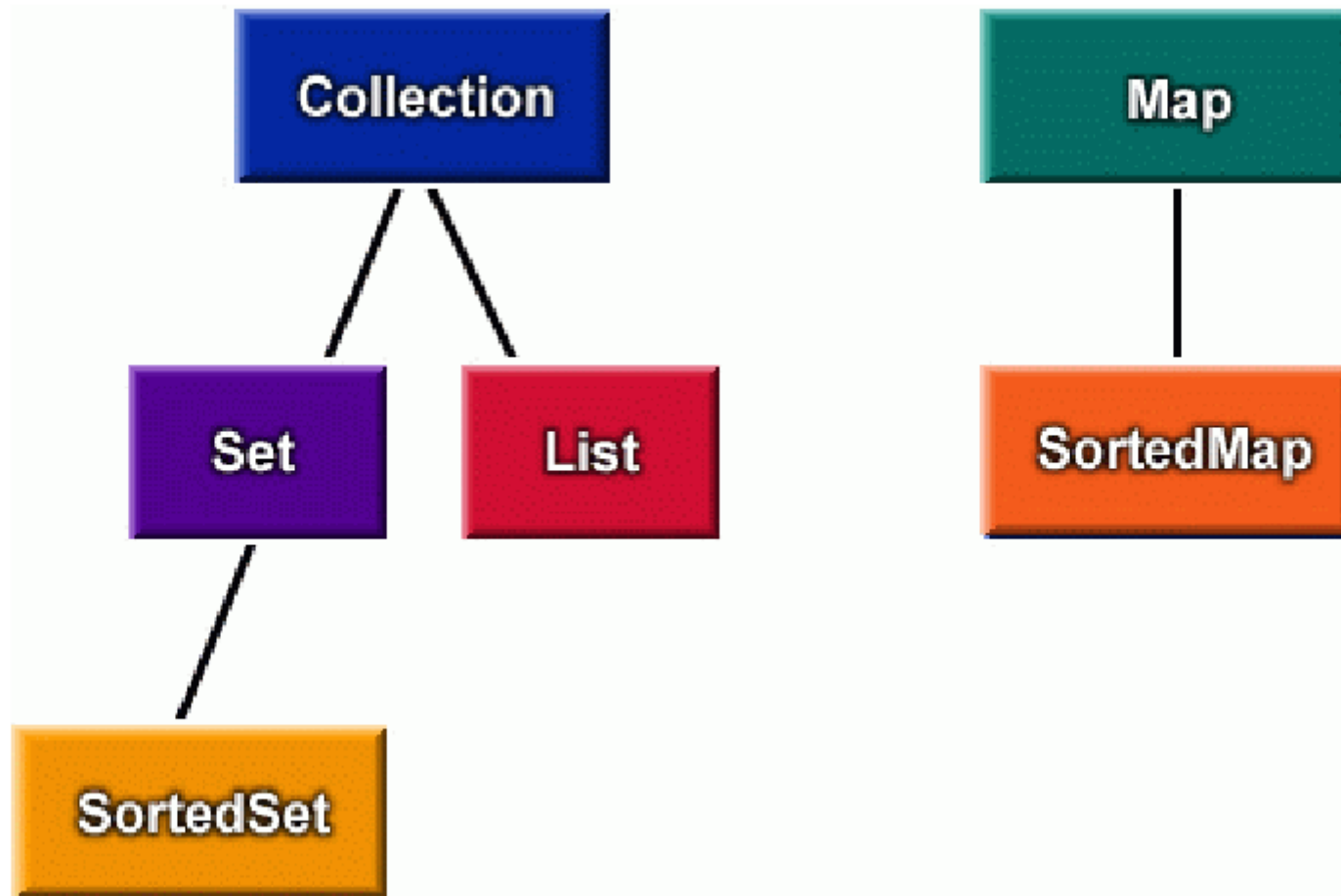
- 1) To allow collections to be manipulated independently of the details of their representation.

Note:

- 1) They are the heart and soul of the collections framework.
- 2) When you understand how to use these interfaces, you know most of what there is to know about the framework.

Core Collection Interfaces

The core collections interfaces are shown below:



Core Collection Interfaces

There are four basic core collection interfaces:

- 1) Collection**
- 2) Set**
- 3) List**
- 4) Map**

Collection

The Collection interface is the root of the collection hierarchy.

Usage: To pass around collections of objects where maximum generality is desired.

Behaviors:

- 1) Basic Operations
- 2) Bulk Operations
- 3) Array Operations

Collection Methods

```
public interface Collection {  
  // Basic Operations  
  int size();  
  boolean isEmpty();  
  boolean contains(Object element);  
  boolean add(Object element); // Optional  
  boolean remove(Object element); // Optional  
  Iterator iterator();  
  
  // Bulk Operations  
  boolean containsAll(Collection c);  
  boolean addAll(Collection c); // Optional  
  boolean removeAll(Collection c); // Optional  
  boolean retainAll(Collection c); // Optional  
  void clear(); // Optional  
  
  // Array Operations  
  Object[] toArray();  
  Object[] toArray(Object a[]);  
}
```


Collection Methods

It has methods to tell you:

- 1) how many elements are in the collection (size, isEmpty)
- 2) to check if a given object is in the collection (contains)
- 3) to add and remove an element from the collection (add, remove),
- 4) and to provide an iterator over the collection (iterator)

Set

A Set is a Collection that cannot contain duplicate elements.

Set models the mathematical set abstraction.

Example:

- 1) Set of Cars - {BMW, Ford, Jeep, Chevrolet, Nissan, Toyota, VW}
- 2) Nationalities in the class - {Chinese, American, Canadian, Indian}

It extends Collection and contains no methods other than those inherited from Collection.

Set

Set extends Collection to add the following functionality:

- a) stronger contract on the behavior of the equals and hashCode operations,**
- b) allowing Set objects with different implementation types to be compared meaningfully.**

Two Set objects are equal if they contain the same elements.

Set Methods

```
boolean retainAll(Collection c); // Optional
void clear();                    // Optional

// Array Operations
Object[] toArray();
Object[] toArray(Object a[]);
}
```

Provides two general purpose implementations:

- 1) `HashSet` – which stores its elements in a hash table, is the best-performing
- 2) `TreeSet` – which stores its elements in a red-black tree, guarantees the order of iteration.

Example: Set

```
import java.util.*;

public class FindDups {

    public static void main(String args[]) {
        Set s = new HashSet();

        for (int i=0; i<args.length; i++)
            if (!s.add(args[i])) System.out.println("Duplicate detected: "+args[i]);

        System.out.println(s.size()+" distinct words detected: "+s);
    }
}
```

List

- 1) New List operations
 - a) Positional access
 - b) Search
 - c) Iteration (ordered, backward)
 - d) Range-view operations
- 2) General Purpose Implementation
 - a) ArrayList
 - b) ListIterator

List implementations in Java

- Two concrete classes for lists:
 - **ArrayList, LinkedList**
- Both implement the List interface
 - Which extends the Collection interface and adds:
E get(int index);
E set(int index, E element);
E remove(int index);
void add(int index, E element);
int indexOf(Object o);
int lastIndexOf(Object o);
List<E> subList(int from, int to);
...
- See Java API documentation for all methods

ArrayList

- The ArrayList class extends AbstractList and implements the List interface.
- ArrayList supports dynamic arrays that can grow as needed.
- ArrayList has the constructors shown here:

ArrayList()

ArrayList(Collection c)

ArrayList(int capacity)

LinkedList

- The LinkedList class extends AbstractSequentialList and implements the List interface. It provides a linked-list data structure.
- It has the two constructors:

LinkedList()

LinkedList(Collection c)

LinkedList

- In addition to the methods that it inherits, the LinkedList class defines some useful methods of its own for manipulating and accessing lists. To add elements to the start of the list, use `addFirst()`; to add elements to the end, use `addLast()`.

`void addFirst(Object obj)`

`void addLast(Object obj)`

To insert items at a specific location, use

`add(int, Object)`

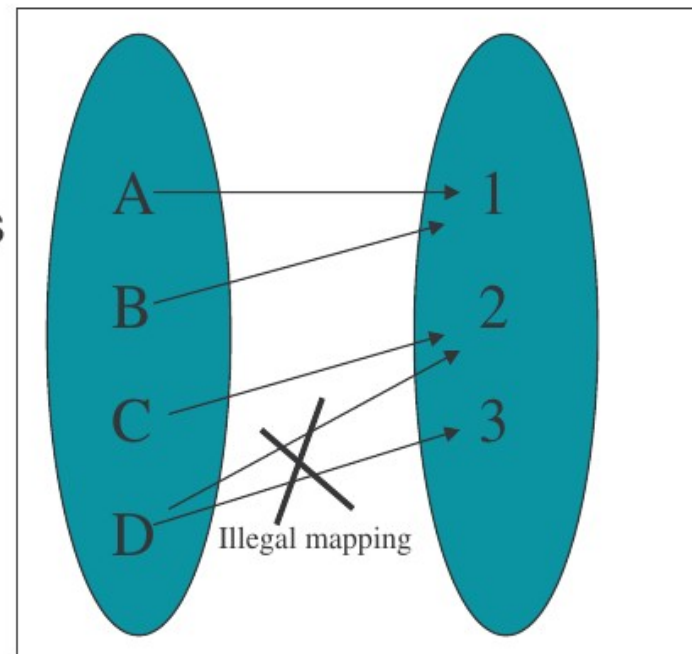
Map

A Map is an object that maps keys to values. Maps cannot contain duplicate keys.

- 1) Each key can map to at most one value
- 2) A map cannot contain duplicate keys.
- 3) Two Map objects are equal if they represent the same key-value mappings

Examples:

- a) Think of a dictionary:
word \leftrightarrow description
- b) address book
name \leftrightarrow phone number



Map

Map

- 1) Collection-view methods allow a Map to be viewed as a Collection
 - a) `keySet` – The Set of keys contained in the Map
 - b) `values` – The Collection of values contained in the Map
 - c) `entrySet` – The Set of key-value pairs contained in the Map
- 2) With all three Collection-views, calling an Iterator's `remove` operation removes the associated entry from the backing Map.
 - a) This is the only safe way to modify a Map during iteration.
- 3) Every object can be used as a hash key
- 4) Two Map objects are equal if they represent the same key-value mappings


SortedSet Interface

- 1) A Set that maintains its elements in ascending order
 - a) according to elements' natural order
 - b) according to a Comparator provided at SortedSet creation time
- 2) Set operations
 - a) Iterator traverses the sorted set in order
- 3) Additional operations
 - a) Range view – range operations
 - b) Endpoints – return the first or last element
 - c) Comparator access
- 4) Location: `java.util.SortedSet`

SortedMap Interface

- 1) A Map that maintains its entries in ascending order
 - a) According to keys' natural order
 - b) According to a Comparator provided at creation time.
- 2) Map operations
 - a) Iterator traverses elements in any of the sorted map's collection-views in key-order.
- 3) Additional Operations
 - a) Range view
 - b) End points
 - c) Comparator access
- 4) Location: `java.util.SortedMap`

General Purpose Implementations

		Implementations			
		Hash Table	Resizeable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet		TreeSet	
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap	

Older Implementations

- 1) The collections framework was introduced in JDK1.2.
- 2) Earlier JDK versions included collection implementations that were not part of any framework
 - a) `java.util.Vector`
 - b) `java.util.Hashtable`
- 3) These implementations were extended to implement the core interfaces but still have all their legacy operations
 - a) Be careful to always manipulate them only through the core interfaces.

Q & A