Java Day6

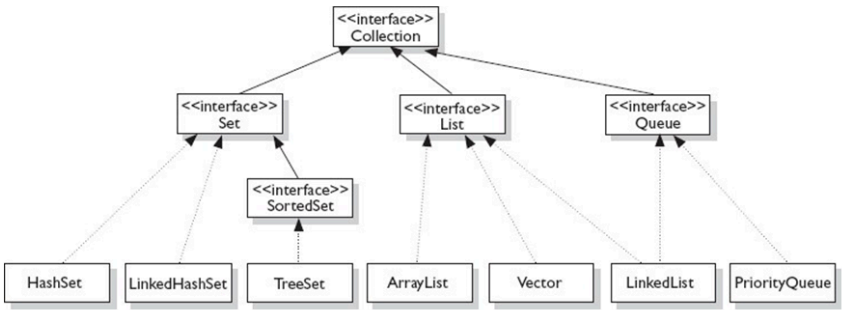
Introduction

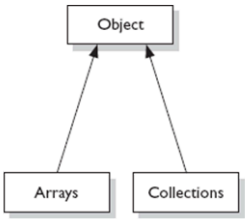
- A Collection is a group of objects
- Collections framework provide a a set of standard utility classes to manage collections
- Collections Framework consists of three parts:
  - Core Interfaces
  - Concrete Implementation
  - Algorithms such as searching and sorting

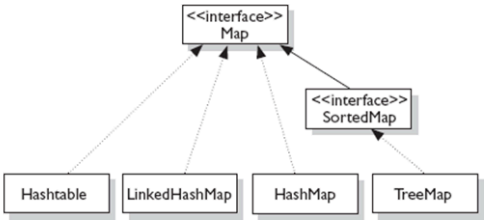
Advantages of Collections

- **Reduces programming effort :** by providing useful data structures and algorithms so you don't have to write them yourself.
- **Increases performance :**by providing high-performance implementations of useful data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be easily tuned by switching implementations.
- **Provides interoperability between unrelated APIs:** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs:** by eliminating the need to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs:** by eliminating the need to produce ad hoc collections APIs.
- **Fosters Software reuse:** by providing a standard interface for collections and algorithms to manipulate them.

Interfaces and their implementation



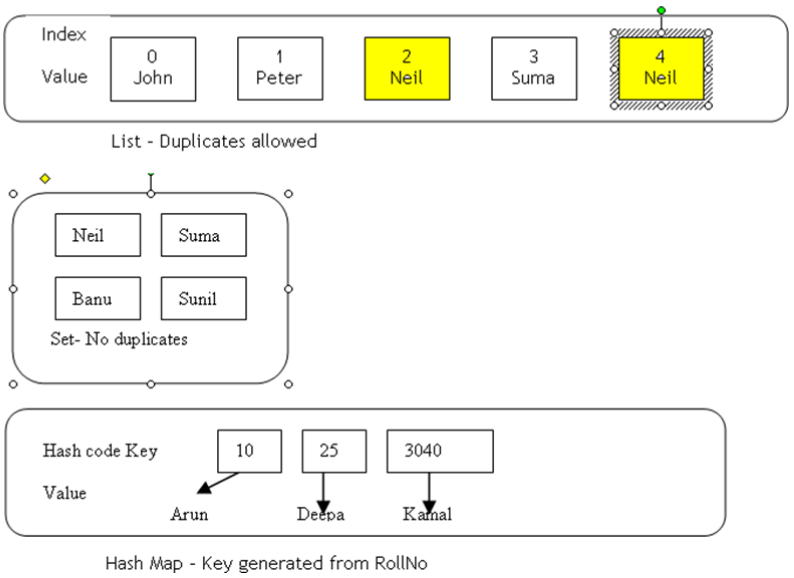




Collection Interfaces

Interfaces	Description
Collection	A basic interface that defines the operations that all the classes that maintain collections of objects typically implement.
Set	Extends the Collection interface for sets that maintain unique element.
SortedSet	Augments the Set interface or Sets that maintain their elements in sorted order.
List	Collections that require position-oriented operations should be created as lists. Duplicates are allowed.
Queue	Things arranged by the order in which they are to be processed.
Map	A basic interface that defines operations that classes that represent mapping of keys to values typically implement.
SortedMap	Extends the Map interface for maps that maintain their mappings in the key order.

Illustration of List, Set and Map



Collection Implementations

		Implementations				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Collection Implementations

- Classes that implement the collection interfaces typically have names of the form <*Implementation-style*><*Interface*>. The general purpose implementations are summarized in the table above.
- **HashSet** A HashSet is an unsorted, unordered Set. It uses the hashcode of the object being inserted, so the more efficient your hashCode() implementation the better access performance you'll get. Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.
- **TreeSet** A TreeSet stores objects in a sorted sequence. It stores its elements in a tree and they are automatically arranged in a sorted order.

Collection Implementations

- **LinkedHashSet** A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. Use this class instead of HashSet when you care about the iteration order. When you iterate through a HashSet the order is unpredictable, while a LinkedHashSet lets you iterate through the elements in the order in which they were inserted.
- **ArrayList** Think of this as a growable array. It gives you fast iteration and fast random access. It is an ordered collection (by index), but not sorted. ArrayList now implements the new RandomAccess interface—a marker interface (meaning it has no methods) that says, "this list supports fast (generally constant time) random access." Choose this over a LinkedList when you need fast iteration but aren't as likely to be doing a lot of insertion and deletion.
- **LinkedList** A LinkedList is ordered by index position, like ArrayList, except that the elements are doubly-linked to one another.

Collection interface methods

Method	Description
`int size();`	Returns number of elements in collection.
`boolean isEmpty();`	Returns true if invoking collection is empty.
`boolean contains(Object element);`	Returns true if element is an element of invoking collection.
`boolean add(Object element);`	Adds element to invoking collection.
`boolean remove(Object element);`	Removes one instance of element from invoking collection
`Iterator iterator();`	Returns an iterator fro the invoking collection
`boolean containsAll(Collection c);`	Returns true if invoking collection contains all elements of *c*; false otherwise.
`boolean addAll(Collection c);`	Adds all elements of *c* to the invoking collection.
`boolean removeAll(Collection c);`	Removes all elements of *c* from the invoking collection
`boolean retainAll(Collection c);`	Removes all elements from the invoking collection except those in *c*.
`void clear();`	Removes all elements from the invoking collection
`Object[] toArray();`	Returns an array that contains all elements stored in the invoking collection
`Object[] toArray(Object a[]);`	Returns an array that contains only those collection elements whose type matches that of *a*.

List

- List interface extends from Collection interface
- It stores elements in a sequential manner
- Elements in the list can be accessed or inserted based on their position
- Starts with zero based index
- Can contain duplicate elements
- An Iterator can be used to access the elements of the List

Please refer documentation and note down the important methods available in List interface

The ArrayList Class

- ArrayList class implements List interface
- It supports dynamic array that can grow dynamically
- Standard arrays are of fixed size. After arrays are created they cannot grow or shrink
- It provides more powerful insertion and search mechanisms than arrays
- Gives faster Iteration and fast random access
- Ordered Collection (by index), but not Sorted

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(0, new Integer(42));
int total = list.get(0).intValue();
```

Refer documentation for the various ways in which an ArrayList can be created and the various methods available in ArrayList

Iterator

- **Iterator is an object that enables you to traverse through a collection**
- **Can be used to remove elements from the collection selectively, if desired**

```
public interface Iterator<E>

 {
 boolean hasNext();
 E next();
 void remove();
 }
```

```
ArrayList<Integer> ai=new ArrayList<Integer>();
Iterator i=ai.iterator();
while (i.hasNext())
System.out.println(i.next());
```

Iterator

- Java provides 2 interfaces that define the methods by which you can access each element of a collection : enumeration & iterators. Enumeration is a legacy interface and is considered obsolete for new code. It is now superceded by the iterator interface.
- The iterator() method returns an iterator to a collection. It is very similar to an Enumeration, but differs in the two respects:
- Iterator allows the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.
- The first point is important: There was *no* safe way to remove elements from a collection while traversing it with an Enumeration. The semantics of this operation were ill defined, and differed from implementation to implementation.
- **boolean hasNext() -** Returns true if there are more elementsObject
- **next() -** Returns next element. Throws NoSuchElementException if there is no next element.
- **void remove() -** Removes current element. Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next()

7

Linked List

- **Implements the List and also the Queue interface**
- **Some Useful Methods**
  - void addFirst(Object x)
  - void addLast(Object x)
  - Object getFirst()
  - Object getLast()
  - Object removeFirst()
  - Object removeLast()

ListIterator

- Used for obtaining a iterator for collections that implement List
- ListIterator gives us the ability to access the collection in either forward or backward direction
- Has both next() and previous() method to access the next and previous element in the List

ArrayList

```
package com.Day6.Collections;

import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.Iterator;
import java.util.List;

/*
Arraylist is a class implementing list interface
Insertion order preserved
Duplicates are allowed
Null also allowed
Heterogenous Data are allowed
 */
/*
 * 1. Declaration
 * 2. How to Add Data to ArrayList
 * 3. How to find the size of the ArrayList
 * 4. How to print ArrayList
 * 5. How to remove element in an ArrayList
 * 6. How to insert element at particular index
 * 7. How to modify/update element in the ArrayList
 * 8. How to access elements from ArrayList
 * 9. How to clear ArrayList
 * 10. How to check whether the ArrayList contains an element
 * 11. How to convert array to ArrayList and ArrayList to array
 */
public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList list=new ArrayList<>();
        list.add("Jannath");
        list.add(31);
        list.add('F');
        list.add(356.123);
        list.add(true);
        list.add("Jannath");
        list.add(null);
        System.out.println(list);
//      list.remove(4);
//      list.remove(356.123);
        list.set(4,false);
        list.add(3,"Sarah");
```

```
        System.out.println(list);
        System.out.println(list.get(5));
        list.clear();
        System.out.println(list);
        System.out.println(list.isEmpty());

//      List<Integer> list=new ArrayList<>();
//      list.add(52);
//      list.add(75);
//      list.add(100);
//      System.out.println(list.size());


        Integer[] array=new Integer[]{1,2,3,4,5};
        List<Integer> list1=new ArrayList<>(Arrays.asList(array));
        System.out.println(list1);

        for(int i=0;i<list1.size();i++){
            System.out.println(list1.get(i));
        }
        for(Integer temp:list1){
            System.out.println(temp);
        }
        Iterator<Integer> it=list1.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}
```

HashSet

```
package com.Day6.Collections;
/*
Hashset is a class implementing set interface
Heterogenous data are allowed
Duplicates are not allowed
```

```
single null value
won't maintain insertion order

 */
/*
 * 1. Declaration
 * 2. Adding Elements
 * 3. Printing HashSet
 * 4. Printing size
 * 5. Remove specific value
 * 6. Inserting Element
 * 7. Accessing specific element
 * 8. Alternative method to access the specific element
 * 9. Read all the elements
 * 10. clear all elements
 * 11. check whether the hashset is empty
 */

import java.util.*;

public class HashSetDemo {
    public static void main(String[] args) {
//      HashSet set=new HashSet<>();
        Set<String> hs=new HashSet<>();
        hs.add("Apple");
        hs.add("Orange");
        hs.add("Banana");
        hs.add("Strawberry");
        System.out.println(hs.size());
        hs.remove("Banana");
        System.out.println(hs);
        Set<String> fruits=new HashSet<>();
        fruits.add("Blueberry");
        fruits.add("Pineapple");
        hs.addAll(fruits);
        System.out.println(hs);

        List<String> list=new ArrayList<>(hs);
```

```
        System.out.println(list.get(2));
        for(String temp:hs){
            System.out.println(temp);
        }

        Iterator<String> it=hs.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }


}
```

HashMap

```
package com.Day6.Collections;
/*
HashMap is a class implementing map interface
Data's are stored in the form of key-value pair
Key should not contains duplicate
Value may contain duplicate

 */


/*
 * 1. Declaration
 * 2. Adding pairs
 * 3. Finding size of hashmap
 * 4. Remove pairs
 * 5. Accessing value through key
 * 6. Retrieve All keys from HashMap
 * 7. Retrieve All values from HashMap
 * 8. Retrieve All keys along with values from HashMap
 * 9. Reading all data from HashMap
 */
```

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class HashMapDemo {
    public static void main(String[] args) {
        Map<Integer,String> hm=new HashMap<>();
        hm.put(101,"Apple");
        hm.put(102,"Strawberry");
        hm.put(103,"Pineapple");
        hm.put(104,"Orange");
        System.out.println(hm.size());
        hm.remove(102);
        System.out.println(hm.get(104));
        System.out.println(hm.keySet());
        System.out.println(hm.values());
        System.out.println(hm.entrySet());
        for(int key:hm.keySet()){
          System.out.println(key+"→"+hm.get(key));
        }
        Iterator<Map.Entry<Integer,String>> it=hm.entrySet().iterator();
        while(it.hasNext()){
            Map.Entry<Integer,String> pair=it.next();
            System.out.println(pair.getKey()+"→"+pair.getValue());
        }
    }
}
```

Program to count the number of occurrence of each element in an Array.

```
package com.Day6.Collections;

import java.util.HashMap;
import java.util.Map;
```

```
public class FrequencyCounter {
    public static void main(String[] args) {
        String[] fruits={"Apple","Orange","Banana","Apple","Orange","Banan
a","Pineapple"};
        Map<String,Integer> freq=new HashMap<>();
        for(String fruit:fruits){
            if(freq.containsKey(fruit)){
                freq.put(fruit,freq.get(fruit)+1);
            }else {
                freq.put(fruit, 1);
            }
        }
        System.out.println(freq);
    }
}
```

Program to remove duplicates from an Array

```
package com.Day6.Collections;

import java.util.HashSet;
import java.util.Set;

public class RemoveDuplicates {
    public static void main(String[] args) {
        String[] fruits={"Apple","Orange","Banana","Apple","Orange","Banan
a","Pineapple"};
        Set<String> hs=new HashSet<>();
        for(String temp:fruits){
            hs.add(temp);
        }
        System.out.println(hs);
    }
}
```